Catalog Manager Reference

This section describes the feature flag bit array, and the data types and functions provided by the Catalog Manager.

Feature Flag Bit Array

Each catalog provides information so that you can determine which features it supports. This information is specified in a feature flag bit array. The bits are defined next.

Bit name

```
kSupportsDNodeNumberBit
kSupportsRecordCreationIDBit
kSupportsAttributeCreationIDBit
kSupportsMatchAllBit

kSupportsBeginsWithBit
kSupportsExactMatchBit
kSupportsEndsWithBit
kSupportsContainsBit
kSupportsOrderedEnumerationBit
kCanSupportNameOrderBit

kCanSupportTypeOrderBit
kSupportSortBackwardsBit
kSupportIndexRatioBit
kSupportsEnumerationContinueBit
kSupportsLookupContinueBit

kSupportsEnumerateAttributeTypeContinueBit
kSupportsEnumeratePseudonymContinueBit
kSupportsAliasesBit
kSupportsPseudonymsBit

kSupportsPartialPathNamesBit
kSupportsAuthenticationBit
kSupportsProxiesBit
kSupportsFindRecordBit
```

Bit descriptions

`kSupportsDNodeNumberBit`
: If this bit is set, you can reference a dNode by using a dNode number in the `RLI` data structure and setting the pathname pointer to `nil`. If this bit is not set, you can reference a dNode only by specifying its pathname information in the `RLI` data structure; in this case, you must set the dNode number to 0.

`kSupportsRecordCreationIDBit`
: If this bit is set, you can reference a record by specifying its record creation ID for most Catalog Manager functions. If this bit is not set, you must reference a record by specifying its record name and record type in its record ID.

`kSupportsAttributeCreationIDBit`
: If this bit is set, you can reference an attribute value by specifying its attribute creation ID and attribute type.

The next five bits indicate what combination of browsing, finding, and matching capabilities a catalog supports when you enumerate the contents of a dNode in that catalog.

`kSupportsMatchAllBit`
: If this bit is set, a catalog supports browsing of record names and record types. When you call the `DirEnumerateGet` function, such a catalog can accept an enumeration specification with the `matchNameHow` and `matchTypeHow` fields set to `kMatchAll`, in which case, a search matches any record name or record type.

`kSupportsBeginsWithBit`
: If this bit is set, a catalog supports finding record names and record types beginning with a certain string. When you call the `DirEnumerateGet` function, such a catalog can accept an enumeration specification with the `matchNameHow` and `matchTypeHow` fields set to `kBeginsWith`; in this case, a search matches any record name or record type that begins with the string pointed to by the `recordName` or `typesList` field, respectively.

`kSupportsExactMatchBit`
: If this bit is set, a catalog supports finding a record based on an exact match with a record name or record type. When you call the `DirEnumerateGet` function, such a catalog can accept an enumeration specification with the `matchNameHow` and `matchTypeHow` fields set to `kMatchExact`; in this case, a search matches only the record name or record type pointed to by the `recordName` or `typesList` field, respectively.

`kSupportsEndsWithBit`
: If this bit is set, a catalog supports finding record names and record types ending with a certain string. When you call the `DirEnumerateGet` function, such a catalog can accept an enumeration specification with the `matchNameHow` and `matchTypeHow` fields set to `kEndingWith`; in this case, a search matches any record name or record type that ends with the string pointed to by the `recordName` or `typesList` field, respectively.

`kSupportsContainsBit`
: If this bit is set, a catalog supports finding record names and record types that contain a certain string. When you call the `DirEnumerateGet` function, such a catalog can accept an enumeration specification with the `matchNameHow` and `matchTypeHow` fields set to `kContaining`; in this case, a search matches any record name or record type that contains the string pointed to by the `recordName` or `typesList` field, respectively.

`kSupportsOrderedEnumerationBit`
: If this bit is set, a catalog returns requested information in a sorted order when you call the `DirEnumerateGet` function. It may return the information sorted by name or by type, in which case one of the two following bits will also be set. The catalog may also return the information in an unspecified sorted order.

`kCanSupportNameOrderBit`
: If this bit is set, a catalog supports the sorting by name option in the `DirEnumerateGet` function.

`kCanSupportTypeOrderBit`
: If this bit is set, a catalog supports the sorting by type option in the `DirEnumerateGet` function.

`kSupportsSortBackwardsBit`
: If this bit is set, a catalog supports the backward sort direction option in the `DirEnumerateGet` function.

`kSupportIndexRatioBit`
: If this bit is set, a catalog supports the index ratio feature in the `DirEnumerateGet` function. That is, the catalog can return the approximate position of a record among all records that match the search criteria in a dNode.

`kSupportsEnumerationContinueBit`
: If this bit is set, a catalog supports the continue feature in the `DirEnumerateGet` function.

`kSupportsLookupContinueBit`
: If this bit is set, a catalog supports the continue feature in the `DirLookupGet` function.

`kSupportsEnumerateAttributeTypeContinueBit`
: If this bit is set, a catalog supports the continue feature in the `DirEnumerateAttributeTypesGet` function.

`kSupportsEnumeratePseudonymContinueBit`
: If this bit is set, a catalog supports the continue feature in the `DirEnumeratePseudonymGet` function.

`kSupportsAliasesBit`
: If this bit is set, a catalog supports the `DirAddAlias` function. It also supports deleting an alias with the `DirDeleteRecord` function and enumerating aliases with the `DirEnumerateGet` function.

`kSupportsPseudonymsBit`
: If this bit is set, a catalog supports the `DirAddPseudonym`, `DirDeletePseudonym`, and `DirEnumeratePseudonymGet` functions. It also supports enumerating pseudonyms with the `DirEnumerateGet` function.

`kSupportsPartialPathnamesBit`
: If this bit is set, you can specify a catalog node by using the dNode number of an intermediate dNode and a partial pathname starting from the intermediate dNode to the target dNode.

`kSupportsAuthenticationBit`
: If this bit is set, a catalog supports all Authentication Manager functions except those that relate to proxies. Support for proxies is specified by a separate bit.

`kSupportsProxiesBit`
: If this bit is set, a catalog supports the Authentication Manager functions that relate to proxies.

`kSupportsFindRecordBit`
: If this bit is set, a catalog supports the `DirFindRecordGet` and `DirFindRecordParse` functions.

You can use the following mask values to set the bits in a variable that specifies the features supported by a given catalog. Such variables are of type `DirGestalt`.

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

Data Types

This section describes the data types that are specific to the Catalog Manager. See the chapter "AOCE Utilities" for descriptions of other data types that you use to provide information to or obtain information from Catalog Manager functions.

The Parameter Block Header

Every Catalog Manager routine takes a pointer to a `DirParamBlock` parameter block as input. The `DirParamBlock` parameter block defines a union of substructures, each of which is a parameter block for one of the Catalog Manager routines. Each routine description in "Catalog Manager Routines" starting on page 8-196 lists the fields of that routine's parameter block. Each parameter block contains the following header.

```
#define  AuthDirParamHeader \
   Ptr             qLink;          /* reserved */\
   long            reserved_H1;    /* reserved */\
   long            reserved_H2;    /* reserved */\
   ProcPtr         ioCompletion;   /* your completion routine */\
   OSErr           ioResult;       /* result code */\
   unsigned long   saveA5;         /* reserved */\
   short           reqCode;        /* CSAM request code*/\
   long            reserved[2];    /* reserved */\
   AddrBlock       serverHint;     /* PowerShare server's AppleTalk address */\
   short           dsRefNum;       /* personal catalog reference number */\
   unsigned long   callID;         /* reserved */\
   AuthIdentity    identity;       /* requester's authentication identity */\
   long            gReserved1;     /* reserved */\
   long            gReserved2;     /* reserved */\
   long            gReserved3;     /* reserved */\
   long            clientData;     /* you define this field */
```

Field descriptions

`qLink`	Reserved.
`reserved_H1`	Reserved.
`reserved_H2`	Reserved.

`ioCompletion`	A pointer to a completion routine that you can provide. When a Catalog Manager function that you called asynchronously completes execution, it calls your completion routine. Set this field to `nil` if you do not wish to provide a completion routine. The function ignores this field if you call it synchronously.
`ioResult`	The result of the function. When you execute the function asynchronously, the function sets this field to 1 as soon as the routine has been queued for execution. When the function completes execution, it sets this field to the actual result code.
`saveA5`	Reserved.
`reqCode`	This field is reserved when you call a Catalog Manager function. However, when the Catalog Manager passes a `DirParamBlock` parameter block to a CSAM, the `reqCode` field contains a constant that identifies which member of the `DirParamBlock` `union` type is being passed
`reserved[2]`	Reserved.
`serverHint`	The AppleTalk address of the PowerShare server to which you want to direct your request. Normally, you specify `nil` for all fields of this structure and the Catalog Manager directs the request to an appropriate PowerShare server. However, PowerShare server administration software (PowerShare Admin) may need to specify a particular server, and the `DirAddADAPDirectory` function requires a specific PowerShare server address. You can obtain the AppleTalk address of a PowerShare server from the `NBPLookup` function. The `AddrBlock` data structure is defined in *Inside Macintosh: Networking*.
`dsRefNum`	The reference number of the personal catalog to which the request applies. The `DirOpenPersonalDirectory` function returns this reference number when you open a personal catalog. If you are not addressing a personal catalog, set this field to 0.
`callID`	Reserved.
`identity`	The authentication identity of the requester. The authentication identity can be either the local identity of the owner of the computer or a specific identity. Typically, you set this field to the local identity to gain transparent access to all installed catalogs. You may also set this field to a specific identity. You can obtain the local identity from the Authentication Manager's `AuthGetLocalIdentity` function and a specific identity from the `AuthBindIdentity` function. See the chapter "Authentication Manager" in this book for more information about obtaining identities. Specify 0 for this field for guest access; that is, no identity. Functions that fail due to an insufficient level of access privilege return either the `kOCEReadAccessDenied` or `kOCEWriteAccessDenied` result code.
`gReserved1`	Reserved.
`gReserved2`	Reserved.
`gReserved3`	Reserved.

`clientData`	Reserved for your use. The Catalog Manager passes the value in this field to your callback routines. If you have the same callback or completion routine processing more than one asynchronous request, your routine can use the `clientData` field to determine for which request it is processing results.

The dNode ID

A dNode ID consists of a dNode number that uniquely identifies a dNode within a catalog plus the name of the dNode. A dNode ID is defined by the `DNodeID` data structure. In the Catalog Manager API, it is not used as a stand-alone data structure; it is a member of the union part of the `DirEnumSpec` data structure, described on page 8-193.

```
struct DNodeID {
   DNodeNum    dNodeNumber;   /* dNode number  */
   long        reserved1;     /* reserved */
   RStringPtr  name;          /* name of the dNode */
   long        reserved2;     /* reserved */
};
```

The Enumeration Choice Type

The bits in a variable of type `DirEnumChoices` indicate types of entities. You use a variable of type `DirEnumChoices` to specify the type of entities about which you want information when you call the `DirEnumerateGet` function.

```
typedef unsigned long DirEnumChoices;
```

The bits in the `DirEnumChoices` data type are defined as follows:

```
enum {
   kEnumDistinguishedNameBit,
   kEnumAliasBit,
   kEnumPseudonymBit,
   kEnumDNodeBit,
   kEnumInvisibleBit
};
```

You can use the following values to set and test the bits in a variable of type `DirEnumChoices`.

```
enum {         /* values of DirEnumChoices */
   kEnumDistinguishedNameMask = 1L<<kEnumDistinguishedNameBit,
   kEnumAliasMask             = 1L<<kEnumAliasBit,
   kEnumPseudonymMask         = 1L<<kEnumPseudonymBit,
   kEnumDNodeMask              = 1L<<kEnumDNodeBit,
   kEnumInvisibleMask          = 1L<<kEnumInvisibleBit
};
```

```
#define kEnumAllMask (kEnumDistinguishedNameMask | kEnumAliasMask |
                    kEnumPseudonymMask | kEnumDNodeMask |
                    kEnumInvisibleMask)
```

Descriptions

`kEnumDistinguishedNameMask`
This setting specifies a record.

`kEnumAliasMask`
This setting specifies an alias.

`kEnumPseudonymMask`
This setting specifies a pseudonym.

`kEnumDNodeMask`
This setting specifies a dNode.

`kEnumInvisibleMask`
As an input, this setting specifies all dNodes, records, aliases, and pseudonyms, both visible and invisible. As an output, it is set in conjunction with either `kEnumDistinguishedNameMask`, `kEnumAliasMask`, `kEnumPseudonymMask`, or `kEnumDNodeMask`, and indicates that the specified entity is invisible.

`kEnumAllMask` As an input, this setting specifies all visible records, aliases, pseudonyms, and dNodes. It is not used as an output.

The Enumeration Specification

The `DirEnumSpec` data structure contains information about either a record, an alias, a pseudonym, or a dNode. The value of the `enumFlag` field indicates the type of entity to which the rest of the information applies as well as the format of that information.

When you want to enumerate the contents of a dNode starting from a specific dNode, record, alias, or pseudonym, you provide a `DirEnumSpec` structure to the `DirEnumerateGet` function that specifies the record, alias, pseudonym, or dNode at which you want the `DirEnumerateGet` function to start the enumeration. The `DirEnumerateParse` function passes a `DirEnumSpec` structure to your callback routine for each record, alias, pseudonym, or dNode that it finds in the buffer.

```
struct DirEnumSpec {
   DirEnumChoices enumFlag;      /* type of entity */
   unsigned short indexRatio;    /* approximate record position */
   union {
      LocalRecordID  recordIdentifier; /* record information */
      DNodeID        dnodeIdentifier;  /* dNode info */
   }u;
};
```

Field descriptions

`enumFlag`	A value that indicates the type of entity about which information is provided in the `u` field. The following constants indicate whether the information applies to a record, an alias, a pseudonym, or a dNode: `kEnumDistinguishedNameMask`, `kEnumAliasMask`, `kEnumPseudonymMask`, or `kEnumDNodeMask`. The `kEnumInvisibleMask` constant indicates whether the entity is invisible or visible.
`indexRatio`	The approximate position, expressed as a percentile ranging from 1 to 100, of a record among all records in a dNode. This is a hint that can be used with a scroll box (or some other mechanism) to show how far you have moved through a list of records. If a catalog does not support this feature, it sets this field to 0.
`u.recordIdentifier`	If the `enumFlag` field is set to `kEnumDistinguishedNameMask`, `kEnumAliasMask` or `kEnumPseudonymMask`, this field contains a `LocalRecordID` data structure. The local record ID specifies a record's name, type, and creation ID.
`u.dnodeIdentifier`	If the `enumFlag` field is set to `kEnumDNodeMask`, this field contains a `DNodeID` data structure. A dNode ID specifies a dNode's name and its dNode number. If a catalog does not support dNode numbers, the `dNodeNumber` field is set to 0.

The Script Structure

The script structure `SLRV`, returned by the `DirEnumerateGet` function, identifies the script, language, and region that the function uses to sort the entries in your buffer.

```
struct SLRV {
   ScriptCode  script;     /*  script code in which entries are sorted */
   short       language;   /*  language code in which entries are sorted */
   short       regionCode; /*  region code in which entries are sorted */
   short       version;    /*  version of AOCE sorting software */
};
```

Field descriptions

`script`	The script code identifies the script that the `DirEnumerateGet` function uses in sorting.
`language`	The language code identifies the language that the `DirEnumerateGet` function uses in sorting.
`regionCode`	The region code identifies the region that the `DirEnumerateGet` function uses in sorting.
`version`	The constant `kCurrentOCESortVersion`. It identifies the version of AOCE sorting software that the `DirEnumerateGet` function uses.

The Matching Criteria Type

You use the `DirMatchWith` data type to indicate a matching mode when you enumerate the contents of a dNode. You always use a variable of type `DirMatchWith` in conjunction with a search string. The `DirMatchWith` variable specifies the criteria that the `DirEnumerateGet` function uses to determine when it has found a match with your search string.

```
typedef unsigned char DirMatchWith;
```

The possible values of the `DirMatchWith` data type are defined as follows:

```
   enum {            /* values of DirMatchWith */
      kMatchAll,
      kExactMatch,
      kBeginsWith,
      kEndingWith,
      kContaining
};
```

Descriptions

`kMatchAll`	Match any string.
`kExactMatch`	Match only those strings that are exactly the same as the search string.
`kBeginsWith`	Match any string that begins with the search string.
`kEndingWith`	Match any string that ends with the search string.
`kContaining`	Match any string that contains the search string.

Catalog Manager Functions

This section describes the Catalog Manager functions that provide services such as getting information about catalogs and dNodes, managing the PowerTalk Setup catalog, managing records and attribute values and types, and controlling access to dNodes, records, and attribute types.

All of the Catalog Manager functions take a pointer to a catalog parameter block as input. Each routine description includes a list of the fields in the parameter block that are used by the function. Each list of parameter block fields has four columns. See the Preface to this book for a description of the type of information that each column contains.

To call a Catalog Manager function from assembly language, push the address of the `DirParamBlock` parameter block and the `async` flag onto the stack using the Pascal calling convention, and place the selector value for the `_oceTBDispatch` trap macro in register D0. Each function description includes the selector value for that function. The function returns its result code in the `ioResult` field of the parameter block.

Getting Information About Catalogs

You can use the functions in this section to get a variety of information about the catalogs that are listed in the PowerTalk Setup catalog. The `DirEnumerateDirectoriesGet` and `DirEnumerateDirectoriesParse` functions work together to provide the catalog name, discriminator value, and feature flags for some or all of the catalogs listed in the PowerTalk Setup catalog. You can discover the features that a specific catalog supports by calling the `DirGetDirectoryInfo` function. The `DirGetLocalNetworkSpec` function provides you with the name of the network on which a PowerShare catalog is located. You can get information about the icons that represent a catalog by calling the `DirGetDirectoryIcon` function. The `DirGetExtendedDirectoriesInfo` function provides additional information about an external catalog that is specific to that catalog.

DirEnumerateDirectoriesGet

The `DirEnumerateDirectoriesGet` function returns information about catalogs that are listed in the PowerTalk Setup catalog.

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

`paramBlock`
: Pointer to a parameter block.

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

Parameter block

→	`ioCompletion`	`ProcPtr`	Your completion routine
←	`ioResult`	`OSErr`	Result code
→	`clientData`	`long`	You define this field
→	`directoryKind`	`OCEDirectoryKind`	Catalog type
→	`startingDirectoryName`	`DirectoryNamePtr`	Starting catalog name
→	`startingDirDiscriminator`	`DirDiscriminator`	Starting discriminator value
→	`includeStartingPoint`	`Boolean`	Begin enumeration with starting point?
↔	`getBuffer`	`Ptr`	Your buffer
→	`getBufferSize`	`unsigned long`	Size of your buffer

See “The Parameter Block Header” on page 8-190 for descriptions of the `ioCompletion`, `ioResult`, and `clientData` fields.

Field descriptions

`directoryKind`
: A value that indicates the type of catalog about which you are requesting information. Use the constant `kDirADAPKind` to request information about PowerShare catalogs. Use the constant `kDirDSAMKind` to request information about external catalogs. To request information about both PowerShare and external catalogs, use the constant `kDirAllKinds`. You can also supply a specific signature value to get information on catalogs having that signature. The function does not return information about personal catalogs.

`startingDirectoryName`
: A pointer to the name of the catalog at which you want the `DirEnumerateDirectoriesGet` function to begin the enumeration. Set this field to `nil` to start with the first catalog. If the `DirEnumerateDirectoriesGet` function completes with the `kOCEMoreData` result code, set this field to the value of the last `dirName` parameter passed to your callback routine by the `DirEnumerateDirectoriesParse` function to continue the enumeration. You must coordinate the value you provide in this field with the value you provide in the `startingDirDiscriminator` field; that is, both values are required, and both must apply to the same catalog.

`startingDirDiscriminator`
: The discriminator value of the catalog at which you want the `DirEnumerateDirectoriesGet` function to begin the enumeration. Set the fields of this structure to 0 to start with the first catalog. If the `DirEnumerateDirectoriesGet` function completes with the `kOCEMoreData` result code, set this field to the value of the last `discriminator` parameter passed to your

	callback routine by the `DirEnumerateDirectoriesParse` function to continue the enumeration. You must coordinate the value you provide in this field with the value you provide in the `startingDirectoryName` field; that is, both values are required, and both must apply to the same catalog
`includeStartingPoint`	A Boolean value that tells the `DirEnumerateDirectoriesGet` function how to interpret the `startingDirectoryName` and `startingDirDiscriminator` fields. Set this field to `true` if you want the `DirEnumerateDirectoriesGet` function to return information about catalogs beginning with the one specified by the `startingDirectoryName` and `startingDirDiscriminator` fields. If you set this field to `false`, the function returns information starting with the catalog immediately *after* the one specified by the `startingDirectoryName` and `startingDirDiscriminator` fields.
`getBuffer`	A pointer to the buffer in which the function stores the name, the discriminator value, and the capability flags for each catalog listed in the PowerTalk Setup catalog. You provide this buffer.
`getBufferSize`	The number of bytes in the buffer.

DESCRIPTION

You call the `DirEnumerateDirectoriesGet` function to obtain information about PowerShare catalogs and external catalogs that are listed in the PowerTalk Setup catalog. You can request information about either PowerShare catalogs or external catalogs, or about both. You can also request information about catalogs that share a specific signature that you specify. For example, if there are several X.500 catalogs listed in the PowerTalk Setup catalog and they used the same signature value, you could request information about that set of catalogs.

For each catalog about which you have requested information, the function places the catalog's name, its discriminator value, and its feature flags in the buffer you provide. If your buffer is not large enough to contain all of the information you requested, the function places as many sets of catalog name, discriminator value, and feature flags as will fit in your buffer and returns the `kOCEMoreData` result code.

When the function completes with either the `noErr` or `kOCEMoreData` result codes, you can provide a pointer to your buffer to the `DirEnumerateDirectoriesParse` function, which extracts the catalog information from the buffer and passes it to a callback routine that you provide.

If your buffer is too small to hold all of the information you requested, you can continue to obtain information by calling the `DirEnumerateDirectoriesGet` function again, after calling the `DirEnumerateDirectoriesParse` function. For the values of the `startingDirectoryName` and `startingDirDiscriminator` fields, use the values that the `DirEnumerateDirectoriesParse` function last passed to the `dirName` and `discriminator` parameters of your callback routine. The `DirEnumerateDirectoriesGet` function will continue the enumeration starting with the next catalog as determined by the value of the `includeStartingPoint` field.

Because personal catalogs are not listed in the PowerTalk Setup catalog, the `DirEnumerateDirectoriesGet` function does not return information about them. To obtain the name, discriminator value, and feature flags of a personal catalog, locate the catalog using the routines in the Standard File Package; open the catalog by calling the `DirOpenPersonalDirectory` function, and call the `DirGetDirectoryInfo` function to get the information.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	$011A

RESULT CODES

`noErr`	0	No error
`kOCEParamErr`	–50	Invalid parameter
`kOCEMoreData`	–1623	More data available

SEE ALSO

The `DirEnumerateDirectoriesParse` function is described next.

The `DirGetDirectoryInfo` function is described on page 8-206.

For an example of continuing the enumeration (using the `DirEnumerateAttributeTypesGet` function) when the buffer is too small to hold all the information you requested, see "Getting Attribute Type Information" on page 8-178.

DirEnumerateDirectoriesParse

The `DirEnumerateDirectoriesParse` function parses the data returned by the `DirEnumerateDirectoriesGet` function and returns information about catalogs, one catalog at a time, by repeatedly calling your callback routine.

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

`paramBlock`
Pointer to a parameter block.

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

Parameter block

→	`ioCompletion`	`ProcPtr`	Your completion routine
←	`ioResult`	`OSErr`	Result code
→	`clientData`	`long`	You define this field
→	`eachDirectory`	`ForEachDirectory`	Your callback routine
→	`getBuffer`	`Ptr`	Your buffer
→	`getBufferSize`	`unsigned long`	Size of your buffer

See "The Parameter Block Header" on page 8-190 for descriptions of the `ioCompletion`, `ioResult`, and `clientData` fields.

Field descriptions

`eachDirectory`	A pointer to your callback routine. The function declaration for this routine is described on page 8-311.
`getBuffer`	A pointer to the buffer containing the information to parse. Use the same buffer that you provided to the `DirEnumerateDirectoriesGet` function.
`getBufferSize`	The number of bytes in the buffer. Use the same value that you provided to the `DirEnumerateDirectoriesGet` function.

DESCRIPTION

You call the `DirEnumerateDirectoriesParse` function to extract the catalog information placed in your buffer by the `DirEnumerateDirectoriesGet` function. You must provide a callback routine that the `DirEnumerateDirectoriesParse` function calls for each set of catalog information that it finds in the buffer. Each time it calls your callback routine, the function passes it the name, discriminator value, and the feature flags of a catalog.

The `DirEnumerateDirectoriesParse` function completes when it has finished parsing the contents of your buffer or when your callback routine returns `true`. The function returns the `kOCEMoreData` result code if it reaches the end of the buffer and finds that the `DirEnumerateDirectoriesGet` function did not return all the data requested. To continue the enumeration, call the `DirEnumerateDirectoriesGet` function again. Get the values of the `dirName` and `discriminator` parameters that the `DirEnumerateDirectoriesParse` function last passed to your callback routine. In your next call to the `DirEnumerateDirectoriesGet` function, use these as the values of the `startingDirectoryName` and `startingDirDiscriminator` fields.

If your callback routine returns `true`, the `DirEnumerateDirectoriesParse` function completes with the `noErr` result code.

Because the `DirEnumerateDirectoriesGet` function returns information about PowerShare and external catalogs only, the `DirEnumerateDirectoriesParse` function can retrieve information only about these types of catalogs. To obtain the name, discriminator value, and feature flags of a personal catalog, locate the catalog using the routines in the Standard File Package; open the catalog by calling the `DirOpenPersonalDirectory` function, and call the `DirGetDirectoryInfo` function to get the information.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	$0106

RESULT CODES

`noErr`	0	No error
`kOCEParamErr`	–50	Invalid parameter
`kOCEMoreData`	–1623	More data available

SEE ALSO

The function declaration for your callback routine is described on page 8-311.

The `DirGetDirectoryInfo` function is described on page 8-206.

The `DirEnumerateDirectoriesGet` function is described on page 8-196.

For an example of continuing the enumeration (using the `DirEnumerateAttributeTypesParse` function) when the buffer is too small to hold all the information you requested, see “Getting Attribute Type Information” on page 8-178.

DirFindRecordGet

The `DirFindRecordGet` function returns information about the records, aliases, and pseudonyms contained in a catalog that you specify.

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

`paramBlock`
Pointer to a parameter block.

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

Parameter block

→	`ioCompletion`	`ProcPtr`	Your completion routine
←	`ioResult`	`OSErr`	Result code
→	`serverHint`	`AddrBlock`	AppleTalk address of the PowerShare server
→	`dsRefNum`	`short`	Personal catalog reference number
→	`identity`	`AuthIdentity`	Requester's authentication identity
→	`clientData`	`long`	You define this field
→	`startingPoint`	`DirEnumSpec*`	Starting point for enumeration
→	`nameMatchString`	`RStringPtr`	Name of record, alias, or pseudonym you want returned
→	`typesList`	`RStringPtr*`	List of types you want returned
→	`typeCount`	`unsigned long`	Number of types in the list
→	`matchNameHow`	`DirMatchWith`	Match criteria for names
→	`matchTypeHow`	`DirMatchWith`	Match criteria for types
↔	`getBuffer`	`Ptr`	Your buffer
→	`getBufferSize`	`unsigned long`	Size of your buffer
→	`directoryName`	`DirectoryNamePtr`	Catalog name
→	`discriminator`	`Discriminator`	Discriminator value

See "The Parameter Block Header" on page 8-190 for descriptions of the `ioCompletion`, `ioResult`, `serverHint`, `dsRefNum`, `identity`, and `clientData` fields.

