Open Directory Reference

Networking > Mac OS X Server

2006-05-23

Contents

Open Directory Reference

Framework:	DirectoryService
Declared in	DirServices.h DirServicesTypes.h DirServicesUtils.h ImageCodec.k.h ImageCompression.k.h
Companion guides	Open Directory Programming Guide Open Directory Plug-in Programming Guide

Overview

This document describes the Open Directory functions, constants and data types for retrieving information stored in directories.

Functions by Task

Open Directory Client Functions

`dsAddAttribute` (page 11)
: Adds an attribute to a record.

`dsAddAttributeValue` (page 12)
: Adds a value to an attribute.

`dsAddChildPIDToReference` (page 13)
: Allows the specified process to use a node reference.

`dsAllocAttributeValueEntry` (page 14)
: Allocates an attribute value entry structure having the specified attribute value.

`dsAppendStringToListAlloc` (page 14)
: Appends a string to a data list.

`dsBuildListFromNodesAlloc` (page 16)
: Fills in a previously allocated data list using one or more data nodes.

`dsBuildFromPath` (page 15)
: Builds a data list from a pathname.

`dsBuildListFromPathAlloc` (page 17)
: Builds a data list from a pathname using a data list that has already been allocated.

`dsBuildListFromStrings` (page 18)
: Builds a data list from strings.

`dsBuildListFromStringsAlloc` (page 18)
: Fills in a previously allocated data list using data from strings.

`dsBuildListFromStringsAllocV` (page 19)
: Fills in a previously allocated data list using data from a list of type `va_list`.

`dsCloseAttributeList` (page 20)
: Disposes of an attribute list reference.

`dsCloseAttributeValueList` (page 20)
: Disposes of an attribute value list reference.

`dsCloseDirNode` (page 21)
: Closes a session with a node.

`dsCloseDirService` (page 22)
: Closes an Open Directory session.

`dsCloseRecord` (page 22)
: Closes a open record.

`dsCreateRecord` (page 23)
: Creates a record.

`dsCreateRecordAndOpen` (page 24)
: Creates a record and opens it.

`dsDataBufferAllocate` (page 25)
: Allocates an Open Directory data buffer.

`dsDataBufferDeAllocate` (page 25)
: Deallocates an Open Directory data buffer.

`dsDataListAllocate` (page 26)
: Allocates a data list.

`dsDataListCopyList` (page 27)
: Copies a data list.

`dsDataListDeallocate` (page 27)
: Deallocates a data list.

`dsDataListGetNodeAlloc` (page 28)
: Gets a data node from a data list.

`dsDataListGetNodeCount` (page 29)
: Gets the number of data nodes in a data list.

`dsDataListInsertAfter` (page 30)
: Inserts a data node in a data list.

`dsDataListMergeListAfter` (page 30)
: Merges two data lists.

`dsDataListDeleteThisNode` (page 28)
: Deletes a data node from a data list.

`dsDataNodeAllocateBlock` (page 31)
: Allocates an Open Directory data node.

`dsDataNodeAllocateString` (page 32)
: Allocates an Open Directory data node using a string.

`dsDataNodeDeAllocate` (page 33)
  Deallocates a data node.

`dsDataNodeGetSize` (page 34)
  Gets the size of a data node's buffer.

`dsDataNodeGetLength` (page 33)
  Gets the length of valid data in a data node's buffer.

`dsDataNodeSetLength` (page 34)
  Sets the length of valid data in a data node's buffer.

`dsDeallocAttributeEntry` (page 35)
  Deallocates an attribute entry structure.

`dsDeallocAttributeValueEntry` (page 35)
  Deallocates an attribute value entry structure.

`dsDeallocRecordEntry` (page 36)
  Deallocates a record entry structure.

`dsDeleteRecord` (page 37)
  Deletes a record.

`dsDoAttributeValueSearch` (page 37)
  Searches a node for records by attribute value.

`dsDoAttributeValueSearchWithData` (page 40)
  Searches for records by attribute type and attribute value.

`dsDoDirNodeAuth` (page 41)
  Performs authentication with a node.

`dsDoDirNodeAuthOnRecordType` (page 43)
  Performs authentication using a record type.

`dsDoMultipleAttributeValueSearch` (page 45)
  Uses multiple attribute values to search a node for records.

`dsDoMultipleAttributeValueSearchWithData` (page 47)
  Searches for records by attribute type and multiple attribute values.

`dsDoPlugInCustomCall` (page 49)
  Exchanges custom information with an Open Directory plug-in.

`dsFindDirNodes` (page 49)
  Finds the registered node names that match a pattern.

`dsFlushRecord` (page 51)
  Writes a record.

`dsGetAttributeEntry` (page 51)
  Gets an attribute entry from a data buffer.

`dsGetAttributeValue` (page 53)
  Gets the value of an attribute from a data buffer.

`dsGetDataLength` (page 54)
  Gets the length of data in a data list.

`dsGetDirNodeCount` (page 54)
  Gets the total number of registered nodes.

`dsGetDirNodeCountWithInfo` (page 55)
  Gets the total number of registered nodes and a change token.

`dsGetDirNodeInfo` (page 56)
: Gets information about a node's attribute types and attribute values.

`dsGetDirNodeList` (page 57)
: Gets the names of registered nodes.

`dsGetDirNodeName` (page 58)
: Gets node names from a buffer.

`dsGetPathFromList` (page 59)
: Gets the path from a data list.

`dsGetRecordAttributeInfo` (page 59)
: Gets information about a record's attribute.

`dsGetRecordAttributeValueByID` (page 60)
: Uses an attribute value ID to obtain the value of an attribute.

`dsGetRecordAttributeValueByIndex` (page 61)
: Uses an index to get the value of an attribute.

`dsGetRecordAttributeValueByValue` (page 62)
: Verifies the existence of an attribute value within a record.

`dsGetRecordEntry` (page 63)
: Gets the next record from a data buffer.

`dsGetRecordList` (page 64)
: Gets a list of records and puts it in a data buffer.

`dsGetRecordReferenceInfo` (page 66)
: Gets a record's name and type and the number of attributes the record has.

`dsGetRecordNameFromEntry` (page 66)
: Gets the name of a record from a record entry structure.

`dsGetRecordTypeFromEntry` (page 67)
: Gets the type of a record from a record entry structure.

`dsIsDirServiceRunning` (page 67)
: Checks to see if Open Directory is running.

`dsOpenDirNode` (page 68)
: Opens a session with a node.

`dsOpenDirService` (page 69)
: Opens an Open Directory session.

`dsOpenDirServiceProxy` (page 70)
: Opens a remote Open Directory session.

`dsOpenRecord` (page 71)
: Opens a record.

`dsReleaseContinueData` (page 72)
: Releases memory allocated for continuation data.

`dsRemoveAttribute` (page 73)
: Removes an attribute from a record.

`dsRemoveAttributeValue` (page 73)
: Removes an attribute value.

`dsSetAttributeValue` (page 74)
: Sets the value of an attribute.

`dsSetAttributeValues` (page 75)
    Sets multiple values for an attribute.

`dsSetRecordName` (page 76)
    Sets the name of a record.

`dsSetRecordType` (page 77)
    Sets the type of a record.

`dsVerifyDirRefNum` (page 77)
    Verifies that a `tDirReference` is valid.

Open Directory Plug-in Functions

`Initialize` (page 78)
    Initializes the plug-in.

`PeriodicTask` (page 78)
    Performs a periodic task.

`ProcessRequest` (page 79)
    Processes requests.

`SetPluginState` (page 79)
    Sets the plug-in's state.

`Shutdown` (page 79)
    Prepares the plug-in for shut down.

`Validate` (page 80)
    Validates the plug-in.

Functions

dsAddAttribute

Adds an attribute to a record.

```
tDirStatus dsAddAttribute (
   tRecordReference inRecordReference,
   tDataNodePtr inNewAttribute,
   tAccessControlEntryPtr inNewAttributeAccess,
   tDataNodePtr inFirstAttributeValue
);
```

Parameters

inRecordReference
    On input, a value of type tRecordReference (page 89) obtained by previously calling `dsOpenRecord` (page 71) or `dsCreateRecordAndOpen` (page 24) that represents the record to which an attribute is to be added.

inNewAttribute
: On input, a value of type `tDataNodePtr` (page 88) that points to a `tDataNode` (page 88) structure allocated by calling `dsDataNodeAllocateBlock` (page 31) or `dsDataNodeAllocateString` (page 32) that contains the name of the attribute that is to be added.

inNewAttributeAccess
: Reserved for this release. On input, set `inNewAttributeAccess` to `NULL`.

inFirstAttributeValue
: On input, a value of type `tDataNodePtr` (page 88) that points to a `tDataNode` (page 88) structure allocated by calling `dsDataNodeAllocateBlock` (page 31) or `dsDataNodeAllocateString` (page 32) that contains the value of the attribute that is to be added. If you don't want to set a value, this parameter can be `NULL`.

function result
: A value of type `tDirStatus` indicating success (`eDSNoErr`) or an error. For a list of possible result codes, see "Result Codes" (page 169).

Discussion

This function adds to the specified record an attribute having the name specified by the `inNewAttribute` parameter and the value pointed to by the `inFirstAttributeValue` parameter.

To change the value of an attribute, call `dsSetAttributeValue` (page 74) or `dsSetAttributeValues` (page 75).

Availability

Available in Mac OS X v10.0 and later.

Declared In

`DirServices.h`

dsAddAttributeValue

Adds a value to an attribute.

```
tDirStatus dsAddAttributeValue (
   tRecordReference inRecordReference,
   tDataNodePtr inAttributeType,
   tDataNodePtr inAttributeValue
);
```

Parameters

inRecordReference
: On input, value of type tRecordReference (page 89) obtained by previously calling `dsOpenRecord` (page 71) that represents the record having an attribute to which a value is to be appended.

inAttributeType
: On input, a value of type `tDataNodePtr` (page 88) that points to a `tDataNode` (page 88) structure allocated by calling `dsDataNodeAllocateBlock` (page 31) or `dsDataNodeAllocateString` (page 32) that contains the type of the attribute to which a value is to be added.

inAttributeValue
: On input, a value of type `tDataNodePtr` (page 88) that points to a `tDataNode` (page 88) structure allocated by calling `dsDataNodeAllocateBlock` (page 31) or `dsDataNodeAllocateString` (page 32) that contains the value that is to be added.

function result

A value of type `tDirStatus` indicating success (`eDSNoErr`) or an error. For a list of possible result codes, see "Result Codes" (page 169).

Discussion

This function adds the specified value to the specified attribute. The attribute must be capable of having more than one value.

To change the value of an attribute, call `dsSetAttributeValue` (page 74) or `dsSetAttributeValues` (page 75).

Availability

Available in Mac OS X v10.0 and later.

Declared In

`DirServices.h`

dsAddChildPIDToReference

Allows the specified process to use a node reference.

```
tDirStatus dsAddChildPIDToReference (
   tDirReference inDirRef,
   SInt32 inValidChildPID,
   UInt32 inValidAPIReferenceToGrantChild
);
```

Parameters

inDirRef

A value of type `tDirReference` (page 89) obtained by previously calling `dsOpenDirService` (page 69) that identifies the Open Directory session.

inValidChildPID

A value of type `long` that specifies the child process ID that is to be granted permission to use the Open Directory reference specified by `inDirReference`.

inValidAPIReferenceToGrantChild

A value of type `unsigned long` containing a node reference obtained by previously calling `dsOpenDirNode` (page 68).

function result

A value of type `tDirStatus` indicating success (`eDSNoErr`) or an error. For a list of possible result codes, see "Result Codes" (page 169).

Discussion

This function allows the child process specified by `inValidChildPID` to use the node reference specified by the `inValidAPIReferenceToGrantChild` parameter. Calling this function allows a child process that your application forks to use a node reference that the parent process has already acquired.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`DirServices.h`

dsAllocAttributeValueEntry

Allocates an attribute value entry structure having the specified attribute value.

```
tAttributeValueEntryPtr dsAllocAttributeValueEntry (
   tDirReference inDirRef,
   UInt32 inAttrValueID,
   void *inAttrValueData,
   UInt32 inAttrValueDataLen
);
```

Parameters

inDirRef

On input, a value of type `tDirReference` (page 89) obtained by calling `dsOpenDirService` (page 69) representing the Open Directory session that is to be associated with the attribute value entry structure, or zero.

inAttrValueID

On input, a value of type `unsigned long` containing an attribute value ID.

inAttrValueData

On input, a pointer an arbitrary value containing the value that is to be incorporated in the attribute value entry structure as an attribute value.

inAttrValueDataLen

On input, the length of valid data in the value pointed to by `inAttrValueData`.

function result

A value of type `tAttributeEntryPtr` (page 85) that points to the new `tAttributeValueEntry` (page 83) structure.

Discussion

This utility function allocates a structure of type `tAttributeValueEntry` (page 83) and returns a pointer to it. The resulting structure can be used to set the value of the attribute identified by `inAttrValueID` by calling `dsSetAttributeValue` (page 74) and passing to it the attribute value entry pointer returned by this function.

The allocated structure contains the attribute value ID specified by `inAttrValueID` and the attribute value pointed to by `inAttrValueData`.

To release the memory associated with `tAttributeValueEntryPtr`, call `dsDeallocAttributeValueEntry` (page 35).

Availability

Available in Mac OS X v10.0 and later.

Declared In

`DirServicesUtils.h`

dsAppendStringToListAlloc

Appends a string to a data list.

```
tDirStatus dsAppendStringToListAlloc (
   tDirReference inDirReferences,
   tDataListPtr inDataList,
   const char *inCString
);
```

Parameters

`inDirReference`

On input, a value of type `tDirReference` (page 89) obtained by calling `dsOpenDirService` (page 69) or `dsOpenDirServiceProxy` (page 70) representing the Open Directory session that is associated with this data list, or zero.

`inDataList`

On input, a value of type `tDataListPtr` (page 87) that points to the data list to which the string specified by `inCString` is to be appended.

`inCString`

On input, a pointer to a null-terminated string containing the value in UTF-8 format that is to be appended to the data list.

`function result`

A value of type `tDirStatus` indicating success (`eDSNoErr`) or an error. For a list of possible result codes, see "Result Codes" (page 169).

Discussion

This utility function appends a string to a data list.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`DirServicesUtils.h`

dsBuildFromPath

Builds a data list from a pathname.

```
tDataListPtr dsBuildFromPath (
   tDirReference inDirReference,
   const char *inPathCString,
   const char *inPathSeparatorCString
);
```

Parameters

`inDirReference`

On input, a value of type `tDirReference` (page 89) obtained by calling `dsOpenDirService` (page 69) or `dsOpenDirServiceProxy` (page 70) that represents the Open Directory session for which the data list is to be built, or zero.

`inPathCString`

On input, a pointer to a null-terminated string containing a pathname in UTF-8 format.

`inPathSeparatorCString`

On input, a pointer to a null-terminated string containing the character that delimits the components of the pathname pointed to by `inPathCString`.

function result

A value of type `tDataListPtr` (page 87) that points to the new data list.

Discussion

This utility function uses a pathname to build a null-terminated data list and returns a pointer to it. Many Open Directory functions take a pointer to a data list as a parameter. For example, you can pass the resulting data list pointer as a parameter to `dsOpenDirNode` (page 68).

When you no longer need the data list, call `dsDataListDeallocate` (page 27) to release the memory associated with it. If the data list is a heap-based data list, you also need to call `free()`.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

NetworkAuthentication

Declared In

`DirServicesUtils.h`

dsBuildListFromNodesAlloc

Fills in a previously allocated data list using one or more data nodes.

```
tDirStatus dsBuildListFromNodesAlloc (
   tDirReference inDirReferences,
   tDataListPtr inDataList,
   tDataNodePtr in1stDataNodePtr,
   ...
);
```

Parameters

inDirReference

On input, a value of type `tDirReference` (page 89) obtained by calling `dsOpenDirService` (page 69) or `dsOpenDirServiceProxy` (page 70) representing the Open Directory session that is associated with the specified data list, or zero.

inDataList

On input, a value of type `tDataListPtr` that points to a data list allocated by calling `dsDataListAllocate` (page 26).

in1stDataNodePtr

On input, a value of type `tDataNodePtr` (page 88) that points to a data node containing data in UTF-8 format. The `in1stDataNodePtr` parameter may be followed by one or more parameters of type `tDataNodePtr`, each pointing to a data node. Each data node may have been allocated by calling `dsDataNodeAllocateBlock` (page 31) or `dsDataNodeAllocateString` (page 32).

function result

A value of type `tDirStatus` indicating success (`eDSNoErr`) or an error. For a list of possible result codes, see "Result Codes" (page 169).

Discussion

This utility function uses information from one or more data nodes to fill in a previously allocated data list. The resulting data list is null-terminated.

When you no longer need the data list, call `dsDataListDeallocate` (page 27) to release the memory associated with it. If the data list is a heap-based data list, you also need to call `free()`.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`DirServicesUtils.h`

dsBuildListFromPathAlloc

Builds a data list from a pathname using a data list that has already been allocated.

```
tDirStatus dsBuildListFromPathAlloc (
   tDirReference inDirReference,
   tDataListPtr inDataList,
   const char *inPathCString,
   const char *inPathSeparatorCString
);
```

Parameters

`inDirReference`
: On input, a value of type `tDirReference` (page 89) obtained by calling `dsOpenDirService` (page 69) or `dsOpenDirServiceProxy` (page 70) that represents the Open Directory session for which the data list is to be built, or zero.

`inDataList`
: On input, a value of type `tDataListPtr` that points to a data list allocated by calling `dsDataListAllocate` (page 26).

`inPathCString`
: On input, a pointer to a null-terminated string containing a pathname in UTF-8 format.

`inPathSeparatorCString`
: On input, a pointer to a null-terminated string containing the character that delimits the components of the pathname pointed to by inPathCString.

`function result`
: A value of type `tDirStatus` indicating success (`eDSNoErr`) or an error. For a list of possible result codes, see "Result Codes" (page 169).

Discussion
This utility function uses previously allocated data list and a pathname to build a null-terminated data list and returns a pointer to it. Many Open Directory functions take a pointer to a data list as a parameter. For example, you can pass the resulting data list pointer as a parameter to `dsOpenDirNode` (page 68).

When you no longer need the data list, call `dsDataListDeallocate` (page 27) to release the memory associated with it. If the data list is a heap-based data list, you also need to call `free()`.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`DirServicesUtils.h`

dsBuildListFromStrings

Builds a data list from strings.

```
tDataListPtr dsBuildListFromStrings (
   tDirReference inDirReference,
   const char *in1stCString,
   ...
);
```

Parameters

inDirReference

On input, a value of type `tDirReference` (page 89) obtained by calling `dsOpenDirService` (page 69) or `dsOpenDirServiceProxy` (page 70) representing the Open Directory session for which the data list is being built, or zero.

in1stCString

On input, a pointer to a null-terminated string containing data in UTF-8 format that is to be added to the data list. The in1stCString parameter may be followed by one or more parameters of type `char *`, each pointing to a C string containing data in UTF-8 format that is to be added to the data list.

function result

A value of type `tDataListPtr` that points to the `tDataList` (page 84) structure that has been created.

Discussion

This utility function uses one or more null-terminated strings to build a data list and returns a pointer to it.

When you no longer need the data list, call `dsDataListDeallocate` (page 27) t o release the memory associated with it. If the data list is a heap-based data list, you also need to call `free()`.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

NetworkAuthentication

Declared In

`DirServicesUtils.h`

dsBuildListFromStringsAlloc

Fills in a previously allocated data list using data from strings.

```
tDirStatus dsBuildListFromStringsAlloc (
   tDirReference inDirReferences,
   tDataListPtr inDataList,
   const char *in1stCString,
   ...
);
```

Parameters

inDirReference

On input, a value of type `tDirReference` (page 89) obtained by calling `dsOpenDirService` (page 69) or `dsOpenDirServiceProxy` (page 70) representing the Open Directory session that is associated with the specified data list, or zero.

inDataList
: On input, a value of type `tDataListPtr` that points to a data list allocated by calling `dsDataListAllocate` (page 26).

in1stCString
: On input, a pointer to a character string that specifies the name of a data node to add to the data list. The `in1stCString` parameter may be followed by one or more additional parameters of type `char *`, each pointing to a C string containing data that is to be added to the data list.

function result
: A value of type `tDirStatus` indicating success (`eDSNoErr`) or an error. For a list of possible result codes, see "Result Codes" (page 169).

Discussion
This utility function fills in a data list using the data in UTF-8 format contained by the specified null-terminated strings.

When you no longer need the data list, call `dsDataListDeallocate` (page 27) to release the memory associated with it. If the data list is a heap-based data list, you also need to call `free()`.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`DirServicesUtils.h`

dsBuildListFromStringsAllocV

Fills in a previously allocated data list using data from a list of type `va_list`.

```
tDirStatus dsBuildListFromStringsAllocV (
   tDirReference inDirRef,
   tDataList *inDataList,
   const char *in1stCString,
   va_list args
);
```

Parameters

inDirRef
: On input, a value of type `tDirReference` (page 89) obtained by calling `dsOpenDirService` (page 69) representing the Open Directory session that is associated with the specified data list, or zero.

inDataList
: On input, a pointer to a value of type `tDataList` (page 84) allocated by calling `dsDataListAllocate` (page 26).

in1stCString
: On input, a pointer to a character string that specifies the name of a data node to add to the data list. The `in1stCString` parameter may be followed by one or more additional parameters of type `char *`, each pointing to a C string containing data that is to be added to the data list.

args
: On input, a value of type `va_list` with additional C strings containing data that is to be added to the date list.

function result

A value of type `tDirStatus` indicating success (`eDSNoErr`) or an error. For a list of possible result codes, see "Result Codes" (page 169).

Discussion

This utility function fills in a data list using the data in UTF-8 format contained by the specified null-terminated string and additional strings in the `va_list` parameter.

When you no longer need the data list, call `dsDataListDeallocate` (page 27) to release the memory associated with it.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`DirServicesUtils.h`

dsCloseAttributeList

Disposes of an attribute list reference.

```
tDirStatus dsCloseAttributeList (
   tAttributeListRef inAttributeListRef
);
```

Parameters

inAttributeListRef

On input, a value of type `tAttributeListRef` (page 86) obtained by a previous call to `dsGetDirNodeInfo` (page 56) or `dsGetRecordEntry` (page 63).

function result

A value of type `tDirStatus` indicating success (`eDSNoErr`) or an error. For a list of possible result codes, see "Result Codes" (page 169).

Discussion

This function disposes of an attribute list reference that was obtained by a previous call to `dsGetDirNodeInfo` (page 56) or `dsGetRecordList` (page 64). You should dispose of an attribute list reference when it is no longer needed.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

NetworkAuthentication

Declared In

`DirServices.h`

dsCloseAttributeValueList

Disposes of an attribute value list reference.

```
tDirStatus dsCloseAttributeValueList (
   tAttributeValueListRef inAttributeValueListRef
);
```

Parameters

`inAttributeValueListRef`

On input, a value of type `tAttributeValueListRef` (page 86) that was obtained by a previous call to `dsGetAttributeEntry` (page 51).

`function result`

A value of type `tDirStatus` indicating success (`eDSNoErr`) or an error. For a list of possible result codes, see "Result Codes" (page 169).

Discussion

This function disposes of an attribute value list reference that was obtained by a previous call to `dsGetAttributeEntry` (page 51). You should dispose of an attribute value list reference when it is no longer needed.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

NetworkAuthentication

Declared In

`DirServices.h`

dsCloseDirNode

Closes a session with a node.

```
tDirStatus dsCloseDirNode (
   tDirNodeReference inDirNodeReference
);
```

Parameters

`inDirNodeReference`

On input, a value of type `tDirNodeReference` (page 89) obtained by previously calling `dsOpenDirNode` (page 68) or `dsOpenDirServiceProxy` (page 70) that identifies the node session that is to be closed.

`function result`

A value of type `tDirStatus` indicating success (`eDSNoErr`) or an error. such as `eDSInvalidReference` if the `tDirNodeReference` is invalid. For a list of other possible result codes, see "Result Codes" (page 169).

Discussion

This function closes a session with the node represented by `inDirNodeReference`.

When the session with the node is closed, `inDirNodeReference` becomes invalid and cannot be used with any other Open Directory function that takes a node reference as a parameter. Any references that were created with `inDirNodeReference` as a parameter, such as record references, attribute list references, and attribute value references become invalid when the session represented by `inDirNodeReference` is closed.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code
NetworkAuthentication

Declared In
`DirServices.h`

dsCloseDirService

Closes an Open Directory session.

```
tDirStatus dsCloseDirService (
   tDirReference inDirReference
);
```

Parameters

`inDirReference`
A value of type `tDirReference` (page 89) obtained by previously calling `dsOpenDirService` (page 69) or `dsOpenDirServiceProxy` (page 70) that identifies the Open Directory session that is to be closed.

`function result`
A value of type `tDirStatus` indicating success (`eDSNoErr`) or an error. For a list of possible result codes, see "Result Codes" (page 169).

Discussion
This function closes the Open Directory session represented by `inDirReference`. Continuation data and child references, such as node, record, attribute list, and attribute value list references, that were created using `inDirReference` become invalid when the session is closed and are released implicitly when this function is called. You must deallocate data lists, data nodes, and data buffers yourself by calling `dsDataListDeallocate` (page 27), `dsDataNodeDeAllocate` (page 33), and `dsDataBufferDeAllocate` (page 25) respectively.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
NetworkAuthentication

Declared In
`DirServices.h`

dsCloseRecord

Closes a open record.

```
tDirStatus dsCloseRecord (
   tRecordReference inRecordReference
);
```

Parameters

`inRecordReference`
On input, value of type tRecordReference (page 89) obtained by previously calling `dsOpenRecord` (page 71) that identifies the record that is to be closed.

function result

A value of type `tDirStatus` indicating success (`eDSNoErr`) or an error. For a list of possible result codes, see "Result Codes" (page 169).

Discussion

This function closes a record that was previously opened by calling `dsOpenRecord` (page 71). Closing the record invalidates the `inRecordReference` parameter so that it cannot be used as a parameter to any other Open Directory function. Any pending changes to the record are flushed at this time.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`DirServices.h`

dsCreateRecord

Creates a record.

```
tDirStatus dsCreateRecord (
   tDirNodeReference inDirNodeReference,
   tDataNodePtr inRecordType,
   tDataNodePtr inRecordName
);
```

Parameters

inDirNodeReference

On input, a value of type `tDirNodeReference` (page 89), obtained by previously calling `dsOpenDirNode` (page 68) that identifies the node in which the record is to be created.

inRecordType

On input, a value of type `tDataNodePtr` (page 88) that points to a `tDataNode` (page 88) structure allocated by calling `dsDataNodeAllocateBlock` (page 31) or `dsDataNodeAllocateString` (page 32) that contains the record type for the record that is to be created. For record type constants, see Standard Record Types (page 143).

inRecordName

On input, a value of type `tDataNodePtr` (page 88) that points to a `tDataNode` (page 88) structure allocated by calling `dsDataNodeAllocateBlock` (page 31) or `dsDataNodeAllocateString` (page 32) that contains the name in UTF-8 format for the record that is to be created.

function result

A value of type `tDirStatus` indicating success (`eDSNoErr`) or an error. For a list of possible result codes, see "Result Codes" (page 169).

Discussion

This function creates in the node represented by `inDirNodeReference` a record having the name and type specified by the data nodes pointed to by the `inRecordType` and `inRecordName` parameters.

To add attributes to the new record, call `dsAddAttribute` (page 11).

This function does not open the created record. To create a record and open it in one step, call `dsCreateRecordAndOpen` (page 24).

Availability

Available in Mac OS X v10.0 and later.

Declared In
`DirServices.h`

dsCreateRecordAndOpen

Creates a record and opens it.

```
tDirStatus dsCreateRecordAndOpen (
   tDirNodeReference inDirNodeReference,
   tDataNodePtr inRecordType,
   tDataNodePtr inRecordName,
   tRecordReference *outRecordReference
);
```

Parameters

inDirNodeReference
: On input, a value of type `tDirReference` (page 89), obtained by calling `dsOpenDirNode` (page 68) that identifies the node in which the record is to be created.

inRecordType
: On input, a value of type `tDataNodePtr` (page 88) that points to a `tDataNode` (page 88) structure allocated by calling `dsDataNodeAllocateBlock` (page 31) or `dsDataNodeAllocateString` (page 32) that contains the record type for the record that is to be created. For record type constants, see Standard Record Types (page 143).

inRecordName
: On input, a value of type `tDataNodePtr` (page 88) that points to a `tDataNode` (page 88) structure allocated by calling `dsDataNodeAllocateBlock` (page 31) or `dsDataNodeAllocateString` (page 32) that contains the name in UTF-8 format for the record that is to be created.

outRecordReference
: On input, a pointer to a value of type tRecordReference (page 89). On output, `outRecordReference` points to a record reference for the created record and that can be provided as a parameter to Open Directory functions that operate on opened records.

function result
: A value of type `tDirStatus` indicating success (`eDSNoErr`) or an error. For a list of possible result codes, see "Result Codes" (page 169).

Discussion
This function creates a record and opens it. On output the `outRecordReference` parameter is a reference to the newly created record that can be passed as a parameter to Open Directory functions that operate on open records.

To add attributes to the new record, call `dsAddAttribute` (page 11).

To create a record without opening it, call `dsCreateRecord` (page 23).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`DirServices.h`

dsDataBufferAllocate

Allocates an Open Directory data buffer.

```
tDataBufferPtr dsDataBufferAllocate (
   tDirReference inDirReference,
   UInt32 inBufferSize
);
```

Parameters

inDirReference

On input, a value of type `tDirReference` (page 89) obtained by calling `dsOpenDirService` (page 69) or `dsOpenDirServiceProxy` (page 70), or zero.

inBufferSize

On input, a value of type `unsigned long` that specifies the length of the buffer that is to be allocated.

function result

A value of type `tDataBufferPtr` (page 87) that points to the allocated `tDataBuffer` (page 84) structure.

Discussion

The utility function allocates an Open Directory data buffer of the specified size and returns a value that points to the allocated buffer.

Open Directory data buffers are used by many Open Directory functions to exchange information between an Open Directory client application and an Open Directory plug-in.

When you no longer need the data buffer, call `dsDataBufferDeAllocate` (page 25) to deallocate the memory that is associated with it.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

NetworkAuthentication

Declared In

`DirServicesUtils.h`

dsDataBufferDeAllocate

Deallocates an Open Directory data buffer.

```
tDirStatus dsDataBufferDeAllocate (
   tDirReference inDirReference,
   tDataBufferPtr inDataBufferPtr
);
```

Parameters

inDirReference

On input, a value of type `tDirReference` (page 89) obtained by calling `dsOpenDirService` (page 69) or `dsOpenDirServiceProxy` (page 70) for which a data buffer is to be deallocated, or zero.

inDataBufferPtr

A value of type `tDataBufferPtr` (page 87) that points to the `tDataBuffer` (page 84) structure that is to be deallocated.

function result

A value of type `tDirStatus` indicating success (`eDSNoErr`) or an error. For a list of possible result codes, see "Result Codes" (page 169).

Discussion

This utility function appends the specified string to the specified data list.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

NetworkAuthentication

Declared In

`DirServicesUtils.h`

dsDataListAllocate

Allocates a data list.

```
tDataListPtr dsDataListAllocate (
   tDirReference inDirReference
);
```

Parameters

inDirReference

On input, a value of type `tDirReference` (page 89) obtained by calling `dsOpenDirService` (page 69) or `dsOpenDirServiceProxy` (page 70) representing the Open Directory session for which the data list is to be allocated, or zero.

function result

A value of type `tDataListPtr` that points to the allocated `tDataList` (page 84) structure. If this function cannot allocate the data list, it returns `NULL`.

Discussion

This utility function allocates an empty data list and returns a value of type `tDataListPtr` that points to it.

Many Open Directory functions return information in a data list and receive information in a data list, such as `dsFindDirNodes` (page 49), `dsGetDirNodeInfo` (page 56), `dsGetRecordList` (page 64), and `dsDoAttributeValueSearch` (page 37).

To add data to the data list, call `dsBuildListFromNodesAlloc` (page 16) or `dsBuildListFromStringsAlloc` (page 18).

When you no longer need the data list, call `dsDataListDeallocate` (page 27) to release the memory associated with it.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`DirServicesUtils.h`

dsDataListCopyList

Copies a data list.

```
tDataListPtr dsDataListCopyList (
   tDirReference inDirReference,
   const tDataList *inDataListSource
);
```

Parameters

inDirReference

On input, a value of type `tDirReference` (page 89) obtained by calling `dsOpenDirService` (page 69) or `dsOpenDirServiceProxy` (page 70) representing the Open Directory session that is associated with the specified data list, or zero.

inDataListSource

On input, a pointer to a `tDataList` (page 84) structure for the data list that is to be copied.

function result

A value of type `tDataListPtr` that points to the copy of the data list. If this function cannot copy the list, it returns `NULL`.

Discussion

This utility function copies a data list and returns a pointer to the copy of the data list.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`DirServicesUtils.h`

dsDataListDeallocate

Deallocates a data list.

```
tDirStatus dsDataListDeallocate (
   tDirReference inDirReference,
   tDataListPtr inDataList
);
```

Parameters

inDirReference

On input, a value of type `tDirReference` (page 89) obtained by calling `dsOpenDirService` (page 69) or `dsOpenDirServiceProxy` (page 70) representing the Open Directory session for which the data list is to be deallocated, or zero.

inDataList

On input, a value of type `tDataListPtr` pointing to the `tDataList` (page 84) structure that is to be deallocated.

function result

A value of type `tDirStatus` indicating success (`eDSNoErr`) or an error. For a list of possible result codes, see "Result Codes" (page 169).

Discussion

This utility function deallocates a data list previously created by calling `dsBuildListFromNodesAlloc` (page 16), `dsBuildFromPath` (page 15), `dsBuildListFromStrings` (page 18), or `dsDataListCopyList` (page 27).

This utility function does not clean up the header structure associated with the `inDataList` parameter, so if the `inDataList` parameter is a true pointer and not the address of a stack variable, you need to call `free(inDataList)`.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
NetworkAuthentication

Declared In
`DirServicesUtils.h`

dsDataListDeleteThisNode

Deletes a data node from a data list.

```
tDirStatus dsDataListDeleteThisNode (
   tDirReference inDirReference,
   tDataListPtr inDataList,
   UInt32 inNodeIndex
);
```

Parameters

`inDirReference`
: On input, a value of type `tDirReference` (page 89) obtained by calling `dsOpenDirService` (page 69) or `dsOpenDirServiceProxy` (page 70) representing the Open Directory session that is associated with the specified data list, or zero.

`inDataList`
: On input, a value of type `tDataListPtr` pointing to the data list from which a data node is to be removed.

`inNodeIndex`
: On input, a value of type `unsigned long` that identifies the data node to remove. Set `inNodeIndex` to 1 to remove the first node. Set `inNodeIndex` to 2 to remove the second node, and so on.

`function result`
: A value of type `tDirStatus` indicating success (`eDSNoErr`) or an error. For a list of possible result codes, see "Result Codes" (page 169).

Discussion
This utility function removes a data node from a data list. The `inNodeIndex` parameter specifies the index of the data node that is to be removed.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`DirServicesUtils.h`

dsDataListGetNodeAlloc

Gets a data node from a data list.

```
tDirStatus dsDataListGetNodeAlloc (
   tDirReference inDirReference,
   const tDataList *inDataListPtr,
   UInt32 inNodeIndex,
   tDataNodePtr *outDataNode
);
```

Parameters

inDirReference
: On input, a value of type `tDirReference` (page 89) obtained by calling `dsOpenDirService` (page 69) or `dsOpenDirServiceProxy` (page 70) representing the Open Directory session that is associated with the specified data list, or zero.

inDataListPtr
: On input, a pointer to a `tDataList` (page 84) structure of the data list from which a data node is to be obtained.

inNodeIndex
: On input, a value of type `unsigned long` that identifies the data node to obtain. Set `inNodeIndex` to 1 to get the first node. Set `inNodeIndex` to 2 to get the second node, and so on.

outDataNode
: On output, a value of type `tDataNodePtr` (page 88) that points to the data node obtained from the data list.

function result
: A value of type `tDirStatus` indicating success (`eDSNoErr`) or an error. For a list of possible result codes, see "Result Codes" (page 169).

Discussion

This utility function obtains a data node from a data list.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`DirServicesUtils.h`

dsDataListGetNodeCount

Gets the number of data nodes in a data list.

```
UInt32 dsDataListGetNodeCount (
   const tDataList *inDataList
);
```

Parameters

inDataListPtr
: On input, a pointer to a value of type `tDataList` (page 84) containing the data nodes that are to be counted.

function result
: The number of data nodes in the data list or an error code. For a list of possible result codes, see "Result Codes" (page 169).

Discussion

This utility function returns the number of data nodes in a data list.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`DirServicesUtils.h`

dsDataListInsertAfter

Inserts a data node in a data list.

```
tDirStatus dsDataListInsertAfter (
   tDirReference inDirReferences,
   tDataListPtr inDataList,
   tDataNodePtr inInsertDataNode,
   const UInt32 inNodeIndex
);
```

Parameters

inDirReference
: On input, a value of type `tDirReference` (page 89) obtained by calling `dsOpenDirService` (page 69) or `dsOpenDirServiceProxy` (page 70) representing the Open Directory session that is associated with the specified data list, or zero.

inDataList
: On input, a value of type `tDataListPtr` pointing to a data list containing a list of nodes.

inInsertDataNode
: On input, a value of type `tDataNodePtr` (page 88) pointing to a data node.

inNodeIndex
: On input, a value of type `const unsigned long` that specifies the data node in the list after which the data node specified by `inInsertDataNode` is to be inserted. If `inNodeIndex` is zero, the data node is inserted at the beginning of the data list.

function result
: A value of type `tDirStatus` indicating success (`eDSNoErr`) or an error. For a list of possible result codes, see "Result Codes" (page 169).

Discussion
This utility function inserts a node into a list of nodes in a data list.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`DirServicesUtils.h`

dsDataListMergeListAfter

Merges two data lists.

```
tDirStatus dsDataListMergeListAfter (
   tDataListPtr inTargetList,
   tDataListPtr inSourceList,
   const UInt32 inNodeIndex
);
```

Parameters

inTargetList
: On input, a value of type `tDataListPtr` pointing to a data list containing data nodes. When this function returns, `inTargetList` contains the data nodes it contained before this function was called as well as the data nodes contained by the data list pointed to by `inSourceList`.

inSourceList
: On input, a value of type `tDataListPtr` pointing to a data list containing data nodes that are to be merged with the data nodes in the data list specified by `inTargetList`.

inNodeIndex
: On input, a value of type `const unsigned long` that specifies the index of the node in the data list pointed to by `inTargetList` after which the data nodes in the list pointed to by `inSourceList` are to be inserted. If `inNodeIndex` is zero, the data nodes are inserted at the beginning of the list.

function result
: A value of type `tDirStatus` indicating success (`eDSNoErr`) or an error. For a list of possible result codes, see "Result Codes" (page 169).

Discussion

This utility function merges two data lists. The data nodes in the data list pointed by the `inSourceList` parameter are merged with the data nodes in the data list pointed to by the `inTargetList` parameter after the data node indicated by the `inNodeIndex` parameter.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`DirServicesUtils.h`

dsDataNodeAllocateBlock

Allocates an Open Directory data node.

```
tDataNodePtr dsDataNodeAllocateBlock (
   tDirReference inDirReference,
   UInt32 inDataNodeSize,
   UInt32 inDataNodeLength,
   tBuffer inDataNodeBuffer
);
```

Parameters

inDirReference
: On input, a value of type `tDirReference` (page 89) obtained by calling `dsOpenDirService` (page 69) or `dsOpenDirServiceProxy` (page 70) representing the Open Directory session for which the data node is to be allocated, or zero.

inDataNodeSize
: On input, a value of type `unsigned long` that specifies the size of `inDataNodeBuffer`.

inDataNodeLength
: On input, a value of type `unsigned long` that specifies the length of valid data in `inDataNodeBuffer`.

inDataNodeBuffer
: On input, a value of type `tBuffer` (page 87) containing the value the data node is to contain.

function result
: A value of type `tDataNodePtr` (page 88) that points to the allocated data node and that can be passed as a parameter to Open Directory functions that require such a value as a parameter. If this function cannot allocate the data node, it returns `NULL`.