Field descriptions

`startingPoint`
A pointer to the record, alias, or pseudonym at which you want the function to start the enumeration. Set this field to `nil` when you call the `DirFindRecordGet` function for the first time. If the function completes with the `kOCEMoreData` result code, you can set this field to the value of the last `enumSpec` parameter passed to your callback routine by the `DirFindRecordParse` function to continue the enumeration from the next record, alias, or pseudonym.

`nameMatchString`
A pointer to the name of the record, alias, or pseudonym about which you want information. You specify the mode in which you want the function to match the name in the `matchNameHow` field. If you specify `kMatchAll` in the `matchNameHow` field, the function ignores this field. The `DirFindRecordGet` function returns only records, aliases, or pseudonyms whose names match the value that you specify according to the match criteria that you specify.

`typesList`
A pointer to an array of pointers. Each element in the array points to a record type about which you want information. Your array may include both AOCE-defined record types and record types that you define. You specify the mode in which you want the function to match the type in the `matchTypeHow` field. If you specify `kMatchAll` in the `matchTypeHow` field, the function ignores this field.

`typeCount`
The number of pointers to record types in your array of types.

`matchNameHow`	A value that specifies the matching mode by which the function determines matches with the name you provide in the `nameMatchString` field. The possible values for exact and wildcard matching are described on page 8-195.
`matchTypeHow`	A value that specifies the matching mode by which the function determines matches with the values you provide in the `typesList` field. The possible values for exact and wildcard matching are described on page 8-195. If you specify `kMatchAll`, the function returns information on each record, alias, or pseudonym whose name matches the value pointed to by the `nameMatchString` field.
`getBuffer`	A pointer to the buffer in which the function stores the requested information. You provide this buffer.
`getBufferSize`	The number of bytes in the buffer.
`directoryName`	A pointer to the name of the catalog whose records you want to enumerate. You provide the name buffer.
`discriminator`	A unique value associated with a catalog that distinguishes it from other catalogs with the same name.

DESCRIPTION

You call the `DirFindRecordGet` function to obtain a list of records, aliases, pseudonyms, or all of these for a catalog that you specify. This function allows you to specify matching criteria for both names and types.

The sort order of the information returned by the function is undefined.

The `DirFindRecordGet` function places a local record ID for each record, alias, or pseudonym that it finds in your buffer. The function provides only whole units of information for each entity. That is, it will not provide the creation ID for a record without also providing its name and type. If your buffer is not large enough to contain all of the information requested, the `DirFindRecordGet` function provides complete information on as many records, aliases, or pseudonyms as will fit and returns the `kOCEMoreData` result code.

When the function completes with either the `noErr` or `kOCEMoreData` result codes, you use a pointer to your buffer as input to the `DirFindRecordParse` function, which extracts the information from the buffer.

If the `DirFindRecordGet` function returned the `kOCEMoreData` result code, you can request additional information by calling it again after calling the `DirFindRecordParse` function. Get the value of the `enumSpec` parameter that the `DirFindRecordParse` function last passed to your callback routine. When you call `DirFindRecordParse` again, use this value in the `startingPoint` field. Use the same values for the `nameMatchString` and `typesList` fields that you used in your original call to the `DirFindRecordGet` function. The `DirFindRecordGet` function will continue the enumeration starting with the next record, alias, or pseudonym.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	$0140

RESULT CODES

`noErr`	0	No error
`kOCEMoreData`	–1623	More data available

SEE ALSO

The `DirFindRecordParse` function is described next.

For an example of continuing the enumeration (using the `DirEnumerateAttributeTypesGet` function) when the buffer is too small to hold all the information you requested, see “Getting Attribute Type Information” on page 8-178.

To obtain the value for the `discriminator` field, call the `DirGetDirectoryInfo` function on page 8-206.

DirFindRecordParse

The `DirFindRecordParse` function parses the data returned by the `DirFindRecordGet` function and returns information on each record, alias, or pseudonym by repeatedly calling your callback routine.

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

`paramBlock`
: Pointer to a parameter block.

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

Parameter block

→	`ioCompletion`	`ProcPtr`	Your completion routine
←	`ioResult`	`OSErr`	Result code
→	`serverHint`	`AddrBlock`	AppleTalk address of the PowerShare server
→	`dsRefNum`	`short`	Personal catalog reference number
→	`identity`	`AuthIdentity`	Requester’s authentication identity
→	`clientData`	`long`	You define this field
→	`startingPoint`	`DirEnumSpec *`	Starting point for enumeration
→	`nameMatchString`	`RStringPtr`	Name of record, alias, or pseudonym you want returned
→	`typesList`	`RStringPtr *`	List of types you want returned

→	`typeCount`	`unsigned long`	Number of types in the list
→	`matchNameHow`	`DirMatchWith`	Match criteria for names
→	`matchTypeHow`	`DirMatchWith`	Match criteria for types
→	`getBuffer`	`Ptr`	Your buffer
→	`getBufferSize`	`unsigned long`	Size of your buffer
→	`directoryName`	`DirectoryNamePtr`	Catalog name
→	`discriminator`	`Discriminator`	Discriminator value
→	`forEachRecordFunc`	`ForEachRecord`	Your callback routine

See "The Parameter Block Header" on page 8-190 for descriptions of the `ioCompletion`, `ioResult`, `serverHint`, `dsRefNum`, `identity`, and `clientData` fields.

Field descriptions

`startingPoint`	Use the value you provided in the `startingPoint` field of the `DirFindRecordGet` function.
`nameMatchString`	Use the value you provided in the `nameMatchString` field of the `DirFindRecordGet` function.
`typesList`	Use the value you provided in the `typesList` field of the `DirFindRecordGet` function.
`typeCount`	Use the value you provided in the `typeCount` field of the `DirFindRecordGet` function.
`matchNameHow`	Use the value you provided in the `matchNameHow` field of the `DirFindRecordGet` function.
`matchTypeHow`	Use the value you provided in the `matchTypeHow` field of the `DirFindRecordGet` function.
`getBuffer`	Use the value you provided in the `getBuffer` field of the `DirFindRecordGet` function.
`getBufferSize`	Use the value you provided in the `getBufferSize` field of the `DirFindRecordGet` function.
`directoryName`	Use the value you provided in the `directoryName` field of the `DirFindRecordGet` function.
`discriminator`	Use the value you provided in the `discriminator` field of the `DirFindRecordGet` function.
`forEachRecordFunc`	A pointer to your callback routine.

DESCRIPTION

You call the `DirFindRecordParse` function to extract the information that the `DirFindRecordGet` function placed in your buffer. You must provide a callback routine that the `DirFindRecordParse` function calls for each record, alias, or pseudonym about which there is information in the buffer. The `DirFindRecordParse` function provides a local record ID for each record, alias, or pseudonym. See the description of your callback routine on page 8-317 for more information.

The `DirFindRecordParse` function completes when it has finished parsing the contents of your buffer or when your callback routine returns `true`. The function returns the `kOCEMoreData` result code if it reaches the end of the buffer and finds that the `DirFindRecordGet` function did not return all the data requested. To continue the enumeration, call the `DirFindRecordGet` function again. In your next call to the `DirFindRecordGet` function, for the value of the `startingPoint` field, use the value that your callback routine last received in the `enumSpec` parameter.

If your callback routine returns `true`, the `DirFindRecordParse` function completes with the `noErr` result code.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	$0141

RESULT CODES

`noErr`	0	No error
`kOCEParamErr`	–50	Invalid parameter
`kOCEMoreData`	–1623	More data available

SEE ALSO

The function declaration for your callback routine is described on page 8-317.

The `DirFindRecordGet` function is described on page 8-201.

For an example of continuing the enumeration (using the `DirEnumerateAttributeTypesParse` function) when the buffer is too small to hold all the information you requested, see "Getting Attribute Type Information" on page 8-178.

DirGetDirectoryInfo

The `DirGetDirectoryInfo` function returns information about a catalog that you specify.

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

`paramBlock`
Pointer to a parameter block.

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

Parameter block

→	`ioCompletion`	`ProcPtr`	Your completion routine
←	`ioResult`	`OSErr`	Result code
→	`serverHint`	`AddrBlock`	AppleTalk address of the PowerShare server
→	`dsRefNum`	`short`	Personal catalog reference number
→	`identity`	`AuthIdentity`	Requester's authentication identity
→	`clientData`	`long`	You define this field
↔	`directoryName`	`DirectoryNamePtr`	The name of the catalog
↔	`discriminator`	`DirDiscriminator`	Discriminator value
←	`features`	`DirGestalt`	Feature flags

See "The Parameter Block Header" on page 8-190 for descriptions of the `ioCompletion`, `ioResult`, `serverHint`, `dsRefNum`, `identity`, and `clientData` fields.

Field descriptions

`directoryName`	A pointer to the name of the catalog about which you want information. You provide the name buffer. You specify the catalog name unless you are requesting information about a personal catalog. In that case, you may provide either the personal catalog's name and discriminator value or its reference number. If you specify its reference number in the `dsRefNum` field, the function returns, in the buffer supplied for the `directoryName` field, the volume name on which the personal catalog resides. To obtain the file specification for the personal catalog, call the `DirMakePersonalDirectoryRLI` function first. Then call `OCEExtractAlias` using the record location information you obtained to extract the File Manager alias for the personal catalog.
`discriminator`	A unique value that distinguishes a catalog from other catalogs with the same name. You specify this field unless you are requesting information about a personal catalog. In that case, you may provide either the personal catalog's name and discriminator value or its reference number. If you specify its reference number in the `dsRefNum` field, the function returns the discriminator value in this field.
`features`	A set of bit flags that describe the features that a catalog supports. The function returns these flags for the catalog that you specify.

DESCRIPTION

You call the `DirGetDirectoryInfo` function to determine the features that a catalog supports before calling other Catalog Manager functions that address that catalog.

In addition to returning a catalog's feature flags, the `DirGetDirectoryInfo` function may also return the name and discriminator value for a catalog. The function first examines the `dsRefNum` field. If you specify a nonzero value for the `dsRefNum` field (that is, if your target catalog is a personal catalog), the `DirGetDirectoryInfo` function returns the name, the discriminator value, and the feature flags for the personal catalog that you identified. If the dsRefNum field is set to 0, the function examines the serverHint field. A special case arises when you request information about a PowerShare catalog and you specify the AppleTalk address of a server for that catalog in the serverHint field. In this case, you do not need to provide the catalog name and discriminator. The function returns those values as well the feature flags.

To test the bits in the features field, you can use the mask values shown on page 8-189.

Note
The DirEnumerateDirectoriesGet function also returns the name, discriminator value, and feature flags for PowerShare and external catalogs. Unlike the DirGetDirectoryInfo function, which requires that you know some information about a specific catalog before you can request additional information about that catalog, the DirEnumerateDirectoriesGet function returns catalog information without you needing to provide any. However, the DirEnumerateDirectoriesGet function returns information only about the PowerShare and external catalogs listed in the PowerTalk Setup catalog. ◆

SPECIAL CONSIDERATIONS

The DirFindADAPDirectoryByNetSearch and DirAddADAPDirectory functions allow you to make a catalog available for your use without adding it to the PowerTalk Setup catalog. You can call the DirGetDirectoryInfo function for any catalog that you have made privately available.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
_oceTBDispatch	$0119

RESULT CODES

noErr	0	No error
kOCEParamErr	–50	Invalid parameter
kOCETargetDirectoryInaccessible	–1613	Target catalog is not currently available

SEE ALSO

The DirEnumerateDirectoriesGet function is described on page 8-196.

You obtain a reference number for a personal catalog from the DirOpenPersonalDirectory function, which is described on page 8-242.

DirGetLocalNetworkSpec

The `DirGetLocalNetworkSpec` function returns the name of the network on which a PowerShare catalog resides.

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

`paramBlock`
: Pointer to a parameter block.

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

Parameter block

→	`ioCompletion`	`ProcPtr`	Your completion routine
←	`ioResult`	`OSErr`	Result code
→	`serverHint`	`AddrBlock`	AppleTalk address of the PowerShare server
→	`dsRefNum`	`short`	Personal catalog reference number
→	`identity`	`AuthIdentity`	Requester's authentication identity
→	`clientData`	`long`	You define this field
→	`directoryName`	`DirectoryNamePtr`	Catalog name
→	`discriminator`	`DirDiscriminator`	Discriminator value
↔	`networkSpec`	`NetworkSpecPtr`	Network name

See "The Parameter Block Header" on page 8-190 for descriptions of the `ioCompletion`, `ioResult`, `serverHint`, `dsRefNum`, `identity`, and `clientData` fields.

Field descriptions

`directoryName`
: A pointer to the name of the PowerShare catalog to which the request applies.

`discriminator`
: The discriminator value of the PowerShare catalog to which the request applies. A catalog discriminator differentiates between two or more catalogs with the same name.

`networkSpec`
: A pointer to a buffer in which the function places the name of the network in which the catalog resides. You provide this buffer. The buffer should be big enough to hold a maximum size `NetworkSpec` data structure.

DESCRIPTION

You call the `DirGetLocalNetworkSpec` function when you want to know the name of the network on which a specific ADAP catalog resides. You provide the catalog name and discriminator value. The function returns in the `networkSpec` field a pointer to the name of the network. The information that this function provides may be useful in an environment containing multiple interconnected networks.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	**Selector**
`_oceTBDispatch`	$0124

RESULT CODES

`noErr`	0	No error
`kOCEParamErr`	–50	Invalid parameter
`kOCETargetDirectoryInaccessible`	–1613	Target catalog is not currently available

SEE ALSO

The `NetworkSpec` data structure is described in the chapter “AOCE Utilities” in this book.

DirGetDirectoryIcon

The `DirGetDirectoryIcon` function returns information about an icon representing a catalog that you specify.

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

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

Parameter block

→	`ioCompletion`	`ProcPtr`	Your completion routine
←	`ioResult`	`OSErr`	Result code
→	`serverHint`	`AddrBlock`	AppleTalk address of the PowerShare server
→	`dsRefNum`	`short`	Personal catalog reference number
→	`identity`	`AuthIdentity`	Requester's authentication identity
→	`clientData`	`long`	You define this field
→	`pRLI`	`PackedRLIPtr`	Target catalog
→	`iconType`	`OSType`	The type of icon
↔	`iconBuffer`	`Ptr`	Your buffer
↔	`bufferSize`	`unsigned long`	Size of buffer on input; data bytes in buffer on output

See "The Parameter Block Header" on page 8-190 for descriptions of the `ioCompletion`, `ioResult`, `serverHint`, `dsRefNum`, `identity`, and `clientData` fields.

Field descriptions

`pRLI`	A pointer to packed record location information for the catalog whose icon you want to obtain. The function ignores this field when you provide a nonzero value in the `dsRefNum` field to specify a personal catalog.
`iconType`	The type of icon about which you want information. Specify one of the following: '`ICN#`', '`icl8`', '`icl4`', '`ics8`' '`ics4`', or '`ics#`'.
`iconBuffer`	A pointer to the buffer in which the function stores the icon data. You provide this buffer.
`bufferSize`	On input, you set this field to the size of the buffer pointed to by the `iconBuffer` field. On output, the function sets this field to the size of the icon it placed in your buffer. If the function completes with the `kOCEBufferTooSmall` result code, it sets this field to the size of the icon.

DESCRIPTION

You call the `DirGetDirectoryIcon` function to get icon information for a catalog so that you may display the icon.

This function is not supported by PowerShare and personal catalogs. A catalog service access module may support this function for its catalog.

If your buffer is not large enough to hold the icon you requested, the function returns the `kOCEBufferTooSmall` result code. In that case, the `bufferSize` field contains the size of the icon. You should increase the size of your buffer to the icon size and call the function again.

SPECIAL CONSIDERATIONS

Apple Computer, Inc., does not publish the size of icon resources. They are subject to change.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	$0121

RESULT CODES

`noErr`	0	No error
`kOCEBufferTooSmall`	–1503	Buffer too small for data requested

SEE ALSO

You can create a `PackedRLI` data structure with the `OCEPackRLI` utility routine. It is described in the chapter “AOCE Utilities” in this book.

For information about the different icon types and the format of the data associated with those types, see the chapter “Finder Interface” in *Inside Macintosh: Macintosh Toolbox Essentials*.

DirGetExtendedDirectoriesInfo

The `DirGetExtendedDirectoriesInfo` function returns extended information about catalogs.

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

`paramBlock`
: Pointer to a parameter block.

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

Parameter block

→	`ioCompletion`	`ProcPtr`	Your completion routine
←	`ioResult`	`OSErr`	Result code
→	`serverHint`	`AddrBlock`	AppleTalk address of the PowerShare server
→	`dsRefNum`	`short`	Personal catalog reference number
→	`identity`	`AuthIdentity`	Requester’s authentication identity
→	`clientData`	`long`	You define this field
↔	`buffer`	`Ptr`	Your output buffer
→	`bufferSize`	`unsigned long`	Size of buffer;
←	`totalEntries`	`unsigned long`	Number of catalogs found
←	`actualEntries`	`unsigned long`	Number of entries returned

See "The Parameter Block Header" on page 8-190 for descriptions of the `ioCompletion`, `ioResult`, `serverHint`, `dsRefNum`, `identity`, and `clientData` fields.

Field descriptions

`buffer`	A pointer to your buffer in which the function stores the information you request.
`bufferSize`	The number of bytes in your buffer. You set this field to the size of your buffer in bytes.
`totalEntries`	The total number of external catalogs that the `DirGetExtendedDirectoriesInfo` function found listed in the PowerTalk Setup catalog.
`actualEntries`	The number of catalogs about which the function has returned information in your buffer.

DESCRIPTION

You call the `DirGetExtendedDirectoriesInfo` function to get information about catalogs. The function provides more information than is available from the `DirEnumerateDirectoriesGet` function. For example, it might return information on the addressing scheme used by an external catalog. Typically, an AOCE address template calls this function to help construct an address for a messaging service access module. Unlike the `DirEnumerateDirectoriesGet` function, `DirGetExtendedDirectoriesInfo` has no associated parse routine. Thus, you must parse the contents of the buffer yourself.

For each catalog, the `DirGetExtendedDirectoriesInfo` function stores information in your buffer in the following format:

```
struct EachDirectoryData {
   PackedRLI      pRLI;          /* packed RLI for catalog */

   OSType         entnType;      /* address type */
   long           hasMailSlot;   /* catalog has mail slot? */
   ProtoRString   realName;      /* real name */
   ProtoRString   comment;       /* comment for display */
   long           length;        /* data length */
   char           data[length];  /* data */
};
```

Field descriptions

`pRLI`	Packed record location information that identifies the catalog.
`entnType`	The address type.
`hasMailSlot`	The `DirGetExtendedDirectoriesInfo` function sets this field to 1 if the catalog is associated with a mail slot. Otherwise, it sets this field to 0.

`realName`	The name of the catalog in its native environment. This may differ from its catalog name within an AOCE system. It is word aligned.
`comment`	Information that the catalog provider stores in its record in the PowerTalk Setup catalog for display to a user. Typically, this information further identifies and describes the catalog to the user. For example, it might say "This catalog is located in Paris, France. You are connected to it via a public packet-switched network." It is word aligned.
`length`	The number of bytes in the `data` field.
`data`	Information about the catalog, padded to an even boundary.

Your buffer must be large enough to accommodate the total number of entries that the function finds. If your buffer is not large enough to hold all of the information, the function completes with the `kOCEMoreData` result code. In that case, use the value of the `totalEntries` field as a guide in allocating a bigger buffer and then call the function again. Because the function returns data that is of variable length for each catalog, this is a trial-and-error method.

Note that there is no way to have the `DirGetExtendedDirectoriesInfo` function return only data it has not previously returned. It always attempts to return information on every catalog that it finds.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	$0136

RESULT CODES

`noErr`	0	No error
`kOCEMoreData`	–1623	More data available

SEE ALSO

The `DirEnumerateDirectoriesGet` function is described on page 8-196.

For information on messaging service access modules, see the chapter "Messaging Service Access Modules" in *Inside Macintosh: AOCE Service Access Modules*.

The chapter "Service Access Module Setup" in *Inside Macintosh: AOCE Service Access Modules* describes address templates.

For an example of using the `DirEnumerateDirectoriesGet` function, see "Getting Extended Catalog Information" beginning on page 8-182.

Getting Information About dNodes

You can use the functions in this section to get a variety of information about dNodes. The `DirEnumerateGet` and `DirEnumerateParse` functions work together to provide information about the contents of a dNode. You can detect changes in a dNode by calling the `DirGetDNodeMetaInfo` function which indicates whether a specific dNode is a leaf node in a catalog tree. If you know the pathname information for a dNode, you can obtain its dNode number and vice versa by using the functions `DirMapDNodeNumberToPathName` and `DirMapPathNameToDNodeNumber`.

DirEnumerateGet

The `DirEnumerateGet` function returns information about the contents of a dNode that you specify. The contents of a dNode include records, aliases, pseudonyms, and dNodes.

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

`paramBlock`
Pointer to a parameter block.

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

Parameter block

→	`ioCompletion`	`ProcPtr`	Your completion routine
←	`ioResult`	`OSErr`	Result code
→	`serverHint`	`AddrBlock`	AppleTalk address of the PowerShare server
→	`dsRefNum`	`short`	Personal catalog reference number
→	`identity`	`AuthIdentity`	Requester's authentication identity
→	`clientData`	`long`	You define this field
→	`aRLI`	`PackedRLIPtr`	Target dNode
→	`startingPoint`	`DirEnumSpec*`	Starting point for enumeration
→	`sortBy`	`DirSortOption`	Return data in name or type order
→	`sortDirection`	`DirSortDirection`	Search forward or backward for info
→	`nameMatchString`	`RStringPtr`	Name of record, alias, pseudonym, or dNode you want returned
→	`typesList`	`RStringPtr*`	List of types you want returned
→	`typeCount`	`unsigned long`	Number of types in the list
→	`enumFlags`	`DirEnumChoices`	Types of entities about which you want information
→	`includeStartingPoint`	`Boolean`	Begin enumeration with starting point?
→	`matchNameHow`	`DirMatchWith`	Match criteria for names
→	`matchTypeHow`	`DirMatchWith`	Match criteria for types
↔	`getBuffer`	`Ptr`	Your buffer
→	`getBufferSize`	`unsigned long`	Size of your buffer
←	`responseSLRV`	`SLRV`	Script information

See "The Parameter Block Header" on page 8-190 for descriptions of the `ioCompletion`, `ioResult`, `serverHint`, `dsRefNum`, `identity`, and `clientData` fields.

Field descriptions

`aRLI`	A pointer to packed record location information that identifies the dNode for which you want a list of records, aliases, pseudonyms, or dNodes. You use the `enumFlags` field to specify the type of entity about which you want information. The function ignores the `aRLI` field when you provide a nonzero value in the `dsRefNum` field to specify a personal catalog.
`startingPoint`	A pointer to the record, alias, pseudonym, or dNode at which you want the function to start the enumeration. You specify the type of entity in the `enumFlag` field of the `DirEnumSpec` data structure and provide either a `LocalRecordID` or a `DNodeID` data structure to identify the specific entity from which you want the function to start returning information. Set this field to `nil` to start with the first record, alias, pseudonym, or dNode in the dNode. If the `DirEnumerateGet` function completes with the `kOCEMoreData` result code, you can continue the enumeration as follows: Set the `startingPoint` field to the value of the last `enumSpec` parameter passed to your callback routine by the `DirEnumerateParse` function.
`sortBy`	A constant that specifies whether the function returns the records and dNodes sorted by name or sorted by type. Set this field to the constant `kSortByName` if you want the data ordered alphabetically by name. Set this field to the constant `kSortByType` if you want the data ordered alphabetically by type.
`sortDirection`	A constant that specifies whether the function returns the information you requested in forward sort order or reverse sort order. Set this field to the constant `kSortForwards` if you want your data in forward sort order. Set it to the constant `kSortBackwards` if you want your data in reverse sort order.
`nameMatchString`	A pointer to the name of the record, alias, pseudonym, or dNode about which you want information. Use the `matchNameHow` field to specify the mode in which you want the function to match the name. If you specify `kMatchAll` in the `matchNameHow` field, the function ignores this field. The `DirEnumerateGet` function returns only records, aliases, pseudonyms, or dNodes whose names match the value that you specify according to the match criteria that you specify.
`typesList`	A pointer to an array of pointers. Each element in the array points to a record type about which you want information. Your array may include both AOCE-defined record types and record types that you define. In the `matchTypeHow` field, specify the mode in which you want the function to match the type. If you specify `kMatchAll` in the `matchTypeHow` field, the function ignores this field.
`typeCount`	The number of pointers to record types in your array of types.

`enumFlags`	A mask value that specifies whether you want the `DirEnumerateGet` function to return information about records, aliases, pseudonyms, dNodes, or some combination of these. The mask constants that you can specify are described in the section "The Enumeration Choice Type" on page 8-192. With the `enumFlag` field of the `DirEnumSpec` data structure and with either a `LocalRecordID` or a `DNodeID` data structure that you provide in that data structure, you identify the specific entity from which you want the function to start returning information.
`includeStartingPoint`	A Boolean value that tells the function how to interpret the `startingPoint` field. Set `includeStartingPoint` to `true` if you want `DirEnumerateGet` to return information beginning with the entity specified by the `startingPoint` field. Set this field to `false` if you want the `DirEnumerateGet` function to return information beginning with the entity immediately after the entity specified by the `startingPoint` field.
`matchNameHow`	A value that specifies the matching mode used to determine matches with the name specified by `nameMatchString`. The possible values for exact and wildcard matching are described on page 8-195.
`matchTypeHow`	A value that specifies the matching mode used to determine matches with the values specified by `typesList`. The possible values for matching are described in "The Matching Criteria Type" on page 8-195. If you specify `kMatchAll`, the function returns information on each instance of a target entity whose name matches the value pointed to by the `nameMatchString` field. (You specify target entities in the `enumFlags` field.)
`getBuffer`	A pointer to the buffer in which the function stores the requested information. You provide this buffer.
`getBufferSize`	The number of bytes in the buffer.
`responseSLRV`	A structure in which the function returns the script code, language code, and region code of the character set that the function used to sort the entries in your buffer.

DESCRIPTION

You call the `DirEnumerateGet` function to obtain a list of records, aliases, pseudonyms, dNodes, or any combination of these for a dNode that you specify. This function allows you to specify a starting point for the enumeration, a sort indicator (by name or by type), and a sort direction, as well as matching criteria for both names and types.

Note that a given catalog may not support the sort indicator that you specify. For example, a catalog may support an ordered enumeration by creation times, but not a sort by name or by type. Your results would come back in an unspecified order. (A catalog indicates its sorting capabilities through its feature flags. See "Feature Flag Bit Array" beginning on page 8-186 for more information.)

The sort order of the data returned to you is determined by the target catalog's sorting capabilities and the value you provide in the `sortDirection` field. If the catalog supports sorting by name or sorting by type, the data is sorted in alphabetical or reverse-alphabetical order. If the catalog supports an unspecified ordered enumeration, the catalog determines the meaning of a forward or backward order. For example, if a catalog supports only an ordered enumeration by creation times, it may return the data in a most recent first or oldest first order.

PowerShare and personal catalogs do not provide secondary sorting. If you specify sorting by name and there are several entities with the same name, those entities are not additionally sorted by type. Similarly, if you specify sorting by type, entities of the same type are not additionally sorted by name. Some external catalogs may have a secondary sort capability; however, the `DirEnumerateGet` function does not provide a way for you to specify a secondary sort order.