Discussion

This utility function allocates an Open Directory data node and returns a pointer to it. Use the data node as a convenient way to pass data, such as record names and authentication types, to Open Directory functions.

To release the memory associated with a data node, call `dsDataNodeDeAllocate` (page 33).

To use a C string to allocate a data node, call `dsDataNodeAllocateString` (page 32).

Availability

Available in Mac OS X v10.0 and later.

Declared In

`DirServicesUtils.h`

dsDataNodeAllocateString

Allocates an Open Directory data node using a string.

```
tDataNodePtr dsDataNodeAllocateString (
   tDirReference inDirReference,
   const char *inCString
);
```

Parameters

inDirReference
: On input, a value of type `tDirReference` (page 89) obtained by calling `dsOpenDirService` (page 69) or `dsOpenDirServiceProxy` (page 70) representing the Open Directory session for which the data node is to be allocated. The value of this parameter is actually ignored in Mac OS X.

inCString
: A pointer to a value of type `char` that specifies the value the data node is to contain.

function result
: A value of type `tDataNodePtr` (page 88) that points to the allocated data node and that can be passed as a parameter to Open Directory functions that require such a value as a parameter. If this function cannot allocate the data node, it returns `NULL`.

Discussion

This utility function uses a C string to allocate an Open Directory data node and returns a pointer to the allocated data node. Use the data node as a convenient way to pass data, such as record names and authentication types, to Open Directory functions.

To release the memory associated with a data node, call `dsDataNodeDeAllocate` (page 33).

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code
NetworkAuthentication

Declared In
`DirServicesUtils.h`

dsDataNodeDeAllocate

Deallocates a data node.

```
tDirStatus dsDataNodeDeAllocate (
   tDirReference inDirReference,
   tDataNodePtr inDataNodePtr
);
```

Parameters

`inDirReference`
: On input, a value of type `tDirReference` (page 89) obtained by calling `dsOpenDirService` (page 69) or `dsOpenDirServiceProxy` (page 70), or zero.

`inDataNodePtr`
: On input, a value of type `tDataNodePtr` (page 88) that points to the `tDataBuffer` (page 84) structure that is to be deallocated.

`function result`
: A value of type `tDirStatus` indicating success (`eDSNoErr`) or an error. For a list of possible result codes, see "Result Codes" (page 169).

Discussion
This utility function deallocates an Open Directory data node that was created by previously calling `dsDataNodeAllocateBlock` (page 31) or `dsDataNodeAllocateString` (page 32).

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
NetworkAuthentication

Declared In
`DirServicesUtils.h`

dsDataNodeGetLength

Gets the length of valid data in a data node's buffer.

```
UInt32 dsDataNodeGetLength (
   tDataNodePtr inDataNodePtr
);
```

Parameters

`inDataNodePtr`
: On input, a value of type `tDataNodePtr` (page 88) that points to the data node for which the length of valid data in the data node's buffer is to be obtained.

function result

A value of type `unsigned long` that contains the length of valid data in the data node's buffer. If this function cannot obtain the length, it returns zero.

Discussion

This utility function gets the length of valid data in the buffer of the data node pointed to by `inDataNodePtr`.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`DirServicesUtils.h`

dsDataNodeGetSize

Gets the size of a data node's buffer.

```
UInt32 dsDataNodeGetSize (
   tDataNodePtr inDataNodePtr
);
```

Parameters

inDataNodePtr

On input, a value of type `tDataNodePtr` (page 88) that points to the `tDataBuffer` (page 84) structure whose buffer size is to be obtained.

function result

A value of type `unsigned long` that contains the size of the buffer. If this function cannot obtain the buffer's size, it returns zero.

Discussion

This utility function obtains the size of a data node's buffer.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`DirServicesUtils.h`

dsDataNodeSetLength

Sets the length of valid data in a data node's buffer.

```
tDirStatus dsDataNodeSetLength (
   tDataNodePtr inDataNodePtr,
   UInt32 inDataNodeLength
);
```

Parameters

inDataNodePtr

On input, a value of type `tDataNodePtr` (page 88) that points to the data node whose buffer size is to be set.

inDataNodeLength

On input, a value of type `unsigned long` that specifies the length of valid data in the buffer.

function result

A value of type `tDirStatus` indicating success (`eDSNoErr`) or an error. For a list of possible result codes, see "Result Codes" (page 169).

Discussion

This utility function sets the length of valid data in the buffer of the data node pointed to by `inDataNodePtr`.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`DirServicesUtils.h`

dsDeallocAttributeEntry

Deallocates an attribute entry structure.

```
tDirStatus dsDeallocAttributeEntry (
   tDirReference inDirRef,
   tAttributeEntryPtr inAttrEntry
);
```

Parameters

inDirRef

On input, a value of type `tDirReference` (page 89) obtained by calling `dsOpenDirService` (page 69) representing the Open Directory session associated with the attribute entry structure that is to be deallocated, or zero.

inAttrEntry

On input, a value of type `tAttributeEntryPtr` (page 85) that points to the `tAttributeValueEntry` (page 83) structure that is to be deallocated.

function result

A value of type `tDirStatus` indicating success or failure.

Discussion

This utility function deallocates an attribute entry structure and the pointer to it that were allocated in order to call `dsGetAttributeEntry` (page 51) or `dsGetRecordAttributeInfo` (page 59).

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

NetworkAuthentication

Declared In

`DirServicesUtils.h`

dsDeallocAttributeValueEntry

Deallocates an attribute value entry structure.

```
tDirStatus dsDeallocAttributeValueEntry (
   tDirReference inDirRef,
   tAttributeValueEntryPtr inAttrValueEntry
);
```

Parameters

inDirRef

On input, a value of type `tDirReference` (page 89) obtained by calling `dsOpenDirService` (page 69) representing the Open Directory session associated with the attribute value entry structure that is to be deallocated, or zero.

inAttrValueEntry

On input, a value of type `tAttributeValueEntryPtr` that points to the `tAttributeValueEntry` (page 83) structure that is to be deallocated.

function result

A value of type `tDirStatus` indicating success or failure.

Discussion

This utility function deallocates an attribute value entry structure that was previously allocated by calling `dsGetAttributeValue` (page 53), `dsGetRecordAttributeValueByID` (page 60), `dsGetRecordAttributeValueByIndex` (page 61), or `dsGetRecordAttributeValueByValue` (page 62).

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

NetworkAuthentication

Declared In

`DirServicesUtils.h`

dsDeallocRecordEntry

Deallocates a record entry structure.

```
tDirStatus dsDeallocRecordEntry (
   tDirReference inDirRef,
   tRecordEntryPtr inRecEntry
);
```

Parameters

inDirRef

A value of type `tDirReference` (page 89) obtained by previously calling `dsOpenDirService` (page 69) that identifies the Open Directory session for the record entry structure that is to be deallocated, or zero.

inRecEntry

On input, a value of type `tRecordEntryPtr` (page 89) that points to the `tRecordEntry` (page 85) structure that is to be deallocated.

function result

A value of type `tDirStatus` indicating success (`eDSNoErr`) or an error. For a list of possible result codes, see "Result Codes" (page 169).

Discussion
This utility function deallocates the specified record entry structure that was allocated by a previous call to `dsGetRecordEntry` (page 63) or `dsGetRecordReferenceInfo` (page 66).

You should always deallocate record entry structures when you no longer need them.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
NetworkAuthentication

Declared In
`DirServicesUtils.h`

dsDeleteRecord

Deletes a record.

```
tDirStatus dsDeleteRecord (
   tRecordReference inRecordReference
);
```

Parameters

`inRecordReference`
On input, a value of type tRecordReference (page 89) obtained by previously calling `dsOpenRecord` (page 71) or `dsCreateRecordAndOpen` (page 24)that represents the record that is to be deleted.

`function result`
A value of type `tDirStatus` indicating success (`eDSNoErr`) or an error. For a list of possible result codes, see "Result Codes" (page 169).

Discussion
This function deletes the specified record. Deleting the record invalidates the record reference. Therefore, before deleting a record, be sure to call `dsCloseAttributeList` (page 20) and `dsCloseAttributeValueList` (page 20) to close any attribute list references and attribute value list references that may have been allocated.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`DirServices.h`

dsDoAttributeValueSearch

Searches a node for records by attribute value.

```
tDirStatus dsDoAttributeValueSearch (
   tDirNodeReference inDirNodeReference,
   tDataBufferPtr inOutDataBuffer,
   tDataListPtr inRecordTypeList,
   tDataNodePtr inAttributeType,
   tDirPatternMatch inPatternMatchType,
   tDataNodePtr inPattern2Match,
   UInt32 *inOutMatchRecordCount,
   tContextData *inOutContinueData
);
```

Parameters

`inDirNodeReference`

On input, a value of type `tDirNodeReference` (page 89), obtained by calling `dsOpenDirNode` (page 68), that identifies the node that is to be searched.

`inOutDataBuffer`

On input, a value of type `tDataBufferPtr` (page 87) created by calling `dsDataBufferAllocate` (page 25) that points to the `tDataBuffer` (page 84) structure in which this function is to place search results. On output, if `inOutMatchRecordCount` points to a value greater than zero, call `dsGetRecordEntry` (page 63), `dsGetAttributeEntry` (page 51), and `dsGetAttributeValue` (page 53) to get the records, attributes, and attribute values from the data buffer.

`inRecordTypeList`

On input, a value of type `tDataListPtr` pointing to a `tDataList` (page 84) structure allocated by calling `dsDataListAllocate` (page 26) that contains a list of record types to search. Set the record type to `kDSStdRecordTypeAll` to search all records. See Standard Record Types (page 143) for other possible values.

`inAttributeType`

On input, a value of type `tDataNodePtr` (page 88) that points to a `tDataNode` (page 88) structure allocated by calling `dsDataNodeAllocateBlock` (page 31) or `dsDataNodeAllocateString` (page 32) that contains an attribute type to search for. To search all attribute types, set the attribute type to `kDSAttributesAll`. See the attribute constants described in the "Constants" (page 129) section for other possible values.

`inPatternMatchType`

On input, a value of type `tDirPatternMatch` specifying a pattern type that controls the way in which the pattern specified by `inPattern2Match` is compared with attribute values. See Pattern Matching Constants (page 137) for possible values. The pattern type may also be defined by the Open Directory plug-in that handles the directory service represented by `inDirNodeReference`.

`inPattern2Match`

On input, a value of type `tDataNodePtr` (page 88) that points to a `tDataNode` (page 88) structure allocated by calling `dsDataNodeAllocateBlock` (page 31) or `dsDataNodeAllocateString` (page 32) that contains the pattern to match.

`inOutMatchRecordCount`

On input, a pointer to a value of type `unsigned long` that specifies the total number of matching records to get across multiple calls to this function. Set this value to zero to get all matching records. On output, `inOutRecordMatchCount` points to the number of records in the data buffer pointed to by `inOutDataBuffer`. Once you start a series of `dsDoAttributeValueSearch` calls, `inOutMatchRecordCount` is ignored as an input parameter.

`inOutContinueData`

On input, a pointer to a value of type `tContextData` (page 87) and set to `NULL`. On output, if the value pointed to by `inOutContinueData` is not `NULL`, get more matching records by calling this function again and pass the value pointed to by `inOutContinueData`. If on output `inOutContinueData` is `NULL`, there are no more records to get. If `inOutContinueData` is not `NULL` and an error occurs or you don't want to get any more matching records, you must call `dsReleaseContinueData` (page 72) to release the memory associated with `inOutContinueData`.

`function result`

A value of type `tDirStatus` indicating success (`eDSNoErr`) or an error. For a list of possible result codes, see "Result Codes" (page 169).

Discussion

This function works across multiple calls to obtain a list of all records having attributes whose values match the specified pattern. Set `inOutRecordMatchCount` to zero to get all matching records. If you want to limit the number of matching records that this function returns, set `inOutRecordMatchCount` to a positive integer value that specifies the limit.

On output, matching records are returned in the buffer pointed to by `inOutDataBuffer` with the number of records in that buffer indicated by the value pointed to by `inOutMatchRecordCount`.

On output, the value pointed to by `inOutContinueData` indicates whether you should call this function again to obtain more matching records. If `inOutContinueData` is `NULL`; you do not need to call this function again. If `inOutContinueData` is not `NULL` and you do not want to continue the search, you must call `dsReleaseContinueData` (page 72) to deallocate the memory that is associated with `inOutContinueData`.

If there are too many records to fit in a single buffer, this function returns a non-null value in the value pointed to by `inOutContinueData`. To get more records, call this function again, passing the pointer to `inOutContinueData` that was returned by the previous call to this function.

If this function returns `eDSBufferTooSmall`, the buffer is too small for a record that is to be returned. You should allocate a larger buffer and try again. When this function returns `eDSBufferTooSmall`, `inOutContinueData` is also set.

If the value pointed to by `inOutContinueData` is not `NULL` and the value returned by this function is zero, more results may be available. Continue calling this function until `inOutContinueData` points to a `NULL` value.

To get a record from the data buffer pointed to by `inOutDataBuffer`, call `dsGetRecordEntry` (page 63). To get information about the record's attributes, call `dsGetAttributeEntry` (page 51). To get the value of a record's attribute, call `dsGetAttributeValue` (page 53).

If `inOutContinueData` is not `NULL` and you no longer need it, call `dsReleaseContinueData` (page 72) to release the memory associated with it.

Special Considerations

In a series of calls to this function, the value of `inOutRecordEntryCount` must be set by the first call. Its value is ignored in the next calls in the series.

See `dsDoAttributeValueSearchWithData` (page 40) to get information about other attribute types and their values.

Availability

Available in Mac OS X v10.0 and later.

Declared In
`DirServices.h`

dsDoAttributeValueSearchWithData

Searches for records by attribute type and attribute value.

```
tDirStatus dsDoAttributeValueSearchWithData (
   tDirNodeReference inDirNodeReference,
   tDataBufferPtr inOutDataBuffer,
   tDataListPtr inRecordTypeList,
   tDataNodePtr inAttributeMatchType,
   tDirPatternMatch inPatternMatchType,
   tDataNodePtr inPatternToMatch,
   tDataListPtr inAttributeTypeRequestList,
   dsBool inAttributeInfoOnly,
   UInt32 *inOutMatchRecordCount,
   tContextData *inOutContinueData
);
```

Parameters

inDirNodeReference

On input, a value of type `tDirNodeReference` (page 89), obtained by calling `dsOpenDirNode` (page 68), that identifies the node that is to be searched.

inOutDataBuffer

On input, a value of type `tDataBufferPtr` (page 87) created by calling `dsDataBufferAllocate` (page 25) that points to the `tDataBuffer` (page 84) structure in which this function is to place search results. On output, if `inOutMatchRecordCount` points to a value greater than zero, call `dsGetRecordEntry` (page 63), `dsGetAttributeEntry` (page 51), and `dsGetAttributeValue` (page 53) to get the records, attributes, and attribute values from the data buffer.

inRecordTypeList

On input, a value of type `tDataListPtr` pointing to a `tDataList` (page 84) structure allocated by calling `dsDataListAllocate` (page 26) that contains a list of the record types to search for. Set the record type to `kDSStdRecordTypeAll` to search all records. For other possible values, see Standard Record Types (page 143).

inAttributeMatchType

On input, a value of type `tDataNodePtr` (page 88) that points to a `tDataNode` (page 88) structure allocated by calling `dsDataNodeAllocateBlock` (page 31) or `dsDataNodeAllocateString` (page 32) that contains an attribute type to search for. To search all attribute types, set the attribute type to `kDSAttributesAll`. For other possible values, see the attribute constants described in the “`Constants`” (page 129) section for other possible values.

inPatternMatchType

On input, a value of type `tDirPatternMatch` specifying a pattern type that controls the way in which the pattern specified by `inPattern2Match` is compared with attribute values. See Pattern Matching Constants (page 137) for possible values. The pattern type may also be defined by the Open Directory plug-in that handles the directory service represented by `inDirNodeReference`.

inPatternToMatch

On input, a value of type `tDataNodePtr` (page 88) that points to a `tDataNode` (page 88) structure allocated by calling `dsDataNodeAllocateBlock` (page 31) or `dsDataNodeAllocateString` (page 32) that contains the pattern to match.

`inAttributeTypeRequestList`
: On input, a value of type `tDataListPtr` (page 87) pointing to a `tDataList` (page 84) structure allocated by calling `dsDataListAllocate` (page 26) that specifies the record attribute types that are to be returned.

`inAttributeInfoOnly`
: On input, a value of type `dsBool` set to `TRUE` if the calling application only wants information about attributes. To get the values of the attributes as well as information about the attributes, set `inAttributeInfoOnly` to `FALSE`.

`inOutMatchRecordCount`
: On input, a pointer to a value of type `long` that specifies the number of matching records to get. On output, `inOutRecordEntryCount` points to the number of records in the data buffer pointed to by `inOutDataBuffer`; the number may be less than the requested number if there were not enough matching records to fill the buffer. The caller cannot change the value of `inOutRecordEntryCount` across multiple calls to this function using the value pointed to by `inOutContinueData`.

`inOutContinueData`
: On input, a pointer to a value of type `tContextData` (page 87) and set to `NULL`. On output, if the value pointed to by `inOutContinueData` is `NULL`, there are no new results in the buffer. If the value pointed to by `inOutContinueData` is not `NULL` on output, pass the value pointed to by `inOutContinueData` to this function again to get the next entries. You must call `dsReleaseContinueData` (page 72) if you don't want to get the remaining records.

`function result`
: A value of type `tDirStatus` indicating success (`eDSNoErr`) or an error. For a list of possible result codes, see "Result Codes" (page 169).

Discussion
This function stores in the data buffer pointed to by `inOutDataBuffer` a list of records having attributes of the type specified by the `inAttributeMatchType` parameter whose values match the specified pattern.

Set `inOutRecordEntryCount` to point to a positive integer value that represents the number of records that are to be returned. You cannot change the value pointed to by `inOutRecordEntryCount` if you call this function with `inOutContinueData` pointing to context data returned by a previous call to this function.

If there are too many records to fit in a single buffer, this function returns a non-null value in the value pointed to by `inOutContinueData`. To get more records, call this function again, passing the pointer to `inOutContinueData` that was returned by the previous call to this function.

To get a record from the data buffer pointed to by `inOutDataBuffer`, call `dsGetRecordEntry` (page 63). To get information about the record's attributes, call `dsGetAttributeEntry` (page 51). To get the value of a record's attribute, call `dsGetAttributeValue` (page 53).