Note

The `enumFlags` field indicates the type of entity about which you want information. You can set it to any combination of the mask constants `kEnumDistinguishedNameMask`, `kEnumAliasMask`, `kEnumPseudonymMask`, and `kEnumDNodeMask` to request information about records, aliases, pseudonyms, and dNodes, respectively. If you want information about all visible entities, set the mask to `kEnumAllMask`. If you want information about all entities, visible and invisible, set the mask to `kEnumInvisibleMask`. ◆

If the `DirEnumerateGet` function is enumerating dNodes, it obtains a dNode name and number for each dNode and places the names and numbers in your buffer. If the `DirEnumerateGet` function is enumerating records, aliases, or pseudonyms, it obtains a local record ID for each record, alias, or pseudonym and places these IDs in your buffer. The function provides only whole units of information for each entity. That is, it will not provide the name for a dNode without also providing the dNode number. Similarly, it will not provide the creation ID for a record without also providing its name and type. If your buffer is not large enough to contain all of the information requested, the `DirEnumerateGet` function provides complete information on as many records, aliases, pseudonyms, or dNodes as will fit and returns the `kOCEMoreData` result code.

When the function completes with either the `noErr` or `kOCEMoreData` result codes, you use a pointer to your buffer as input to the `DirEnumerateParse` function, which extracts the information from the buffer.

If the `DirEnumerateGet` function returns the `kOCEMoreData` result code, you can request additional information by calling it again after calling the `DirEnumerateParse` function. In your next call to the `DirEnumerateGet` function, for the value of the `startingPoint` field, use the value that your callback routine last received in the `enumSpec` parameter. Use the same values for the `aRLI`, `nameMatchString`, and `typesList` fields that you used in your original call to the `DirEnumerateGet` function. The `DirEnumerateGet` function continues the enumeration starting with the next entity as determined by the value of the `includeStartingPoint` field.

To enumerate the contents of the root node of a PowerShare or external catalog, construct a `PackedRLI` data structure in which the dNode number is set to `kRootDNodeNumber` and the pointer to the pathname is set to `nil`. Then set the `aRLI` field to point to your `PackedRLI` data structure.

SPECIAL CONSIDERATIONS

If you target a PowerShare or personal catalog and you specify sorting by type, you can provide only one type in the types list. If you provide more than one type, the function returns an error.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	**Selector**
`_oceTBDispatch`	0x111

RESULT CODES

`noErr`	0	No error
`kOCEReadAccessDenied`	–1540	Identity lacks read access privileges
`kOCEUnknownID`	–1567	Authentication identity is not valid
`kOCETargetDirectoryInaccessible`	–1613	Target catalog is not currently available
`kOCENoSuchDNode`	–1615	Can't find specified dNode
`kOCEMoreData`	–1623	More data available
`kOCEStreamCreationErr`	–1625	Error in creating connection to server

SEE ALSO

The `PackedRLI` data structure is described in the chapter "AOCE Utilities" in this book.

To create a `PackedRLI` data structure use the `OCEPackRLI` utility routine, also described in the chapter "AOCE Utilities."

The `DirEnumSpec` data structure is described on page 8-193.

The `DirEnumerateParse` function is described next.

Feature flags are described in "Feature Flag Bit Array" beginning on page 8-186.

The enumeration mask constants are described in the section "The Enumeration Choice Type" on page 8-192.

For an example of continuing the enumeration (using the `DirEnumerateAttributeTypesGet` function) when the buffer is too small to hold all the information you requested, see "Getting Attribute Type Information" beginning on page 8-178.

DirEnumerateParse

The `DirEnumerateParse` function parses the data returned by the `DirEnumerateGet` function and returns information on each record, alias, pseudonym, or dNode by repeatedly calling your callback routine.

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

`paramBlock`
: Pointer to a parameter block.

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

Parameter block

→	`ioCompletion`	`ProcPtr`	Your completion routine
←	`ioResult`	`OSErr`	Result code
→	`serverHint`	`AddrBlock`	AppleTalk address of the PowerShare server
→	`dsRefNum`	`short`	Personal catalog reference number
→	`identity`	`AuthIdentity`	Requester's authentication identity
→	`clientData`	`long`	You define this field
→	`aRLI`	`PackedRLIPtr`	Target dNode
→	`eachEnumSpec`	`ForEachDirEnumSpec`	Your callback routine
→	`getBuffer`	`Ptr`	Your buffer
→	`getBufferSize`	`unsigned long`	Size of your buffer

See "The Parameter Block Header" on page 8-190 for descriptions of the `ioCompletion`, `ioResult`, `serverHint`, `dsRefNum`, `identity`, and `clientData` fields.

Field descriptions

`aRLI`	The pointer to the dNode for which you want a list of records, aliases, pseudonyms, or dNodes. Use the same value that you provided to the associated `DirEnumerateGet` function.
`eachEnumSpec`	The pointer to your callback routine. The function declaration for this routine is described on page 8-315.
`getBuffer`	A pointer to the buffer containing the information to parse. Use the same buffer that you provided to the `DirEnumerateGet` function.
`getBufferSize`	The number of bytes in the buffer. Use the same value that you provided to the associated `DirEnumerateGet` function.

DESCRIPTION

You call the `DirEnumerateParse` function to extract the information placed in your buffer by the `DirEnumerateGet` function. You must provide a callback routine that the `DirEnumerateParse` function calls for each record, alias, pseudonym, or dNode about which there is information in the buffer. The `DirEnumerateParse` function provides the dNode name and number if the entity about which it returns information is a dNode. It provides a local record ID if the entity is a record, an alias, or a pseudonym. See the description of your callback routine on page 8-315 for more information.

The `DirEnumerateParse` function completes when it has finished parsing the contents of your buffer or when your callback routine returns `true`. The function returns the `kOCEMoreData` result code if it reaches the end of the buffer and finds that the `DirEnumerateGet` function did not return all the data requested. To continue the enumeration, call the `DirEnumerateGet` function again. For the value of the `startingPoint` field, use the value that your callback routine last received in the `enumSpec` parameter.

If your callback routine returns `true`, the `DirEnumerateParse` function completes with the `noErr` result code.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	0x101

RESULT CODES

`noErr`	0	No error
`kOCEParamErr`	–50	Invalid parameter
`kOCEMoreData`	–1623	More data available

SEE ALSO

The function declaration for your callback routine is described on page 8-315.

The `DirEnumerateGet` function is described on page 8-215.

The `PackedRLI` data structure is described in the chapter "AOCE Utilities" in this book.

For an example of continuing the enumeration (using the `DirEnumerateAttributeTypesParse` function) when the buffer is too small to hold all the information you requested, see "Getting Attribute Type Information" beginning on page 8-178.

DirGetDNodeMetaInfo

The `DirGetDNodeMetaInfo` function returns a numeric value that you can use to determine whether a dNode has changed since you last called this function.

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

`paramBlock`
Pointer to a parameter block.

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

Parameter block

→	`ioCompletion`	`ProcPtr`	Your completion routine
←	`ioResult`	`OSErr`	Result code
→	`serverHint`	`AddrBlock`	AppleTalk address of the PowerShare server
→	`dsRefNum`	`short`	Personal catalog reference number
→	`identity`	`AuthIdentity`	Requester's authentication identity
→	`clientData`	`long`	You define this field
→	`pRLI`	`PackedRLIPtr`	Target dNode
←	`metaInfo`	`DirMetaInfo`	Comparison value

See "The Parameter Block Header" on page 8-190 for descriptions of the `ioCompletion`, `ioResult`, `serverHint`, `dsRefNum`, `identity`, and `clientData` fields.

Field descriptions

`pRLI` A pointer to packed record location information that identifies the dNode to which the request applies. The function ignores this field when you provide a non-zero value in the `dsRefNum` field to specify a personal catalog.

`metaInfo` A numeric value that the `DirGetDNodeMetaInfo` function returns. The Catalog Manager updates this value when a catalog node changes. You use it to determine if the catalog node has changed.

DESCRIPTION

You call the `DirGetDNodeMetaInfo` function to find out if there has been a change in the content of a dNode that you specify. The function returns the `metaInfo` value associated with the dNode. You must call the function once to get an initial value. When you call the function again, compare the initial value with the new value. If the values match, the dNode has not changed since your previous call to the `DirGetDNodeMetaInfo` function. Any change in the information associated with that dNode causes the value of the `metaInfo` field to change. Records, aliases, pseudonyms, or dNodes may have been added, deleted or renamed. Attribute types or attribute values may have been added, deleted, or changed. Access controls for the dNode, its records, or attribute types may have changed.

If you detect a change in a dNode, you should do whatever is appropriate in your application to update the information you need. For example, you can call the `DirEnumerate` function to retrieve current information for the dNode. If your application is displaying information about the dNode, you can refresh your window.

The `metaInfo` field contains the following structure:

```
struct DirMetaInfo {
   unsigned long  info[4];
};
```

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	**Selector**
`_oceTBDispatch`	0x118

RESULT CODES

`noErr`	0	No error
`kOCEParamErr`	–50	Invalid parameter
`kOCETargetDirectoryInaccessible`	–1613	Target catalog is not currently available
`kOCENoSuchDNode`	–1615	Can’t find specified dNode

SEE ALSO

The `PackedRLI` data structure is described in the chapter “AOCE Utilities” in this book.

You can create a `PackedRLI` data structure with the `OCEPackRLI` utility routine. It is also described in the chapter “AOCE Utilities.”

DirMapDNodeNumberToPathName

The `DirMapDNodeNumberToPathName` function returns pathname information for a dNode that you specify.

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

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

Parameter block

→	`ioCompletion`	`ProcPtr`	Your completion routine
←	`ioResult`	`OSErr`	Result code
→	`serverHint`	`AddrBlock`	AppleTalk address of the PowerShare server
→	`dsRefNum`	`short`	Personal catalog reference number
→	`identity`	`AuthIdentity`	Requester's authentication identity
→	`clientData`	`long`	You define this field
→	`directoryName`	`DirectoryNamePtr`	Catalog name
→	`discriminator`	`DirDiscriminator`	Discriminator value
→	`dNodeNumber`	`DNodeNum`	The dNode number
↔	`path`	`PackedPathNamePtr`	Your buffer
↔	`lengthOfPathName`	`unsigned short`	Length of your buffer, pathname

See "The Parameter Block Header" on page 8-190 for descriptions of the `ioCompletion`, `ioResult`, `serverHint`, `dsRefNum`, `identity`, and `clientData` fields.

Field descriptions

`directoryName`	A pointer to the name of the catalog in which the target dNode resides.
`discriminator`	The discriminator value of the catalog in which the dNode resides. This value differentiates two or more catalogs with the same name.
`dNodeNumber`	The dNode number whose pathname you want to obtain.
`path`	A pointer to a buffer in which the function stores packed pathname information. You must provide a buffer big enough to hold all of the path information that the function returns. A buffer size of `kPathNameMaxBytes` can hold any packed pathname. Before you can read the packed pathname information, you must unpack it with the `OCEUnpackPathName` routine.
`lengthOfPathName`	This field is used for both input and output. You set this field to the size of your buffer in bytes before you call the `DirMapDNodeNumberToPathName` function. The function sets this field to the number of bytes in the pathname information that you requested. If the function completes successfully, this field represents the number of bytes that the function placed in your buffer. If your buffer is too small to hold the entire pathname, the function returns a `kOCEMoreData` result code and does not store any information in your buffer. If this occurs, the value in this field represents the minimum size of a buffer capable of holding the packed pathname information. You must increase the size of your buffer to at least the minimum size and call the function again.

DESCRIPTION

You call the `DirMapDNodeNumberToPathName` function when you know a dNode number and want to obtain the corresponding full pathname. If the catalog you specify does not support dNode numbers (this includes all personal catalogs), the function returns the `kOCENoSuchDNode` error.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	0x123

RESULT CODES

`noErr`	0	No error
`kOCEParamErr`	–50	Invalid parameter
`kOCEUnknownID`	–1567	Authentication identity is not valid
`kOCETargetDirectoryInaccessible`	–1613	Target catalog is not currently available
`kOCENoSuchDNode`	–1615	Can't find specified dNode
`kOCEMoreData`	–1623	Buffer too small
`kOCEStreamCreationErr`	–1625	Error in creating connection to server

SEE ALSO

The `OCEUnpackPathName` routine is described in the chapter “AOCE Utilities” in this book.

The `PackedPathName` data structure is also described in the chapter “AOCE Utilities.”

To obtain the dNode number when you know the pathname, use the `DirMapPathNameToDNodeNumber` function, described next.

DirMapPathNameToDNodeNumber

The `DirMapPathNameToDNodeNumber` function returns the dNode number for a dNode identified by a pathname and catalog that you specify.

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

`paramBlock`
: Pointer to a parameter block.

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

Parameter block

→	`ioCompletion`	`ProcPtr`	Your completion routine
←	`ioResult`	`OSErr`	Result code
→	`serverHint`	`AddrBlock`	AppleTalk address of the PowerShare server
→	`dsRefNum`	`short`	Personal catalog reference number
→	`identity`	`AuthIdentity`	Requester's authentication identity
→	`clientData`	`long`	You define this field
→	`directoryName`	`DirectoryNamePtr`	Catalog name
→	`discriminator`	`DirDiscriminator`	Discriminator value
←	`dNodeNumber`	`DNodeNum`	DNode number
→	`path`	`PackedPathNamePtr`	Pathname

See "The Parameter Block Header" on page 8-190 for descriptions of the `ioCompletion`, `ioResult`, `serverHint`, `dsRefNum`, `identity`, and `clientData` fields.

Field descriptions

`directoryName`	The name of the catalog containing the dNode whose dNode number you want to obtain.
`discriminator`	The discriminator value of the catalog containing the dNode whose dNode number you want to obtain. This value differentiates two or more catalogs with the same name.
`dNodeNumber`	A number that uniquely identifies a dNode within a catalog. The function returns this number.
`path`	A pointer to the buffer that contains the packed pathname for the dNode whose dNode number you want to obtain. You create a packed pathname with the `OCEPackPathName` utility routine.

DESCRIPTION

You call the `DirMapPathNameToDNodeNumber` function when you know the path of a particular dNode and you want to obtain its dNode number. If the catalog you specify does not support dNode numbers (this includes all personal catalogs), the function returns the `kOCENoSuchDNode` error.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	0x122

RESULT CODES

`noErr`	0	No error
`kOCEParamErr`	–50	Invalid parameter
`kOCETargetDirectoryInaccessible`	–1613	Target catalog is not currently available
`kOCENoSuchDNode`	–1615	Can't find specified dNode

SEE ALSO

The `OCEPackPathName` routine is described in the chapter "AOCE Utilities" in this book.

The `PackedPathName` data structure is also described in the chapter "AOCE Utilities."

To obtain the pathname when you know the DNode number, use the `DirMapDNodeNumberToPathName` function, described on page 8-223.

DirGetDNodeInfo

The `DirGetDNodeInfo` function indicates whether a dNode that you specify can contain records and whether it is a foreign node.

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

`paramBlock`
Pointer to a parameter block.

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

Parameter block

→	`ioCompletion`	`ProcPtr`	Your completion routine
←	`ioResult`	`OSErr`	Result code
→	`serverHint`	`AddrBlock`	AppleTalk address of the PowerShare server
→	`dsRefNum`	`short`	Personal catalog reference number
→	`identity`	`AuthIdentity`	Requester's authentication identity
→	`clientData`	`long`	You define this field
→	`pRLI`	`PackedRLIPtr`	Target dNode
←	`descriptor`	`DirNodeKind`	DNode descriptor
↔	`networkSpec`	`NetworkSpecPtr`	Network name

See "The Parameter Block Header" on page 8-190 for descriptions of the `ioCompletion`, `ioResult`, `serverHint`, `dsRefNum`, `identity`, and `clientData` fields.

Field descriptions

`pRLI`	A pointer to packed record location information that identifies the catalog and dNode to which the request applies. The function ignores this field when you provide a nonzero value in the `dsRefNum` field to specify a personal catalog.
`descriptor`	A value that the function returns by which you can determine whether the dNode you specified can contain records and whether it is a foreign node. Use the mask `kCanContainRecords` to determine whether the dNode can contain records. To find out if the dNode you specified is a foreign node, use the mask `kForeignNode`.
`networkSpec`	A pointer to the name of the network in which the dNode resides. The function sets this field only if the dNode can contain records.

DESCRIPTION

The `DirGetDNodeInfo` function is usually called by PowerShare Admin software. Most applications do not need to use this function. However, messaging service access modules may call it to determine if a dNode is a foreign dNode. Foreign dNodes represent external messaging systems that are connected to an AOCE system.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	0x125

RESULT CODES

`noErr`	0	No error
`kOCEParamErr`	–50	Invalid parameter
`kOCETargetDirectoryInaccessible`	–1613	Target catalog is not currently available
`kOCENoSuchDNode`	–1615	Can’t find specified dNode
`kOCEStreamCreationErr`	–1625	Error in creating connection to server

SEE ALSO

The `NetworkSpec` data structure is described in the chapter “AOCE Utilities” in this book.

The `PackedRLI` data structure is also described in the chapter “AOCE Utilities.”

You can create a `PackedRLI` data structure with the `OCEPackRLI` utility routine. It is also described in the chapter “AOCE Utilities.”

Maintaining the PowerTalk Setup Catalog

A catalog that is listed in the PowerTalk Setup catalog is available for use by any application that uses the Catalog Manager. Setup templates use the `DirAddADAPDirectory` and `DirRemoveDirectory` routines to add and remove records that represent PowerShare catalogs from the PowerTalk Setup catalog. The `DirRemoveDirectory` function also removes records that represent external catalogs. For information on adding records that represent external catalogs, see the chapter "Access Module Setup" in *Inside Macintosh: AOCE Service Access Modules*.

Note
A shorthand way of saying that a record representing a catalog is added or removed from the PowerTalk Setup catalog is to say that the *catalog* is added or removed from the PowerTalk Setup catalog. However, a catalog itself is never added or removed from the PowerTalk Setup catalog; only records that represent catalogs are added or removed. ◆

The `DirNetSearchADAPDirectoryGet` and `DirNetSearchADAPDirectoryParse` routines work together to provide information about all of the PowerShare catalogs on a network.

If you know a PowerShare catalog's name and discriminator value, you can call the `DirFindADAPDirectoryByNetSearch` function to locate a catalog and add it to the PowerTalk Setup catalog if you choose.

The `DirAddADAPDirectory` and `DirFindADAPDirectoryByNetSearch` functions provide the option of making a PowerShare catalog temporarily available for the PowerTalk Key Chain's use, without adding it to the PowerTalk Setup catalog. This condition of private availability lasts only until the computer is restarted.

The `DirGetOCESetupRefnum` function provides the reference number of the PowerTalk Setup catalog.

To get information about all of the catalogs that are listed in the PowerTalk Setup catalog, you can call the `DirEnumerateDirectoriesGet` function. It is described in the section "Getting Information About Catalogs" beginning on page 8-196.

DirAddADAPDirectory

The `DirAddADAPDirectory` function makes a PowerShare catalog that you specify available for use with other Catalog Manager functions. At your option, it also adds the catalog to the PowerTalk Setup catalog.

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

`paramBlock`
Pointer to a parameter block.

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

Parameter block

→	`ioCompletion`	`ProcPtr`	Your completion routine
←	`ioResult`	`OSErr`	Result code
→	`serverHint`	`AddrBlock`	AppleTalk address of the PowerShare server
→	`clientData`	`long`	You define this field
→	`directoryName`	`DirectoryNamePtr`	Name of the catalog
→	`discriminator`	`DirDiscriminator`	Discriminator value
→	`addToOCESetup`	`Boolean`	Add to PowerTalk Setup?
←	`directoryRecordCID`	`CreationID`	Creation ID of catalog

See "The Parameter Block Header" on page 8-190 for descriptions of the `ioCompletion`, `ioResult`, `serverHint`, and `clientData` fields.

Field descriptions

`directoryName`	A pointer to the name of the PowerShare catalog that you want to use.
`discriminator`	A value that differentiates two or more catalogs with the same name.
`addToOCESetup`	A Boolean value that specifies whether you want to add the catalog to the PowerTalk Setup catalog. Set this field to `true` if you want to add the catalog to the PowerTalk Setup catalog.
`directoryRecordCID`	The creation ID of the record representing the PowerShare catalog that you specify in the `directoryName` and `discriminator` fields. The function creates a record for the catalog, adds it to the PowerTalk Setup catalog, and returns the record creation ID only when you set `addToOCESetup` to `true`.

DESCRIPTION

You call the `DirAddADAPDirectory` function when you want to make a PowerShare catalog that is not listed in the PowerTalk Setup catalog available for use with other Catalog Manager functions. You must specify a valid AppleTalk address in the `serverHint` field in the parameter block header for this function. If the `serverHint` field is set to `nil` or does not point to a PowerShare server for that catalog, the `DirAddADAPDirectory` function returns an error.

Note

The PowerTalk Key Chain uses this function. In general, there is no reason for an application to use this function. ◆

When the function completes successfully, you can use the catalog with other Catalog Manager functions. If you set the `addToOCESetup` field to `true`, the function adds the catalog to the PowerTalk Setup catalog. All catalogs listed in the PowerTalk Setup catalog are visible to the `DirEnumerateDirectories` function and thus available to any application using the services of the Catalog Manager. Furthermore, the catalogs listed in the PowerTalk Setup catalog remain available until they are explicitly removed by the `DirRemoveDirectory` function.

If you set `addToOCESetup` to `false`, the `DirAddADAPDirectory` function makes the catalog available to you privately, and you may specify it when you call other Catalog Manager functions. This availability lasts until the computer is restarted. Once the computer is restarted, the catalog is no longer available to you. A catalog that you do not add to the PowerTalk Setup catalog is not visible to the `DirEnumerateDirectories` function; therefore, it is not available to other applications nor is it visible to a user.

If you want to use a PowerShare catalog that is not listed in the PowerTalk Setup catalog, but you do not know the address of a PowerShare server for that catalog, you can call the `DirFindADAPDirectoryByNetSearch` function.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	**Selector**
`_oceTBDispatch`	0x137

RESULT CODES

`noErr`	0	No error
`kOCEParamErr`	–50	Invalid parameter
`kOCEAlreadyExists`	–1510	The catalog being added already exists
`kOCETargetDirectoryInaccessible`	–1613	Target catalog is not currently available

SEE ALSO

The `DirEnumerateDirectoriesGet` function is described on page 8-196.

The `DirFindADAPDirectoryByNetSearch` function is described next.

The `DirRemoveDirectory` function is described on page 8-237.

For more information on the PowerTalk Setup catalog, see “Identities and the PowerTalk Setup Catalog” on page 8-166.

DirFindADAPDirectoryByNetSearch

The `DirFindADAPDirectoryByNetSearch` function locates a PowerShare catalog that you specify on a network and makes it available for use with other Catalog Manager functions. At your option, it also adds the catalog to the PowerTalk Setup catalog.

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

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

Parameter block

→	`ioCompletion`	`ProcPtr`	Your completion routine
←	`ioResult`	`OSErr`	Result code
→	`clientData`	`long`	You define this field
→	`directoryName`	`DirectoryNamePtr`	Catalog name
→	`discriminator`	`DirDiscriminator`	Discriminator value
→	`addToOCESetup`	`Boolean`	Add to setup list?
←	`directoryRecordCID`	`CreationID`	Creation ID of catalog record

See "The Parameter Block Header" on page 8-190 for descriptions of the `ioCompletion`, `ioResult`, and `clientData` fields.

Field descriptions

`directoryName`	The name of the PowerShare catalog that you want to find.
`discriminator`	The discriminator value for the named catalog. This value differentiates two or more catalogs with the same name.
`addToOCESetup`	A Boolean value that indicates whether you want to add the catalog to the PowerTalk Setup catalog. Set this field to `true` if you want to the catalog to the PowerTalk Setup catalog.
`directoryRecordCID`	The creation ID of the record representing the catalog that you specify in the `directoryName` and `discriminator` fields. The function creates a record for the catalog, adds it to the PowerTalk Setup catalog, and returns the record creation ID only when you set `addToOCESetup` to `true`.

DESCRIPTION

You call the `DirFindADAPDirectoryByNetSearch` function when you want to use a PowerShare catalog and the catalog is not listed in the PowerTalk Setup catalog. You must provide the catalog name and discriminator value. The function searches the network for the catalog.

Note
The PowerTalk Key Chain uses this function. In general, there is no reason for an application to use this function. ◆

If the function finds the catalog, you can use it with other Catalog Manager functions. If you set the `addToOCESetup` field to `true`, the function adds the catalog to the PowerTalk Setup catalog. All catalogs listed in the PowerTalk Setup catalog are visible to the `DirEnumerateDirectories` function and thus are available to any application using the services of the Catalog Manager. Furthermore, the catalogs listed in the PowerTalk Setup catalog remain available until they are explicitly removed by the `DirRemoveDirectory` function.

If you set `addToOCESetup` to `false`, the `DirFindADAPDirectoryByNetSearch` function makes the catalog available to you privately and you may specify it when you call other Catalog Manager functions. This availability lasts until the computer is restarted. Once the computer is restarted, the catalog is no longer available to you. Catalogs that you do not choose to add to the PowerTalk Setup catalog are not visible to the `DirEnumerateDirectories` function; therefore, they are not available to other applications nor are they visible to a user.

SPECIAL CONSIDERATIONS

The `DirFindADAPDirectoryByNetSearch` function makes a networkwide search for the PowerShare catalog that you specify. Because this function consumes expensive network resources, you should use it very sparingly. If you know a catalog's name, discriminator value, and the Apple talk address of a PowerShare server for the catalog, you should use the `DirAddADAPDirectory` function. It too, makes a catalog available for your use and can add it to the PowerTalk Setup catalog.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	0x107

RESULT CODES

`noErr`	0	No error
`kOCEAlreadyExists`	–1510	The catalog being added already exists
`kOCETargetDirectoryInaccessible`	–1613	Target catalog is not currently available

SEE ALSO

The `DirAddADAPDirectory` function is described on page 8-229.

For more information on the PowerTalk Setup catalog, see “Identities and the PowerTalk Setup Catalog” on page 8-166.

The `DirRemoveDirectory` function is described on page 8-237.

The `DirNetSearchADAPDirectoriesGet` function, described next, retrieves the return address of a PowerShare catalog on a network. By saving and using this address you can eliminate the need to search for a particular catalog with the `DirFindADAPDirectoryByNetSearch` function each time the computer is rebooted.

DirNetSearchADAPDirectoriesGet

The `DirNetSearchADAPDirectoriesGet` function returns information about the PowerShare catalogs on a network.

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

`paramBlock`
Pointer to a parameter block.

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

Parameter block

→	`ioCompletion`	`ProcPtr`	Your completion routine
←	`ioResult`	`OSErr`	Result code
→	`clientData`	`long`	You define this field
↔	`getBuffer`	`Ptr`	Your buffer
→	`getBufferSize`	`unsigned long`	Size of your buffer

See “The Parameter Block Header” on page 8-190 for descriptions of the `ioCompletion`, `ioResult`, and `clientData` fields.

Field descriptions

`getBuffer`	A pointer to a buffer in which the function stores information about each PowerShare catalog on the network: its name, discriminator value, feature flags, and the AppleTalk address of its server. You provide this buffer.
`getBufferSize`	The number of bytes in the buffer.

DESCRIPTION

You call the `DirNetSearchADAPDirectoriesGet` function to obtain a list of the PowerShare catalogs on a network.

If the buffer you provide is not large enough to contain all of the information, the `DirNetSearchADAPDirectoriesGet` function returns the `kOCEMoreData` result code.

If your buffer is too small to hold all of the information you requested, you must allocate a bigger buffer and call the `DirNetSearchADAPDirectoriesGet` function again to get it all. At each call, the function attempts to return all the information you have requested, starting from the beginning. Therefore, you will get duplicate information on subsequent calls.

When the function completes with either the `noErr` or `kOCEMoreData` result codes, you use a pointer to your buffer as input to the `DirNetSearchADAPDirectoriesParse` function, which extracts the catalog information from the buffer.

SPECIAL CONSIDERATIONS

The `DirNetSearchADAPDirectoriesGet` function makes a networkwide search for PowerShare catalogs. Because this search consumes expensive network resources, you should use this function very sparingly.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	0x108

RESULT CODES

`noErr`	0	No error
`kOCEMoreData`	–1623	More data available

SEE ALSO

The `DirNetSearchADAPDirectoriesParse` function is described next.

DirNetSearchADAPDirectoriesParse

The `DirNetSearchADAPDirectoriesParse` function parses the data returned by the `DirNetSearchADAPDirectoriesGet` function and returns information on each PowerShare catalog by repeatedly calling your callback routine.

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

`paramBlock`
Pointer to a parameter block.

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

Parameter block

→	`ioCompletion`	`ProcPtr`	Your completion routine
←	`ioResult`	`OSErr`	Result code
→	`clientData`	`long`	You define this field
→	`getBuffer`	`Ptr`	Your buffer
→	`getBufferSize`	`unsigned long`	Size of your buffer
→	`eachADAPDirectory`	`ForEachADAPDirectory`	Your callback routine

See “The Parameter Block Header” on page 8-190 for descriptions of the `ioCompletion`, `ioResult`, and `clientData` fields.

Field descriptions

`getBuffer` A pointer to the buffer containing the information to parse. Use the same buffer that you provided to the `DirNetSearchADAPDirectoriesGet` function.

`getBufferSize` The number of bytes in the buffer. Use the same value that you provided to the `DirNetSearchADAPDirectoriesGet` function.

`eachADAPDirectory`
A pointer to your callback routine. The function declaration for this routine is described on page 8-318.

DESCRIPTION

You call the `DirNetSearchADAPDirectoriesParse` function to extract the information about PowerShare catalogs placed in your buffer by the `DirNetSearchADAPDirectoriesGet` function. You must provide a callback routine that the `DirNetSearchADAPDirectoriesParse` function calls for each set of catalog information in the buffer. The `DirNetSearchADAPDirectoriesParse` function passes your callback routine the following information about each catalog: a catalog name and discriminator value, its feature flags, and the AppleTalk address of a PowerShare server for that catalog.

The `DirNetSearchADAPDirectoriesParse` function completes when it has finished parsing the contents of your buffer or when your callback routine returns `true`. The function returns the `kOCEMoreData` result code if it reaches the end of the buffer and finds that the `DirNetSearchADAPDirectoriesGet` function did not return all the data requested.

If your callback routine returns `true`, the `DirNetSearchADAPDirectoriesParse` function completes with the `noErr` result code.

Once you have the name and discriminator value for a PowerShare catalog, you can call the `DirAddADAPDirectory` function to make the catalog available for use and, if you choose, to add it to the PowerTalk Setup catalog.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	**Selector**
`_oceTBDispatch`	0x105

RESULT CODES

`noErr`	0	No error
`kOCEMoreData`	–1623	More data available

SEE ALSO

The function declaration for your callback routine is described on page 8-318.

The `DirNetSearchADAPDirectoriesGet` function is described on page 8-234.

The `DirAddADAPDirectory` function is described on page 8-229.

DirRemoveDirectory

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

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

`paramBlock`
Pointer to a parameter block.

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

Parameter block

→	`ioCompletion`	`ProcPtr`	Your completion routine
←	`ioResult`	`OSErr`	Result code
→	`serverHint`	`AddrBlock`	AppleTalk address of the PowerShare server
→	`dsRefNum`	`short`	Personal catalog reference number
→	`identity`	`AuthIdentity`	Requester's authentication identity
→	`clientData`	`long`	You define this field
→	`directoryRecordCID`	`CreationID`	Creation ID of catalog

See "The Parameter Block Header" on page 8-190 for descriptions of the `ioCompletion`, `ioResult`, `serverHint`, `dsRefNum`, `identity`, and `clientData` fields.

Field descriptions

`directoryRecordCID`
The creation ID of a record in the PowerTalk Setup catalog. This record represents the catalog that you want to remove.

DESCRIPTION

You call the `DirRemoveDirectory` function to remove an external or PowerShare catalog that you specify from the PowerTalk Setup catalog.

A catalog that you remove from the PowerTalk Setup catalog is no longer visible to the `DirEnumerateDirectoriesGet` function. You cannot specify it in calls to other Catalog Manager functions until you again add it to the PowerTalk Setup catalog or make it available privately to your application through the `DirAddADAPDirectory` or the `DirFindADAPDirectoryByNetSearch` function.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	0x135

RESULT CODES

`noErr`	0	No error
`kOCEDirectoryNotFoundErr`	–7945	Can't find specified catalog

SEE ALSO

Use the `DirAddDSAMDirectory` function, which is described in the chapter "Catalog Service Access Modules" in *Inside Macintosh: AOCE Service Access Modules*, to add a catalog to the PowerTalk Setup catalog.

You can also use the `DirAddADAPDirectory` function, which is described on page 8-229, to add a catalog to the PowerTalk Setup catalog.

The `DirFindADAPDirectoryByNetSearch` function is described on page 8-232.

The `DirEnumerateDirectoriesGet` function is described on page 8-196.

DirGetOCESetupRefnum

The `DirGetOCESetupRefnum` function returns the reference number of the PowerTalk Setup catalog.

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

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

Parameter block

→	`ioCompletion`	`ProcPtr`	Your completion routine
←	`ioResult`	`OSErr`	Result code
←	`dsRefNum`	`short`	PowerTalk Setup catalog reference number
→	`identity`	`AuthIdentity`	Requester's authentication identity
→	`clientData`	`long`	You define this field
←	`oceSetupRecordCID`	`CreationID`	Creation ID of the record identifying the PowerTalk Setup catalog

See "The Parameter Block Header" on page 8-190 for descriptions of the `ioCompletion`, `ioResult`, `dsRefNum`, `identity`, and `clientData` fields.

Field descriptions

`oceSetupRecordCID`	The creation ID of the record identifying the PowerTalk Setup catalog.

DESCRIPTION

You call the `DirGetOCESetupRefnum` function if you need to read from or write to the PowerTalk Setup catalog. The function returns the `dsRefNum` value for the PowerTalk Setup catalog. You need this value to perform operations on the PowerTalk Setup catalog.

The function also returns the creation ID of the record that contains summary information about the contents of the PowerTalk Setup catalog.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	0x128

RESULT CODES

`noErr`	0	No error

SEE ALSO

For more information on the PowerTalk Setup catalog and local identity, see "Identities and the PowerTalk Setup Catalog" on page 8-166 as well as the chapter "Authentication Manager" in this book.

Creating, Opening, and Closing Personal Catalogs

A personal catalog is a Hierarchical File System (HFS) file. You can use the functions in this section to create new personal catalogs as well as to open and close existing personal catalogs. In addition, the `DirMakePersonalDirectoryRLI` function provides information you can use to locate a personal catalog that you opened even if it has been closed, moved, or renamed.

You can use File Manager functions to browse for a personal catalog. Use the constants `kPersonalDirectoryFileType` and `kPersonalDirectoryFileCreator` to specify the file type and file creator, respectively, for a personal catalog.

DirCreatePersonalDirectory

The `DirCreatePersonalDirectory` function creates a new personal catalog.

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

`paramBlock`
Pointer to a parameter block.

Parameter block

←	`ioResult`	`OSErr`	Result code
→	`fsSpec`	`FSSpecPtr`	File system specification
→	`fdType`	`OSType`	File type
→	`fdCreator`	`OSType`	File creator

See "The Parameter Block Header" on page 8-190 for a description of the `ioResult` field.

Field descriptions

`fsSpec`	A pointer to the file system specification record that identifies the personal catalog you want to create. You can obtain the file system specification record from the `FSMakeFSSpec` function.
`fdType`	The file type for the new personal catalog. If you want to create an ordinary personal catalog, set this field to the constant `kPersonalDirectoryFileType`. If you want to create an information card, set this field to the constant `kBusinessCardFileType`.
`fdCreator`	The file creator for the new personal catalog. Set this field to the constant `kPersonalDirectoryFileCreator`, for both an ordinary personal catalog and an information card.

DESCRIPTION

You call the `DirCreatePersonalDirectory` function to create a personal catalog.

You can provide values for the file creator and file type other than those specified in the field descriptions above. However, if you do so, the Finder and AOCE software will not be able to display the icons that represent the personal catalog or information card to the user.

To open the new personal catalog, use the `DirOpenPersonalDirectory` function, described next.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	0x11F

RESULT CODES

`noErr`	0	No error
`dupFNErr`	–48	Filename already exists
`kOCEParamErr`	–50	Invalid parameter
`dirNFErr`	–120	Catalog not found

SEE ALSO

An information card is a personal catalog containing a single record. For more information about information cards, see the section “Introduction to AOCE Catalogs” beginning on page 8-162.

For information about file system specification records, see the chapter “File Manager” in *Inside Macintosh: Files*.

DirOpenPersonalDirectory

The `DirOpenPersonalDirectory` function opens a personal catalog and returns a reference number for it.

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

`paramBlock`
Pointer to a parameter block.

Parameter block

←	`ioResult`	`OSErr`	Result code
←	`dsRefNum`	`short`	Personal catalog reference number
→	`fsSpec`	`FSSpecPtr`	File system specification
→	`accessRequested`	`char`	Permissions requested
←	`accessGranted`	`char`	Permissions granted
←	`features`	`DirGestalt`	Feature flags

See "The Parameter Block Header" on page 8-190 for a description of the `ioResult` and `dsRefNum` fields.

Field descriptions

`fsSpec` A pointer to a file system specification record for the personal catalog that you want to open.

`accessRequested`
The access that you are requesting for this personal catalog. Set this field to `fsRdPerm` if you are requesting permission to read the personal catalog. If you also want permission to write to the personal catalog, set this field to `fsRdWrPerm`.

`accessGranted`
The catalog access that the Catalog Manager grants. The function returns either `fsRdPerm` or `fsRdWrPerm` in this field, granting you read-only or read/write access, respectively.

`features` A set of bit flags indicating the features that the personal catalog supports. The bit flags are described in "Feature Flag Bit Array" beginning on page 8-186.

DESCRIPTION

You call the `DirOpenPersonalDirectory` function to open a personal catalog (including information cards, which are a type of personal catalog). In the `dsRefNum` field of the parameter block header, the function returns the reference number that uniquely identifies the personal catalog. You must use this reference number in all subsequent Catalog Manager requests directed to this personal catalog.

The function also returns the access that you have to the personal catalog file and a set of bit flags that specify what features the personal catalog supports.

SPECIAL CONSIDERATIONS

If the user moves a personal catalog to a computer whose operating system uses a different script system from the one last used to sort the catalog, the personal catalog must be resorted before the Catalog Manager can open it. If the `DirOpenPersonalDirectory` function returns the error `kOCEVersionErr`, you must call the `SDPSortPersonalDirectory` function to resort the personal catalog and then call the `DirOpenPersonalDirectory` function again to open the catalog.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	0x11E

RESULT CODES

`noErr`	0	No error
`tmfoErr`	–42	Too many files open
`fnfErr`	–43	File not found
`kOCEParamErr`	–50	Invalid parameter
`permErr`	–54	Permissions error
`dirNFErr`	–120	Catalog not found
`kOCEVersionErr`	–1504	Need to sort personal catalog

SEE ALSO

For a description of catalog feature flags, see "Feature Flag Bit Array" beginning on page 8-186.

To close a personal catalog that you have opened, use the `DirClosePersonalDirectory` function, described next.

The `SDPSortPersonalDirectory` function is described in the chapter "Standard Catalog Package" in this book.

For information about file system specifications, see the chapter "File Manager" in *Inside Macintosh: Files*.

DirClosePersonalDirectory

The `DirClosePersonalDirectory` function closes an open personal catalog.

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

`paramBlock`
Pointer to a parameter block.

Parameter block

←	`ioResult`	`OSErr`	Result code
→	`dsRefNum`	`short`	Reference number

Field descriptions

`ioResult`	The result of the function.
`dsRefNum`	The catalog reference number that identifies the personal catalog to be closed. After this function successfully completes execution, that reference number is no longer valid.

DESCRIPTION

You call the `DirClosePersonalDirectory` function to close any personal catalog that has been opened by the `DirOpenPersonalDirectory` function. This includes information cards, which are a type of personal catalog.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	0x131

RESULT CODES

`noErr`	0	No error
`kOCERefNumBad`	–1624	Reference number is not valid

SEE ALSO

The `DirOpenPersonalDirectory` function is described on page 8-242**.**

DirMakePersonalDirectoryRLI

The `DirMakePersonalDirectoryRLI` function provides you with packed record location information for a personal catalog that you specify.

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

`paramBlock`
Pointer to a parameter block.

Parameter block

←	`ioResult`	`OSErr`	Result code
→	`dsRefNum`	`short`	Reference number
→	`fromFSSpec`	`FSSpecPtr`	Catalog to search
→	`pRLIBufferSize`	`unsigned short`	Size of your buffer
←	`pRLISize`	`unsigned short`	Size of the `PackedRLI` data structure
↔	`pRLI`	`PackedRLIPtr`	Your buffer

See "The Parameter Block Header" on page 8-190 for a description of the `ioResult` and `dsRefNum` fields.

Field descriptions

`fromFSSpec`	A pointer to a file system specification record. It specifies the folder within which the personal catalog must reside for the Alias Manager to find it. Set this field to `nil` if you do not want to limit the Alias Manager's search.
`pRLIBufferSize`	The size, in bytes, of the buffer that you provide for the packed record location information.
`pRLISize`	The length of the packed record location information. If the function returns the `noErr` result code, this is the number of bytes of data that the function placed in your buffer. If the function returns the `kOCEMoreData` result code, you can use the value of this field to determine how large a buffer is required, allocate a buffer of that size, and call the function again.
`pRLI`	A pointer to the buffer in which the function stores the packed record location information. You provide this buffer.

DESCRIPTION

You call the `DirMakePersonalDirectoryRLI` function to obtain record location information for a personal catalog. You identify the personal catalog about which you want record location information by setting the `dsRefNum` field in the parameter block header to the personal catalog's reference number. You obtain the reference number from the `DirOpenPersonalDirectory` function.

You can use the record location information to find the personal catalog if it has been closed, moved, or renamed. For example, if you are developing an electronic mail application, you might have to handle the following sequence of events. A user may open a personal catalog and copy an address from it to a letter being prepared. The user may then close the personal catalog and send the letter at a later time. To send the letter, you may need additional information from the personal catalog. You can locate it using the record location information that the `DirMakePersonalDirectoryRLI` function returns to you.

To make sure that you can locate a personal catalog even if it has been moved, renamed, or closed, call the `DirMakePersonalDirectoryRLI` function after opening the personal catalog. The function actually creates an alias for the personal catalog and returns record location information for the alias. To find the personal catalog, pass the `PackedRLI` data structure returned by the `DirMakePersonalDirectoryRLI` function to the `OCEExtractAlias` utility routine, which returns an alias record. Pass that alias record to the `ResolveAlias` function, which locates the personal catalog and returns its file system specification. You can then open the personal catalog with the `DirOpenPersonalDirectory` function using the `FSSpec` data structure returned by the `ResolveAlias` function.

If you provided a file system specification record in the `fromFSSpec` field, the `ResolveAlias` routine looks for the catalog only in that folder and any folders enclosed within it. This results in a speedier search. However, if the personal catalog has been moved elsewhere, the `ResolveAlias` routine cannot find it and returns an error.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	**Selector**
`_oceTBDispatch`	0x132

RESULT CODES

`noErr`	0	No error
`kOCEParamErr`	–50	Invalid parameter
`kOCEMoreData`	–1623	More data available
`kOCERefNumBad`	–1624	Reference number is not valid

SEE ALSO

The `PackedRLI` data structure is described in the chapter “AOCE Utilities” in this book.

You can create a `PackedRLI` data structure with the `OCEPackRLI` utility routine. It is also described in the chapter “AOCE Utilities.”

The `OCEExtractAlias` utility routine is also described in the chapter “AOCE Utilities.”

The `ResolveAlias` routine is described in the chapter “Alias Manager” in *Inside Macintosh: Files*.

The `DirOpenPersonalDirectory` function is described on page 8-242.

Managing Records

The functions described in this section provide the following services:

- adding and deleting records
- adding and deleting pseudonyms
- listing the pseudonyms for a record
- detecting a change in a record
- setting and obtaining a record's name and type
- adding an alias for a record

You can also list all of the records, pseudonyms, and aliases within a dNode. To do this, use the `DirEnumerateGet` and `DirEnumerateParse` functions described in "Getting Information About dNodes" beginning on page 8-214.

DirAddRecord

The `DirAddRecord` function adds a new record to a dNode in a catalog that you specify.

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

`paramBlock`
Pointer to a parameter block.

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

Parameter block

→	`ioCompletion`	`ProcPtr`	Your completion routine
←	`ioResult`	`OSErr`	Result code
→	`serverHint`	`AddrBlock`	AppleTalk address of the PowerShare server
→	`dsRefNum`	`short`	Personal catalog reference number
→	`identity`	`AuthIdentity`	Requester's authentication identity
→	`clientData`	`long`	You define this field
↔	`aRecord`	`RecordIDPtr`	Target record
→	`allowDuplicate`	`Boolean`	Allow duplicate record name?

See "The Parameter Block Header" on page 8-190 for descriptions of the `ioCompletion`, `ioResult`, `serverHint`, `dsRefNum`, `identity`, and `clientData` fields.

Field descriptions

`aRecord`	A pointer to a partially specified record ID for the record that you want to add. If you want to add a record to a personal catalog, you must provide the record's name and type. If you want to add a record to a PowerShare or external catalog, you must specify everything in the record ID except the `cid` field. The function places the creation ID for the new record in the `cid` field. If a catalog does not support creation IDs, the function sets the `cid` field to 0.
`allowDuplicate`	A Boolean value specifying whether the function should create a record if another record, alias, or pseudonym with the same name and type already exists. Set this field to `true` if you want the function to create the new record without checking the dNode for a duplicate name and type. If you set the `allowDuplicate` field to `false`, the function checks the name and type of all records, aliases, and pseudonyms in the dNode and returns the `kOCENoDupAllowed` result code if it finds a duplicate name.

DESCRIPTION

You call the `DirAddRecord` function to add a record to a dNode.

SPECIAL CONSIDERATIONS

If you set the `allowDuplicate` field to `false`, the function will not add the record if a record with the same name and type already exists. However, this does not guarantee that a duplicate record will not be created by a requester who sets the `allowDuplicate` field to `true`. The prohibition on duplicates applies only at the time you call this function; it does not guarantee that the record name and record type will be unique at a later time.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	$0109

RESULT CODES

`noErr`	0	No error
`kOCEParamErr`	–50	Invalid parameter
`kOCEWriteAccessDenied`	–1541	Identity lacks write access privileges
`kOCETargetDirectoryInaccessible`	–1613	Target catalog is not currently available
`kOCENoSuchDNode`	–1615	Can't find specified dNode
`kOCENoDupAllowed`	–1641	Duplicate name and type

SEE ALSO

The `RecordID` data structure is described in the chapter "AOCE Utilities" in this book.

DirDeleteRecord

The `DirDeleteRecord` function deletes the record that you specify.

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

`paramBlock`
Pointer to a parameter block.

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

Parameter block

→	`ioCompletion`	`ProcPtr`	Your completion routine
←	`ioResult`	`OSErr`	Result code
→	`serverHint`	`AddrBlock`	AppleTalk address of the PowerShare server
→	`dsRefNum`	`short`	Personal catalog reference number
→	`identity`	`AuthIdentity`	Requester's authentication identity
→	`clientData`	`long`	You define this field
→	`aRecord`	`RecordIDPtr`	Target record

See "The Parameter Block Header" on page 8-190 for descriptions of the `ioCompletion`, `ioResult`, `serverHint`, `dsRefNum`, `identity`, and `clientData` fields.

Field descriptions

`aRecord` A pointer to a record ID that identifies the record you want to delete. You must provide the record location information unless the record exists in a personal catalog. If the catalog in which the record resides supports record creation IDs, you must provide the creation ID; otherwise, you must provide the record name and type.

DESCRIPTION

You call `DirDeleteRecord` to delete a record within a catalog. The function also deletes any pseudonyms for the record. The function does not automatically delete aliases that point to the record you want to delete.

Note

Although, you can call the `DirDeleteRecord` function to delete a record that is an alias for another record, there is no way to automatically identify any aliases that point to a record you have deleted. The situation is much the same as for HFS files. When a file is deleted, its aliases remain intact, but of course the aliases return an error if someone attempts to use them. ◆

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	$010A

RESULT CODES

`noErr`	0	No error
`kOCEParamErr`	–50	Invalid parameter
`kOCEWriteAccessDenied`	–1541	Identity lacks write access privileges
`kOCETargetDirectoryInaccessible`	–1613	Target catalog is not currently available
`kOCENoSuchDNode`	–1615	Can't find specified dNode
`kOCEBadRecordId`	–1617	Record name or record type doesn't match creation ID
`kOCENoSuchRecord`	–1618	Can't find specified record

SEE ALSO

The `RecordID` data structure is described in the chapter "AOCE Utilities" in this book.

DirGetRecordMetaInfo

The `DirGetRecordMetaInfo` function returns a numeric value that you can use to determine if a record has changed since you last called this function.

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

`paramBlock`
Pointer to a parameter block.

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

Parameter block

→	`ioCompletion`	`ProcPtr`	Your completion routine
←	`ioResult`	`OSErr`	Result code
→	`serverHint`	`AddrBlock`	AppleTalk address of the PowerShare server
→	`dsRefNum`	`short`	Personal catalog reference number
→	`identity`	`AuthIdentity`	Requester's authentication identity
→	`clientData`	`long`	You define this field
→	`aRecord`	`RecordIDPtr`	Target record
←	`metaInfo`	`DirMetaInfo`	Comparison value

See "The Parameter Block Header" on page 8-190 for descriptions of the `ioCompletion`, `ioResult`, `serverHint`, `dsRefNum`, `identity`, and `clientData` fields.

Field descriptions

`aRecord`	A pointer to a record ID that identifies the record to which the request applies. You must provide the record location information unless the record exists in a personal catalog. If the catalog in which the record resides supports record creation IDs, you must provide the creation ID; otherwise, you must provide the record name and type.
`metaInfo`	A numeric value returned by `DirGetRecordMetaInfo`. The Catalog Manager updates this value when the content of a record changes. You use it to determine if the record has changed.

DESCRIPTION

You call the `DirGetRecordMetaInfo` function to find out if there has been a change in the contents of a record that you specify. The function returns the `metaInfo` value associated with the record. You must call the function once to get an initial value. When you call the function again, compare the initial value with the new value. If the values match, the record has not changed since your previous call to the `DirGetRecordMetaInfo` function. Any change to the value of the `metaInfo` field indicates a change in the information associated with that record. Attribute types may have been added or deleted; attribute values may have been added, deleted, or changed; or access controls for the records or its attribute types may have changed.

If you detect a change in a record, you should do whatever is appropriate in your application to update the information you need. For example, you can call the `DirEnumerateAttributeTypes` function, followed by the `DirLookupGet` and `DirLookupParse` functions, to retrieve current information about attribute types and values in the record. If your application is displaying information about the record, you can refresh your window.

The `metaInfo` field contains the following structure:

```
struct DirMetaInfo {
   long  info[4];
};
```

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	$0116

RESULT CODES

`noErr`	0	No error
`kOCEParamErr`	–50	Invalid parameter
`kOCETargetDirectoryInaccessible`	–1613	Target catalog is not currently available
`kOCENoSuchDNode`	–1615	Can't find specified dNode
`kOCEBadRecordID`	–1617	Record name or record type doesn't match creation ID
`kOCENoSuchRecord`	–1618	Can't find specified record

SEE ALSO

The `RecordID` data structure is described in the chapter "AOCE Utilities" in this book.

DirGetNameAndType

The `DirGetNameAndType` function returns a record's name and type.

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

`paramBlock`
Pointer to a parameter block.

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

Parameter block

→	`ioCompletion`	`ProcPtr`	Your completion routine
←	`ioResult`	`OSErr`	Result code
→	`serverHint`	`AddrBlock`	AppleTalk address of the PowerShare server
→	`dsRefNum`	`short`	Personal catalog reference number
→	`identity`	`AuthIdentity`	Requester's authentication identity
→	`clientData`	`long`	You define this field
↔	`aRecord`	`RecordIDPtr`	Target record

See "The Parameter Block Header" on page 8-190 for descriptions of the `ioCompletion`, `ioResult`, `serverHint`, `dsRefNum`, `identity`, and `clientData` fields.

Field descriptions

`aRecord`	A pointer to a record ID that identifies the record whose name and type you are requesting. You must provide the record creation ID. Unless the catalog containing the record is a personal catalog, you must also provide packed record location information. The `name` and `type` buffers that you provide must be large enough to hold a maximum-length `RString` data structure. If the function is successful, it places the record's name and type in these buffers.

DESCRIPTION

If you know the creation ID of a record and the catalog and dNode in which it resides, you can use the `DirGetNameAndType` function to obtain its name and type. A record's name and type may change, but its creation ID always remains the same. You can store the record creation ID as an always valid value and use it as needed to retrieve the changeable name and type.

You may also prefer to store only the record creation ID because it requires less memory and use this function to retrieve the record name and type when you need it.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
_oceTBDispatch	$0114

RESULT CODES

`noErr`	0	No error
`kOCEParamErr`	–50	Invalid parameter
`kOCEReadAccessDenied`	–1540	Identity lacks read access privileges
`kOCETargetDirectoryInaccessible`	–1613	Target catalog is not currently available
`kOCENoSuchDNode`	–1615	Can't find specified dNode
`kOCENoSuchRecord`	–1618	Can't find specified record

SEE ALSO

The `RecordID` data structure is described in the chapter "AOCE Utilities" in this book.

The `RString` data structure is also described in the chapter "AOCE Utilities."

You use the `DirSetNameAndType` function to change a record's name and type. It is described next.

DirSetNameAndType

The `DirSetNameAndType` function changes the name, the type, or both the name and type of a record that you specify.

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

`paramBlock`
Pointer to a parameter block.

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

Parameter block

→	`ioCompletion`	`ProcPtr`	Your completion routine
←	`ioResult`	`OSErr`	Result code
→	`serverHint`	`AddrBlock`	AppleTalk address of the PowerShare server
→	`dsRefNum`	`short`	Personal catalog reference number
→	`identity`	`AuthIdentity`	Requester's authentication identity
→	`clientData`	`long`	You define this field
→	`aRecord`	`RecordIDPtr`	Target record
→	`allowDuplicate`	`Boolean`	Are duplicate name and type OK?
→	`newName`	`RStringPtr`	New record name
→	`newType`	`RStringPtr`	New record type

See "The Parameter Block Header" on page 8-190 for descriptions of the `ioCompletion`, `ioResult`, `serverHint`, `dsRefNum`, `identity`, and `clientData` fields.

Field descriptions

`aRecord` A pointer to a record ID that identifies the record whose name or type you want to change.You must provide the record location information unless the record exists in a personal catalog. If the catalog in which the record resides supports record creation IDs, you must provide the creation ID; otherwise, you must provide the record name and type.