When you no longer need `inOutContinueData`, call `dsReleaseContinueData` (page 72) to release the memory associated with it.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`DirServices.h`

dsDoDirNodeAuth

Performs authentication with a node.

```
tDirStatus dsDoDirNodeAuth (
   tDirNodeReference inDirNodeReference,
   tDataNodePtr inDirNodeAuthName,
   dsBool inDirNodeAuthOnlyFlag,
   tDataBufferPtr inAuthStepData,
   tDataBufferPtr outAuthStepDataResponse,
   tContextData *inOutContinueData
);
```

Parameters

inDirNodeReference

On input, a value of type `tDirNodeReference` (page 89) obtained by previously calling `dsOpenDirNode` (page 68) that representing the node that is to be authenticated.

inDirNodeAuthName

On input, a value of type `tDataNodePtr` (page 88) pointing to a `tDataNode` (page 88) structure allocated by calling `dsDataNodeAllocateBlock` (page 31) or `dsDataNodeAllocateString` (page 32) containing the authentication method to use. Authentication methods vary from user to user. Examples include `kDSStdAuthSetPasswd`, `kDSStdAuthSetPasswdAsRoot`, and `kDSStdAuthChangePasswd` to set or change a password and `kDSStdAuthNodeNativeNoClearText` to authenticate a user. If changes will be made to the node after authentication, the value of the `inDirNodeAuthOnlyFlag` parameter should be `FALSE`. For other possible values, see Authentication Constants (page 129).

inDirNodeAuthOnlyFlag

On input, a value of type `dsBool` that indicates whether the result of authentication will be used in the future. A file server that is only authenticating a user should set this parameter to `TRUE` to indicate that once the user is authenticated, the result will not be used in the future. An application that might make changes to the node after authentication would set this parameter to `FALSE` to indicate that the result may be used in the future.

inAuthStepData

On input, this parameter contains the data necessary for this step in the authentication process. This parameter is a value of type `tDataBufferPtr` (page 87) created by calling `dsDataBufferAllocate` (page 25) that points to a `tDataBuffer` (page 84) structure.

outAuthStepDataResponse

On output, this parameter contains the plug-in's response. If the authentication was not successful, the buffer contains a plug-in–defined value. If there are more steps in the authentication process, the buffer contains a plug-in–defined value that is used in the next step of the authentication process. This parameter is a value of type `tDataBufferPtr` (page 87) created by calling `dsDataBufferAllocate` (page 25) pointing to a `tDataBuffer` (page 84) structure.

inOutContinueData

On input, a pointer to a value of type `tContextData` (page 87) and set to `NULL`. On output, if the value pointed to by `inOutContinueData` is `NULL`, there are no more steps in the authentication process. If `inOutContinueData` is not `NULL` on output, there are more steps to complete. Call this function again and pass to it the value pointed to by `inOutContinueData`. Call `dsReleaseContinueData` (page 72) if the value pointed to by `inOutContinueData` is not `NULL` and you do not want to complete the authentication process.

function result

A value of type `tDirStatus` indicating successful authentication (`eDSNoErr`) or an error, such as `eDSAuthFailed`. Other authentication result codes include `eDSAuthMethodNotSupported`, `eDSAuthInBuffFormatError`, `eDSAuthNoSuchEntity`, `eDSAuthBadPassword`, `eDSAuthContinueDataBad`, `eDSAuthUnknownUser`, `eDSAuthCannotRecoverPasswd`, `eDSAuthFailedClearTextOnly`, `eDSAuthNoAuthServerFound`, `eDSAuthServerError`, `eDSAuthNewPasswordRequired`, `eDSAuthPasswordExpired`, `eDSAuthPasswordQualityCheckFailed`, `eDSAuthAccountDisabled`, `eDSAuthAccountExpired`, and `eDSAuthAccountInactive`. For an explanation of these result codes, see "Result Codes" (page 169).

Discussion

This function performs a variety of authentication tasks, such as authenticating a user, setting a password, and changing a password, depending on the value of the `inDirNodeAuthName` parameter.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

NetworkAuthentication

Declared In

`DirServices.h`

dsDoDirNodeAuthOnRecordType

Performs authentication using a record type.

```
tDirStatus dsDoDirNodeAuthOnRecordType (
   tDirNodeReference inDirNodeReference,
   tDataNodePtr inDirNodeAuthName,
   dsBool inDirNodeAuthOnlyFlag,
   tDataBufferPtr inAuthStepData,
   tDataBufferPtr outAuthStepDataResponse,
   tContextData *inOutContinueData,
   tDataNodePtr inRecordType
);
```

Parameters

inDirNodeReference

On input, a value of type `tDirNodeReference` (page 89) obtained by previously calling `dsOpenDirNode` (page 68) that representing the node that is to be authenticated.

inDirNodeAuthName

On input, a value of type `tDataNodePtr` (page 88) pointing to a `tDataNode` (page 88) structure allocated by calling `dsDataNodeAllocateBlock` (page 31) or `dsDataNodeAllocateString` (page 32) containing the authentication method to use. Authentication methods vary from record to record. Examples include `kDSStdAuthSetPasswd`, `kDSStdAuthSetPasswdAsRoot`, and `kDSStdAuthChangePasswd` to set or change a password and `kDSStdAuthNodeNativeNoClearText` to authenticate a user. If changes will be made to the node after authentication, the value of the `inDirNodeAuthOnlyFlag` parameter should be `FALSE`. For other possible values, see Authentication Constants (page 129).

`inDirNodeAuthOnlyFlag`

On input, a value of type `dsBool` that indicates whether the result of authentication will be used in the future. A file server that is only authenticating a user should set this parameter to `TRUE` to indicate that once the user is authenticated, the result will not be used in the future. An application that might make changes to the node after authentication would set this parameter to `FALSE` to indicate that the result may be used in the future.

`inAuthStepData`

On input, this parameter contains the data necessary for this step in the authentication process. This parameter is a value of type `tDataBufferPtr` (page 87) created by calling `dsDataBufferAllocate` (page 25) that points to a `tDataBuffer` (page 84) structure.

`outAuthStepDataResponse`

On output, this parameter contains the plug-in's response. If the authentication was not successful, the buffer contains a plug-in–defined value. If there are more steps in the authentication process, the buffer contains a plug-in–defined value that is used in the next step of the authentication process. This parameter is a value of type `tDataBufferPtr` (page 87) created by calling `dsDataBufferAllocate` (page 25) pointing to a `tDataBuffer` (page 84) structure.

`inOutContinueData`

On input, a pointer to a value of type `tContextData` (page 87) and set to `NULL`. On output, if the value pointed to by `inOutContinueData` is `NULL`, there are no more steps in the authentication process. If `inOutContinueData` is not `NULL` on output, there are more steps to complete. Call this function again and pass to it the value pointed to by `inOutContinueData`. Call `dsReleaseContinueData` (page 72) if the value pointed to by `inOutContinueData` is not `NULL` and you do not want to complete the authentication process.

`inRecordType`

On input, a value of type `tDataNodePtr` (page 88) that points to a `tDataBuffer` (page 84) structure allocated by calling `dsDataBufferAllocate` (page 25) containing the type of the record to use for authentication. Currently, the only record types that are allowed are `kDSStdRecordTypeComputers` and `kDSStdRecordTypeUsers`.If this parameter is `NULL`, `dsDoDirNodeAuth` (page 41) is called and a record type of `kDSStdRecordTypeUsers` is used.

`function result`

A value of type `tDirStatus` indicating successful authentication (`eDSNoErr`) or an error, such as `eDSAuthFailed`. Other authentication result codes include `eDSAuthMethodNotSupported`, `eDSAuthInBuffFormatError`, `eDSAuthNoSuchEntity`, `eDSAuthBadPassword`, `eDSAuthContinueDataBad`, `eDSAuthUnknownUser`, `eDSAuthCannotRecoverPasswd`, `eDSAuthFailedClearTextOnly`, `eDSAuthNoAuthServerFound`, `eDSAuthServerError`, `eDSAuthNewPasswordRequired`, `eDSAuthPasswordExpired`, `eDSAuthPasswordQualityCheckFailed`, `eDSAuthAccountDisabled`, `eDSAuthAccountExpired`, and `eDSAuthAccountInactive`. For an explanation of these result codes, see "Result Codes" (page 169).

Discussion

This function uses a record type of `kDDStdRecordTypeUsers` or `kDSStdRecordTypeComputers` to perform authentication. Specifying a record type of `kDSStdRecordTypeUsers` is equivalent to calling `dsDoDirNodeAuth` (page 41). Records of type `kDSStdRecordTypeUsers` and `kDSStdRecordTypeComputers` are the only records that can be used for authentication.

Version Notes

Introduced in Mac OS X v10.3.

Availability

Available in Mac OS X v10.3 and later.

Declared In

`DirServices.h`

dsDoMultipleAttributeValueSearch

Uses multiple attribute values to search a node for records.

```
tDirStatus dsDoMultipleAttributeValueSearch (
   tDirNodeReference inDirNodeReference,
   tDataBufferPtr inOutDataBuffer,
   tDataListPtr inRecordTypeList,
   tDataNodePtr inAttributeType,
   tDirPatternMatch inPatternMatchType,
   tDataListPtr inPatterns2Match,
   UInt32 *inOutMatchRecordCount,
   tContextData *inOutContinueData
);
```

Parameters

`inDirNodeReference`

On input, a value of type `tDirNodeReference` (page 89), obtained by calling `dsOpenDirNode` (page 68), that identifies the node that is to be searched.

`inOutDataBuffer`

On input, a value of type `tDataBufferPtr` (page 87) created by calling `dsDataBufferAllocate` (page 25) that points to the `tDataBuffer` (page 84) structure in which this function is to place search results. On output, if `inOutMatchRecordCount` points to a value greater than zero, call `dsGetRecordEntry` (page 63), `dsGetAttributeEntry` (page 51), and `dsGetAttributeValue` (page 53) to get the records, attributes, and attribute values from the data buffer.

`inRecordTypeList`

On input, a value of type `tDataListPtr` pointing to a `tDataList` (page 84) structure allocated by calling `dsDataListAllocate` (page 26) that contains a list of record types to search. Set the record type to `kDSStdRecordTypeAll` to search all records. See Standard Record Types (page 143) for other possible values.

`inAttributeType`

On input, a value of type `tDataNodePtr` (page 88) that points to a `tDataNode` (page 88) structure allocated by calling `dsDataNodeAllocateBlock` (page 31) or `dsDataNodeAllocateString` (page 32) that contains an attribute type to search for. To search all attribute types, set the attribute type to `kDSAttributesAll`. See the attribute constants described in the "Constants" (page 129) section for other possible values.

`inPatternMatchType`

On input, a value of type `tDirPatternMatch` specifying a pattern type that controls the way in which the pattern specified by `inPattern2Match` is compared with attribute values. See Pattern Matching Constants (page 137) for possible values. The pattern type may also be defined by the Open Directory plug-in that handles the directory service represented by `inDirNodeReference`.

`inPatterns2Match`

On input, a value of type `tDataListPtr` (page 87) that points to a list of patterns to match.

inOutMatchRecordCount
: On input, a pointer to a value of type `unsigned long` that specifies the total number of matching records to get across multiple calls to this function. Set this value to zero to get all matching records. On output, `inOutRecordMatchCount` points to the number of records in the data buffer pointed to by `inOutDataBuffer`. Once you start a series of `dsDoMultipleAttributeValueSearch` calls, `inOutMatchRecordCount` is ignored as an input parameter.

inOutContinueData
: On input, a pointer to a value of type `tContextData` (page 87) and set to `NULL`. On output, if the value pointed to by `inOutContinueData` is not `NULL`, get more matching records by calling this function again and pass the value pointed to by `inOutContinueData`. If on output `inOutContinueData` is `NULL`, there are no more records to get. If `inOutContinueData` is not `NULL` and an error occurs or you don't want to get any more matching records, you must call `dsReleaseContinueData` (page 72) to release the memory associated with `inOutContinueData`.

function result
: A value of type `tDirStatus` indicating success (`eDSNoErr`) or an error. A result code of `eDSBufferTooSmall` indicates you should allocate a larger buffer and call this function again. For a list of possible result codes, see "Result Codes" (page 169).

Discussion

This function works across multiple calls to obtain a list of all records having an attribute whose value matches one of the patterns specified by `inPatters2Match`. Set `inOutRecordMatchCount` to zero to get all matching records. If you want to limit the number of matching records that this function returns, set `inOutRecordMatchCount` to a positive integer value that specifies the limit.

On output, matching records are returned in the buffer pointed to by `inOutDataBuffer` with the number of records in that buffer indicated by the value pointed to by `inOutDataBuffer`.

On output, the value pointed to by `inOutContinueData` indicates whether you should call this function again to obtain more matching records. If `inOutContinueData` is `NULL`; you do not need to call this function again. If `inOutContinueData` is not `NULL` and you do not want to continue the search, you must call `dsReleaseContinueData` (page 72) to deallocate the memory that is associated with `inOutContinueData`.

If there are too many records to fit in a single buffer, this function returns a non-null value in the value pointed to by `inOutContinueData`. To get more records, call this function again, passing the pointer to `inOutContinueData` that was returned by the previous call to this function.

If the value pointed to by `inOutContinueData` is not `NULL` and the value returned by this function is zero, more results may be available. Continue calling this function until `inOutContinueData` points to a `NULL` value.

To get a record from the data buffer pointed to by `inOutDataBuffer`, call `dsGetRecordEntry` (page 63). To get information about the record's attributes, call `dsGetAttributeEntry` (page 51). To get the value of a record's attribute, call `dsGetAttributeValue` (page 53).

If `inOutContinueData` is not `NULL` and you no longer need it, call `dsReleaseContinueData` (page 72) to release the memory associated with it.

Special Considerations

In a series of calls to this function, the value of `inOutRecordEntryCount` must be set by the first call. Its value is ignored in the next calls in the series.

See `dsDoAttributeValueSearchWithData` (page 40) to get information about other attribute types and their values.

Version Notes
Introduced in Mac OS X v10.4.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`DirServices.h`

dsDoMultipleAttributeValueSearchWithData

Searches for records by attribute type and multiple attribute values.

```
tDirStatus dsDoMultipleAttributeValueSearchWithData (
   tDirNodeReference inDirNodeReference,
   tDataBufferPtr inOutDataBuffer,
   tDataListPtr inRecordTypeList,
   tDataNodePtr inAttributeMatchType,
   tDirPatternMatch inPatternMatchType,
   tDataListPtr inPatternsToMatch,
   tDataListPtr inAttributeTypeRequestList,
   dsBool inAttributeInfoOnly,
   UInt32 *inOutMatchRecordCount,
   tContextData *inOutContinueData
);
```

Parameters

`inDirNodeReference`

On input, a value of type `tDirNodeReference` (page 89), obtained by calling `dsOpenDirNode` (page 68), that identifies the node that is to be searched.

`inOutDataBuffer`

On input, a value of type `tDataBufferPtr` (page 87) created by calling `dsDataBufferAllocate` (page 25) that points to the `tDataBuffer` (page 84) structure in which this function is to place search results. On output, if `inOutMatchRecordCount` points to a value greater than zero, call `dsGetRecordEntry` (page 63), `dsGetAttributeEntry` (page 51), and `dsGetAttributeValue` (page 53) to get the records, attributes, and attribute values from the data buffer.

`inRecordTypeList`

On input, a value of type `tDataListPtr` pointing to a `tDataList` (page 84) structure allocated on the stack or by calling `dsDataListAllocate` (page 26) that contains a list of the record types to search for. Set the record type to `kDSStdRecordTypeAll` to search all records. For other possible values, see Standard Record Types (page 143).

`inAttributeMatchType`

On input, a value of type `tDataNodePtr` (page 88) that points to a `tDataNode` (page 88) structure allocated by calling `dsDataNodeAllocateBlock` (page 31) or `dsDataNodeAllocateString` (page 32) that contains the attribute type to search for. To search all attribute types, set the attribute type to `kDSAttributesAll`. See the attribute constants described in the "Constants" (page 129) section for other possible values.

inPatternMatchType
: On input, a value of type `tDirPatternMatch` specifying a pattern type that controls the way in which the pattern specified by `inPattern2Match` is compared with attribute values. See Pattern Matching Constants (page 137) for possible values. The pattern type may also be defined by the Open Directory plug-in that handles the directory service represented by `inDirNodeReference`.

inPatternsToMatch
: On input, a value of type `tDataListPtr` (page 87) that points to a list of patterns to match.

inAttributeTypeRequestList
: On input, a value of type `tDataListPtr` pointing to a `tDataList` (page 84) structure allocated by calling `dsDataListAllocate` (page 26) that specifies the record attribute types that are to be returned.

inAttributeInfoOnly
: On input, a value of type `dsBool` set to `TRUE` if the calling application only wants information about attributes. To get the values of the attributes as well as information about the attributes, set `inAttributeInfoOnly` to `FALSE`.

inOutMatchRecordCount
: On input, a pointer to a value of type `long` that specifies the number of matching records to get. On output, `inOutRecordEntryCount` points to the number of records in the data buffer pointed to by `inOutDataBuffer`; the number may be less than the requested number if there were not enough matching records to fill the buffer. The caller cannot change the value of `inOutRecordEntryCount` across multiple calls to this function using the value pointed to by `inOutContinueData`.

inOutContinueData
: On input, a pointer to a value of type `tContextData` (page 87) and set to `NULL`. On output, if the value pointed to by `inOutContinueData` is `NULL`, there are no new results in the buffer. If the value pointed to by `inOutContinueData` is not `NULL` on output, pass the value pointed to by `inOutContinueData` to this function again to get the next entries. You must call `dsReleaseContinueData` (page 72) if you don't want to get the remaining records.

function result
: A value of type `tDirStatus` indicating success (`eDSNoErr`) or an error. A result code of `eDSBufferTooSmall` indicates you should allocate a larger buffer and call this function again. For a list of possible result codes, see "Result Codes" (page 169).

Discussion

This function stores in the data buffer pointed to by `inOutDataBuffer` a list of records having attributes of the type specified by the `inAttributeMatchTypes` parameter whose values match the specified pattern.

Set `inOutRecordEntryCount` to point to a positive integer value that represents the number of records that are to be returned. You cannot change the value pointed to by `inOutRecordEntryCount` if you call this function with `inOutContinueData` pointing to context data returned by a previous call to this function.

If there are too many records to fit in a single buffer, this function returns a non-null value in the value pointed to by `inOutContinueData`. To get more records, call this function again, passing the pointer to `inOutContinueData` that was returned by the previous call to this function.

To get a record from the data buffer pointed to by `inOutDataBuffer`, call `dsGetRecordEntry` (page 63). To get information about the record's attributes, call `dsGetAttributeEntry` (page 51). To get the value of a record's attribute, call `dsGetAttributeValue` (page 53).

When you no longer need `inOutContinueData`, call `dsReleaseContinueData` (page 72) to release the memory associated with it.

Version Notes
Introduced in Mac OS X v10.4.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`DirServices.h`

dsDoPlugInCustomCall

Exchanges custom information with an Open Directory plug-in.

```
tDirStatus dsDoPlugInCustomCall (
   tDirNodeReference inDirNodeReference,
   UInt32 inCustomRequestCode,
   tDataBufferPtr inCustomRequestData,
   tDataBufferPtr outCustomRequestResponse
);
```

Parameters

`inDirNodeReference`
: On input, a value of type `tDirNodeReference` (page 89), obtained by calling `dsOpenDirNode` (page 68), that identifies the open node for which custom information is to be exchanged.

`inCustomRequestCode`
: On input, a value of type `unsigned long`, containing a request code that is to be sent to the plug-in.

`inCustomRequestData`
: On input, a value of type `tDataBufferPtr` (page 87) created by calling `dsDataBufferAllocate` (page 25) that points to a `tDataBuffer` (page 84) structure containing data that is to be sent to the plug-in.

`outCustomRequestResponse`
: On input, a value of type `tDataBufferPtr` (page 87) created by calling `dsDataBufferAllocate` (page 25) that points to a `tDataBuffer` (page 84) structure. On output, the buffer contains the plug-in's response to the information that was sent.

`function result`
: A value of type `tDirStatus` indicating success (`eDSNoErr`) or an error. For a list of possible result codes, see "Result Codes" (page 169).

Discussion
This function exchanges custom information with the Open Directory plug-in for the node represented by `inDirNodeReference`.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`DirServices.h`

dsFindDirNodes

Finds the registered node names that match a pattern.

```
tDirStatus dsFindDirNodes (
   tDirReference inDirReference,
   tDataBufferPtr inOutDataBufferPtr,
   tDataListPtr inNodeNamePattern,
   tDirPatternMatch inPatternMatchType,
   UInt32 *outDirNodeCount,
   tContextData *inOutContinueData
);
```

Parameters

inDirReference

On input, a value of type `tDirReference` (page 89) obtained by calling `dsOpenDirService` (page 69).

inOutDataBufferPtr

On input, a value of type `tDataBufferPtr` (page 87) created by calling `dsDataBufferAllocate` (page 25) that points to a `tDataBuffer` (page 84) structure in which the results are to be returned. On output, call `dsGetDirNodeName` (page 58) to extract the results from the data buffer pointed to by `inOutDataBufferPtr`.

inNodeNamePattern

On input, a value of type `tDataListPtr` pointing to a `tDataList` (page 84) structure containing the pattern that is to be matched. Set this parameter to `NULL` and `inPatternMatchType` to the appropriate constant to get the contacts search node (`eDSContactsSearchNodeName`), network search node (`eDSNetworkSearchNodeName`), authentication search node (`eDSAuthenticationSearchNodeName`), the node for the local NetInfo domain (`eDSLocalNodeNames`), or locally hosted nodes (`eDSLocalHostedNodes`).

inPatternMatchType

On input, a value of type `tDirPatternMatch` specifying a pattern type that controls the way in which the pattern specified by `inNodeNamePattern` is compared with registered node names. See Pattern Matching Constants (page 137) for possible values.

outDirNodeCount

On output, a pointer to a value of type `unsigned long` in which this function has stored the number of registered node names in the data buffer pointed to by `inOutDataBufferPtr`.

inOutContinueData

On input, a pointer to a value of type `tContextData` (page 87) and set to `NULL`. On output, if `inOutContinueData` points to a value that is `NULL`, there is no more response data to get. If `inOutContinueData` points to a value that is not `NULL`, there is more response data, which you can get by calling this function again and passing the context data pointed to by `inOutContinueData`. If `inOutContinueData` points to a value that is not `NULL` and you do not want to get the remaining response data, you must call `dsReleaseContinueData` (page 72) to deallocate the memory associated with `inOutContinueData`.

function result

A value of type `tDirStatus` indicating success (`eDSNoErr`) or an error. For a list of possible result codes, see "Result Codes" (page 169).

Discussion

This function searches the list of nodes that have been registered by Open Directory plug-ins for the directory service represented by `inDirReference` for names that match a pattern. It places the names that match the pattern in the data buffer pointed to by `inOutDataBufferPtr`. Use the `inNodeNamePattern` parameter to specify pattern to match and the `inPatternMatchType` parameter to specify how the pattern is to be matched or to specify that a search node is to be found.

On output, `outDirNodeCount` contains the number of matching registered node names that this function has found. Call `dsGetDirNodeName` (page 58) to extract the names from the data buffer.

On output, if `inOutContinueData` points to a value that is not `NULL`, there are more matching registered node names for this function to find even if `outDirNodeCount` points to a zero value. To get another buffer of matching registered node names, call this function again and pass to it the context data pointed to by `inOutContinueData`. If you do not want to get another buffer of matching node names, you must call `dsReleaseContinueData` (page 72) to deallocate the context data pointed to by `inOutContinueData`.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
NetworkAuthentication

Declared In
`DirServices.h`

dsFlushRecord

Writes a record.

```
tDirStatus dsFlushRecord (
   tRecordReference inRecordReference
);
```

Parameters

`inRecordReference`
: On input, value of type tRecordReference (page 89) obtained by previously calling `dsOpenRecord` (page 71).

`function result`
: A value of type `tDirStatus` indicating success (`eDSNoErr`) or an error. For a list of possible result codes, see "Result Codes" (page 169).

Discussion
This function requests the directory service to write the record. The directory service may comply with the request or may choose to ignore it.

The value returned by this function does not reflect whether the record was actually written.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`DirServices.h`

dsGetAttributeEntry

Gets an attribute entry from a data buffer.

```
tDirStatus dsGetAttributeEntry (
   tDirNodeReference inDirNodeReference,
   tDataBufferPtr inOutDataBuffer,
   tAttributeListRef inAttributeListRef,
   UInt32 inAttributeInfoIndex,
   tAttributeValueListRef *outAttributeValueListRef,
   tAttributeEntryPtr *outAttributeInfoPtr
);
```

Parameters

inDirNodeReference

On input, a value of type `tDirNodeReference` (page 89) obtained by calling `dsOpenDirNode` (page 68) representing the node associated with the data in the buffer pointed to by `inOutDataBuffer`.

inOutDataBuffer

On input, a value of type `tDataBufferPtr` (page 87) pointing to a `tDataBuffer` (page 84) structure containing data returned, for example, by a previous call to `dsGetDirNodeInfo` (page 56) or `dsGetRecordList` (page 64).

inAttributeListRef

On input, a value of type `tAttributeListRef` (page 86) obtained by previously calling `dsGetDirNodeInfo` (page 56) or `dsGetRecordEntry` (page 63).

inAttributeInfoIndex

On input, a value of type `unsigned long`. Set `inAttributeInfoIndex` to 1 to get the first attribute entry. Set `inAttributeInfoIndex` to 2 to get the second attribute entry, and so on.

outAttributeValueListRef

On output, a pointer to a value of type `tAttributeValueListRef` (page 86). Pass the pointer to `outAttributeValueListRef` to `dsGetAttributeValue` (page 53) to get the value of the attribute.

outAttributeInfoPtr

On output, a pointer to a value of type `tAttributeEntryPtr` (page 85) that points to a `tAttributeEntry` (page 82) structure in which this function stores information about the attribute specified by `inAttributeInfoIndex`. The information includes the number of attribute values, the maximum size of the attribute's value, and the attribute's signature.

function result

A value of type `tDirStatus` indicating success (`eDSNoErr`) or an error. For a list of possible result codes, see "Result Codes" (page 169).

Discussion

This function uses an attribute list reference to extract an attribute entry from a data buffer previously obtained by calling `dsGetDirNodeInfo` (page 56) or `dsGetRecordEntry` (page 63). The function stores the information in the `tAttributeEntry` (page 82) structure pointed to by `outAttributeInfoPtr`.

To get the value(s) of the attribute, call `dsGetAttributeValue` (page 53) and pass to it the data buffer pointed to by `inOutDataBuffer` and the attribute value list reference pointed to by `outAttributeValueListRef`.

When you no longer need the attribute value list pointed to by `outAttributeValueListRef`, call `dsCloseAttributeValueList` (page 20).

When you no longer need the `outAttributeInfoPtr` parameter, call `dsDeallocAttributeEntry` (page 35) to deallocate the `tAttributeEntry` (page 82) structure and its pointer.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code
NetworkAuthentication

Declared In
`DirServices.h`

dsGetAttributeValue

Gets the value of an attribute from a data buffer.

```
tDirStatus dsGetAttributeValue (
   tDirNodeReference inDirNodeReference,
   tDataBufferPtr inOutDataBuffer,
   UInt32 inAttributeValueIndex,
   tAttributeValueListRef inAttributeValueListRef,
   tAttributeValueEntryPtr *outAttributeValue
);
```

Parameters

`inDirNodeReference`
On input, a value of type `tDirNodeReference` (page 89) obtained by calling `dsOpenDirNode` (page 68) that represents the node for which the search was conducted.

`inOutDataBuffer`
On input, a value of type `tDataBufferPtr` (page 87) pointing to a `tDataBuffer` (page 84) structure that was previously filled in, for example, by calling `dsDoAttributeValueSearch` (page 37), `dsGetDirNodeInfo` (page 56), or `dsGetRecordList` (page 64).

`inAttributeValueIndex`
On input, a value of type `unsigned long`. Set `inAttributeValueIndex` to 1 to get the first attribute value. Set `inAttributeValueIndex` to 2 to get the second attribute value, and so on.

`inAttributeValueListRef`
On input, a value of type `tAttributeValueListRef` (page 86) obtained by calling `dsGetAttributeEntry` (page 51) that represents a `tAttributeValueEntry` (page 83) structure containing an attribute value ID and the value of the attribute represented by the attribute value ID.

`outAttributeValue`
On output, a pointer to a value of type `tAttributeValueEntryPtr` that points to an application-allocated `tAttributeValueEntry` (page 83) structure containing the attribute value ID and the value of the attribute.

`function result`
A value of type `tDirStatus` indicating success (`eDSNoErr`) or an error. For a list of possible result codes, see "Result Codes" (page 169).

Discussion
This function obtains the value of an attribute from a data buffer previously filled in, for example, by calling `dsDoAttributeValueSearch` (page 37) or `dsGetRecordList` (page 64), and stores the value in a `tAttributeValueEntry` (page 83) structure.

When you no longer need the attribute value list pointed to by `inAttributeValueListRef`, call `dsCloseAttributeValueList` (page 20). When you no longer need `outAttributeValue`, call `dsDeallocAttributeValueEntry` (page 35).

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
NetworkAuthentication

Declared In
`DirServices.h`

dsGetDataLength

Gets the length of data in a data list.

```
UInt32 dsGetDataLength (
   const tDataList *inDataList
);
```

Parameters

`inDataListPtr`
: On input, a pointer to a value of type `tDataList` (page 84) whose length is to be obtained.

`function result`
: The length of data in the specified data list or an error code. For a list of possible result codes, see "Result Codes" (page 169).

Discussion
This utility function obtains the length in bytes of data in a data list.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`DirServicesUtils.h`

dsGetDirNodeCount

Gets the total number of registered nodes.

```
tDirStatus dsGetDirNodeCount (
   tDirReference inDirReference,
   UInt32 *outDirectoryNodeCount
);
```

Parameters

`inDirReference`
: A value of type `tDirReference` (page 89) obtained by previously calling `dsOpenDirService` (page 69) or `dsOpenDirServiceProxy` (page 70).

`outDirectoryNodeCount`
: On output, a pointer to a value of type `unsigned long` containing the total number of registered nodes that are available to the Open Directory session represented by `inDirReference`.

`function result`
: A value of type `tDirStatus` indicating success (`eDSNoErr`) or an error. For a list of possible result codes, see "Result Codes" (page 169).

Discussion

This function gets the total number of registered nodes that are available to the Open Directory session represented by `inDirReference`. If you need to know whether directory names have changed even if the count has not changed, see `dsGetDirNodeCountWithInfo` (page 55).

Availability

Available in Mac OS X v10.0 and later.

Declared In

`DirServices.h`

dsGetDirNodeCountWithInfo

Gets the total number of registered nodes and a change token.

```
tDirStatus dsGetDirNodeCountWithInfo (
   tDirReference inDirReference,
   UInt32 *outDirectoryNodeCount,
   UInt32 *outDirectoryNodeChangeToken
);
```

Parameters

`inDirReference`

A value of type `tDirReference` (page 89) obtained by previously calling `dsOpenDirService` (page 69) or `dsOpenDirServiceProxy` (page 70).

`outDirectoryNodeCount`

On output, a pointer to a value of type `unsigned long` containing the total number of registered nodes that are available to the Open Directory session represented by `inDirReference`.

`outChangeToken`

On output, a pointer to a value of type `unsigned long` containing the change token. Save the value pointed to by `outChangeToken` and compare it with the next value received when you call this function again to see if there has been a change.

`function result`

A value of type `tDirStatus` indicating success (`eDSNoErr`) or an error. For a list of possible result codes, see "Result Codes" (page 169).

Discussion

This function gets the total number of registered nodes that are available to the Open Directory session represented by `inDirReference`, as well as a change token. To learn whether the node names have changed even though the number of registered nodes remains the same, call this function and get another change token. Compare the original and the new change token. The two change tokens will not be equal if there has been a change in the name of a registered node or to the number of registered nodes. If the change tokens are not equal, you may want to call `dsGetDirNodeList` (page 57) to get a new list of registered nodes.

The change token is only guaranteed to be *different* if the node names have changed. Do not assume that the new change token will be incremented or decremented relative to the value of the original change token.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`DirServices.h`

dsGetDirNodeInfo

Gets information about a node's attribute types and attribute values.

```
tDirStatus dsGetDirNodeInfo (
   tDirNodeReference inDirNodeReference,
   tDataListPtr inDirNodeInfoTypeList,
   tDataBufferPtr inOutDataBuffer,
   dsBool inAttributeInfoOnly,
   UInt32 *outAttributeInfoCount,
   tAttributeListRef *outAttributeListRef,
   tContextData *inOutContinueData
);
```

Parameters

inDirNodeReference

On input, a value of type `tDirNodeReference` (page 89), obtained by previously calling `dsOpenDirNode` (page 68), that identifies the node for which information is to be obtained.

inDirNodeInfoTypeList

On input, a value of type `tDataListPtr` (page 87) pointing to a `tDataList` (page 84) structure containing the attribute types for which information is requested. To get information about all attribute types, pass a `tDataList` (page 84) structure whose list is `kDSAttributesAll`.

inOutDataBuffer

On input, a value of type `tDataBufferPtr` (page 87) created by calling `dsDataBufferAllocate` (page 25) that points to a `tDataBuffer` (page 84) structure. On output, the `tDataBuffer` (page 84) structure contains the requested attribute type information for the specified node. If the input value of `inAttributeInfoOnly` is `FALSE`, the data buffer also contains attribute values. Call `dsGetAttributeEntry` (page 51) to extract attribute information from the buffer. Then call `dsGetAttributeValue` (page 53) to get the value of an attribute.

inAttributeInfoOnly

On input, a value of type `dsBool` set to `TRUE` if you only want attribute information. To get the values of the requested attributes as well as information about the attributes, set `inAttributeInfoOnly` to `FALSE`.

outAttributeInfoCount

On output, a pointer to a value of type `unsigned long` containing the number of attribute types in the data buffer pointed to by `inOutDataBuffer`.

outAttributeListRef

On input, a pointer to a value of type `tAttributeListRef` (page 86). When this function returns, use the attribute list reference pointed to by `outAttributeListRef` to call `dsGetAttributeEntry` (page 51) to get the attribute type information. Use information provided by calling `dsGetAttributeEntry` to call `dsGetAttributeValue` (page 53) to get the value of an attribute.

inOutContinueData

On input, a pointer to a value of type `tContextData` (page 87) and set to `NULL`. On output, if `inOutContinueData` points to a value that is `NULL`, there is no more response data to get. If `inOutContinueData` points to a value that is not `NULL`, there is more response data, which you can get by calling this function again and passing the context data pointed to by `inOutContinueData`. If `inOutContinueData` points to a value that is not `NULL` and you do not want to get the remaining response data, you must call `dsReleaseContinueData` (page 72) to deallocate the memory associated with `inOutContinueData`.

function result

A value of type `tDirStatus` indicating success (`eDSNoErr`) or an error. For a list of possible result codes, see "Result Codes" (page 169).

Discussion

This function gets attribute type information about a node, which may include attribute types for storing the authentication methods the node supports, the types of records the node contains, `kDS1AttrReadOnlyNode`, which indicates whether the node supports write operations, `kDSNAttrNodePath`, which indicates the node's name, and `kDSNAttrSubNodes`, which indicates nodes that are children of this node in the hierarchy.

You should call `dsCloseAttributeList` (page 20) when you no longer need the attribute list reference pointed to by `outAttributeListRef`.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`DirServices.h`

dsGetDirNodeList

Gets the names of registered nodes.

```
tDirStatus dsGetDirNodeList (
   tDirReference inDirReference,
   tDataBufferPtr inOutDataBufferPtr,
   UInt32 *outDirNodeCount,
   tContextData *inOutContinueData
);
```

Parameters

inDirReference

On input, a value of type `tDirReference` (page 89) obtained by previously calling `dsOpenDirService` (page 69) or `dsOpenDirServiceProxy` (page 70).

inOutDataBufferPtr

On input, a value of type `tDataBufferPtr` (page 87) created by calling `dsDataBufferAllocate` (page 25) that points to a `tDataBuffer` (page 84) structure. On output, the `tDataBuffer` (page 84) structure contains the requested list of registered node names. Call `dsGetDirNodeName` (page 58) to get a name from the buffer.

outDirNodeCount

On output, a pointer to a value of type `unsigned long` in which this function has stored the number of registered directory names in the data buffer pointed to by `inOutDataBufferPtr`.

inOutContinueData

On input, a pointer to a value of type `tContextData` (page 87) and set to `NULL`. On output, if `inOutContinueData` points to a value that is `NULL`, there is no more response data to get. If `inOutContinueData` points to a value that is not `NULL`, there is more response data, which you can get by calling this function again and passing the context data pointed to by `inOutContinueData`. If `inOutContinueData` points to a value that is not `NULL` and you do not want to get the remaining response data, you must call `dsReleaseContinueData` (page 72) to deallocate the memory associated with `inOutContinueData`.

function result

A value of type `tDirStatus` indicating success (`eDSNoErr`) or an error. A result code of `eDSBufferTooSmall` indicates you should allocate a larger buffer and call this function again. For a list of possible result codes, see "Result Codes" (page 169).

Discussion

This function fills a data buffer with the names of registered nodes. Call `dsGetDirNodeName` (page 58) to extract the names from the buffer.

On output, if `inOutContinueData` points to a value that is not `NULL`, there are more registered node names to get even if `outDirNodeCount` points to a zero value. To get another buffer of registered node names, call this function again and pass to it the context data pointed to by `inOutContinueData`.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`DirServices.h`

dsGetDirNodeName

Gets node names from a buffer.

```
tDirStatus dsGetDirNodeName (
   tDirReference inDirReference,
   tDataBufferPtr inOutDataBuffer,
   UInt32 inDirNodeIndex,
   tDataListPtr *inOutDataList
);
```

Parameters

inDirReference

On input, a value of type `tDirReference` (page 89) obtained by previously calling `dsOpenDirService` (page 69) or `dsOpenDirServiceProxy` (page 70).

inOutDataBuffer

On input, a value of type `tDataBufferPtr` (page 87) that points to a `tDataBuffer` (page 84) structure containing the results of calling `dsFindDirNodes` (page 49) or `dsGetDirNodeList` (page 57).

inDirNodeIndex

On input, a value of type `unsigned long`. Set `inDirNodeIndex` to 1 to get the first name. Set `inDirNodeIndex` to 2 to get the second name, and so on.

inOutDataList

On input, a value of type `tDataListPtr` pointing to a value that is `NULL` or that can be overwritten. On output, the data list contains the full pathname of the node specified by `inDirNodeIndex`. You can reuse the data list for other purposes, but when you no longer need the data list, call `dsDataListDeallocate` (page 27) to deallocate it. The data list is heap-based, you also need to call `free()`.

function result

A value of type `tDirStatus` indicating success (`eDSNoErr`) or an error. For a list of possible result codes, see "Result Codes" (page 169).

Discussion
This function parses a buffer of node names obtained by calling `dsFindDirNodes` (page 49) or `dsGetDirNodeList` (page 57).

The `inOutDataBuffer` parameter points to the data buffer that contains node names. The `inDirNodeIndex` parameter specifies which node name to get, and the `inOutDataList` parameter specifies the address of the application-defined `tDataList` (page 84) structure in which this function is to place the node name.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
NetworkAuthentication

Declared In
`DirServices.h`

dsGetPathFromList

Gets the path from a data list.

```
char * dsGetPathFromList (
   tDirReference inDirReference,
   const tDataList *inDataList,
   const char *inDelimiter
);
```

Parameters

`inDirReference`
: On input, a value of type `tDirReference` (page 89) obtained by calling `dsOpenDirService` (page 69) or `dsOpenDirServiceProxy` (page 70) representing the Open Directory session associated with the data list from which a path is to be obtained, or zero.

`inDataList`
: On input, a pointer to a value of type `tDataList` (page 84) containing the path to get.

`inDelimiter`
: On input, a pointer to a character string containing the character that delimits the components of the path in the data list pointed to by the `inDataList` parameter.

`function result`
: A pointer to a character string that contains the path that was obtained from the data list.

Discussion
This utility function gets the path from a data list. The path is in UTF-8 format.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`DirServicesUtils.h`

dsGetRecordAttributeInfo

Gets information about a record’s attribute.

```
tDirStatus dsGetRecordAttributeInfo (
   tRecordReference inRecordReference,
   tDataNodePtr inAttributeType,
   tAttributeEntryPtr *outAttributeInfoPtr
);
```

Parameters

`inRecordReference`

On input, a value of type tRecordReference (page 89) obtained by previously calling `dsOpenRecord` (page 71) representing the record for which the record's attribute type information is to be obtained.

`inAttributeType`

On input, a value of type `tDataNodePtr` (page 88) that points to a `tDataBuffer` (page 84) structure allocated by calling `dsDataBufferAllocate` (page 25) containing the attribute type for which information is to be obtained. Call `dsGetRecordList` (page 64) to find out the record's attribute types.