`allowDuplicate`
A Boolean value that specifies whether you want to change the name and type even if this change results in a duplicate name and type. If you set this field to `true`, the function does not check the dNode for a duplicate name and type. It simply locates the record and executes the change.

`newName` A pointer to a buffer that contains the new name for the record. You provide this buffer.

`newType` A pointer to a buffer that contains the new type for the record. You provide this buffer.

DESCRIPTION

If `allowDuplicate` is set to `false`, the `DirSetNameAndType` function returns the `kOCENoDupAllowed` result code if it finds another record with the same name and the same type as the new name and type that you specified.

To change the record name without changing the type, set the value of the new type to the current type. To change the record type without changing the name, set the value of the new name to the current name.

If either the `newName` or `newType` field is set to `nil`, the function returns the `kOCEParamErr` result code.

SPECIAL CONSIDERATIONS

If you set the `allowDuplicate` field to `false`, the function will not set the new name and type if a record with the same name and type already exists. However, this does not guarantee that a duplicate record will not be created by a requester who sets the `allowDuplicate` field to `true`. The prohibition on duplicates applies only at the time you call this function; it does not guarantee that the record name and record type will be unique at a later time.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	**Selector**
`_oceTBDispatch`	$0115

RESULT CODES

`noErr`	0	No error
`kOCEParamErr`	–50	Invalid parameter
`kOCEWriteAccessDenied`	–1541	Identity lacks write access privileges
`kOCEReadAccessDenied`	–1540	Identity lacks read access privileges
`kOCETargetDirectoryInaccessible`	–1613	Target catalog is not currently available
`kOCENoSuchDNode`	–1615	Can’t find specified dNode
`kOCEBadRecordID`	–1617	Record name or record type doesn’t match creation ID
`kOCENoSuchRecord`	–1618	Can’t find specified record
`kOCEOperationNotSupported`	–1626	Specified catalog does not support this operation
`kOCENoDupAllowed`	–1641	Duplicate name and type

SEE ALSO

The `RecordID` data structure is described in the chapter “AOCE Utilities” in this book.

DirAddPseudonym

The `DirAddPseudonym` function adds an alternative name and type for a record that you specify.

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

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

Parameter block

→	`ioCompletion`	`ProcPtr`	Your completion routine
←	`ioResult`	`OSErr`	Result code
→	`serverHint`	`AddrBlock`	AppleTalk address of the PowerShare server
→	`dsRefNum`	`short`	Personal catalog reference number
→	`identity`	`AuthIdentity`	Requester's authentication identity
→	`clientData`	`long`	You define this field
→	`aRecord`	`RecordIDPtr`	Target record
→	`pseudonymName`	`RStringPtr`	Alternative name
→	`pseudonymType`	`RStringPtr`	Alternative type
→	`allowDuplicate`	`Boolean`	Are duplicate name and type OK?

See "The Parameter Block Header" on page 8-190 for descriptions of the `ioCompletion`, `ioResult`, `serverHint`, `dsRefNum`, `identity`, and `clientData` fields.

Field descriptions

`aRecord`	A pointer to a record ID that identifies the record for which you want to add a pseudonym. You must provide the record location information unless the record exists in a personal catalog. If the catalog in which the record resides supports record creation IDs, you must provide the creation ID; otherwise, you must provide the record name and type.
`pseudonymName`	A pointer to the alternative name that you want to add.
`pseudonymType`	A pointer to the alternative type that you want to add.
`allowDuplicate`	A Boolean value that indicates whether the function will add a name and type if another record, alias, or pseudonym with the same name and type already exists in the dNode. Set this field to `true` if you want the function to add the new pseudonym without checking for a duplicate name and type. If you set the `allowDuplicate` field to `false`, the function checks the name and type fields of all

records, aliases, and pseudonyms in the dNode and returns the `kOCENoDupAllowed` result code if it finds a duplicate.

DESCRIPTION

You call the `DirAddPseudonym` function when you want to add an alternative name and type for a record. You can discover all of the existing pseudonyms for a record by calling the `DirEnumeratePseudonymGet` and `DirEnumeratePseudonymParse` functions.

Pseudonyms are automatically deleted when the target record is deleted.

You must specify both a name and a type. If either the `pseudonymName` or `pseudonymType` field is set to `nil`, the function returns the `kOCEParamErr` result code.

SPECIAL CONSIDERATIONS

If you set the `allowDuplicate` field to `false`, the function will not add the pseudonym if a pseudonym with the same name and type already exists. However, this does not guarantee that a duplicate pseudonym will not be created by a requester who sets the `allowDuplicate` field to `true`. The prohibition on duplicates applies only at the time you call this function; it does not guarantee that the pseudonym name and pseudonym type will be unique at a later time.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	**Selector**
`_oceTBDispatch`	$010F

RESULT CODES

`noErr`	0	No error
`kOCEParamErr`	–50	Invalid parameter
`kOCEWriteAccessDenied`	–1541	Identity lacks write access privileges
`kOCETargetDirectoryInaccessible`	–1613	Target catalog is not currently available
`kOCEBadRecordID`	–1617	Record name or record type doesn't match creation ID
`kOCENoSuchRecord`	–1618	Can't find specified record
`kOCEStreamCreationErr`	–1625	Error in connection to server
`kOCENoDupAllowed`	–1641	Duplicate name and type

SEE ALSO

The `RecordID` data structure is described in the chapter "AOCE Utilities" in this book.

The `DirEnumeratePseudonymGet` and `DirEnumeratePseudonymParse` functions are described on page 8-259 and page 8-262 respectively.

You can use the `DirEnumerateGet` and `DirEnumerateParse` functions, described on page 8-215 and page 8-220, respectively, to enumerate all of the pseudonyms that exist in a dNode.

To remove a pseudonym that you have added, use the `DirDeletePseudonym` function, described next.

DirDeletePseudonym

The `DirDeletePseudonym` function deletes an alternative name and type of a record that you specify.

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

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

Parameter block

→	`ioCompletion`	`ProcPtr`	Your completion routine
←	`ioResult`	`OSErr`	Result code
→	`serverHint`	`AddrBlock`	AppleTalk address of the PowerShare server
→	`dsRefNum`	`short`	Personal catalog reference number
→	`identity`	`AuthIdentity`	Requester's authentication identity
→	`clientData`	`long`	You define this field
→	`aRecord`	`RecordIDPtr`	Target record
→	`pseudonymName`	`RStringPtr`	Alternative name
→	`pseudonymType`	`RStringPtr`	Alternative type

See "The Parameter Block Header" on page 8-190 for descriptions of the `ioCompletion`, `ioResult`, `serverHint`, `dsRefNum`, `identity`, and `clientData` fields.

Field descriptions

`aRecord`	A pointer to a record ID that identifies the record whose alternative name and type you want to delete. You must provide the record location information unless the record exists in a personal catalog. If the catalog in which the record resides supports record creation IDs, you must provide the creation ID; otherwise, you must provide the record name and type.
`pseudonymName`	A pointer to the alternative name that you want to delete.
`pseudonymType`	A pointer to the alternative type that you want to delete.

DESCRIPTION

If you no longer want to refer to a record by an alternate name or type, you can call this function to delete the name and the type.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	**Selector**
`_oceTBDispatch`	$0110

RESULT CODES

`noErr`	0	No error
`kOCEParamErr`	–50	Invalid parameter
`kOCEWriteAccessDenied`	–1541	Identity lacks write access privileges
`kOCETargetDirectoryInaccessible`	–1613	Target catalog is not currently available
`kOCEBadRecordID`	–1617	Record name or record type doesn't match creation ID
`kOCENoSuchRecord`	–1618	Can't find specified record
`kOCENoSuchPseudonym`	–1620	Can't find specified pseudonym
`kOCEStreamCreationErr`	–1625	Error in creating connection to server

SEE ALSO

The `RecordID` data structure is described in the chapter "AOCE Utilities" in this book.

To add a pseudonym to a record, use the `DirAddPseudonym` function, described on page 8-256.

DirEnumeratePseudonymGet

The `DirEnumeratePseudonymGet` function returns information about the pseudonyms for a record that you specify.

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

`paramBlock`
Pointer to a parameter block.

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

Parameter block

→	`ioCompletion`	`ProcPtr`	Your completion routine
←	`ioResult`	`OSErr`	Result code
→	`serverHint`	`AddrBlock`	AppleTalk address of the PowerShare server
→	`dsRefNum`	`short`	Personal catalog reference number
→	`identity`	`AuthIdentity`	Requester's authentication identity
→	`clientData`	`long`	You define this field
→	`aRecord`	`RecordIDPtr`	Target record
→	`startingName`	`RStringPtr`	Name to start enumeration from
→	`startingType`	`RStringPtr`	Type to start enumeration from
→	`includeStartingPoint`	`Boolean`	Begin enumeration with starting point?
↔	`getBuffer`	`Ptr`	Your buffer
→	`getBufferSize`	`unsigned long`	Size of your buffer

See "The Parameter Block Header" on page 8-190 for descriptions of the `ioCompletion`, `ioResult`, `serverHint`, `dsRefNum`, `identity`, and `clientData` fields.

Field descriptions

`aRecord`	A pointer to a record ID that identifies the record for which you want to obtain pseudonyms. You must provide the record location information unless the record is in a personal catalog. If the catalog in which the record resides supports record creation IDs, you must provide the creation ID; otherwise, you must provide the record name and type.
`startingName`	A pointer to the alternative name from which you want the function to begin the enumeration. Set this field to `nil` to start with the first alternative name for the record. If the `DirEnumeratePseudonymGet` function completes with the `kOCEMoreData` result code, you can continue the enumeration by setting this field to the value of the `name` field in the last `recordID` parameter passed to your callback routine from the `DirEnumeratePseudonymParse` function. You must coordinate the value you provide in this field with the value you provide in the `startingType` field; that is, both values are required, and both must belong to the same pseudonym.
`startingType`	A pointer to the alternative type from which you want the function to begin the enumeration. Set this field to `nil` to start with the first alternative type for the record. If the `DirEnumeratePseudonymGet` function completes with the `kOCEMoreData` result code, you can continue the enumeration by setting this field to the value of the `type` field in the last `recordID` parameter passed to your callback routine from the `DirEnumeratePseudonymParse` function. You must coordinate the value you provide in this field with the value you provide in the `startingName` field; that is, both values are required, and both must belong to the same pseudonym.

`includeStartingPoint`	A Boolean value that tells the function how to interpret the `startingName` and `startingType` fields. Set this field to `true` if you want the `DirEnumeratePseudonymGet` function to return information about pseudonyms beginning with the one specified by the `startingName` and `startingType` fields. If you set this field to `false`, the function returns information starting with the pseudonym after the one specified by the `startingName` and `startingType` fields.
`getBuffer`	A pointer to the buffer in which the function stores the list of pseudonyms that you requested. You provide this buffer.
`getBufferSize`	The number of bytes in the buffer.

DESCRIPTION

You call the `DirEnumeratePseudonymGet` function to obtain a list of the pseudonyms for a record that you specify.

If the buffer you provide is not large enough to contain all of the information you requested, the `DirEnumeratePseudonymGet` function returns the `kOCEMoreData` result code.

When the function completes with either the `noErr` or `kOCEMoreData` result codes, you use a pointer to your buffer as input to the `DirEnumeratePseudonymParse` function, which extracts the pseudonyms from the buffer.

If the `DirEnumeratePseudonymGet` function returns the `kOCEMoreData` result code, you can request additional information by calling the `DirEnumeratePseudonymGet` function again, after calling the `DirEnumeratePseudonymParse` function. As the values of the `startingName` and `startingType` fields, use the values of the `name` and `type` fields in the last `recordID` parameter passed to your callback routine from the `DirEnumeratePseudonymParse` function. The `DirEnumeratePseudonymGet` function will continue the enumeration starting with the next record as determined by the value of the `includeStartingPoint` field.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	**Selector**
`_oceTBDispatch`	$0113

RESULT CODES

`noErr`	0	No error
`kOCEParamErr`	–50	Invalid parameter
`kOCEReadAccessDenied`	–1540	Identity lacks read access privileges
`kOCETargetDirectoryInaccessible`	–1613	Target catalog is not currently available
`kOCENoSuchDNode`	–1615	Can’t find specified dNode
`kOCENoSuchRecord`	–1618	Can't find specified record
`kOCEMoreData`	–1623	More data available

SEE ALSO

The `RecordID` data structure is described in the chapter “AOCE Utilities” in this book.

The `DirEnumeratePseudonymParse` function is described next.

For an example of continuing the enumeration (using the `DirEnumerateAttributeTypesGet` function) when the buffer is too small to hold all the information you requested, see “Getting Attribute Type Information” beginning on page 8-178.

DirEnumeratePseudonymParse

The `DirEnumeratePseudonymParse` function parses the data returned by the `DirEnumeratePseudonymGet` function and returns a pointer to each pseudonym by repeatedly calling your callback routine.

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

`paramBlock`
Pointer to a parameter block.

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

Parameter block

→	`ioCompletion`	`ProcPtr`	Your completion routine
←	`ioResult`	`OSErr`	Result code
→	`serverHint`	`AddrBlock`	AppleTalk address of the PowerShare server
→	`dsRefNum`	`short`	Personal catalog reference number
→	`identity`	`AuthIdentity`	Requester's authentication identity
→	`clientData`	`long`	You define this field
→	`aRecord`	`RecordIDPtr`	Target record
→	`eachRecordID`	`ForEachRecordID`	Your callback routine
→	`getBuffer`	`Ptr`	Your buffer
→	`getBufferSize`	`unsigned long`	Size of your buffer

See "The Parameter Block Header" on page 8-190 for descriptions of the `ioCompletion`, `ioResult`, `serverHint`, `dsRefNum`, `identity`, and `clientData` fields.

Field descriptions

`aRecord`	A pointer to a record ID that identifies the record for which you want to obtain pseudonyms. Use the same value that you provided to the `DirEnumeratePseudonymGet` function.
`eachRecordID`	A pointer to your callback routine. The function declaration for this routine is described on page 8-309.
`getBuffer`	A pointer to the buffer containing the information to parse. Use the same buffer that you provided to the `DirEnumeratePseudonymGet` function.
`getBufferSize`	The number of bytes in the buffer. Use the same value that you provided to the `DirEnumeratePseudonymGet` function.

DESCRIPTION

You call the `DirEnumeratePseudonymParse` function to extract the pseudonyms placed in a buffer by the `DirEnumeratePseudonymGet` function. You must provide a callback routine that the `DirEnumeratePseudonymParse` function calls for each pseudonym it finds in the buffer.

The `DirEnumeratePseudonymParse` function completes when it has finished parsing the contents of your buffer or when your callback routine returns `true`. The function returns the `kOCEMoreData` result code if it reaches the end of the buffer and finds that the `DirEnumeratePseudonymGet` function did not return all the data requested. If you want to continue the enumeration, you can call the `DirEnumeratePseudonymGet` function again. In your next call to the `DirEnumeratePseudonymGet` function, set the `startingName` and `startingType` fields to the values of the `name` and `type` fields of the last `recordID` parameter passed to your callback routine from the `DirEnumeratePseudonymParse` function.

If your callback routine returns `true`, the `DirEnumeratePseudonymParse` function completes with the `noErr` result code.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	**Selector**
`_oceTBDispatch`	$0104

RESULT CODES

`noErr`	0	No error
`kOCEParamErr`	–50	Invalid parameter
`kOCEMoreData`	–1623	More data available

SEE ALSO

The function declaration for your callback routine is described on page 8-309.

The `RecordID` data structure is described in the chapter "AOCE Utilities" in this book.

The `DirEnumeratePseudonymGet` function is described on page 8-259.

For an example of continuing the enumeration (using the `DirEnumerateAttributeTypesParse` function) when the buffer is too small to hold all the information you requested, see "Getting Attribute Type Information" beginning on page 8-178.

DirAddAlias

The `DirAddAlias` function adds an alias record to a catalog.

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

`paramBlock`
Pointer to a parameter block.

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

Parameter block

→	`ioCompletion`	`ProcPtr`	Your completion routine
←	`ioResult`	`OSErr`	Result code
→	`serverHint`	`AddrBlock`	AppleTalk address of the PowerShare server
→	`dsRefNum`	`short`	Personal catalog reference number
→	`identity`	`AuthIdentity`	Requester's authentication identity
→	`clientData`	`long`	You define this field
↔	`aRecord`	`RecordIDPtr`	Target record
→	`allowDuplicate`	`Boolean`	Is duplicate name and type OK?

See “The Parameter Block Header” on page 8-190 for descriptions of the `ioCompletion`, `ioResult`, `serverHint`, `dsRefNum`, `identity`, and `clientData` fields.

Field descriptions

`aRecord`	A pointer to a partially specified record ID for the alias record that you want to add. If you want to add an alias record to a personal catalog, you must provide the alias record’s name and type. If you want to add an alias record to an external catalog, you must specify everything in the record ID except the `cid` field. The function places the creation ID for the new alias record in the `cid` field. If the catalog does not support creation IDs, the function sets the `cid` field to `nil`.
`allowDuplicate`	A Boolean value specifying whether the function should create an alias if another record, alias, or pseudonym with the same name and type already exists. Set this field to `true` if you want the function to create the alias without checking the dNode for a duplicate name and type. If you set the `allowDuplicate` field to `false`, the function checks the name and type of all records, aliases, and pseudonyms in the dNode and returns the `kOCENoDupAllowed` result code if it finds a duplicate.

DESCRIPTION

This function works just like the `DirAddRecord` function in that it adds a record. It also marks the new record as an alias. Your application is responsible for storing the information you need to resolve the alias. You should add to the new alias record an attribute whose type is referenced by the attribute type index `kAliasAttrTypeNum` and whose value is a `DSSpec` structure that points to the record that this is an alias to.

You can enumerate aliases with the `DirEnumerateGet` and `DirEnumerateParse` functions. You can use the `DirDeleteRecord` function to remove an alias record that you added.

The catalog feature bit flag `kSupportsAliasMask` indicates whether a catalog supports the `DirAddAlias` function.

SPECIAL CONSIDERATIONS

If you set the `allowDuplicate` field to `false`, the function will not add the alias if a record, pseudonym, or alias with the same name and type already exists. However, this does not guarantee that a duplicate alias will not be created by a requester who sets the `allowDuplicate` field to `true`. The prohibition on duplicates applies only at the time you call this function; it does not guarantee that the name and type will be unique at a later time.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	$011C

RESULT CODES

`noErr`	0	No error
`kOCEParamErr`	–50	Invalid parameter
`kOCEWriteAccessDenied`	–1541	Identity lacks write access privileges
`kOCETargetDirectoryInaccessible`	–1613	Target catalog is not currently available
`kOCENoSuchDNode`	–1615	Can’t find specified dNode
`kOCENoDupAllowed`	–1641	Same name and type already exists

SEE ALSO

The `RecordID` data structure is described in the chapter “AOCE Utilities” in this book.

The `DirDeleteRecord` function is described on page 8-249.

For a description of catalog feature flags, see “Feature Flag Bit Array” beginning on page 8-186.

The `DirEnumerateGet` function is described on page 8-215 `and the` `DirEnumerateParse` function is described on page 8-220.

The `DirAddRecord` function is described on page 8-247.

For more information on aliases and pseudonyms, see “Aliases and Pseudonyms” on page 8-165.

Managing Attribute Types and Values

The functions described in this section provide the following services:

- adding and deleting attribute values
- changing and verifying attribute values
- searching for an occurrence of specific data in an attribute value
- reading the attribute values in a record or records
- deleting an attribute type
- listing the attribute types in a record

DirAddAttributeValue

The `DirAddAttributeValue` function adds an attribute value to an existing record.

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

`paramBlock`
: Pointer to a parameter block.

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

Parameter block

→	`ioCompletion`	`ProcPtr`	Your completion routine
←	`ioResult`	`OSErr`	Result code
→	`serverHint`	`AddrBlock`	AppleTalk address of the PowerShare server
→	`dsRefNum`	`short`	Personal catalog reference number
→	`identity`	`AuthIdentity`	Requester's authentication identity
→	`clientData`	`long`	You define this field
→	`aRecord`	`RecordIDPtr`	Target record
↔	`attr`	`AttributePtr`	Attribute structure

See "The Parameter Block Header" on page 8-190 for descriptions of the `ioCompletion`, `ioResult`, `serverHint`, `dsRefNum`, `identity`, and `clientData` fields.

Field descriptions

`aRecord`
: A pointer to a record ID that identifies the record to which you want to add an attribute value. You must provide the record location information unless the record exists in a personal catalog. If the catalog in which the record resides supports record creation IDs, you must provide the creation ID; otherwise, you must provide the record name and type.

`attr`
: A pointer to an `Attribute` data structure. You must completely specify the `type` and `value` substructures of the `Attribute` data structure. The function returns the attribute creation ID.

DESCRIPTION

You call the `DirAddAttributeValue` function to add an attribute value to a record that you specify. PowerShare and personal catalogs do not check for already existing attributes having the same type and value; they simply add the attribute you specify. Therefore, you may add duplicate attribute values to PowerShare and personal catalog records. The Catalog Manager assigns them unique attribute creation IDs.

If the attribute type that you specify does not already exist within the record, the function first adds the new attribute type and then adds the value.

For PowerShare and personal catalogs, you can specify an attribute value up to `kAttrValueMaxBytes` bytes in length. If you specify an attribute value that is larger than `kAttrValueMaxBytes` bytes, the function returns the `kOCEAttributeValueTooBig` result code. The maximum size for an attribute value stored in an external catalog is undefined.

SPECIAL CONSIDERATIONS

Note that there is no function in the Catalog Manager API that explicitly adds an attribute type. To add a new attribute type to a record, begin by setting all fields of the `value` substructure to 0 or `nil` and the `type` substructure to the attribute type that you want to add. Then call the `DirAddAttributeValue` function. If that attribute type already exists within the record, the function returns without an error result code.

Although the Catalog Manager imposes no restrictions on the number of attribute values of a particular attribute type that you can add to a record, the Finder has limited ability to display multivalued attribute types. See the chapter "AOCE Templates" in this book for more information.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	$010B

RESULT CODES

`noErr`	0	No error
`kOCEParamErr`	–50	Invalid parameter
`kOCEWriteAccessDenied`	–1541	Identity lacks write access privileges
`kOCETargetDirectoryInaccessible`	–1613	Target catalog is not currently available
`kOCENoSuchDNode`	–1615	Can't find specified dNode
`kOCENoSuchRecord`	–1618	Can't find specified record
`kOCEAttributeValueTooBig`	–1621	Attribute value larger than `kAttrValueMaxBytes` bytes

SEE ALSO

The `RecordID` data structure is described in the chapter "AOCE Utilities" in this book.

The `Attribute` data structure is also described in the chapter "AOCE Utilities."

You can delete an attribute value with the `DirDeleteAttributeValue` function, described next.

You can delete an attribute type within a record with the `DirDeleteAttributeType` function, described on page 8-284.

DirDeleteAttributeValue

The `DirDeleteAttributeValue` function deletes an attribute value from a record that you specify.

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

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

Parameter block

→	`ioCompletion`	`ProcPtr`	Your completion routine
←	`ioResult`	`OSErr`	Result code
→	`serverHint`	`AddrBlock`	AppleTalk address of the PowerShare server
→	`dsRefNum`	`short`	Personal catalog reference number
→	`identity`	`AuthIdentity`	Requester's authentication identity
→	`clientData`	`long`	You define this field
→	`aRecord`	`RecordIDPtr`	Target record
→	`attr`	`AttributePtr`	Target attribute value

See "The Parameter Block Header" on page 8-190 for descriptions of the `ioCompletion`, `ioResult`, `serverHint`, `dsRefNum`, `identity`, and `clientData` fields.

Field descriptions

`aRecord`	A pointer to a record ID that identifies the record in which the target attribute value is located. You must provide the record location information unless the record exists in a personal catalog. If the catalog in which the record resides supports record creation IDs, you must provide the creation ID; otherwise, you must provide the record name and type.
`attr`	A pointer to an `Attribute` data structure. If you want to delete an attribute value from a catalog that supports attribute creation IDs, you identify the attribute value to be deleted by specifying the attribute creation ID and attribute type. To delete an attribute value from a catalog that does not support attribute creation IDs, specify the attribute type and attribute value.

DESCRIPTION

You call the `DirDeleteAttributeValue` function to delete an attribute value from a record. Deleting the last attribute value of a given attribute type does not delete the attribute type.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	**Selector**
`_oceTBDispatch`	$010C

RESULT CODES

`noErr`	0	No error
`kOCEParamErr`	–50	Invalid parameter
`kOCEWriteAccessDenied`	–1541	Identity lacks write access privileges
`kOCETargetDirectoryInaccessible`	–1613	Target catalog is not currently available
`kOCENoSuchDNode`	–1615	Can't find specified dNode
`kOCENoSuchRecord`	–1618	Can't find specified record
`kOCENoSuchAttributeValue`	–1619	Can't find specified attribute value

SEE ALSO

The `RecordID` data structure is described in the chapter "AOCE Utilities" in this book.

The `Attribute` data structure is also described in the chapter "AOCE Utilities."

You can add an attribute value to a record with the `DirAddAttributeValue` function, described on page 8-267.

You can delete an attribute type with the `DirDeleteAttributeType` function, described on page 8-284.

DirChangeAttributeValue

The `DirChangeAttributeValue` function changes an attribute value that you specify.

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

`paramBlock`
Pointer to a parameter block.

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

Parameter block

→	`ioCompletion`	`ProcPtr`	Your completion routine
←	`ioResult`	`OSErr`	Result code
→	`serverHint`	`AddrBlock`	AppleTalk address of the PowerShare server
→	`dsRefNum`	`short`	Personal catalog reference number
→	`identity`	`AuthIdentity`	Requester's authentication identity
→	`clientData`	`long`	You define this field
→	`aRecord`	`RecordIDPtr`	Target record
→	`currentAttr`	`AttributePtr`	Existing attribute value
→	`newAttr`	`AttributePtr`	New attribute value

See "The Parameter Block Header" on page 8-190 for descriptions of the `ioCompletion`, `ioResult`, `serverHint`, `dsRefNum`, `identity`, and `clientData` fields.

Field descriptions

`aRecord`	A pointer to a record ID that identifies the record containing the attribute value that you want to change. You must provide the record location information unless the record exists in a personal catalog. If the catalog in which the record resides supports record creation IDs, you must provide the creation ID; otherwise, you must provide the record name and type.
`currentAttr`	A pointer to the `Attribute` data structure that specifies the attribute value that you want to change. For catalogs that support attribute creation IDs, you identify the attribute value to change by providing its attribute creation ID and attribute type; otherwise, you need to provide the attribute value and attribute type.
`newAttr`	A pointer to an `Attribute` data structure that contains the new attribute value. In the `value` field, you provide the new attribute value and its length in bytes. You must also provide a type in the `type` field or a `ParamErr` is returned.

DESCRIPTION

You call the `DirChangeAttributeValue` function to change an attribute value without changing its associated attribute creation ID. If you want to assign a new attribute creation ID to the attribute value, use the `DirDeleteAttributeValue` function to delete the old value and the `DirAddAttributeValue` function to add the new value.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	$010D

RESULT CODES

`noErr`	0	No error
`kOCEParamErr`	–50	Invalid parameter
`kOCEWriteAccessDenied`	–1541	Identity lacks write access privileges
`kOCETargetDirectoryInaccessible`	–1613	Target catalog is not currently available
`kOCENoSuchDNode`	–1615	Can’t find specified dNode
`kOCENoSuchRecord`	–1618	Can’t find specified record
`kOCENoSuchAttributeValue`	–1619	Can’t find specified attribute value

SEE ALSO

The `RecordID` data structure is described in the chapter “AOCE Utilities” in this book.