`outAttributeInfoPtr`

On output, a pointer to a value of type `tAttributeEntryPtr` (page 85) that points to an application-allocated `tAttributeEntry` (page 82) structure containing the information about the attribute pointed to by `inAttributeType`.

`function result`

A value of type `tDirStatus` indicating success (`eDSNoErr`) or an error. For a list of possible result codes, see "Result Codes" (page 169).

Discussion

This function gets information about a record's attribute. The information consists of the number of attribute values, data size, maximum value size, and signature.

When you no longer need the `outAttributeInfoPtr` parameter, call `dsDeallocAttributeEntry` (page 35) to deallocate the `tAttributeValueEntry` (page 83) structure and its pointer.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`DirServices.h`

dsGetRecordAttributeValueByID

Uses an attribute value ID to obtain the value of an attribute.

```
tDirStatus dsGetRecordAttributeValueByID (
   tRecordReference inRecordReference,
   tDataNodePtr inAttributeType,
   UInt32 inValueID,
   tAttributeValueEntryPtr *outEntryPtr
);
```

Parameters

`inRecordReference`

On input, a value of type tRecordReference (page 89) obtained by previously calling `dsOpenRecord` (page 71) representing the record that has an attribute whose value is to be obtained.

inAttributeType

On input, a value of type `tDataNodePtr` (page 88) pointing to a `tDataNode` (page 88) structure allocated by calling `dsDataNodeAllocateBlock` (page 31) or `dsDataNodeAllocateString` (page 32) that contains the type of the attribute whose value is to be obtained.

inValueID

On input, a value of type `unsigned long` containing the attribute value ID of the value to get. Call `dsGetAttributeEntry` (page 51) to get an attribute value ID.

outEntryPtr

On output, a pointer to a value of type `tAttributeValueEntryPtr` (page 86) that points to a `tAttributeValueEntry` (page 83) structure allocated by calling `dsAllocAttributeValueEntry` (page 14) containing the requested attribute value.

function result

A value of type `tDirStatus` indicating success (`eDSNoErr`) or an error. For a list of possible result codes, see "Result Codes" (page 169).

Discussion

This function uses an attribute value ID to obtain the value of an attribute for the record represented by `inRecordReference`.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`DirServices.h`

dsGetRecordAttributeValueByIndex

Uses an index to get the value of an attribute.

```
tDirStatus dsGetRecordAttributeValueByIndex (
   tRecordReference inRecordReference,
   tDataNodePtr inAttributeType,
   UInt32 inValueIndex,
   tAttributeValueEntryPtr *outEntryPtr
);
```

Parameters

inRecordReference

On input, a value of type tRecordReference (page 89) obtained by previously calling `dsOpenRecord` (page 71) representing the record that has an attribute whose value is to be obtained.

inAttributeType

On input, a value of type `tDataNodePtr` (page 88) that points to a `tDataNode` (page 88) structure allocated by calling `dsDataNodeAllocateBlock` (page 31) or `dsDataNodeAllocateString` (page 32) that contains the type of the attribute whose value is to be obtained.

inValueIndex

On input, a value of type `unsigned long` that specifies the index of the attribute value that is to be obtained. Call `dsGetRecordAttributeInfo` (page 59) to find out how many values the attribute has. Set inValueID to 1 to get the first value; set `inValueID` to 2 to get the second value, and so on.

outEntryPtr

On output, a value of type `tAttributeValueEntryPtr` that points to a `tAttributeValueEntry` (page 83) structure containing the requested attribute value.

function result

A value of type `tDirStatus` indicating success (`eDSNoErr`) or an error. For a list of possible result codes, see "Result Codes" (page 169).

Discussion

This function uses an index to obtain the value of an attribute for the record represented by `inRecordReference`.

To determine whether an attribute can have multiple values, call `dsGetRecordAttributeInfo` (page 59), which returns a value that points to the attribute's value count.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`DirServices.h`

dsGetRecordAttributeValueByValue

Verifies the existence of an attribute value within a record.

```
tDirStatus dsGetRecordAttributeValueByValue (
   tRecordReference inRecordReference,
   tDataNodePtr inAttributeType,
   tDataNodePtr inAttributeValue,
   tAttributeValueEntryPtr *outEntryPtr
);
```

Parameters

inRecordReference

On input, a value of type tRecordReference (page 89) obtained by previously calling `dsOpenRecord` (page 71) representing the record that has an attribute whose value is to be obtained.

inAttributeType

On input, a value of type `tDataNodePtr` (page 88) that points to a `tDataNode` (page 88) structure allocated by calling `dsDataNodeAllocateBlock` (page 31) or `dsDataNodeAllocateString` (page 32) that contains the type of the attribute whose value is to be obtained.

inValueValue

On input, a value of type `tDataNodePtr` (page 88) that specifies the value that is to be verified.

outEntryPtr

On output, a value of type `tAttributeValueEntryPtr` that points to a `tAttributeValueEntry` (page 83) structure containing the attribute value.

function result

A value of type `tDirStatus` indicating that the value was obtained (`eDSNoErr`); any value other than `eDSNoErr` indicates failure. For a list of possible result codes, see "Result Codes" (page 169).

Discussion

This function verifies the existence the specified attribute value for the record specified by `inRecordReference`. It also returns the value's ID, which is useful if you want to remove this value by calling `dsRemoveAttributeValue` (page 73) or change it by calling `dsSetAttributeValue` (page 74).

Version Notes

Introduced in Mac OS X v10.4.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`DirServices.h`

dsGetRecordEntry

Gets the next record from a data buffer.

```
tDirStatus dsGetRecordEntry (
   tDirNodeReference inDirNodeReference,
   tDataBufferPtr inOutDataBuffer,
   UInt32 inRecordEntryIndex,
   tAttributeListRef *outAttributeListRef,
   tRecordEntryPtr *outRecordEntryPtr
);
```

Parameters

inDirNodeReference
: On input, a value of type `tDirNodeReference` (page 89), obtained by calling `dsOpenDirNode` (page 68), that identifies the node in which the record specified by `inRecordEntryIndex` resides.

inOutDataBuffer
: On input, a value of type `tDataBufferPtr` (page 87) that points to a `tDataBuffer` (page 84) structure containing data obtained by previously calling `dsGetRecordList` (page 64), `dsDoAttributeValueSearch` (page 37), `dsDoAttributeValueSearchWithData` (page 40), `dsDoMultipleAttributeValueSearch` (page 45), or `dsDoMultipleAttributeValueSearchWithData` (page 47).

inRecordEntryIndex
: On input, a value of type `unsigned long` that specifies the next record to get. Set `inRecordEntryIndex` to 1 to get the first record. Set `inRecordEntryIndex` to 2 to get the second record, and so on.

outAttributeListRef
: On input, a pointer to a value of type `tAttributeListRef` (page 86). On output, to get information about the record's attributes, pass the value pointed to by `outAttributeListRef` as a parameter when calling `dsGetAttributeEntry` (page 51).

outRecordEntryPtr
: On output, `outRecordEntryPtr` points to a `tRecordEntry` (page 85) structure that contains the record entry specified by the `inRecordEntryIndex`. When you no longer need the record entry structure, call `dsDeallocRecordEntry` (page 36).

function result
: A value of type `tDirStatus` indicating success (`eDSNoErr`) or an error. For a list of possible result codes, see "Result Codes" (page 169).

Discussion
This function gets the name, type and number of attribute types for a record from the data buffer pointed to by `inOutDataBuffer` and puts it in the `tRecordEntry` (page 85) structure pointed to by `outRecordEntryPtr`.

This function also returns a pointer to an attribute list reference that can be used to get information about a record's attributes by calling `dsGetAttributeEntry` (page 51). Calling `dsGetAttributeEntry` returns an attribute value list reference that can be used to call `dsGetAttributeValue` (page 53) to get the value of an attribute.

You should call `dsCloseAttributeList` (page 20) when you no longer need the attribute list reference pointed to by `outAttributeListRef`. You should call `dsDeallocRecordEntry` (page 36) when you no longer need the record entry structure pointed to by `outRecordEntryPtr`.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
NetworkAuthentication

Declared In
`DirServices.h`

dsGetRecordList

Gets a list of records and puts it in a data buffer.

```
tDirStatus dsGetRecordList (
   tDirNodeReference inDirNodeReference,
   tDataBufferPtr inOutDataBuffer,
   tDataListPtr inRecordNameList,
   tDirPatternMatch inPatternMatchType,
   tDataListPtr inRecordTypeList,
   tDataListPtr inAttributeTypeList,
   dsBool inAttributeInfoOnly,
   UInt32 *inOutRecordEntryCount,
   tContextData *inOutContinueData
);
```

Parameters

inDirNodeReference
On input, a value of type `tDirNodeReference` (page 89) obtained by calling `dsOpenDirNode` (page 68).

inOutDataBuffer
On input, a value of type `tDataBufferPtr` (page 87) created by calling `dsDataBufferAllocate` (page 25) that points to a `tDataBuffer` (page 84) structure into which this function is to place the requested list of records. If `inOutRecordEntryCount` points to a positive value greater than zero when this function returns, parse the records in the buffer pointed to by `inOutDataBuffer` by calling `dsGetRecordEntry` (page 63).

inRecordNameList
On input, a value of type `tDataListPtr` (page 87) specifying the record names to get.

inPatternMatchType
On input, a value of type `tDirPatternMatch` specifying a pattern type that controls the way in which the pattern specified by `inRecordNameList` is compared with record names. See Pattern Matching Constants (page 137) for possible values. The pattern type may also be defined by the Open Directory plug-in that handles the directory service represented by `inDirNodeReference`. The `inPatternMatchType` parameter is ignored if `inRecordNameList` is set to get all records.

inRecordTypeList
: On input, a value of type `tDataListPtr` pointing to an `tDataList` (page 84) structure containing the types of records to get. One way to allocate the data list is to call `dsDataListAllocate` (page 26).

inAttributeTypeList
: On input, a value of type `tDataListPtr` pointing to a `tDataList` (page 84) structure that contains the attribute types of the records that are to be obtained. If you want all attribute types, create the data list using `kDSAttributesAll`. To get all standard attribute types, create the data list using `kDSAttributesStandardAll`. To get all native attribute types, create the data list using `kDSAttributesNativeAll`.

inAttributeInfoOnly
: On input, a value of type `dsBool`. Set `inAttributeInfoOnly` to `TRUE` if you only want attribute type information. To get attribute type information as well as attribute values, set `inAttributeInfoOnly` to `FALSE`.

inOutRecordEntryCount
: On input, a pointer to a value of type `unsigned long` that specifies the total number of records to get across what may be multiple calls to this function in order to get the complete list of records, or zero if you want to get all matching records. On output, `inOutRecordEntryCount` points to the number of records this function has stored in the data buffer pointed to by `inOutDataBufferPtr`. Once you start a series of `dsGetRecordList` calls, `inOutMatchRecordCount` is ignored as an input parameter.

inOutContinueData
: On input, a pointer to a value of type `tContextData` (page 87) and set to `NULL`. On output, if the value pointed to by `inOutContinueData` is `NULL`, there are no more records to get. On output, if the value pointed to by `inOutContinueData` is not `NULL`, fill the data buffer pointed to by `inOutDataBuffer` with the next records by calling this function again and passing the context data pointed to by `inOutContinueData`. If you don't want to get the remaining records, you must call `dsReleaseContinueData` (page 72) to deallocate the memory pointed to by `inOutContinueData`.

function result
: A value of type `tDirStatus` indicating success (`eDSNoErr`) or an error. An result code of `eDSBufferTooSmall` occurs if the buffer is too small to fit the first record to be returned. In this case, call this function again after allocating a larger buffer. For a list of other possible result codes, see "Result Codes" (page 169).

Discussion

This function stores in the data buffer pointed to by `inOutDataBuffer` a list of records having the specified data types and values. Call `dsGetRecordEntry` (page 63) to parse the records in the buffer.

Set `inOutRecordEntryCount` to point to a positive integer value that represents the number of records that are to be returned. You cannot change the value pointed to by `inOutRecordEntryCount` if you call this function with `inOutContinueData` pointing to context data returned by a previous call to this function.

If there are too many records to fit in the data buffer pointed to by `inOutDataBuffer`, `inOutContinueData` points to a non-null value when this function returns. To get more records, call this function again, passing the pointer to the `inOutContinueData` parameter that was returned by the previous call to this function. To get all records, continue calling this function until `inOutContinueData` points to a null value.

If the value pointed to by `inOutContinueData` is not `NULL` and you do not want to get more records, call `dsReleaseContinueData` (page 72) to release the memory associated with `inOutContinueData`.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code
NetworkAuthentication

Declared In
`DirServices.h`

dsGetRecordNameFromEntry

Gets the name of a record from a record entry structure.

```
tDirStatus dsGetRecordNameFromEntry (
   tRecordEntryPtr inRecEntryPtr,
   char **outRecName
);
```

Parameters

`inRecEntryPtr`
: On input, a value of type `tRecordEntryPtr` (page 89) that points to the `tRecordEntry` (page 85) structure that contains the name that is to be obtained.

`outRecName`
: On output, a pointer to a value that points to a character string containing the record's name in UTF-8 format.

`function result`
: A value of type `tDirStatus` indicating success (`eDSNoErr`) or an error. For a list of possible result codes, see "Result Codes" (page 169).

Discussion
This utility function gets the name of the record in a record entry structure.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`DirServicesUtils.h`

dsGetRecordReferenceInfo

Gets a record's name and type and the number of attributes the record has.

```
tDirStatus dsGetRecordReferenceInfo (
   tRecordReference inRecordReference,
   tRecordEntryPtr *outRecordInfo
);
```

Parameters

`inRecordReference`
: On input, value of type tRecordReference (page 89) obtained by previously calling `dsOpenRecord` (page 71) that represents the record reference information is to be obtained.

`outRecordInfo`
: On output, a pointer to a value of type `tRecordEntryPtr` (page 89) that points to a `tRecordEntry` (page 85) structure containing the record information for the specified record. Call `dsDeallocRecordEntry` (page 36) when you no longer need the record entry structure.

function result

A value of type `tDirStatus` indicating success (`eDSNoErr`) or an error. For a list of possible result codes, see "Result Codes" (page 169).

Discussion

This function gets information about the record represented by `inRecordReference` and stores it in the `tRecordEntry` (page 85) structure pointed to by `outRecordInfo`.

The information includes the number of attributes the record has and the name and type of the record.

You should call `dsDeallocRecordEntry` (page 36) when you no longer need the record entry structure pointed to by `outRecordInfo`.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`DirServices.h`

dsGetRecordTypeFromEntry

Gets the type of a record from a record entry structure.

```
tDirStatus dsGetRecordTypeFromEntry (
   tRecordEntryPtr inRecEntryPtr,
   char **outRecType
);
```

Parameters

inRecEntryPtr

On input, a value of type `tRecordEntryPtr` (page 89) that points to the `tRecordEntry` (page 85) structure that contains the type that is to be obtained.

outRecType

On output, a pointer to a value that points to a character string containing the record's type.

function result

A value of type `tDirStatus` indicating success (`eDSNoErr`) or an error. For a list of possible result codes, see "Result Codes" (page 169).

Discussion

This utility function gets the type of the record in a record entry structure.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`DirServicesUtils.h`

dsIsDirServiceRunning

Checks to see if Open Directory is running.

```
tDirStatus dsIsDirServiceRunning
```

Parameters

function result

A value of type `tDirStatus` indicating that Open Directory is running (`eDSNoErr`) or that Open Directory is not running (`eServerNotRunning`).

Discussion

This function checks to see if Open Directory is running.

Prior to Mac OS X 10.2, Open Directory did not start until when an application called `dsOpenDirService` (page 69). With Mac OS X 10.2, Open Directory starts up when the system starts up and is always running.

Applications that run on Mac OS X 10.0 and Mac OS X 10.1 should call this function before calling `dsOpenDirService` (page 69). If Open Directory is not running, you can display a progress indicator to assure the user that your application is still running while Open Directory starts up, and then call `dsOpenDirService`.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`DirServices.h`

dsOpenDirNode

Opens a session with a node.

```
tDirStatus dsOpenDirNode (
   tDirReference inDirReference,
   tDataListPtr inDirNodeName,
   tDirNodeReference *outDirNodeReference
);
```

Parameters

inDirReference

On input, a value of type `tDirReference` (page 89) obtained by previously calling `dsOpenDirService` (page 69) or `dsOpenDirServiceProxy` (page 70).

inDirNodeName

On input, a value of type `tDataListPtr` that points to a `tDataList` (page 84) structure containing the name of the node to open. You can get the name of the node by calling `dsGetDirNodeList` (page 57) or by calling, for example, `dsBuildListFromStrings` (page 18) to construct the name yourself.

outDirNodeReference

On input, a pointer to a value of type `tDirNodeReference` (page 89). On output, the value pointed to by `outDirNodeReference` is a node reference that represents the session context for the contents of the opened node. Provide it as a parameter to Open Directory functions that manipulate nodes, such as `dsGetDirNodeInfo` (page 56), `dsDoDirNodeAuth` (page 41), `dsGetRecordList` (page 64), `dsGetRecordEntry` (page 63), `dsOpenRecord` (page 71), and `dsGetAttributeEntry` (page 51).

function result

A value of type `tDirStatus` indicating success (`eDSNoErr`) or an error. For a list of possible result codes, see "Result Codes" (page 169).

Discussion
This function opens a session with the node whose name is specified in the `tDataList` (page 84) structure pointed to by the `inDirNodeName` parameter. Opening a session with a node allows you to perform operations on the opened node, such as creating, listing, and deleting records.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
NetworkAuthentication

Declared In
`DirServices.h`

dsOpenDirService

Opens an Open Directory session.

```
tDirStatus dsOpenDirService (
   tDirReference *outDirReference
);
```

Parameters

`outDirReference`

On input, a pointer to a value of type `tDirReference` (page 89). On output, the value pointed to by `outDirReference` identifies this session and is passed as a parameter to many Open Directory functions.

`function result`

A value of type `tDirStatus` indicating success (`eDSNoErr`) or an error. For a list of possible result codes, see "Result Codes" (page 169).

Discussion
When this function returns, use the value pointed to by `outDirReference` when calling Open Directory functions that require an Open Directory reference as a parameter.

With Mac OS X 10.2, Open Directory starts up when the system starts up and is always running. Prior to Mac OS X 10.2, the DirectoryService daemon was not started until an application called this function for the first time. Applications that run on Mac OS X 10.0 and Mac OS X 10.1 should call `dsIsDirServiceRunning` (page 67) to learn whether Open Directory is running. If it's not running, you can display a progress indicator while Open Directory starts up and then call this function.

Special Considerations

You can establish multiple Open Directory sessions by calling this function multiple times.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
NetworkAuthentication

Declared In
`DirServices.h`

dsOpenDirServiceProxy

Opens a remote Open Directory session.

```
tDirStatus dsOpenDirServiceProxy (
   tDirReference *outDirRef,
   const char *inIPAddress,
   UInt32 inIPPort,
   tDataNodePtr inAuthMethod,
   tDataBufferPtr inAuthStepData,
   tDataBufferPtr outAuthStepDataResponse,
   tContextData *ioContinueData
);
```

Parameters

outDirRef

On input, a pointer to a value of type `tDirReference` (page 89). On output, the value pointed to by `outDirReference` identifies this session and is passed as a parameter to many Open Directory functions.

inIPAddress

On input, a pointer to a null-terminated string contain the fully qualified domain name or the IP address in dotted decimal format of the Open Directory machine with which a TCP/IP connection is to be established.

inIPPort

On input, a value of type `unsigned long` containing the port number on which the connection is to be made, or zero which allows the default port number to be used. The default port number is 625.

inAuthMethod

On input, a value of type `tDataNodePtr` (page 88) pointing to a `tDataNode` (page 88) structure allocated by calling `dsDataNodeAllocateBlock` (page 31) or `dsDataNodeAllocateString` (page 32) containing the authentication method to use. You can expect these authentication methods to be supported by any plug-in that handles authentication: `kDSStdAuthNodeNativeNoClearText` and `kDSStdAuthNodeNativeClearTextOK`. For other possible values, see Authentication Constants (page 129).

inAuthStepData

On input, a value of type `tDataBufferPtr` (page 87) created by calling `dsDataBufferAllocate` (page 25) pointing to a `tDataBuffer` (page 84) structure that contains the data necessary for this step in the authentication process. For the first step in the authentication process, `inAuthStepData` typically consists of four bytes specifying the length of a username, followed by the user name in UTF-8 encoding, followed by four bytes specifying the length of the password, followed by the password in UTF-8 encoding.

outAuthStepDataResponse

On output, a value of type `tDataBufferPtr` (page 87) created by calling `dsDataBufferAllocate` (page 25) pointing to a `tDataBuffer` (page 84) structure that contains the authentication response.

inOutContinueData

On input, a pointer to a value of type `tContextData` (page 87) and set to `NULL`. On output, if the value pointed to by `inOutContinueData` is `NULL`, there are no more steps in the authentication process. If `inOutContinueData` is not `NULL` on output, there are more steps to complete. Call this function again and pass to it the value pointed to by `inOutContinueData`. Call `dsReleaseContinueData` (page 72) if the value pointed to by `inOutContinueData` is not `NULL` and you do not want to complete the authentication process.

function result

A value of type `tDirStatus` indicating success (`eDSNoErr`) or an error. For a list of possible result codes, see "Result Codes" (page 169).