The `Attribute` data structure is also described in the chapter “AOCE Utilities.”

DirVerifyAttributeValue

The `DirVerifyAttributeValue` function indicates whether the specified attribute value exists in the record.

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

`paramBlock`
: Pointer to a parameter block.

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

Parameter block

→	`ioCompletion`	`ProcPtr`	Your completion routine
←	`ioResult`	`OSErr`	Result code
→	`serverHint`	`AddrBlock`	AppleTalk address of the PowerShare server
→	`dsRefNum`	`short`	Personal catalog reference number
→	`identity`	`AuthIdentity`	Requester’s authentication identity
→	`clientData`	`long`	You define this field
→	`aRecord`	`RecordIDPtr`	Target record
→	`attr`	`AttributePtr`	Target attribute value

See “The Parameter Block Header” on page 8-190 for descriptions of the `ioCompletion`, `ioResult`, `serverHint`, `dsRefNum`, `identity`, and `clientData` fields.

Field descriptions

`aRecord`	A pointer to a record ID that identifies the record in which the target attribute value resides. You must provide the record location information unless the record exists in a personal catalog. If the catalog in which the record resides supports record creation IDs, you must provide the creation ID; otherwise, you must provide the record name and type.
`attr`	A pointer to an `Attribute` data structure. You must specify the `type` and `value` fields. You may also provide the attribute creation ID if you know it. Otherwise, set the `cid` field of this structure to 0.

DESCRIPTION

If you provide the attribute creation ID, the `DirVerifyAttributeValue` function verifies that an attribute value having the specified attribute type, attribute creation ID, and actual attribute value exists in the record you specify.

SPECIAL CONSIDERATIONS

If you set the attribute creation ID to 0, the function verifies that an attribute value having the specified actual attribute value and attribute type exists in the record you specify and returns its attribute creation ID in the `cid` field of your `Attribute` data structure. Note that duplicate attribute values may exist in the same record. The attribute creation ID that the function returns may belong to any of the duplicate attribute values.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	$010E

RESULT CODES

`noErr`	0	No error
`kOCEParamErr`	–50	Invalid parameter
`kOCEReadAccessDenied`	–1540	Identity lacks read access privileges
`kOCETargetDirectoryInaccessible`	–1613	Target catalog is not currently available
`kOCENoSuchDNode`	–1615	Can’t find specified dNode
`kOCENoSuchRecord`	–1618	Can’t find specified record
`kOCENoSuchAttributeValue`	–1619	Can’t find specified attribute value

SEE ALSO

The `RecordID` data structure is described in the chapter “AOCE Utilities” in this book.

The `Attribute` data structure is also described in the chapter “AOCE Utilities.”

DirFindValue

The `DirFindValue` function searches the records in a dNode that you specify for an occurrence of an attribute value.

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

`paramBlock`
Pointer to a parameter block.

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

Parameter block

→	`ioCompletion`	`ProcPtr`	Your completion routine
←	`ioResult`	`OSErr`	Result code
→	`serverHint`	`AddrBlock`	AppleTalk address of the PowerShare server
→	`dsRefNum`	`short`	Personal catalog reference number
→	`identity`	`AuthIdentity`	Requester's authentication identity
→	`clientData`	`long`	You define this field
→	`aRLI`	`PackedRLIPtr`	Target dNode
→	`aRecord`	`LocalRecordIDPtr`	Target record
→	`attrType`	`AttributeTypePtr`	Target attribute type
→	`startingRecord`	`LocalRecordIDPtr`	Record to start search from
→	`startingAttribute`	`AttributePtr`	Attribute value to start search from
↔	`recordFound`	`LocalRecordIDPtr`	Record containing matching data
←	`attributeFound`	`Attribute`	Attribute containing matching data
→	`matchSize`	`unsigned long`	Length of data to match
→	`matchingData`	`Ptr`	Data to match
→	`sortDirection`	`DirSortDirection`	Search forward or backward?

See "The Parameter Block Header" on page 8-190 for descriptions of the `ioCompletion`, `ioResult`, `serverHint`, `dsRefNum`, `identity`, and `clientData` fields.

Field descriptions

`aRLI` A pointer to the dNode in which you want to search for an attribute value. The function ignores this field when you provide a nonzero value in the `dsRefNum` field to specify a personal catalog.

`aRecord` A pointer to a local record ID. If you set this field to a nonzero value, the search for matching data is restricted to the record you specify here. Set this field to `nil` if you do not want to restrict your search to a single record.

`attrType`	A pointer to an attribute type. If you set this field to a nonzero value, the search for matching data is restricted to the attribute type that you specify here. Set this field to `nil` if you do not want to restrict your search to a single attribute type.
`startingRecord`	A pointer to a local record ID. Set this field to `nil` if you wish to start the search with the first record in the dNode. If you have already called the `DirFindValue` function and found an occurrence of matching data, you can set this field to the value of the `recordFound` field to search for the next occurrence.
`startingAttribute`	A pointer to an `Attribute` data structure. Set this field to `nil` if you wish to start the search with the first attribute value in the first record to be searched. If you have already called the `DirFindValue` function, you can set this field to the value of the `attributeFound` field to search for the next occurrence.
`recordFound`	A pointer to the local record ID that identifies the record in which the `DirFindValue` function found a matching attribute value. You can set the `startingRecord` field to this field in a subsequent call to the `DirFindValue` function.
`attributeFound`	An `Attribute` data structure that specifies the attribute value within which the function found matching data. You can set the `startingAttribute` field to the address of this structure in a subsequent call to the `DirFindValue` function. If you are searching a PowerShare or personal catalog, the function returns only the attribute creation ID in the `Attribute` data structure. If the catalog in which you are searching does not support attribute creation IDs, the function may return a complete attribute value. In that case, you must provide a buffer large enough to hold a maximum size attribute value as part of your `Attribute` data structure.
`matchSize`	The number of bytes of data to be matched.
`matchingData`	A pointer to a buffer that contains the data to be matched.
`sortDirection`	A constant that specifies the search direction. Set this field to `kSortForwards` to have the function search in a forward direction through the dNode and the record for a match. Set this field to `kSortBackwards` to have the function search in a backward direction.

DESCRIPTION

The `DirFindValue` function examines up to the first 32 bytes of data in an attribute value to find a match when it is searching in a PowerShare or personal catalog. The match is to any string that begins with the search string.

The function returns the type and creation ID of the attribute value in which it finds a match. You may call `DirLookupGet` to obtain the complete attribute value.

IMPORTANT

Personal catalogs and the PowerShare catalog server do not support the `DirFindValue` function. An external catalog may or may not support this function, and it may match more or less than 32 characters. ▲

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	**Selector**
`_oceTBDispatch`	$0126

RESULT CODES

`noErr`	0	No error

SEE ALSO

The `DirLookupGet` function is described next.

The `PackedRLI` data structure is described in the chapter "AOCE Utilities" in this book.

You can create a `PackedRLI` data structure with the `OCEPackRLI` utility routine. It is described in the chapter "AOCE Utilities."

The `LocalRecordID` data structure is also described in the chapter "AOCE Utilities."

The `Attribute` data structure is also described in the chapter "AOCE Utilities."

DirLookupGet

The `DirLookupGet` function returns the attribute values of the attribute types that you specify for a list of records that you provide.

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

`paramBlock`
: Pointer to a parameter block.

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

Parameter block

→	`ioCompletion`	`ProcPtr`	Your completion routine
←	`ioResult`	`OSErr`	Result code
→	`serverHint`	`AddrBlock`	Address of the PowerShare server
→	`dsRefNum`	`short`	Personal catalog reference number
→	`identity`	`AuthIdentity`	Requester's authentication identity
→	`clientData`	`long`	You define this field
→	`aRecordList`	`RecordIDPtr*`	List of record IDs
→	`attrTypeList`	`AttributeTypePtr*`	List of attribute types
→	`recordIDCount`	`unsigned long`	Number of IDs in list
→	`attrTypeCount`	`unsigned long`	Number of types in list
→	`includeStartingPoint`	`Boolean`	Begin enumeration with starting point?
↔	`getBuffer`	`Ptr`	Your buffer
→	`getBufferSize`	`unsigned long`	Size of your buffer
→	`startingRecordIndex`	`unsigned long`	Record to start from
→	`startingAttrTypeIndex`	`unsigned long`	Attribute type to start from
→	`startingAttribute`	`Attribute`	Attribute value to start from

See "The Parameter Block Header" on page 8-190 for descriptions of the `ioCompletion`, `ioResult`, `serverHint`, `dsRefNum`, `identity`, and `clientData` fields.

Field descriptions

`aRecordList` A pointer to an array of pointers to record IDs. The record IDs represent the records in which you want to look up attribute values. You must provide record location information in each record ID unless the record exists in a personal catalog. If the catalog in which the record resides supports record creation IDs, you must provide the record creation ID; otherwise, you must provide the record name and type. The record IDs in your list should be unique.

For PowerShare catalogs, all of the records that you specify must reside in the same dNode. This is not necessarily true for external catalogs.

`attrTypeList` A pointer to an array of pointers to attribute types. The attribute types are those for which you want to look up attribute values. The attribute types in your list should be unique.

`recordIDCount` The number of elements in your array of pointers to record IDs.

`attrTypeCount` The number of elements in your array of pointers to attribute types.

`includeStartingPoint`
A Boolean value that determines how the `DirLookupGet` function interprets the `startingRecordIndex`, `startingAttrTypeIndex`, and `startingAttribute` fields. Set this field to `true` if you want the `DirLookupGet` function to return information from the record, attribute type, and attribute value specified by the starting fields. If you set this field to `false`, the function returns information starting with the record, attribute type, and attribute value immediately following the one specified by the starting fields.

`getBuffer`	A pointer to the buffer in which the function stores the requested information. You provide this buffer.
`getBufferSize`	The number of bytes in the buffer.
`startingRecordIndex`	An index into the array of pointers to record IDs. It represents the record at which the `DirLookupGet` function begins the lookup. To start at the first record specified by the array, set this value to 1. The value of the `startingRecordIndex` field must always be less than or equal to the value of the `recordIDCount` field.
`startingAttrTypeIndex`	An index into the array of pointers to attribute types. It represents the attribute type at which the function begins the lookup. To start at the first attribute type specified by the array, set this value to 1. The value of the `startingAttrTypeIndex` field must always be less than or equal to the value of the `attrTypeCount` field.
`startingAttribute`	An `Attribute` data structure that specifies the attribute value at which the `DirLookupGet` function begins the lookup. If the catalog in which you are requesting the lookup supports creation IDs, the attribute creation ID is a sufficient specification. If you set the attribute creation ID to 0, the function begins the search from the first attribute value of the type specified by the `startingAttrTypeIndex` field. If you specify a nonzero attribute creation ID and the function does not find an attribute value with a matching creation ID, `DirLookupGet` terminates with a `kOCENoSuchAttributeValue` result code. You should not call `DirLookupParse` after getting this error. If the catalog in which you are requesting the lookup does not support attribute creation IDs, you must specify the entire structure. Set every field in the structure to 0 if you want the function to begin the search from the first attribute value of the type specified by the `startingAttrTypeIndex` field.

DESCRIPTION

You call the `DirLookupGet` function to obtain the attribute values of particular attribute types in records that you specify. You must specify a record, an attribute type, and an attribute value at which you want the function to start the lookup.

The `DirLookupGet` function places the requested attribute values in your buffer. It also provides the local record IDs identifying the records in which the attribute values are located. Last, it provides the attribute type and associated access control information that apply to each attribute value. If the buffer you provide is not large enough to contain all of the information requested, the `DirLookupGet` function returns the `kOCEMoreData` result code.

When the `DirLookupGet` function completes with either the `noErr` or `kOCEMoreData` result codes, call the `DirLookupParse` function to extract the attribute information from your buffer. You can pass `DirLookupParse` the same parameter that you passed to `DirLookupGet`.

If the `DirLookupGet` function completes with the `kOCEMoreData` result code, you may wish to continue the lookup. When the `DirLookupParse` function completes, it returns values in the `lastRecordIndex`, `lastAttributeIndex`, and `lastAttribute` fields. You may use these as the values of the `startingRecordIndex`, `startingAttrTypeIndex`, and `startingAttribute` fields on a subsequent call to the `DirLookupGet` function. Therefore, you can simply pass the same parameter block to the `DirLookupGet` function as you passed to the `DirLookupParse` function and call the `DirLookupGet` function to continue retrieving information from the point at which it stopped during its previous invocation.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	$0117

RESULT CODES

`noErr`	0	No error
`kOCEParamErr`	–50	Invalid parameter
`kOCETargetDirectoryInaccessible`	–1613	Target catalog is not currently available
`kOCENoSuchDNode`	–1615	Can’t find specified dNode
`kOCENoSuchRecord`	–1618	Can’t find specified record
`kOCENoSuchAttributeValue`	–1619	Can’t find specified attribute value
`kOCEMoreData`	–1623	More data available
`kOCEBadStartingRecord`	–1638	Starting record index out of range
`kOCEBadStartingAttribute`	–1639	Starting attribute index out of range
`kOCERLIsDontMatch`	–1645	RLIs of different records in the record list are not the same

SEE ALSO

The `DirLookupParse` function is described next.

The `RecordID` data structure is described in the chapter “AOCE Utilities” in this book.

The `Attribute` data structure is also described in the chapter “AOCE Utilities.”

For an example of continuing the enumeration using the `DirLookupGet` function when the buffer is too small to hold all the information you requested, see “Getting Attribute Value Information” beginning on page 8-174.

DirLookupParse

The `DirLookupParse` function parses the data returned by the `DirLookupGet` function and returns each attribute value to your application by repeatedly calling your callback routine. It also returns information about record IDs and attribute types when you specify callback routines for these purposes.

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

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

Parameter block

→	`ioCompletion`	`ProcPtr`	Your completion routine
←	`ioResult`	`OSErr`	Result code
→	`serverHint`	`AddrBlock`	AppleTalk address of PowerShare server
→	`dsRefNum`	`short`	Personal catalog reference number
→	`identity`	`AuthIdentity`	Requester’s authentication identity
→	`clientData`	`long`	You define this field
→	`aRecordList`	`RecordIDPtr*`	List of record IDs
→	`attrTypeList`	`AttributeTypePtr*`	List of attribute types
→	`eachRecordID`	`ForEachLookupRecordID`	Your callback routine for record information
→	`eachAttrType`	`ForEachAttrTypeLookup`	Your callback routine for attribute type information
→	`eachAttrValue`	`ForEachAttrValue`	Your callback routine for attribute values
→	`recordIDCount`	`unsigned long`	Number of IDs in list
→	`attrTypeCount`	`unsigned long`	Number of types in list
→	`getBuffer`	`Ptr`	Your buffer
→	`getBufferSize`	`unsigned long`	Size of your buffer
←	`lastRecordIndex`	`unsigned long`	Last record ID retrieved
←	`lastAttributeIndex`	`unsigned long`	Last attribute type retrieved
←	`lastAttribute`	`Attribute`	Last attribute value retrieved
←	`attrSize`	`unsigned long`	Length of the attribute value that was too big to fit

See “The Parameter Block Header” on page 8-190 for descriptions of the `ioCompletion`, `ioResult`, `serverHint`, `dsRefNum`, `identity`, and `clientData` fields.

Field descriptions

`aRecordList`	A pointer to an array of pointers to record IDs. Use the same value that you provided to the corresponding `DirLookupGet` function.
`attrTypeList`	A pointer to an array of pointers to attribute types.Use the same value that you provided to the corresponding `DirLookupGet` function.
`eachRecordID`	A pointer to your callback routine for record information. The function declaration for this routine is described on page 8-312. The `DirLookupParse` function calls this routine for each record ID in the buffer. If you are looking up attribute values in a single record, you may not want to provide this callback routine. Set this field to `nil` if you do not want to specify this callback routine.
`eachAttrType`	A pointer to your callback routine for attribute type information. The `DirLookupParse` function passes your callback routine a pointer to an attribute type and the access control mask that applies to the requester for that attribute type. The attribute type always belongs in the record identified in the most recent call to your `MyForEachRecordID` routine. You may set this field to `nil` if you do not want to specify this callback routine. If you are looking up only one attribute type, or you prefer to read the type from the `Attribute` data structure that the `DirLookupParse` function passes to the `MyForEachAttrValue` routine, you may not want to provide this callback routine. However, it is recommended that you supply this callback routine to get the access control information for a given attribute type. Access controls may prohibit you from reading an attribute value. In that case, the `DirLookupParse` function does not call your `MyForEachAttrValue` callback routine even though the attribute value exists. If you do not supply the `MyForEachAttrType` callback routine, you have no way of knowing whether attribute values of the requested type exist for which you are denied read access. The function declaration for this routine is described on page 8-313.
`eachAttrValue`	A pointer to your callback routine for attribute values (`MyForEachAttrValue`). You must provide this callback routine. The function declaration for this routine is described on page 8-314.
`recordIDCount`	The number of elements in the array of pointers to record IDs. Use the same value that you provided to the corresponding `DirLookupGet` function.
`attrTypeCount`	The number of elements in the array of pointers to attribute types. Use the same value that you provided to the corresponding `DirLookupGet` function.
`getBuffer`	A pointer to the buffer containing the information to parse. Use the same buffer that you provided to the `DirLookupGet` function.
`getBufferSize`	The number of bytes in the buffer.
`lastRecordIndex`	The index value of the last record that the `DirLookupParse` function retrieved from your buffer.

`lastAttributeIndex`	The index value of the last attribute type that the `DirLookupParse` function retrieved from your buffer.
`lastAttribute`	An `Attribute` data structure that specifies the last attribute value that the `DirLookupParse` function retrieved from your buffer.
`attrSize`	The length of an attribute value that is too large to fit in the buffer. If your buffer is too small to hold an attribute value, the `DirLookupParse` function sets this field to the length of the attribute value that cannot fit in your buffer and returns the `kOCEMoreAttrValue` result code. In this case, the value in this field represents the minimum size of a buffer capable of holding the attribute value. You must increase the size of your buffer to at least the minimum size and call the `DirLookupGet` function again. When the function does not return the `kOCEMoreAttrValue` result code, this field is undefined.

DESCRIPTION

You call the `DirLookupParse` function to extract the information on attribute values, attribute types, and records placed in a buffer by the `DirLookupGet` function.

When you provide callback routines for record and attribute type information, the `DirLookupGet` function returns the record IDs and attribute types in the same order as you provided in your list of record IDs and attribute types.

You should provide a callback routine for record information if you request information on more than one record. All of the attribute values that the `DirLookupParse` function passes to your callback routine for attribute values (`MyForEachAttrValue`) belong to the record identified in the most recent call to your callback routine for record information (`MyForEachRecordID`). If you do not provide this routine, you cannot determine the record to which an attribute type or value belongs. In addition, a callback routine for record information allows you to distinguish between the case where a record exists but an attribute type does not exist and the case where a record does not exist.

Although it is optional, you should provide a callback routine for attribute types (`MyForEachAttrType`) because it receives access control information about every attribute type in your buffer. If you do not have read access to an attribute type, the `DirLookupParse` function does not call your callback routine for the corresponding attribute values even though those attribute values are present in your buffer. By providing a callback routine for attribute types, you can detect the presence of attribute values for which you do not have read access.

The `DirLookupParse` function returns the `kOCEMoreData` result code if it reaches the end of the buffer and finds that the `DirLookupGet` function did not return all the data requested. In this case, you can call the `DirLookupGet` function again to retrieve more data. The `DirLookupParse` function sets the values of the `lastRecordIndex`, `lastAttributeIndex`, and `lastAttribute` fields in the parameter block to indicate the last items it retrieved from your buffer. These fields correspond to the `startingRecordIndex`, `startingAttrTypeIndex`, and `startingAttribute` fields in the `DirLookupGet` function's parameter block. You can use the same parameter block when you call the `DirLookupGet` function again, and it will continue retrieving data at the point where it stopped the first time you called it.

If the `DirLookupParse` function returns the `kOCEMoreAttrValue` result code, you must increase the size of your buffer before calling the `DirLookupGet` function again. There are two conditions in which an attribute value may not fit in your buffer. The first occurs when your buffer already contains some data and the remaining space is insufficient to store the next `Attribute` data structure. In this case, `DirLookupGet` returns the `kOCEMoreData` result code. Such an attribute value will be stored in your buffer the next time you call the `DirLookupGet` function. The second condition occurs when the size of an attribute value exceeds the size of your buffer. Such an attribute value will not fit even when your buffer is empty. In this second case, the `DirLookupGet` function completes with the `kOCEMoreData` result code; the corresponding `DirLookupParse` function call stores the length of the oversized attribute value in the `attrSize` field and returns the `kOCEMoreAttrValue` result code. Before calling the `DirLookupGet` function again, you must increase the size of your buffer to accommodate the oversized attribute value.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	**Selector**
`_oceTBDispatch`	$0102

RESULT CODES

`noErr`	0	No error
`kOCEParamErr`	–50	Invalid parameter
`kOCEReadAccessDenied`	–1540	Identity lacks read access privileges
`kOCEMoreData`	–1623	More data available
`kOCEMoreAttrValue`	–1640	Buffer too small for a single attribute value

SEE ALSO

The `DirLookupGet` function is described on page 8-276.

The function declaration for your callback routine that processes record information is described on page 8-312.

The function declaration for your callback routine that processes attribute type information is described on page 8-313.

The function declaration for your callback routine that processes attribute value information is described on page 8-314.

For an example of continuing the enumeration using the `DirLookupParse` function when the buffer is too small to hold all the information you requested, see “Getting Attribute Value Information” beginning on page 8-174.

DirDeleteAttributeType

The `DirDeleteAttributeType` function deletes an attribute type and its associated values from a particular record.

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

`paramBlock`
Pointer to a parameter block.

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

Parameter block

→	`ioCompletion`	`ProcPtr`	Your completion routine
←	`ioResult`	`OSErr`	Result code
→	`serverHint`	`AddrBlock`	AppleTalk address of the PowerShare server
→	`dsRefNum`	`short`	Personal catalog reference number
→	`identity`	`AuthIdentity`	Requester's authentication identity
→	`clientData`	`long`	You define this field
→	`aRecord`	`RecordIDPtr`	Target record
→	`attrType`	`AttributeTypePtr`	Target attribute type

See "The Parameter Block Header" on page 8-190 for descriptions of the `ioCompletion`, `ioResult`, `serverHint`, `dsRefNum`, `identity`, and `clientData` fields.

Field descriptions

`aRecord` A pointer to a record ID that identifies the record in which the target attribute type is located. You must provide the record location information unless the record exists in a personal catalog. If the catalog in which the record resides supports record creation IDs, you must provide the creation ID; otherwise, you must provide the record name and type.

`attrType` A pointer to the attribute type that you want to delete.

DESCRIPTION

You call the `DirDeleteAttributeType` function to delete an existing attribute type. If any attribute values exist for that type, the function first deletes the values and then deletes the attribute type. If you do not have access privileges to delete the attribute type, the function returns the `kOCEWriteAccessDenied` result code.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	**Selector**
`_oceTBDispatch`	$0130

RESULT CODES

`noErr`	0	No error
`kOCEWriteAccessDenied`	–1541	Identity lacks write access privileges
`kOCETargetDirectoryInaccessible`	–1613	Target catalog is not currently available
`kOCENoSuchRecord`	–1618	Can’t find specified record
`kOCENoSuchAttributeType`	–1642	Can’t find specified attribute type

SEE ALSO

The `RecordID` data structure is described in the chapter “AOCE Utilities” in this book.

The `AttributeType` data structure is also described in the chapter “AOCE Utilities.”

Access controls are discussed in the section “Getting Access Controls” beginning on page 8-169.

Records and attributes are described in the section “Catalog Records and Attributes” beginning on page 8-164.

DirEnumerateAttributeTypesGet

The `DirEnumerateAttributeTypesGet` function returns information about the attribute types in a record.

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

`paramBlock`
Pointer to a parameter block.

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

Parameter block

→	`ioCompletion`	`ProcPtr`	Your completion routine
←	`ioResult`	`OSErr`	Result code
→	`serverHint`	`AddrBlock`	AppleTalk address of PowerShare server
→	`dsRefNum`	`short`	Personal catalog reference number
→	`identity`	`AuthIdentity`	Requester's authentication identity
→	`clientData`	`long`	You define this field
→	`aRecord`	`RecordIDPtr`	Target record
→	`startingAttrType`	`AttributeTypePtr`	Attribute type to start from
→	`includeStartingPoint`	`Boolean`	Begin enumeration with starting point?
↔	`getBuffer`	`Ptr`	Your buffer
→	`getBufferSize`	`unsigned long`	Size of your buffer

See "The Parameter Block Header" on page 8-190 for descriptions of the `ioCompletion`, `ioResult`, `serverHint`, `dsRefNum`, `identity`, and `clientData` fields.

Field descriptions

`aRecord` A pointer to a record ID that identifies the record about which you are requesting attribute type information. You must provide the record location information unless the record exists in a personal catalog. If the catalog in which the record resides supports record creation IDs, you must provide the creation ID; otherwise, you must provide the record name and type.

`startingAttrType`
A pointer to the attribute type within the record at which you want the `DirEnumerateAttributeTypesGet` function to begin the enumeration. Set this field to `nil` to start with the first attribute type in the record. If the `DirEnumerateAttributeTypesGet` function completes with the `kOCEMoreData` result code, you can continue the enumeration by setting this field to the value of the last `attrType` parameter passed to your callback routine by the `DirEnumerateAttributeTypesParse` function.

`includeStartingPoint`
A Boolean value that determines how this function interprets the `startingAttrType` field. Set this field to `true` if you want the `DirEnumerateAttributeTypesGet` function to return information about attribute types beginning with the one you specify in the `startingAttrType` field. If you set this field to `false`, the function returns information starting with the attribute type immediately after the one specified by the `startingAttrType` field.

`getBuffer` A pointer to the buffer in which the function stores the attribute types it found in the specified record. You provide this buffer.

`getBufferSize` The number of bytes in the buffer.

DESCRIPTION

You call the `DirEnumerateAttributeTypesGet` function to obtain a list of the attribute types that are present in a particular record.

If your buffer is not large enough to contain all of the information requested, the `DirEnumerateAttributeTypesGet` function returns the `kOCEMoreData` result code.

When the `DirEnumerateAttributeTypesGet` function completes with either the `noErr` or `kOCEMoreData` result codes, you provide a pointer to your buffer to the `DirEnumerateAttributeTypesParse` function, which extracts the attribute type information from the buffer.

If your buffer is too small to hold all of the requested information, you can get additional information by calling the `DirEnumerateAttributeTypesGet` function again. As the value of the `startingAttrType` field, use the value of the last `attrType` parameter passed to your callback routine by the `DirEnumerateAttributeTypesParse` function. The `DirEnumerateAttributeTypesGet` function will continue the enumeration starting with the next attribute type as determined by the value of the `includeStartingPoint` field.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	$0112

RESULT CODES

`noErr`	0	No error
`kOCEParamErr`	–50	Invalid parameter
`kOCEReadAccessDenied`	–1540	Identity lacks read access privileges
`kOCETargetDirectoryInaccessible`	–1613	Target catalog is not currently available
`kOCENoSuchDNode`	–1615	Can’t find specified dNode
`kOCENoSuchRecord`	–1618	Can’t find specified record
`kOCEMoreData`	–1623	More data available
`kOCEDirectoryNotFoundErr`	–1630	Can’t find catalog

SEE ALSO

The `RecordID` data structure is described in the chapter “AOCE Utilities” in this book.

The `AttributeType` data structure is also described in the chapter “AOCE Utilities.”

The `DirEnumerateAttributeTypesParse` function is described next.