Discussion

This function opens a remote Open Directory session on another machine running Mac OS X 10.2 or later. When the authentication process completes successfully, this function returns an Open Directory reference that can be used for all subsequent calls to Open Directory functions, such as `dsOpenDirNode` (page 68) and `dsFindDirNodes` (page 49), on the remote machine. These calls will be dispatched automatically over the TCP connection to the remote DirectoryService daemon. Any calls using child references obtained by calling functions such as `dsFindDirNodes` will also be sent to the remote DirectoryService daemon.

Special Considerations

You can establish multiple remote Open Directory sessions by calling this function multiple times.

Version Notes

Available in Mac OS X v10.2 and later.

Availability

Available in Mac OS X v10.2 and later.

Declared In

`DirServices.h`

dsOpenRecord

Opens a record.

```
tDirStatus dsOpenRecord (
   tDirNodeReference inDirNodeReference,
   tDataNodePtr inRecordType,
   tDataNodePtr inRecordName,
   tRecordReference *outRecordReference
);
```

Parameters

inDirNodeReference

On input, a node reference obtained by calling `dsOpenDirNode` (page 68).

inRecordType

On input, a value of type `tDataNodePtr` (page 88) that points to a `tDataBuffer` (page 84) structure allocated by calling `dsDataBufferAllocate` (page 25) containing the type of the record to open. For possible values, see Standard Record Types (page 143).

inRecordName

On input, a value of type `tDataNodePtr` (page 88) that points to a `tDataBuffer` (page 84) structure allocated by calling `dsDataBufferAllocate` (page 25) containing the name in UTF-8 format of the record to open.

outRecordReference

On output, a pointer to a value of type tRecordReference (page 89) that you can pass to other Open Directory functions that operate on records, such as `dsGetRecordReferenceInfo` (page 66), `dsFlushRecord` (page 51), `dsSetRecordName` (page 76), and `dsCloseRecord` (page 22).

function result

A value of type `tDirStatus` indicating success (`eDSNoErr`) or an error. For a list of possible result codes, see "Result Codes" (page 169).

Discussion

This function opens a record and returns in the value pointed to by the `outRecordReference` parameter a record reference that you can use in subsequent calls to Open Directory functions that manipulate records.

A record must be open before you can perform operations on the record, such as setting its name, adding attributes, setting attribute values, and deleting the record.

To close an open record, call `dsCloseRecord` (page 22).

Availability

Available in Mac OS X v10.0 and later.

Declared In

`DirServices.h`

dsReleaseContinueData

Releases memory allocated for continuation data.

```
tDirStatus dsReleaseContinueData (
   tDirReference inDirReference,
   tContextData inContinueData
);
```

Parameters

inDirReference

On input, a value of type `tDirReference` (page 89) if the `inContinueData` parameter was generated by, for example, `dsGetRecordList` (page 64), `dsGetDirNodeInfo` (page 56), `dsDoAttributeValueSearch` (page 37), or `dsDoAttributeValueSearchWithData` (page 40). This parameter is a value of type `tDirReference` (page 89) if the `inContinueData` parameter was generated by, for example, `dsGetDirNodeList` (page 57) or `dsFindDirNodes` (page 49).

inContinueData

On input, a value of type `tContextData` (page 87) that is to be released.

function result

A value of type `tDirStatus` indicating success (`eDSNoErr`) or an error. For a list of possible result codes, see "Result Codes" (page 169).

Discussion

This function releases the memory allocated for continuation data. Continuation data is returned when any Open Directory function cannot return all of the requested information in one call, such as `dsDoDirNodeAuth` (page 41), `dsFindDirNodes` (page 49), `dsGetDirNodeInfo` (page 56), `dsGetDirNodeList` (page 57), `dsGetRecordList` (page 64), `dsDoAttributeValueSearch` (page 37), or `dsDoAttributeValueSearchWithData` (page 40).

If your application does not call again the function that returned the continuation data and provide to it the continuation data, your application should call `dsReleaseContinueData` (page 72) to free the memory allocated to the continuation data.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code
NetworkAuthentication

Declared In
`DirServices.h`

dsRemoveAttribute

Removes an attribute from a record.

```
tDirStatus dsRemoveAttribute (
   tRecordReference inRecordReference,
   tDataNodePtr inAttribute
);
```

Parameters

`inRecordReference`
: On input, a value of type tRecordReference (page 89) obtained by previously calling `dsOpenRecord` (page 71).

`inAttribute`
: On input, a value of type `tDataNodePtr` (page 88) that points to a `tDataNode` (page 88) structure allocated by calling `dsDataNodeAllocateBlock` (page 31) or `dsDataNodeAllocateString` (page 32) that contains the name of the attribute that is to be removed.

`function result`
: A value of type `tDirStatus` indicating success (`eDSNoErr`) or an error. For a list of possible result codes, see "Result Codes" (page 169).

Discussion
This function removes the specified attribute from the specified record. Any attribute value list references, attribute value entry structures, and attribute entry structures that have been created for this attribute are still valid because they use a buffer that has already been filled with data. Calling `dsGetRecordAttributeValueByID` (page 60), `dsGetRecordAttributeValueByIndex` (page 61), or `dsGetRecordAttributeValueByValue` (page 62) after the attribute has been removed generates an error because the attribute no longer exists.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`DirServices.h`

dsRemoveAttributeValue

Removes an attribute value.

```
tDirStatus dsRemoveAttributeValue (
   tRecordReference inRecordReference,
   tDataNodePtr inAttributeType,
   UInt32 inAttributeValueID
);
```

Parameters

inRecordReference

On input, a value of type tRecordReference (page 89) obtained by previously calling `dsOpenRecord` (page 71) that represents the record having an attribute whose value is to be removed.

inAttributeType

On input, a value of type `tDataNodePtr` (page 88) that points to a `tDataNode` (page 88) structure allocated by calling `dsDataNodeAllocateBlock` (page 31) or `dsDataNodeAllocateString` (page 32) that contains the type of the attribute whose value is to be removed.

inAttributeValueID

On input, a value of type `unsigned long` that specifies the attribute value ID of the attribute whose value is to be removed. Call `dsGetAttributeValue` (page 53) to get the attribute value ID.

function result

A value of type `tDirStatus` indicating success (`eDSNoErr`) or an error. For a list of possible result codes, see "Result Codes" (page 169).

Discussion

This function removes the value of the attribute that has the type specified by the data node pointed to by `inAttributeType` and the attribute value ID specified by `inAttributeValueID` for the record represented by `inRecordReference`. Any attribute value list references, attribute value entry structures, and attribute entry structures that have been created for the removed attribute are still valid because they manage offsets into a buffer that already contains data. Calling `dsGetRecordAttributeValueByID` (page 60), `dsGetRecordAttributeValueByIndex` (page 61), or `dsGetRecordAttributeValueByValue` (page 62) after the attribute has been removed generates an error because the attribute no longer exists.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`DirServices.h`

dsSetAttributeValue

Sets the value of an attribute.

```
tDirStatus dsSetAttributeValue (
   tRecordReference inRecordReference,
   tDataNodePtr inAttributeType,
   tAttributeValueEntryPtr inAttributeValuePtr
);
```

Parameters

inRecordReference

On input, value of type tRecordReference (page 89) obtained by previously calling `dsOpenRecord` (page 71) representing the record that has an attribute whose value is to be set.

inAttributeType

On input, a value of type `tDataNodePtr` (page 88) that points to a `tDataNode` (page 88) structure allocated by calling `dsDataNodeAllocateBlock` (page 31) or `dsDataNodeAllocateString` (page 32) that contains the type of the attribute whose value is to be set.

inAttributeValuePtr

On input, a value of type `tAttributeValueEntryPtr` (page 86) that points to a `tAttributeValueEntry` (page 83) structure created by calling `dsAllocAttributeValueEntry` (page 14) that contains the value that is to be set and its attribute value ID. The attribute value ID is the ID of an existing value of this attribute for the record specified by `inRecordReference` obtained by calling `dsGetRecordAttributeValueByIndex` (page 61), `dsGetRecordAttributeValueByValue` (page 62), or `dsGetAttributeValue` (page 53).

function result

A value of type `tDirStatus` indicating success (`eDSNoErr`) or an error. For a list of possible result codes, see "Result Codes" (page 169).

Discussion

This function uses an attribute value ID to replace an existing attribute value with a new value. If the attribute is a multi-value attribute, this function sets only the value of the attribute specified by the attribute value ID without affecting any other values. Unlike `dsSetAttributeValues`, this function does not create an attribute if it does not already exist.

The `inAttributeType` parameter points to a data node that specifies the type of the attribute whose value is to be set, and `inAttributeValuePtr` points to an attribute entry structure that contains the value that is to be set and the value's attribute value ID.

When you no longer need `inAttributeValuePtr`, you should call `dsDeallocAttributeValueEntry` (page 35) to release the memory associated with it.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`DirServices.h`

dsSetAttributeValues

Sets multiple values for an attribute.

```
tDirStatus dsSetAttributeValues (
   tRecordReference inRecordReference,
   tDataNodePtr inAttributeType,
   tDataListPtr inAttributeValuesPtr
);
```

Parameters

inRecordReference

On input, value of type tRecordReference (page 89) obtained by previously calling `dsOpenRecord` (page 71) representing the record whose values are to be set.

inAttributeType

On input, a value of type `tDataNodePtr` (page 88) that points to a `tDataNode` (page 88) structure allocated by calling `dsDataNodeAllocateBlock` (page 31) or `dsDataNodeAllocateString` (page 32) that contains the type of the attribute whose values are to be set.

inAttributeValuesPtr
: On input, a value of type `tDataListPtr` (page 87) that points to the list of values that are to be set.

function result
: A value of type `tDirStatus` indicating success (`eDSNoErr`) or an error. For a list of possible result codes, see "Result Codes" (page 169).

Discussion
This function sets multiple values for the attribute specified by `inAttributeType` for the record specified by `inRecordReference`. This function replaces all of the values of the attribute with a new list of values. If the attribute does not exist, it is created with the specified list of values.

The `inAttributeType` parameter points to a data node that specifies the type of the attribute whose values are to be set, and `inAttributeValuesPtr` points to a list of values that are to be set.

You should call `dsDeallocAttributeValueEntry` (page 35) to release the memory associated with `inAttributeValuesPtr` when you no longer need it.

When you no longer need it, call `dsDataListDeallocate` (page 27) to release the memory associated with `inAttributeValuesPtr`. If `inAttributeValuesPtr` is heap-based, you also need to call `free()`.

Version Notes
Introduced in Mac OS X v10.4.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`DirServices.h`

dsSetRecordName

Sets the name of a record.

```
tDirStatus dsSetRecordName (
   tRecordReference inRecordReference,
   tDataNodePtr inNewRecordName
);
```

Parameters

inRecordReference
: On input, value of type tRecordReference (page 89) obtained by previously calling `dsOpenRecord` (page 71) or `dsCreateRecordAndOpen` (page 24) that represents the record whose name is to be set.

inNewRecordName
: On input, a value of type `tDataNodePtr` (page 88) that points to a `tDataNode` (page 88) structure allocated by calling `dsDataNodeAllocateBlock` (page 31) or `dsDataNodeAllocateString` (page 32) containing the record name in UTF-8 format that is to be set.

function result
: A value of type `tDirStatus` indicating success (`eDSNoErr`) or an error. For a list of possible result codes, see "Result Codes" (page 169).

Discussion
This function sets the name of a record.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`DirServices.h`

dsSetRecordType

Sets the type of a record.

```
tDirStatus dsSetRecordType (
   tRecordReference inRecordReference,
   tDataNodePtr inNewRecordType
);
```

Parameters

`inRecordReference`

On input, value of type tRecordReference (page 89) obtained by previously calling `dsOpenRecord` (page 71) that represents the record whose type is to be set.

`inNewRecordType`

On input, a value of type `tDataNodePtr` (page 88) that points to a `tDataNode` (page 88) structure allocated by calling `dsDataNodeAllocateBlock` (page 31) or `dsDataNodeAllocateString` (page 32) containing the record type that is to be set.

`function result`

A value of type `tDirStatus` indicating success (`eDSNoErr`) or an error. For a list of possible result codes, see "Result Codes" (page 169).

Discussion
This deprecated function sets a record's type. For record type constants, see Pattern Matching Constants (page 137). Not all plug-ins support setting a record's type.

Version Notes
Deprecated in Mac OS X v10.2.

Availability
Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.2.

Declared In
`DirServices.h`

dsVerifyDirRefNum

Verifies that a `tDirReference` is valid.

```
tDirStatus dsVerifyDirRefNum (
   tDirReference inDirReference
);
```

Parameters

`inRecordReference`

On input, the `tDirReference` that is to be verified.

function result

A value of type `tDirStatus` indicating that the `tDirReference is valid` (`eDSNoErr`) or an error. For a list of possible result codes, see "Result Codes" (page 169).

Discussion

This function verifies that a `tDirReference` is valid.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`DirServices.h`

Initialize

Initializes the plug-in.

```
ComponentResult ADD_IMAGECODEC_BASENAME() Initialize
```

Parameters

return result

Value of type `long`. If the `Initialize` routine completes successfully, it should return `eDSNoErr`. If it encounters an error, it should return `ePlugInInitError`.

Discussion

The DirectoryService daemon calls a plug-in's Initialize entry point so that the plug-in can initialize and prepare itself to run. The plug-in might, for example, open network ports and any files it requires.

An Open Directory plug-in's `Initialize` routine is called only once after all Open Directory plug-ins that can be loaded are loaded. If the plug-in cannot initialize itself and returns `ePlugInInitError`, the plug-in remains in the "failed to init" state.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCodec.k.h`

PeriodicTask

Performs a periodic task.

```
long PeriodicTask (void);
```

Parameters

return result

Value of type `long`. If the `PeriodicTask` routine completes successfully, it should return `eDSNoErr`. If it encounters an error, it should return `ePlugInPeriodicTaskError`.

Discussion

The DirectoryService daemon calls a plug-in's PeriodicTask entry point every two minutes. The plug-in can use its `PeriodicTask` routine to perform tasks that need to be performed on a recurring basis. If a plug-in has no tasks for its `PeriodicTask` routine to perform, the `PeriodicTask` routine should immediately return a result code of `eDSNoErr`.

Plug-ins that do not implement their own thread management may want to use the `PeriodicTask` routine to perform a task on a regular basis.

For another way of setting timers, see `sHeader` (page 121).

ProcessRequest

Processes requests.

```
long ProcessRequest (void *inData);
```

Parameters

`inData`

Pointer to an arbitrary value containing the request that is to be processed.

`return result`

Value of type `long`. If the `ProcessRequest` routine completes successfully, it should return `eDSNoErr`. If it encounters an error, it should return an appropriate result code as described in "Result Codes" (page 169).

Discussion

The DirectoryService daemon calls a plug-in's `ProcessRequest` entry point when Open Directory client applications make requests for directory service. The `inData` parameter points to the request, which consists of a structure whose first byte always identifies the type of request. The remaining fields of the structure vary depending on the request type.

SetPluginState

Sets the plug-in's state.

```
long SetPluginState (unsigned long inNewState);
```

Parameters

`inNewState`

A value of type `unsigned long` that describes the plug-in's new state. See the ePluginState Constants (page 168) enumeration for appropriate values.

`return result`

Value of type `long`. If the `SetPluginState` routine completes successfully, it should return `eDSNoErr`. If it encounters an error, it should return an appropriate result code as described in "Result Codes" (page 169).

Discussion

The DirectoryService daemon calls a plug-in's `SetPluginState` entry point to inform the plug-in that its state has changed to the state specified by the `inNewState` parameter.

Shutdown

Prepares the plug-in for shut down.

```
long Shutdown (void);
```

Parameters

return result

Value of type `long`. If the `Shutdown` routine completes successfully, it should return `eDSNoErr`. If it encounters an error, it should return an appropriate result code as described in "Result Codes" (page 169).

Discussion

The DirectoryService daemon calls a plug-in's `Shutdown` entry point so that the plug-in can prepare itself for shut down. The plug-in should close any files that it opened, close network connections that it opened, and deallocate memory that it allocated for its use while it was running.

Validate

Validates the plug-in.

```
ComponentResult ADD_GRAPHICSIMPORT_BASENAME() Validate
```

Parameters

inSignature

Value of type `unsigned long` that uniquely identifies the plug-in.

return result

Value of type `long`. If the `Validate` routine completes successfully, it should return `eDSNoErr`. If it encounters an error, it should return an appropriate result code as described in "Result Codes" (page 169).

Discussion

The DirectoryService daemon calls a plug-in's Validate routine after the plug-in loads in order to pass to the plug-in a unique signature. The plug-in uses the signature to identify itself when it calls any of the Open Directory callback routines, which are described in the section PeriodicTask (page 78).

Availability

Available in Mac OS X v10.1 and later.

Declared In

`ImageCompression.k.h`

Callbacks

This section describes Open Directory callback routines that Open Directory plug-ins can call.

DSDebugLog

Writes information in the log file.

```
sInt32 DSDebugLog (
   const char *inFormat,
   va_list inArgs);
```

Parameters

inFormat
: Pointer to a character array that specifies the format that is to be used to write the data. For additional information, see `sprintf`(3).

inArgs
: Value of type `va_list` that specifies the values that are to be written in the format specified by `inFormat`.

return result
: Value of type `sInt32`. If the `DSDebugLog` callback routine completes successfully, it returns `eDSNoErr`. If the `DSDebugLog` callback routine cannot complete successfully, it returns an appropriate result code as described in "Result Codes" (page 169).

Discussion
The `DSDebugLog` callback routine writes the data specified by `inArgs` using the format specified by `inFormat` in the Open Directory log file, `/Library/Logs/DirectoryService/DirectoryService.debug.log`.

A wrapper function, `CShared:LogIt`, is also available writing to the log file. See the sample code for an example of its use.

DSRegisterNode

Registers a node.

```
long DSRegisterNode (
   unsigned long inSignature,
   tDataList *inNode,
   eDirNodeType inNodeType);
```

Parameters

inSignature
: Value of type `unsigned long` obtained by the plug-in when its `Validate` (page 80) routine was called and that uniquely identifies the plug-in.

inNode
: Pointer to a value of type `tDataList` (page 84) that specifies the name of the node that is to be registered.

inNodeType
: Value of type `eDirNodeType` that specifies the type of the node that is to be registered. See the Discussion section below for possible values.

return result
: Value of type `unsigned long`. If the `DSRegisterNode` callback routine completes successfully, it returns `eDSNoErr`. If the `DSRegisterNode` callback routine cannot complete successfully (for example, if the specified node is already registered or if `inNode` contains a node name that has invalid characters), it returns an appropriate result code as described in "Result Codes" (page 169).