For an example of continuing the enumeration using the `DirEnumerateAttributeTypesGet` function when the buffer is too small to hold all the information you requested, see “Getting Attribute Type Information” beginning on page 8-178.

DirEnumerateAttributeTypesParse

The `DirEnumerateAttributeTypesParse` function parses the data returned by the `DirEnumerateAttributeTypesGet` function and returns each attribute type to your application by repeatedly calling your callback routine.

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

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

Parameter block

→	`ioCompletion`	`ProcPtr`	Your completion routine
←	`ioResult`	`OSErr`	Result code
→	`serverHint`	`AddrBlock`	AppleTalk address of the PowerShare server
→	`dsRefNum`	`short`	Personal catalog reference number
→	`identity`	`AuthIdentity`	Requester's authentication identity
→	`clientData`	`long`	You define this field
→	`aRecord`	`RecordIDPtr`	Target record
→	`eachAttrType`	`ForEachAttrType`	Your callback routine
→	`getBuffer`	`Ptr`	Your buffer
→	`getBufferSize`	`unsigned long`	Size of your buffer

See "The Parameter Block Header" on page 8-190 for descriptions of the `ioCompletion`, `ioResult`, `serverHint`, `dsRefNum`, `identity`, and `clientData` fields.

Field descriptions

`aRecord`	A pointer to a record ID that identifies the record about which you want to obtain attribute type information. Use the same value that you provided to the corresponding `DirEnumerateAttributeTypesGet` function.
`eachAttrType`	A pointer to your callback routine. The function declaration for this routine is described on page 8-310.
`getBuffer`	A pointer to the buffer containing the attribute types to parse. Use the same buffer that you provided to the corresponding `DirEnumerateAttributeTypesGet` function.
`getBufferSize`	The number of bytes in the buffer. Use the same value that you provided to the corresponding `DirEnumerateAttributeTypesGet` function.

DESCRIPTION

You call the `DirEnumerateAttributeTypesParse` function to extract the attribute types placed in a buffer by the `DirEnumerateAttributeTypesGet` function. You must provide a callback routine that the `DirEnumerateAttributeTypesParse` function calls for each attribute type that it finds in the buffer.

The `DirEnumerateAttributeTypesParse` function completes when it has finished parsing the contents of your buffer or when your callback routine returns `true`. The function returns the `kOCEMoreData` result code if it reaches the end of the buffer and finds that the `DirEnumerateAttributeTypesGet` function did not return all the data requested. If you want to continue the enumeration, you can call the `DirEnumerateAttributeTypesGet` function again. In your next call to the `DirEnumerateAttributeTypesGet` function, set `startingAttrType` to the value of the last `attrType` parameter passed to your callback routine by the `DirEnumerateAttributeTypesParse` function.

If your callback routine returns `true`, the `DirEnumerateAttributeTypesParse` function completes with the `noErr` result code.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	**Selector**
`_oceTBDispatch`	$0103

RESULT CODES

`noErr`	0	No error
`kOCEParamErr`	–50	Invalid parameter
`kOCEMoreData`	–1623	More data available

SEE ALSO

The `RecordID` data structure is described in the chapter “AOCE Utilities” in this book.

The function declaration for your callback routine is described on page 8-310.

The `DirEnumerateAttributeTypesGet` function is described on page 8-285.

For an example of continuing the enumeration using the `DirEnumerateAttributeTypesParse` function when the buffer is too small to hold all the information you requested, see “Getting Attribute Type Information” beginning on page 8-178.

Reading Access Controls for dNodes, Records, and Attribute Types

The functions in this section identify requestor categories and access controls. There are five categories of requestors. You use the `OCEGetAccessControlDSSpec` function to read category masks that identify the type of requestor about which you want information. See the section "Getting Access Controls" beginning on page 8-169 for more information on access controls and requestor categories.

You can use the other functions described in this section to read the access control masks for a dNode, a record, or an attribute type.

OCEGetAccessControlDSSpec

The `OCEGetAccessControlDSSpec` function returns a `DSSpec` that you can use to get access controls.

```
pascal DSSpec *OCEGetAccessControlDSSpec (const CategoryMask
                                               categoryBitMask);
```

`categoryBitMask`
: A value indicating the type of `DSSpec` you want `OCEGetAccessControlDSSpec` to return.

DESCRIPTION

Given one of the `categoryBitMask` values, the `OCEGetAccessControlDSSpec` function returns to you a pointer to a `DSSpec` structure that corresponds to the particular `categoryBitMask` value. You can then use, in a `DirGet`*xxx*`AccessControlGet` function, the `DSSpec` that is returned to you. The `categoryBitMask` value identifies a type of requestor, such as owner or guest. For more information on how to use the `categoryBitMask` value to obtain a `DSSpec` structure, see "Types of Requesters" beginning on page 8-169.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	**Selector**
`__OCEUtils`	$0345

RESULT CODES

`noErr`	0	No error

SEE ALSO

The `CategoryMask` data type is described in “Getting Access Controls” beginning on page 8-169.

The `DSSpec` data structure is described in the chapter “Utility Manager” in this book.

DirGetDNodeAccessControlGet

The `DirGetDNodeAccessControlGet` function returns access control information for a dNode that you specify.

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

`paramBlock`
Pointer to a parameter block.

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

Parameter block

→	`ioCompletion`	`ProcPtr`	Your completion routine
←	`ioResult`	`OSErr`	Result code
→	`serverHint`	`AddrBlock`	AppleTalk address of the PowerShare server
→	`dsRefNum`	`short`	Personal catalog reference number
→	`identity`	`AuthIdentity`	Requester’s authentication identity
→	`clientData`	`long`	You define this field
→	`pRLI`	`PackedRLIPtr`	Target dNode
→	`forCurrentUserOnly`	`Boolean`	Return only requester’s access controls?
→	`startingPoint`	`DSSpec*`	Object to start from
→	`includeStartingPoint`	`Boolean`	Begin enumeration with starting point?
↔	`getBuffer`	`Ptr`	Your buffer
→	`getBufferSize`	`unsigned long`	Size of your buffer

See “The Parameter Block Header” on page 8-190 for descriptions of the `ioCompletion`, `ioResult`, `serverHint`, `dsRefNum`, `identity`, and `clientData` fields.

Field descriptions

`pRLI` A pointer to packed record location information that specifies the dNode about which you want access control information. The function ignores this field when you specify a personal catalog in the `dsRefNum` field.

`forCurrentUserOnly`
A Boolean value that indicates what access control information the function returns. Set this field to `true` if you want only access control information for the requester specified in the `identity` field. If you set this field to `false`, the function returns access control information for each object on the dNode's access control list.

`startingPoint` A pointer to the object on the access control list from which the function begins to retrieve information. Set this field to `nil` to start with the first object. If the function completes with the `kOCEMoreData` result code, you can set this field to the value that `DirGetDNodeAccessControlParse` last passed to the `dsObj` parameter of your callback routine. Then call `DirGetDNodeAccessControlGet` again to continue to obtain information. The function ignores this field if you set `forCurrentUserOnly` to `true`.

`includeStartingPoint`
A Boolean value that determines how the function interprets the `startingPoint` field. Set this field to `true` if you want the function to return access control information beginning with the object specified by the `startingPoint` field. If you set this field to `false`, the function returns information starting with the object after the one specified by the `startingPoint` field. The function ignores this field if you set `forCurrentUserOnly` to `true`.

`getBuffer` A pointer to the buffer in which the function stores the access control information for the dNode you specify. You provide this buffer.

`getBufferSize` The size, in bytes, of your buffer.

DESCRIPTION

You call the `DirGetDNodeAccessControlGet` function to obtain access control information for a dNode that you specify. The information consists of catalog service specifications that identify the objects on the access control list and of the access control masks that apply to these objects. The mask specifies which access privileges the object possesses.

If the buffer you provide is not large enough to contain all of the information requested, the function returns the `kOCEMoreData` result code.

When the function completes with either the `noErr` or `kOCEMoreData` result codes, you use a pointer to your buffer as input to the `DirGetDNodeAccessControlParse` function, which extracts the information from the buffer.

If your buffer is too small to hold all of the requested information, you can get additional information by calling the `DirGetDNodeAccessControlGet` function again, after calling the `DirGetDNodeAccessControlParse` function. Set the `startingPoint` field to the value that `DirGetDNodeAccessControlParse` last passed to the `dsObj` parameter of your callback routine. The `DirGetDNodeAccessControlGet` function will continue to return information starting with the next entry on the dNode's access control list as determined by the value of the `includeStartingPoint` field.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	$012A

RESULT CODES

`noErr`	0	No error
`kOCEParamErr`	–50	Invalid parameter
`kOCETargetDirectoryInaccessible`	–1613	Target catalog is not currently available
`kOCENoSuchdNode`	–1615	Can't find specified dNode
`kOCEMoreData`	–1623	More data available

SEE ALSO

The `PackedRLI` data structure is described in the chapter "AOCE Utilities" in this book.

You can create a `PackedRLI` data structure with the `OCEPackRLI` utility routine. It is also described in the chapter "AOCE Utilities."

The catalog service specification, defined by the `DSSpec` data structure, is also described in the chapter "AOCE Utilities."

The `DirGetDNodeAccessControlParse` function is described next.

For information on the types of objects and the types of access controls specified in the access control mask, see "Getting Access Controls" beginning on page 8-169.

For an example of continuing the enumeration (using the `DirEnumerateAttributeTypesGet` function) when the buffer is too small to hold all the information you requested, see "Getting Attribute Type Information" beginning on page 8-178.

DirGetDNodeAccessControlParse

The `DirGetDNodeAccessControlParse` function parses the access control information returned by the `DirGetDNodeAccessControlGet` function and returns a pointer to each object and its access control mask by repeatedly calling your callback routine.

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

`paramBlock`
Pointer to a parameter block.

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

Parameter block

→	`ioCompletion`	`ProcPtr`	Your completion routine
←	`ioResult`	`OSErr`	Result code
→	`serverHint`	`AddrBlock`	AppleTalk address of the PowerShare server
→	`dsRefNum`	`short`	Personal catalog reference number
→	`identity`	`AuthIdentity`	Requester's authentication identity
→	`clientData`	`long`	You define this field
→	`pRLI`	`PackedRLIPtr`	Target dNode
→	`eachObject`	`ForEachDNodeAccessControl`	Your callback routine
→	`forCurrentUserOnly`	`Boolean`	Return only requester's access info?
→	`startingPoint`	`DSSpec*`	Object to start from
→	`includeStartingPoint`	`Boolean`	Begin enumeration with starting point?
→	`getBuffer`	`Ptr`	Your buffer
→	`getBufferSize`	`unsigned long`	Size of your buffer

See "The Parameter Block Header" on page 8-190 for descriptions of the `ioCompletion`, `ioResult`, `serverHint`, `dsRefNum`, `identity`, and `clientData` fields.

Field descriptions

`pRLI` A pointer to packed record location information that specifies the dNode about which you want access control information. Use the same value that you provided to the `DirGetDNodeAccessControlGet` function.

`eachObject` A pointer to your callback routine. The Catalog Manager passes your callback routine the value that you provide in the `clientData` field, a pointer to a catalog service specification that identifies the object to which the access control masks apply, the active access control mask for the target dNode, and the default access control masks for newly created records and attribute types within that dNode. The function declaration for this routine is described on page 8-319.

`forCurrentUserOnly`	Use the same value that you provided to the `DirGetDNodeAccessControlGet` function.
`startingPoint`	Use the same value that you provided to the `DirGetDNodeAccessControlGet` function.
`includeStartingPoint`	Use the same value that you provided to the `DirGetDNodeAccessControlGet` function.
`getBuffer`	A pointer to the buffer containing the information to parse. Use the same buffer that you provided to the `DirGetDNodeAccessControlGet` function.
`getBufferSize`	The size, in bytes, of your buffer. Use the same value that you provided to the `DirGetDNodeAccessControlGet` function.

DESCRIPTION

You call the `DirGetDNodeAccessControlParse` function to extract the access control information placed in your buffer by the `DirGetDNodeAccessControlGet` function. You must provide a callback routine that the `DirGetDNodeAccessControlParse` function calls for each entry it finds in the buffer.

The `DirGetDNodeAccessControlParse` function completes when it has finished parsing the contents of your buffer or when your callback routine returns `true`. The function returns the `kOCEMoreData` result code if it reaches the end of the buffer and finds that the `DirGetDNodeAccessControlGet` function did not return all the data requested. If you want to continue to obtain information, you can call the `DirGetDNodeAccessControlGet` function again. Set the value of the `startingPoint` field to the value that `DirGetDNodeAccessControlParse` last passed to the `dsObj` parameter of your callback routine.

If your callback routine returns `true`, the `DirGetDNodeAccessControlParse` function completes with the `noErr` result code.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	$012F

RESULT CODES

`noErr`	0	No error
`kOCEParamErr`	–50	Invalid parameter
`kOCEMoreData`	–1623	More data available

SEE ALSO

The `PackedRLI` data structure is described in the chapter "AOCE Utilities" in this book.

The `DSSpec` data structure is also described in the chapter "AOCE Utilities."

The `DirGetDNodeAccessControlGet` function is described on page 8-291.

The function declaration for your callback routine is described on page 8-319.

For information on the types of access controls specified in the access control mask, see "Getting Access Controls" beginning on page 8-169.

For an example of continuing the enumeration (using the `DirEnumerateAttributeTypesParse` function) when the buffer is too small to hold all the information you requested, see "Getting Attribute Type Information" beginning on page 8-178.

DirGetRecordAccessControlGet

The `DirGetRecordAccessControlGet` function returns the access controls of a record that you specify.

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

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

Parameter block

→	`ioCompletion`	`ProcPtr`	Your completion routine
←	`ioResult`	`OSErr`	Result code
→	`serverHint`	`AddrBlock`	AppleTalk address of PowerShare server
→	`dsRefNum`	`short`	Personal catalog reference number
→	`identity`	`AuthIdentity`	Requester's authentication identity
→	`clientData`	`long`	You define this field
→	`aRecord`	`RecordIDPtr`	Target record
→	`forCurrentUserOnly`	`Boolean`	Return only requester's access controls?
→	`startingPoint`	`DSSpec*`	Object to start from
→	`includeStartingPoint`	`Boolean`	Begin enumeration with starting point?
↔	`getBuffer`	`Ptr`	Your buffer
→	`getBufferSize`	`unsigned long`	Size of your buffer

See "The Parameter Block Header" on page 8-190 for descriptions of the `ioCompletion`, `ioResult`, `serverHint`, `dsRefNum`, `identity`, and `clientData` fields.

Field descriptions

`aRecord`	A pointer to a record ID that identifies the record about which you want access control information. You must provide the record location information unless the record resides in a personal catalog. If the catalog in which the record resides supports record creation IDs, you must provide the record creation ID; otherwise, you must provide the record name and type.
`forCurrentUserOnly`	A Boolean value that indicates what access control information the function returns. Set this field to `true` if you want access controls only for the user specified in the `identity` field. If you set this field to `false`, the function returns access controls for each object on the record's access control list.
`startingPoint`	A pointer to the object from which the function begins returning information. Set this field to `nil` to start with the first object. If the function completes with the `kOCEMoreData` result code, you can set this field to the value of the last `dsObj` parameter passed to your callback routine by the parse function and call the function again to continue to obtain information. The function ignores this field if you set `forCurrentUserOnly` to `true`.
`includeStartingPoint`	A Boolean value that determines how the function interprets the `startingPoint` field. Set this field to `true` if you want the function to return access control information beginning with the object specified by the `startingPoint` field. If you set this field to `false`, the function returns information starting with the object after the one specified by the `startingPoint` field. The function ignores this field if you set `forCurrentUserOnly` to `true`.
`getBuffer`	A pointer to the buffer in which the function stores the access control information for the record you specify. You provide this buffer.
`getBufferSize`	The size, in bytes, of your buffer.

DESCRIPTION

You call the `DirGetRecordAccessControlGet` function to obtain access control information for a record that you specify. The information consists of catalog service specifications that identify the objects on the access control list and of the access control masks that apply to these objects. The mask specifies which access privileges the object possesses.

If the buffer you provide is not large enough to contain all of the information requested, the function returns the `kOCEMoreData` result code.

When the function completes with either the `noErr` or `kOCEMoreData` result codes, you use a pointer to your buffer as input to the `DirGetRecordAccessControlParse` function, which extracts the information from the buffer.

If your buffer is too small to hold all of the requested information, you can get additional information by calling the `DirGetRecordAccessControlGet` function again, after calling the `DirGetRecordAccessControlParse` function. Set the value of the `startingPoint` field to the value that `DirGetRecordAccessControlParse` last passed to the `dsObj` parameter of your callback routine. The `DirGetRecordAccessControlGet` function will continue to return information starting with the next entry as determined by the value of the `includeStartingPoint` field.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	**Selector**
`_oceTBDispatch`	$012C

RESULT CODES

`noErr`	0	No error
`kOCEParamErr`	–50	Invalid parameter
`kOCETargetDirectoryInaccessible`	–1613	Target catalog is not currently available
`kOCENoSuchdNode`	–1615	Can’t find specified dNode
`kOCENoSuchRecord`	–1618	Can’t find specified record
`kOCEMoreData`	–1623	More data available

SEE ALSO

The `RecordID` data structure is described in the chapter “AOCE Utilities” in this book.

The `DSSpec` data structure is also described in the chapter “AOCE Utilities.”

The `DirGetRecordAccessControlParse` function is described next.

For information on the types of access controls specified in the access control mask, see “Getting Access Controls” beginning on page 8-169.

For an example of continuing the enumeration (using the `DirEnumerateAttributeTypesGet` function) when the buffer is too small to hold all the information you requested, see “Getting Attribute Type Information” beginning on page 8-178.

DirGetRecordAccessControlParse

The `DirGetRecordAccessControlParse` function parses the access control information returned by the `DirGetRecordAccessControlGet` function and returns a pointer to each object and its access control mask by repeatedly calling your callback routine.

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

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

Parameter block

→	`ioCompletion`	`ProcPtr`	Your completion routine
←	`ioResult`	`OSErr`	Result code
→	`serverHint`	`AddrBlock`	AppleTalk address of the PowerShare server
→	`dsRefNum`	`short`	Personal catalog reference number
→	`identity`	`AuthIdentity`	Requester's authentication identity
→	`clientData`	`long`	You define this field
→	`aRecord`	`RecordIDPtr`	Target record
→	`eachObject`	`ForEachRecordAccessControl`	Your callback routine
→	`forCurrentUserOnly`	`Boolean`	Return only requester's access info?
→	`startingPoint`	`DSSpec*`	Object to start from
→	`includeStartingPoint`	`Boolean`	Begin enumeration with starting point?
→	`getBuffer`	`Ptr`	Your buffer
→	`getBufferSize`	`unsigned long`	Size of your buffer

See "The Parameter Block Header" on page 8-190 for descriptions of the `ioCompletion`, `ioResult`, `serverHint`, `dsRefNum`, `identity`, and `clientData` fields.

Field descriptions

`aRecord`	A pointer to a record ID that identifies the record about which you want access control information. Use the same value that you provided to the `DirGetRecordAccessControlGet` function.
`eachObject`	A pointer to your callback routine. The Catalog Manager passes your callback routine the value that you provide in the `clientData` field, a pointer to a catalog service specification that identifies the object to which the access control masks apply, the active access control masks for the target record and the dNode in which it resides, and the default access control mask for newly created attribute types within that record. The function declaration for this routine is described on page 8-320.
`forCurrentUserOnly`	Use the same value that you provided to the `DirGetRecordAccessControlGet` function.
`startingPoint`	Use the same value that you provided to the `DirGetRecordAccessControlGet` function.
`includeStartingPoint`	Use the same value that you provided to the `DirGetRecordAccessControlGet` function.

`getBuffer`	A pointer to the buffer containing the information to parse. Use the same buffer that you provided to the `DirGetRecordAccessControlGet` function.
`getBufferSize`	The size, in bytes, of your buffer. Use the same value that you provided to the `DirGetRecordAccessControlGet` function.

DESCRIPTION

You call the `DirGetRecordAccessControlParse` function to extract the access control information placed in your buffer by the `DirGetRecordAccessControlGet` function. You must provide a callback routine that the `DirGetRecordAccessControlParse` function calls for each entry it finds in the buffer.

The `DirGetRecordAccessControlParse` function completes when it has finished parsing the contents of your buffer or when your callback routine returns `true`. The function returns the `kOCEMoreData` result code if it reaches the end of the buffer and finds that the `DirGetRecordAccessControlGet` function did not return all the data requested. If you want to continue to obtain information, you can call the `DirGetRecordAccessControlGet` function again. Set the value of the `startingPoint` field to the value that `DirGetRecordAccessControlParse` last passed to the `dsObj` parameter of your callback routine.

If your callback routine returns `true`, the `DirGetRecordAccessControlParse` function completes with the `noErr` result code.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	$0134

RESULT CODES

`noErr`	0	No error
`kOCEParamErr`	–50	Invalid parameter
`kOCEMoreData`	–1623	More data available

SEE ALSO

The `RecordID` data structure is described in the chapter “AOCE Utilities” in this book.

The `DSSpec` data structure is also described in the chapter “AOCE Utilities.”

The `DirGetRecordAccessControlGet` function is described on page 8-296.

The function declaration for your callback routine is described on page 8-320.

For information on the types of access controls specified in the access control mask, see "Getting Access Controls" beginning on page 8-169.

For an example of continuing the enumeration (using the `DirEnumerateAttributeTypesParse` function) when the buffer is too small to hold all the information you requested, see "Getting Attribute Type Information" beginning on page 8-178.

DirGetAttributeAccessControlGet

The `DirGetAttributeAccessControlGet` function returns access control information for an attribute type that you specify.

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

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

Parameter block

→	`ioCompletion`	`ProcPtr`	Your completion routine
←	`ioResult`	`OSErr`	Result code
→	`serverHint`	`AddrBlock`	AppleTalk address of PowerShare server
→	`dsRefNum`	`short`	Personal catalog reference number
→	`identity`	`AuthIdentity`	Requester's authentication identity
→	`clientData`	`long`	You define this field
→	`aRecord`	`RecordIDPtr`	Target record
→	`aType`	`AttributeTypePtr`	Target attribute type
→	`forCurrentUserOnly`	`Boolean`	Return only requester's access info?
→	`startingPoint`	`DSSpec*`	Object to start from
→	`includeStartingPoint`	`Boolean`	Begin enumeration with starting point?
↔	`getBuffer`	`Ptr`	Your buffer
→	`getBufferSize`	`unsigned long`	Size of your buffer

See "The Parameter Block Header" on page 8-190 for descriptions of the `ioCompletion`, `ioResult`, `serverHint`, `dsRefNum`, `identity`, and `clientData` fields.

Field descriptions

`aRecord`	A pointer to a record ID that identifies the record in which the target attribute type resides. You must provide the record location information unless the record resides in a personal catalog. If the catalog in which the record resides supports record creation IDs, you must provide the creation ID; otherwise, you must provide the record name and type.
`aType`	A pointer to the attribute type about which you are requesting access control information.
`forCurrentUserOnly`	A Boolean value that indicates what access controls the function returns. Set this field to `true` if you want access controls only for the user specified in the `identity` field. If you set this field to `false`, the function returns access controls for each object on the attribute type's access control list.
`startingPoint`	A pointer to the object from which the function begins to return information. Set this field to `nil` to start with the first object. If the function completes with the `kOCEMoreData` result code, you can set this field to the value of the last `dsObj` parameter passed to your callback routine by the parse function and call the function again to continue to obtain information. The function ignores this field if you set `forCurrentUserOnly` to `true`.
`includeStartingPoint`	A Boolean value that determines how the function interprets the `startingPoint` field. Set this field to `true` if you want the function to return access control information beginning with the object specified by the `startingPoint` field. If you set this field to `false`, the function returns information starting with the object after the one specified by the `startingPoint` field. The function ignores this field if you set `forCurrentUserOnly` to `true`.
`getBuffer`	A pointer to the buffer in which the function stores the access control information for the attribute type you specify. You provide this buffer.
`getBufferSize`	The size, in bytes, of your buffer.

DESCRIPTION

You call the `DirGetAttributeAccessControlGet` function to obtain access control information for an attribute type that you specify. The information consists of catalog service specifications that identify the objects on the access control list and of the access control masks that apply to these objects. The mask specifies which access privileges the object possesses.

If the buffer you provide is not large enough to contain all of the information requested, the function returns the `kOCEMoreData` result code.

When the function completes with either the `noErr` or `kOCEMoreData` result code, you use a pointer to your buffer as input to the `DirGetAttributeAccessControlParse` function, which extracts the information from the buffer.

If your buffer is too small to hold all of the information requested, you can get additional information by calling the `DirGetAttributeAccessControlGet` function again, after calling the `DirGetAttributeAccessControlParse` function. Set the value of the `startingPoint` field to the value that `DirGetAttributeAccessControlParse` last passed to the `dsObj` parameter of your callback routine. The `DirGetAttributeAccessControlGet` function will continue to return information starting with the next object as determined by the value of the `includeStartingPoint` field.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	$012E

RESULT CODES

`noErr`	0	No error
`kOCEParamErr`	–50	Invalid parameter
`kOCETargetDirectoryInaccessible`	–1613	Target catalog is not currently available
`kOCENoSuchdNode`	–1615	Can't find specified dNode
`kOCENoSuchRecord`	–1618	Can't find specified record
`kOCENoSuchPseudonym`	–1619	Can't find specified pseudonym
`kOCEMoreData`	–1623	More data available

SEE ALSO

The `RecordID` data structure is described in the chapter "AOCE Utilities" in this book.

The `AttributeType` data structure is also described in the chapter "AOCE Utilities."

The `DSSpec` data structure is also described in the chapter "AOCE Utilities."

The `DirGetAttributeAccessControlParse` function is described next.

For information on the types of access controls specified in the access control mask, see "Getting Access Controls" beginning on page 8-169.

For an example of continuing the enumeration (using the `DirEnumerateAttributeTypesGet` function) when the buffer is too small to hold all the information you requested, see "Getting Attribute Type Information" beginning on page 8-178.

DirGetAttributeAccessControlParse

The `DirGetAttributeAccessControlParse` function parses the access control information returned by the `DirGetAttributeAccessControlGet` function and returns a pointer to each object and its access control mask by repeatedly calling your callback routine.

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

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

Parameter block

→	`ioCompletion`	`ProcPtr`	Your completion routine
←	`ioResult`	`OSErr`	Result code
→	`serverHint`	`AddrBlock`	AppleTalk address of the PowerShare server
→	`dsRefNum`	`short`	Personal catalog reference number
→	`identity`	`AuthIdentity`	Requester’s authentication identity
→	`clientData`	`long`	You define this field
→	`aRecord`	`RecordIDPtr`	Target record
→	`aType`	`AttributeTypePtr`	Target attribute type
→	`eachObject`	`ForEachAttributeAccessControl`	Your callback routine
→	`forCurrentUserOnly`	`Boolean`	Return only requester’s access controls?
→	`startingPoint`	`DSSpec*`	Object to start from
→	`includeStartingPoint`	`Boolean`	Begin enumeration with starting point?
→	`getBuffer`	`Ptr`	Your buffer
→	`getBufferSize`	`unsigned long`	Size of your buffer

See “The Parameter Block Header” on page 8-190 for descriptions of the `ioCompletion`, `ioResult`, `serverHint`, `dsRefNum`, `identity`, and `clientData` fields.

Field descriptions

`aRecord`	A pointer to a record ID that identifies the record in which the target attribute type resides. Use the same value that you provided to the `DirGetAttributeAccessControlGet` function.
`aType`	A pointer to an attribute type about which you are requesting access control information. Use the same value that you provided to the `DirGetAttributeAccessControlGet` function.

`eachObject`	A pointer to your callback routine. The Catalog Manager passes your callback routine the value that you provide in the `clientData` field, a pointer to a catalog service specification that identifies the object to which the access control masks apply, and the active access control masks for the target attribute type as well as those for the dNode and record within which it resides. The function declaration for this routine is described on page 8-319.
`forCurrentUserOnly`	Use the same value that you provided to the `DirGetAttributeAccessControlGet` function.
`startingPoint`	Use the same value that you provided to the `DirGetAttributeAccessControlGet` function.
`includeStartingPoint`	Use the same value that you provided to the `DirGetAttributeAccessControlGet` function.
`getBuffer`	A pointer to the buffer containing the information to parse. Use the same buffer that you provided to the `DirGetAttributeAccessControlGet` function.
`getBufferSize`	The size, in bytes, of your buffer. Use the same value that you provided to the `DirGetAttributeAccessControlGet` function.

DESCRIPTION

You call the `DirGetAttributeAccessControlParse` function to extract the access control information placed in your buffer by the `DirGetAttributeAccessControlGet` function. You must provide a callback routine that the `DirGetAttributeAccessControlParse` function calls for each entry it finds in the buffer.

The `DirGetAttributeAccessControlParse` function completes when it has finished parsing the contents of your buffer or when your callback routine returns `true`. The function returns the `kOCEMoreData` result code if it reaches the end of the buffer and finds that the `DirGetAttributeAccessControlGet` function did not return all the data requested. If you want to continue to obtain information, you can call the `DirGetAttributeAccessControlGet` function again. Set the value of the `startingPoint` field to the value that `DirGetAttributeAccessControlParse` last passed to the `dsObj` parameter of your callback routine.

If your callback routine returns `true`, the `DirGetAttributeAccessControlParse` function completes with the `noErr` result code.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	**Selector**
`_oceTBDispatch`	$0138

RESULT CODES

`noErr`	0	No error
`kOCEParamErr`	–50	Invalid parameter
`kOCEMoreData`	–1623	More data available

SEE ALSO

The `RecordID` data structure is described in the chapter “AOCE Utilities” in this book.

The `AttributeType` data structure is also described in the chapter “AOCE Utilities.”

The `DSSpec` data structure is also described in the chapter “AOCE Utilities.”

The `DirGetAttributeAccessControlGet` function is described on page 8-301.

The function declaration for your callback routine is described on page 8-319.

For information on the types of access controls specified in the access control mask, see “Getting Access Controls” beginning on page 8-169.

For an example of continuing the enumeration (using the `DirEnumerateAttributeTypesParse` function) when the buffer is too small to hold all the information you requested, see “Getting Attribute Type Information” beginning on page 8-178.

Cancelling a Catalog Manager Function

You use the function described in this section to cancel a Catalog Manager function that has not completed execution.

DirAbort

The `DirAbort` function cancels a currently executing Catalog Manager function.

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

`paramBlock`
Pointer to a parameter block.

Parameter block

←	`ioResult`	`OSErr`	Result code
→	`serverHint`	`AddrBlock`	AppleTalk address of the PowerShare server
→	`dsRefNum`	`short`	Personal catalog reference number
→	`identity`	`AuthIdentity`	Requester’s authentication identity
→	`pb`	`union DirParamBlock*`	Function to cancel

See "The Parameter Block Header" on page 8-190 for descriptions of the `ioResult`, `serverHint`, `dsRefNum`, and `identity` fields.

Field descriptions

`pb`	A pointer to the `DirParamBlock` parameter block for the function you want to cancel.

DESCRIPTION

You call the `DirAbort` function to cancel a Catalog Manager function that has not completed execution. If the function that you want to cancel addresses a PowerShare catalog or a personal catalog, the Catalog Manager attempts the cancel operation. If the function that you want to cancel addresses an external catalog, the CSAM driver attempts the cancel operation. If the Catalog Manager or the CSAM driver does not support the `DirAbort` function for the executing function that you specify, the function returns the `kOCEAbortNotSupportedForThisCall` result code.

PowerShare and personal catalogs support the `DirAbort` function for the `DirFindADAPDirectoryByNetSearch` and `DirNetSearchADAPDirectoriesGet` functions only.

IMPORTANT

Because the `DirAbort` function makes references to fields in the parameter block associated with the function that you want to cancel, you must not alter or dispose of that parameter block before the `DirAbort` function has completed. ▲

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	$011B

RESULT CODES

`noErr`	0	No error
`kOCEAbortNotSupportedForThisCall`	–1631	Abort not supported

SEE ALSO

The `DirFindADAPDirectoryByNetSearch` function is described on page 8-232.

The `DirNetSearchADAPDirectoriesGet` function is described on page 8-234.

Application-Defined Functions

This section contains descriptions of the completion routine you can provide when you call the Catalog Manager asynchronously and of the callback routines that you provide to Catalog Manager functions that parse a buffer's contents.

The information on callback routines in this introduction applies to all callback routines and is not repeated in individual routine descriptions.

The Catalog Manager manages all of the buffers associated with pointers that it passes to a callback routine. You must copy the data in these buffers if you want to refer to it after your callback routine completes execution.

When a callback routine returns `false`, the parse function continues parsing the results in your buffer. When a callback routine returns `true`, the parse function completes with a `noErr` result code. If a parse function invokes a callback routine and passes it the last item in the buffer and the callback routine returns `false`, the parse routine completes with either a `noErr` or a `kOCEMoreData` result code, depending on the result code of the corresponding "get" function.

A Catalog Manager function always calls a callback routine at deferred-task time so that it will work properly if the computer is using virtual memory. Because these functions restore the value of your application's A5 register before calling a callback routine, a callback routine has access to your application's global variables. Your callback routine can allocate memory if you make a synchronous call to the function that invokes it.

See "Callback Routines" on page 8-168 for more information about the restrictions that apply to callback routines.

MyCompletionRoutine

You may provide a completion routine when you call a Catalog Manager function asynchronously.

```
void MyCompletionRoutine (DirParamBlockPtr paramBlk);
```

`paramBlk`	A pointer to the parameter block that you provided to the Catalog Manager function that is calling your completion routine.

DESCRIPTION

You can provide a completion routine to any Catalog Manager functions that you can call asynchronously by passing a pointer to the completion routine in the `ioCompletion` field of the `DirParamBlock` parameter block. If you provide a completion routine, it executes when the asynchronous request completes execution.

The Catalog Manager saves the value of your A5 register at the time you call a Catalog Manager routine and then restores the A5 value before calling the completion routine.

The Catalog Manager always calls completion routines in deferred-task time. Running at deferred-task time is a safe practice when using virtual memory.

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

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

```
PROCEDURE MyCompletion(paramBlk: DirParamBlockPtr);
```

Note that if you do not want to specify a completion routine for an asynchronous function call, you can specify `nil` in the `ioCompletion` field and poll the `ioResult` field of the parameter block header. When you call a Catalog Manager function asynchronously, the function sets the `ioResult` field in the parameter block to 1 to indicate that the routine has not yet completed execution. When the routine completes execution, it sets the `ioResult` field to the actual function result. If you poll, you should do so within a loop that calls the `WaitNextEvent` routine so that other processes get execution time. If you poll in a tight loop, you may cause a deadlock condition.

ASSEMBLY-LANGUAGE INFORMATION

If you write it in assembly language, your completion routine gets a pointer to the parameter block in the A0 register and the Catalog Manager function result code in the D0 register. The function result code is also available in the `ioResult` field of the parameter block.

MyForEachRecordID

The `MyForEachRecordID` function is a callback routine you must provide if you call the `DirEnumeratePseudonymParse` function.

```
pascal Boolean MyForEachRecordID (long clientData,
                                  const RecordID *recordID);
```

`clientData`
: The value that you provided in the `clientData` field of the parameter block that you passed to the `DirEnumeratePseudonymParse` function. You can use this parameter for whatever purpose you choose. For example, if you make multiple asynchronous calls to the `DirEnumeratePseudonymParse` function, you can use this parameter to match calls to this routine with a particular call to the `DirEnumeratePseudonymParse` function.

`recordID`
: A pointer to a record ID containing the name, the type, and the creation ID of a pseudonym. The record location information is unspecified because the pseudonym resides in the same catalog and dNode as the target record.

DESCRIPTION

The `DirEnumeratePseudonymParse` function calls your callback routine for each pseudonym it finds in a buffer previously filled by the `DirEnumeratePseudonymGet` function.

SEE ALSO

Read the introduction to "Application-Defined Functions" on page 8-308 for important information that applies to all callback routines.

The `DirEnumeratePseudonymParse` function is described on page 8-262.

The `RecordID` data structure is described in the chapter "AOCE Utilities" in this book.

MyForEachAttrType

The `MyForEachAttrType` function is a callback routine you must provide if you call the `DirEnumerateAttributeTypesParse` function.

```
pascal Boolean MyForEachAttrType (long clientData,
                                  const AttributeType *attrType);
```

`clientData`	The value that you provided in the `clientData` field of the parameter block that you passed to the `DirEnumerateAttributeTypesParse` function. You can use this field for whatever purpose you choose. For example, if you make multiple asynchronous calls to the `DirEnumerateAttributeTypesParse` function, you can use this field to match calls to this routine with a particular call to the `DirEnumerateAttributeTypesParse` function.
`attrType`	A pointer to an `AttributeType` data structure.

DESCRIPTION

The `DirEnumerateAttributeTypesParse` function calls your callback routine for each attribute type that it finds in a buffer previously filled by the `DirEnumerateAttributeTypesGet` function.

SEE ALSO

Read the introduction to "Application-Defined Functions" on page 8-308 for important information that applies to all callback routines.

The `DirEnumerateAttributeTypesParse` function is described on page 8-288

The `AttributeType` data structure is described in the chapter "AOCE Utilities" in this book.

MyForEachDirectory

The `MyForEachDirectory` function is a callback routine you must provide if you call the `DirEnumerateDirectoriesParse` function.

```
pascal Boolean MyForEachDirectory (long clientData,
                                   const DirectoryName *dirName,
                                   const DirDiscriminator *discriminator,
                                   DirGestalt features);
```

`clientData`
: The value that you provided in the `clientData` field of the parameter block that you passed to the `DirEnumerateDirectoriesParse` function. You can use this parameter for whatever purpose you choose. For example, if you make multiple asynchronous calls to the `DirEnumerateDirectoriesParse` function, you can use this parameter to match calls to this routine with a particular call to the `DirEnumerateDirectoriesParse` function.

`dirName` A pointer to the name of a catalog.

`discriminator`
: A pointer to a `DirDiscriminator` data structure that differentiates between catalogs that share the same name.

`features` A set of flags that indicates the features that the catalog supports.

DESCRIPTION

The `DirEnumerateDirectoriesParse` function calls your callback routine for each catalog entry that it finds in a buffer previously filled by the `DirEnumerateDirectoriesGet` function. Your callback routine receives a pointer to the catalog's name, a pointer to its discriminator value, and the feature flags that indicate what features the catalog supports.

The `DirEnumerateDirectoriesParse` function does not supply information about personal catalogs.

SEE ALSO

Read the introduction to "Application-Defined Functions" on page 8-308 for important information that applies to all callback routines.

The `DirEnumerateDirectoriesParse` function is described on page 8-199.

The `DirDiscriminator` data structure is described in the chapter "AOCE Utilities" in this book.

For a description of catalog feature flags, see "Feature Flag Bit Array" beginning on page 8-186.

MyForEachLookupRecordID

The `MyForEachLookupRecordID` function is a callback routine that you may provide if you call the `DirLookupParse` function and you want to get record information.

```
pascal Boolean MyForEachLookupRecordID (long clientData,
                                 const RecordID *recordID);
```

`clientData`
: The value that you provided in the `clientData` field of the parameter block that you passed to the `DirLookupParse` function. You can use this parameter for whatever purpose you choose. For example, if you make multiple asynchronous calls to the `DirLookupParse` function, you can use this field to match calls to this routine with a particular call to the `DirLookupParse` function.

`recordID` A pointer to a record ID.

DESCRIPTION

The `DirLookupParse` function calls your callback routine for each record ID that it finds in a buffer previously filled by the `DirLookupGet` function.

This callback routine is optional. If you look up attribute values only in a single record, you may not want to provide this routine. However, then you cannot distinguish between the case where a record exists but an attribute type does not exist and the case where a record does not exist.

If you look up attribute values in multiple records, you need to provide this routine to associate attribute types and attribute values with the record to which they belong.

SEE ALSO

Read the introduction to “Application-Defined Functions” on page 8-308 for important information that applies to all callback routines.

The `RecordID` data structure is described in the chapter “AOCE Utilities” in this book.

The `DirLookupParse` function is described on page 8-279.

MyForEachAttrTypeLookup

The `MyForEachAttrTypeLookup` function is a callback routine which you may provide if you call the `DirLookupParse` function and you want to retrieve attribute type information.

```
pascal Boolean MyForEachAttrTypeLookup (long clientData,
                                        const AttributeType *attrType,
                                        AccessMask myAttrAccMask);
```

`clientData`
: The value that you provided in the `clientData` field of the parameter block that you passed to the `DirLookupParse` function. You can use this parameter for whatever purpose you choose. For example, if you make multiple asynchronous calls to the `DirLookupParse` function, you can use this parameter to match calls to this routine with a particular call to the `DirLookupParse` function.

`attrType` A pointer to an `AttributeType` data structure.

`myAttrAccMask`
: The requester's access control mask for this attribute type. If the `myAttrAccMask` parameter indicates that you do not have read access permission for this attribute type, the `DirLookupParse` function does not call your `MyForEachAttrValue` callback routine for this attribute type.

DESCRIPTION

The `DirLookupParse` function calls this callback routine for each attribute type that it finds in a buffer previously filled by the `DirLookupGet` function.

If you provided a callback routine for record ID information (`MyForEachLookupRecordID`), you can associate the attribute type that the function passes here with the record ID that the `DirLookupParse` function most recently passed to your `MyForEachLookupRecordID` callback routine.

This callback routine is optional. However, it provides access control information about each attribute type that you requested. If you do not have read access to an attribute type that you requested, you can still detect the presence of attribute values of that type in a record. However, you cannot read those attribute values because the `DirLookupParse` function does not call your attribute value callback routine (`MyForEachAttrValue`) when you lack read access to the attribute type. If you do not provide this routine, you have no way of knowing that attribute values that you requested exist in a record when you lack read access to their attribute type.

SEE ALSO

Read the introduction to “Application-Defined Functions” on page 8-308 for important information that applies to all callback routines.

The `AttributeType` data structure is described in the chapter “AOCE Utilities” in this book.

The values of the access mask are described in “Getting Access Controls” beginning on page 8-169.

The `DirLookupParse` function is described on page 8-279.

The `DirLookupGet` function is described on page 8-276.

The `MyForEachLookupRecordID` routine is described on page 8-312.

The `MyForEachAttrValue` routine is described next.

MyForEachAttrValue

The `MyForEachAttrValue` function is a callback routine you must provide if you call the `DirLookupParse` function.

```
pascal Boolean MyForEachAttrValue (long clientData,
                                    const Attribute *attribute);
```

`clientData`	The value that you provided in the `clientData` field of the parameter block that you passed to the `DirLookupParse` function. You can use this parameter for whatever purpose you choose. For example, if you make multiple asynchronous calls to the `DirLookupParse` function, you can use this parameter to match calls to this routine with a particular call to the `DirLookupParse` function.
`attribute`	A pointer to an `Attribute` data structure that specifies an attribute value. Note that this attribute value has the attribute type specified in the most recent call to your `MyForEachAttrTypeLookup` callback routine, and it is contained in the record specified by the most recent call to your `MyForEachLookupRecordID` callback routine.

DESCRIPTION

The `DirLookupParse` function calls this callback routine for each attribute value that it finds in a buffer previously filled by the `DirLookupGet` function.

If you provided a callback routine for record ID information (`MyForEachLookupRecordID`), you can associate the attribute value that the function passes here with the local record ID that the `DirLookupParse` function most recently passed to your `MyForEachLookupRecordID` callback routine. If you did not provide a callback routine for record ID information and you are looking up attribute values in multiple records, you must devise your own system of matching attribute values with the records to which they belong.

If you provided a callback routine for attribute type information (`MyForEachAttrTypeLookup`), you can associate the attribute value that the function passes here with the access controls that the `DirLookupParse` function most recently passed to your `MyForEachAttrTypeLookup` callback routine. If you did not provide a callback routine for attribute type information, you cannot detect the presence of attribute values in a record when you lack read access to their attribute types. The `DirLookupParse` function does not call this callback routine for an attribute value if you do not have read access to its attribute type.

SEE ALSO

Read the introduction to "Application-Defined Functions" on page 8-308 for important information that applies to all callback routines.

The `DirLookupParse` function is described on page 8-279.

The `DirLookupGet` function is described on page 8-276.

The `Attribute` data structure is described in the chapter "AOCE Utilities" in this book.

The `MyForEachLookupRecordID` routine is described on page 8-312.

The `MyForEachAttrTypeLookup` routine is described on page 8-313.

MyForEachDirEnumSpec

The `MyForEachDirEnumSpec` function is a callback routine you must provide if you call the `DirEnumerateParse` function.

```
pascal Boolean MyForEachDirEnumSpec (long clientData,
                                     const DirEnumSpec *enumSpec);
```

`clientData`
: The value that you provided in the `clientData` field of the parameter block that you passed to the `DirEnumerateParse` function.You can use this parameter for whatever purpose you choose. For example, if you make multiple asynchronous calls to the `DirEnumerateParse` function, you can use this parameter to match calls to this routine with a particular call to the `DirEnumerateParse` function.

`enumSpec`
: A pointer to an enumeration specification data structure. The value of the `enumFlag` field of the `DirEnumSpec` data structure indicates the type of entity about which information is being returned. Use the mask constants `kEnumDistinguishedNameMask`, `kEnumAliasMask`, `kEnumPseudonymMask`, and `kEnumDNodeMask` to determine if the

entity is a record, alias, pseudonym, or dNode, respectively. Use the mask constant `kEnumInvisibleMask` to determine if the entity is visible or invisible.

If the `DirEnumerateParse` function is returning information about a dNode or an invisible dNode, the `u` field of the `DirEnumSpec` structure contains a `DNodeID` data structure. The dNode ID consists of the name of the dNode and its dNode number. If the catalog does not support dNode numbers, the dNode number is set to 0.

If the `DirEnumerateParse` function is returning information about a record, an alias, or a pseudonym, the `u` field of the `DirEnumSpec` structure contains a `LocalRecordID` data structure. The local record ID consists of the record's creation ID, name, and type. If the catalog does not support creation IDs, the creation ID is set to 0.

DESCRIPTION

The `DirEnumerateParse` function calls your callback routine for each record, alias, pseudonym, and dNode about which it finds information in a buffer previously filled by the `DirEnumerateGet` function.

Invisible dNodes are typically foreign dNodes, that is, they represent external messaging systems within an AOCE system. Invisible records are typically those that are used in administering an AOCE system. Usually, you would not display information about these to a user. It is up to your application to consider how to handle information about invisible entities.

SEE ALSO

Read the introduction to "Application-Defined Functions" on page 8-308 for important information that applies to all callback routines.

The `DirEnumerateParse` function is described on page 8-220.

The `DirEnumerateGet` function is described on page 8-215.

The `LocalRecordID` data structure is described in the chapter "AOCE Utilities" in this book.

The `DirEnumSpec` data structure is described on page 8-193.

The `DNodeID` data structure is described on page 8-192.

MyForEachRecord

The `MyForEachRecord` function is a callback routine you must provide if you call the `DirFindRecordParse` function.

```
pascal Boolean MyForEachRecord (long clientData,
                                const DirEnumSpec *enumSpec,
                                pRLI PackedRLIPtr);
```

`clientData`
: The value that you provided in the `clientData` field of the parameter block that you passed to the `DirFindRecordParse` function.You can use this parameter for whatever purpose you choose. For example, if you make multiple asynchronous calls to the `DirFindRecordParse` function, you can use this parameter to match calls to this routine with a particular call to the `DirFindRecordParse` function.

`enumSpec`
: A pointer to an enumeration specification data structure. The value of the `enumFlag` field of the `DirEnumSpec` data structure indicates the type of entity about which information is being returned. Use the mask constants `kEnumDistinguishedNameMask`, `kEnumAliasMask`, and `kEnumPseudonymMask` to determine if the entity is a record, alias, or pseudonym, respectively.

`pRLI`
: A pointer to packed record location information that specifies the dNode within which the record, alias, or pseudonym is located.

DESCRIPTION

The `DirFindRecordParse` function calls your callback routine for each record, alias, and pseudonym about which it finds information in a buffer previously filled by the `DirFindRecordGet` function.

SEE ALSO

Read the introduction to “Application-Defined Functions” on page 8-308 for important information that applies to all callback routines.

The `DirFindRecordParse` function is described on page 8-204.

The `DirFindRecordGet` function is described on page 8-201.

The `LocalRecordID` data structure is described in the chapter “AOCE Utilities” in this book.

The `DirEnumSpec` data structure is described on page 8-193.

MyForEachADAPDirectory

The `MyForEachADAPDirectory` function is a callback routine you must provide if you call the `DirNetSearchADAPDirectoriesParse` function.

```
pascal Boolean MyForEachADAPDirectory (long clientData,
                           const DirectoryName *directoryName,
                           const DirDiscriminator *discriminator,
                           DirGestalt features,
                           AddrBlock serverHint);
```

`clientData`
: The value that you provided in the `clientData` field of the parameter block that you passed to the `DirNetSearchADAPDirectoriesParse` function. You can use this parameter for whatever purpose you choose. For example, if you make multiple asynchronous calls to the `DirNetSearchADAPDirectoriesParse` function, you can use this parameter to match calls to this routine with a particular call to the `DirNetSearchADAPDirectoriesParse` function.

`directoryName`
: A pointer to the name of the catalog.

`discriminator`
: A pointer to the value that differentiates two or more catalogs with the same name.

`features`
: A set of feature bit flags for the catalog.

`serverHint`
: The AppleTalk address of a PowerShare server that serves the catalog specified in the `directoryName` and `discriminator` fields.

DESCRIPTION

The `DirNetSearchADAPDirectoriesParse` function calls your callback routine for each PowerShare catalog that it finds in a buffer previously filled by the `DirNetSearchADAPDirectoriesGet` function.

SEE ALSO

Read the introduction to “Application-Defined Functions” on page 8-308 for important information that applies to all callback routines.

The `DirNetSearchADAPDirectoriesParse` function is described on page 8-236.

The `DirDiscriminator` data structure is described in the chapter “AOCE Utilities” in this book.

For a description of catalog feature flags, see “Feature Flag Bit Array” beginning on page 8-186.

The `AddrBlock` data structure is described in the header file AppleTalk.h.

MyForEachDNodeAccessControl

The `MyForEachDNodeAccessControl` function is a callback routine you must provide when you call the `DirGetDNodeAccessControlParse` function.

```
pascal Boolean MyForEachDNodeAccessControl (long clientData,
                        const DSSpec *dsObj,
                        AccessMask activeDnodeAccMask,
                        AccessMask defaultRecordAccMask,
                        AccessMask defaultAttributeAccMask);
```

`clientData`
: The value that you provided in the `clientData` field of the parameter block that you passed to the `DirGetDNodeAccessControlParse` function. You can use this parameter for whatever purpose you choose. For example, if you make multiple asynchronous calls to the `DirGetDNodeAccessControlParse` function, you can use this parameter to match calls to this routine with a particular call to the `DirGetDNodeAccessControlParse` function.

`dsObj`
: A pointer to the object to which the access control mask applies.

`activeDnodeAccMask`
: A mask that specifies the access controls that apply to the object in relation to the dNode.

`defaultRecordAccMask`
: A mask that specifies the default access controls that apply to new records within the dNode.

`defaultAttributeAccMask`
: A mask that specifies the default access controls that apply to new attribute types within records in the dNode.

DESCRIPTION

The `DirGetDNodeAccessControlParse` function calls your callback routine for each entry that it finds in a buffer previously filled by the `DirGetDNodeAccessControlGet` function.

SEE ALSO

Read the introduction to “Application-Defined Functions” on page 8-308 for important information that applies to all callback routines.

The `DirGetDNodeAccessControlGet` function is described on page 8-291.

The `DirGetDNodeAccessControlParse` function is described on page 8-294.

The values of the access mask are described in “Getting Access Controls” beginning on page 8-169.

MyForEachRecordAccessControl

The `MyForEachRecordAccessControl` function is a callback routine you must provide when you call the `DirGetRecordAccessControlParse` function.

```
pascal Boolean MyForEachRecordAccessControl (long clientData,
                        const DSSpec *dsObj,
                        AccessMask activeDnodeAccMask,
                        AccessMask activeRecordAccMask,
                        AccessMask defaultAttributeAccMask);
```

`clientData`
: The value that you provided in the `clientData` field of the parameter block that you passed to the `DirGetRecordAccessControlParse` function. You can use this parameter for whatever purpose you choose. For example, if you make multiple asynchronous calls to the `DirGetRecordAccessControlParse` function, you can use this parameter to match calls to this routine with a particular call to the `DirGetRecordAccessControlParse` function.

`dsObj`
: A pointer to the object to which the access control mask applies.

`activeDnodeAccMask`
: A mask that specifies the access controls that apply to the object in relation to the dNode that contains the record.

`activeRecordAccMask`
: A mask that specifies the access controls that apply to the object in relation to the record.

`defaultAttributeAccMask`
: A mask that specifies the default access controls that apply to new attribute types within the record.

DESCRIPTION

The `DirGetRecordAccessControlParse` function calls your callback routine for each entry that it finds in a buffer previously filled by the `DirGetRecordAccessControlGet` function.

Read the introduction to “Application-Defined Functions” on page 8-308 for important information that applies to all callback routines.

SEE ALSO

The `DirGetRecordAccessControlGet` function is described on page 8-296.

The `DirGetRecordAccessControlParse` function is described on page 8-298.

The values of the access mask are described in “Getting Access Controls” beginning on page 8-169.

MyForEachAttributeAccessControl

The `MyForEachAttributeAccessControl` function is a callback routine you must provide when you call the `DirGetAttributeAccessControlParse` function.

```
pascal Boolean MyForEachAttributeAccessControl (long clientData,
                        const DSSpec *dsObj,
                        AccessMask activeDnodeAccMask,
                        AccessMask activeRecordAccMask,
                           AccessMask activeAttributeAccMask);
```

`clientData`
: The value that you provided in the `clientData` field of the parameter block that you passed to the `DirGetAttributeAccessControlParse` function. You can use this field for whatever purpose you choose. For example, if you make multiple asynchronous calls to the `DirGetAttributeAccessControlParse` function, you can use this field to match calls to this routine with a particular call to the `DirGetAttributeAccessControlParse` function.

`dsObj` A pointer to the object to which the access control mask applies.

`activeDnodeAccMask`
: A mask that specifies the access controls that apply to the object in relation to the dNode containing the record that contains the attribute type.

`activeRecordAccMask`
: A mask that specifies the access controls that apply to the object in relation to the record that contains the attribute type.

`activeAttributeAccMask`
: A mask that specifies the access controls that apply to the object in relation to the attribute type.

DESCRIPTION

The `DirGetAttributeAccessControlParse` function calls your callback routine for each entry that it finds in a buffer previously filled by the `DirGetAttributeAccessControlGet` function.

Read the introduction to “Application-Defined Functions” on page 8-308 for important information that applies to all callback routines.

SEE ALSO

The `DirGetAttributeAccessControlGet` function is described on page 8-301.

The `DirGetAttributeAccessControlParse` function is described on page 8-304.

The values of the access mask are described in “Getting Access Controls” beginning on page 8-169.