Discussion
The `DSRegisterNode` callback routine registers the specified node.

The `eDirNodeType` enumeration defines values for the `inNodeType` parameter:

```
typedef enum {     kUnknownNodeType= 0x00000000,     kDirNodeType    = 0x00000001,
     kLocalNodeType  = 0x00000002 } eDirNodeType;
```

The local node (`kLocalNodeType`) is queried by default in response to an Open Directory request and is always the first node that is queried. Only one node can be registered as the local node at any one time. A directory node (`kDirNodeType`) is any other node that is to be registered for Open Directory.

Registrations are valid for the period of time that Open Directory is running. If Open Directory stops and is started again, the node must be registered again.

The plug-in is responsible for keeping the list of registered nodes accurate. It can use the `PeriodicTask` (page 78) entry point to update the list on a regular basis.

DSUnregisterNode

Unregisters a node.

```
long DSUnregisterNode (
   unsigned long inSignature,
   tDataList *inNode);
```

Parameters

inSignature
: Value of type `unsigned long` obtained by the plug-in when its `Validate` (page 80) routine was called and that uniquely identifies the plug-in.

inNode
: Pointer to a value of type `tDataList` (page 84) that specifies the name of the node that is to be unregistered.

result
: Value of type `unsigned long`. If the `DSUnregisterNode` callback routine completes successfully, it returns `eDSNoErr`. If the `DSUnregisterNode` callback routine cannot complete successfully, it returns an error.

Discussion
The `DSUnregisterNode` callback routine unregisters the specified node.

Data Types

Open Directory Structures

This section describes structures used by the Open Directory client.

tAttributeEntry

A structure used to store information about an attribute.

```
typedef struct
{
    unsigned long fReserved1;
    tAccessControlEntry fReserved2;
    unsigned long fAttributeValueCount;
    unsigned long fAttributeDataSize;
    unsigned long fAttributeValueMaxSize;
    tDataNode fAttributeSignature;
} tAttributeEntry;
typedef tAttributeEntry *tAttributeEntryPtr;
```

Fields

`fReserved1`
    Reserved.

`fReserved2`
    Reserved.

`fAttributeValueCount`
    Number of values associated with this attribute.

`fAttributeDataSize`
    Total byte count of all attribute values.

`fAttributeValueMaxSize`
    Maximum size of a value of this attribute type.

`fAttributeSignature`
    Byte sequence that uniquely represents this attribute type. The byte sequence is typically a collection of Unicode characters.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`DirServicesTypes.h`

tAttributeValueEntry

A structure used to get and set the value of an attribute by attribute value ID.

```
typedef struct
{
    unsigned long fAttributeValueID;
    tDataNode fAttributeValueData;
} tAttributeValueEntry;
```

Fields

`fAttributeValueID`
    Unique ID for this attribute value.

`fAttributeValueData`
    Value of type `tDataNode` (page 88) containing the value of this attribute.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`DirServicesTypes.h`

tDataBuffer

A structure that provides a standard format for passing information between Open Directory and applications.

```
typedef struct
{
    unsigned long fBufferSize;
    unsigned long fBufferLength;
    char fBufferData[1];
} tDataBuffer;
```

Fields

`fBufferSize`

Number of bytes allocated for this structure. The value of `fBufferSize` should be set when `tDataBuffer` is created.

`fBufferLength`

Number of meaningful bytes in `fBufferData`. You should call `dsDataNodeSetLength` (page 34) to adjust this value each time you change the value of the `fBufferData` field.

`fBufferData`

Array of characters.

Discussion

A `tDataBuffer` structure is typically used to exchange strings, node names, and attribute types. Call `dsDataBufferAllocate` (page 25) to allocate a data buffer. Call `dsDataBufferDeAllocate` (page 25) to release the memory associated with a data buffer when it is no longer needed.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`DirServicesTypes.h`

tDataList

An ordered list of `tDataNode` structures.

```
typedef struct
{
    unsigned long fDataNodeCount;
    tDataNodePtr fDataListHead;
} tDataList;
```

Fields

`fDataNodeCount`

Number of data nodes in this data list structure.

`fDataListHead`

First pointer to a data node in this data list structure.

Discussion

This structure is used to represent lists of items, such as nodes, full pathnames, attribute type lists, and lists of record names. All items in a data list must be in UTF-8 format.

Do not manipulate `tDataList` structures directly. Instead, use the data list utility functions such as `dsBuildFromPath` (page 15), `dsDataListAllocate` (page 26), `dsDataListGetNodeAlloc` (page 28), `dsAppendStringToListAlloc` (page 14), and `dsDataListDeallocate` (page 27).

See also tDataListPtr (page 87).

Availability
Available in Mac OS X v10.0 and later.

Declared In
DirServicesTypes.h

tRecordEntry

A structure used to store information about a record, including its name, type, and number of attributes.

```
typedef struct
{
    unsigned long fReserved1;
    tAccessControlEntry fReserved2;
    unsigned long fRecordAttributeCount;
    tDataNode fRecordNameAndType;
}   tRecordEntry;
```

Fields

`fReserved1`
Reserved.

`fReserved2`
Reserved.

`fRecordAttributeCount`
Number of attribute types.

`fRecordNameAndType`
Value of type tDataNode (page 88) containing the record's primary name in UTF-8 format and its type.

Availability
Available in Mac OS X v10.0 and later.

Declared In
DirServicesTypes.h

Other Open Directory Data Types

Data types used by the Open Directory Client.

tAttributeEntryPtr

A pointer to a `tAttributeEntry` structure.

```
typedef tAttributeEntry *tAttributeEntryPtr;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`DirServicesTypes.h`

tAttributeListRef

A reference used to get attribute entries.

```
typedef unsigned long tAttributeListRef;
```

Discussion
You receive a `tAttributeListRef` by calling `dsGetDirNodeInfo` (page 56) or `dsGetRecordEntry` (page 63).Pass the reference to `dsGetAttributeEntry` (page 51). Dispose of the reference by calling `dsCloseAttributeList` (page 20).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`DirServicesTypes.h`

tAttributeValueEntryPtr

A pointer to a `tAttributeValueEntry` structure.

```
typedef tAttributeValueEntry *tAttributeValueEntryPtr;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`DirServicesTypes.h`

tAttributeValueListRef

A reference used to get attribute value entries.

```
typedef unsigned long tAttributeValueListRef;
```

Discussion
You receive a `tAttributeValueListRef` when you call `dsGetAttributeEntry` (page 51). Pass the reference to `dsGetAttributeValue` (page 53). Dispose of the reference by calling `dsCloseAttributeValueList` (page 20).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`DirServicesTypes.h`

tBuffer

A pointer to an arbitrary value used to create data nodes.

```
typedef void * tBuffer;
```

Discussion
The `tBuffer` data type is used by `dsDataNodeAllocateBlock` (page 31) to create data nodes.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`DirServicesTypes.h`

tContextData

A pointer to an arbitrary value used to exchange continuation data.

```
typedef void * tContextData;
```

Discussion
When the results of calling an Open Directory function exceed the size of the response buffer, the function returns a value of type `tContextData`. Your application can get the next buffer of results by calling the function again and passing the continuation data as a parameter.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`DirServicesTypes.h`

tDataBufferPtr

A pointer to a value of type `tDataBuffer`.

```
typedef tDataBuffer *tDataBufferPtr;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`DirServicesTypes.h`

tDataListPtr

A pointer to a value of type `tDataList`.

```
typedef tDataList *tDataListPtr;
```

Discussion
To allocate a data list, call `dsDataListAllocate` (page 26). To build a data list from one or more data nodes, call `dsBuildListFromNodesAlloc` (page 16); to build a data list from one or more C strings, call `dsBuildListFromStrings` (page 18). Or copy a data list by calling `dsDataListCopyList` (page 27).

To release the memory associated with a data list when it is no longer needed, call `dsDataListDeallocate` (page 27). If the data list is heap-based, you also need to call `free()`.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`DirServicesTypes.h`

tDataNode

A value of type `tDataBuffer`.

```
typedef tDataBuffer tDataNode;
```

Discussion
The `tDataNode` data type provides a standard format for passing information to Open Directory functions. It is typically used to contain strings, nodes, and attribute types that are exchanged between Open Directory and an Open Directory client.

See also `tDataNodePtr` (page 88).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`DirServicesTypes.h`

tDataNodePtr

A pointer to a value of type `tDataNode`.

```
typedef tDataNode tDataNodePtr;
```

Discussion
Call `dsDataNodeAllocateBlock` (page 31) or `dsDataNodeAllocateString` (page 32) to allocate a data node.

Call `dsDataNodeDeAllocate` (page 33) to release the memory associated with a data node when it is no longer needed.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`DirServicesTypes.h`

tDirNodeReference

A value returned when a node is opened.

```
typedef unsigned long tDirNodeReference;
```

Discussion
Open Directory functions that operate on nodes, records, and attributes require a `tDirNodeReference` as a parameter. Call `dsOpenDirNode` (page 68) to open a node. Call `dsCloseDirNode` (page 21) to close the node and dispose of the reference when you no longer need it.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`DirServicesTypes.h`

tDirReference

A value returned when an Open Directory session is opened.

```
typedef unsigned long tDirReference;
```

Discussion
You receive a `tDirReference` by calling by `dsOpenDirService` (page 69) or `dsOpenDirServiceProxy` (page 70) to open an Open Directory session. You call `dsCloseDirService` (page 22) to close the session and dispose of the reference when you no longer need it.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`DirServicesTypes.h`

tRecordEntryPtr

A pointer to a value of type `tRecordEntry`.

```
typedef tRecordEntry *tRecordEntryPtr;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`DirServicesTypes.h`

tRecordReference

A value returned when a record is opened.

```
typedef unsigned long tRecordReference;
```

Discussion
You receive a `tRecordReference` by calling `dsCreateRecordAndOpen` (page 24) or `dsOpenRecord` (page 71). Closing the record causes the record reference to be invalidated.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`DirServicesTypes.h`

Request Structures

This section describes the structures that Open Directory passes to a plug-in's `ProcessRequest` entry point in order to work with directory nodes.

sAddAttribute

Structure received when an Open Directory client calls `dsAddAttribute`.

```
typedef struct {
    uInt32              fType;
    sInt32              fResult;
    tRecordReference    fInRecRef;
    tDataNodePtr        fInNewAttr;
    tAccessControlEntryPtrfInNewAttrAccess;
    tDataNodePtr        fInFirstAttrValue;
} sAddAttribute;
```

Fields

`fType`
: Always `kAddAttribute`.

`fResult`
: Value of type `sInt32` that the plug-in sets to `eDSNoErr` before returning to indicate that it was able to add the attribute. If an error occurs, the plug-in sets `fResult` to a value listed in "Result Codes" (page 169).

`fInRecRef`
: Value of type `tRecordReference` (page 89) representing the record to which the attribute is to be added. The record must have been previously opened when the plug-in's routine for processing `sOpenRecord` (page 122) structures was called.

`fInNewAttr`
: Value of type `tDataNodePtr` (page 88) that points to a value of type `tDataNode` (page 88) containing the name of the attribute that is to be added.

`fInNewAttrAccess`
: Reserved for this release. Client applications are advised to set this value to `NULL`. For this release, plug-ins should ignore the value of this field.

`fInFirstAttrValue`
: Value of type `tDataListPtr` (page 87) that points to a value of type `tDataNode` (page 88) containing the first value of the attribute that is being added. The `tDataNode` may contain an empty string or `fInFirsAttrValue` may be `NULL` to indicate that the client application does not want to set the attribute's value.

Discussion

When an Open Directory plug-in receives an `sAddAttribute` structure, it uses the `fInRecRef` field of the `sAddAttribute` structure to determine the record to which an attribute is to be added, the `fInNewAttr` field to obtain the name of the attribute that is to be added, and the `fInFirstAttrValue` field as the added attribute's first value.

If the plug-in can the add the attribute, it adds the attribute, sets its first value, sets `fResult` to `eDSNoErr`, and returns.

If the plug-in cannot add the attribute, it sets `fResult` to an appropriate result code as described in "Result Codes" (page 169) and returns.

sAddAttributeValue

Structure when an Open Directory client calls `dsAddAttributeValue`.

```
typedef struct {
    uInt32          fType;
    sInt32          fResult;
    tRecordReferencefInRecRef;
    tDataNodePtr    fInAttrType;
    tDataNodePtr    fInAttrValue;
} sAddAttributeValue;
```

Fields

`fType`
: Always `kAddAttributeValue`.

`fResult`
: Value of type `sInt32` that the plug-in sets to `eDSNoErr` before returning to indicate that it was able to add the value to the attribute. If an error occurs, the plug-in sets `fResult` to a value listed in "Result Codes" (page 169).

`fInRecRef`
: Value of type `tRecordReference` (page 89) representing the record for which a value is to be add to an attribute.

`fInAttrType`
: Value of type `tDataNodePtr` (page 88) that points to a value of type `tDataNode` (page 88) containing the type of the attribute to which a value is to be added.

`fInAttrValue`
: Value of type `tDataNodePtr` (page 88) that points to a value of type `tDataNode` (page 88) containing the value that is to be added.

Discussion

The DirectoryService daemon calls a plug-in's `ProcessRequest` entry point and passes an `sAddAttributeValue` structure when an Open Directory client calls `dsAddAttributeValue`.

The plug-in should verify that the attribute is capable of having multiple values. It then uses the `fInRecRef` field of the `sAddAttributeValue` structure to determine the record that has the attribute to which a value is to be added, the `fInAttrType` field to determine the type of the attribute to which a value is to be added, and the `fInAttrValue` field to get the value to that is to be added.

If the plug-in can add the specified value to the specified attribute, it adds the value and creates a unique attribute value ID for it, sets `fResult` to `eDSNoErr`, and returns.

If the plug-in cannot add the value to the attribute, it sets `fResult` to an appropriate result code as described in "Result Codes" (page 169) and returns.

sCloseAttributeList

Structure received when an Open Directory client calls `dsCloseAttributeList`.

```
typedef struct {
    uInt32          fType;
    sInt32          fResult;
    tAttributeListReffInAttributeListRef;
} sCloseAttributeList;
```

Fields

`fType`
    Always `kCloseAttributeList`.

`fResult`
    Value of type `sInt32` that the plug-in sets to `eDSNoErr` before returning to indicate that it was able to close the specified attribute list reference. If an error occurs, the plug-in sets `fResult` to a value listed in "Result Codes" (page 169).

`fInAttributeListRef`
    Value of type `tAttributeListRef` (page 86) representing the attribute list reference that is to be closed.

Discussion

The DirectoryService daemon calls a plug-in's `ProcessRequest` entry point and passes an `sCloseAttributeList` structure when an Open Directory client calls `dsCloseAttributeList` to dispose of an attribute list reference.

If the attribute value list is valid, the plug-in disposes of it, sets `fResult` to `eDSNoErr`, and returns. If the attribute list reference is invalid, the plug-in sets `fResult` to an appropriate result code as described in "Result Codes" (page 169) and returns.

sCloseAttributeValueList

Structure received when an Open Directory client calls `dsCloseAttributeValueList`.

```
typedef struct {
    uInt32                  fType;
    sInt32                  fResult;
    tAttributeValueListReff InAttributeValueListRef;
} sCloseAttributeValueList;
```

Fields

`fType`

Always `kCloseAttributeValueList`.

`fResult`

Value of type `sInt32` that the plug-in sets to `eDSNoErr` before returning to indicate that it was able to close the specified attribute value list reference. If an error occurs, the plug-in sets `fResult` to a value listed in "Result Codes" (page 169).

`fInAttributeValueListRef`

Value of type `tAttributeValueListRef` (page 86) representing the attribute value list reference that is to be closed.

Discussion

The DirectoryService daemon calls a plug-in's `ProcessRequest` entry point and passes an `sCloseAttributeValueList` structure when an Open Directory client calls `dsCloseAttributeValueList` to dispose of an attribute value list reference.

If the attribute value list reference is valid, the plug-in disposes of it, sets `fResult` to `eDSNoErr`, and returns. If the attribute value list reference is invalid, the plug-in sets `fResult` to an appropriate result code as described in "Result Codes" (page 169) and returns.

sCloseDirNode

Structure received when an Open Directory client calls `dsCloseDirNode`.

```
typedef struct {
    uInt32          fType;
    sInt32          fResult;
    tDirReference   fInNodeRef;
} sCloseDirNode;
```

Fields

`fType`

Always `kCloseDirNode`.

`fResult`

Value of type `sInt32s` that the plug-in sets to `eDSNoErr` before returning to indicate that it was able to close the directory node session specified by `fInNodeRef`. If an error occurs, the plug-in sets `fResult` to a value listed in "Result Codes" (page 169).

`fInNodeRef`

Value of type `tDirReference` (page 89) that identifies the directory node session that is to be closed. The directory node reference was created when the client application opened the directory node session that is to be closed.

Discussion

The DirectoryService daemon calls a plug-in's `ProcessRequest` entry point and passes an `sCloseDirNode` structure when an Open Directory client calls `dsCloseDirNode` to close a session with a directory node.

When an Open Directory plug-in receives a request to close a directory node session, it uses the fInNodeRef field to determine whether fInNodeRef represents a valid directory node that the client application has opened.

If the directory node reference is valid, the plug-in invalidates all record references, attribute references, attribute value references, and continuation data values that are associated with the directory node reference specified by fInNodeRef. The plug-in sets fResult to eDSNoErr and returns.

If the plug-in cannot close the node (for example, because it is invalid), it sets fResult to an appropriate result code as described in "Result Codes" (page 169) and returns.

sCloseRecord

Structure received when an Open Directory client application calls dsCloseRecord.

```
typedef struct {
    uInt32          fType;
    sInt32          fResult;
    tRecordReferencefInRecRef;
} sCloseRecord;
```

Fields

fType

Always kCloseRecord.

fResult

Value of type sInt32 that the plug-in sets to eDSNoErr before returning to indicate that it was able to close the record. If an error occurs, the plug-in sets fResult to a value listed in "Result Codes" (page 169).

fInRecRef

Value of type tRecordReference (page 89) representing the record that is to be closed. The plug-in created the value of fInRecRef when it was called to process a request to open the record.

Discussion

The DirectoryService daemon calls a plug-in's ProcessRequest entry point and passes an sCloseRecord structure when an Open Directory client calls dsCloseRecord to close a record.

If the record can be closed, the plug-in ensures that any changes for the record that are cached in memory are saved to disk, invalidates the record reference specified in the fInRecRef field, invalidates any attribute list references and any attribute value list references associated with the record, sets fResult to eDSNoErr, and returns.

If the plug-in cannot close the record, it sets fResult to an appropriate result code as described in "Result Codes" (page 169) and returns.

sCreateRecord

Structure received when an Open Directory client calls dsCreateRecord or dsCreateRecordAndOpen.

```
typedef struct {
    uInt32              fType;
    sInt32              fResult;
    tDirNodeReference   fInNodeRef;
    tDataNodePtr        fInRecType;
    tDataNodePtr        fInRecName;
    bool                fInOpen;
    tRecordReference    fOutRecRef;
} sCreateRecord;
```

Fields

`fType`
: Always `kCreateRecord` or `kCreateRecordAndOpen`.

`fResult`
: Value of type `sInt32` that the plug-in sets to `eDSNoErr` before returning to indicate that it was able to create the record. If an error occurs, the plug-in sets `fResult` to a value listed in "Result Codes" (page 169).

`fInNodeRef`
: Value of type `tDirNodeReference` (page 89) that identifies the directory node in which the record is to be created. The directory node reference was created when the client application opened a session with the directory node.

`fInRecType`
: Value of type `tDataNodePtr` (page 88) that points to a `tDataNode` (page 88) structure containing the type that is to be assigned to the created record.

`fInRecName`
: Value of type `tDataNodePtr` (page 88) that points to a `tDataNode` (page 88) structure containing the name that is to be assigned to the record that is created.

`fInOpen`
: Boolean whose value is `TRUE` if the client application wants to create the record and open it. Otherwise, the value of `fInOpen` is `FALSE` to indicate that the client application wants to create the record without opening it.

`fOutRecRef`
: Value of type `tRecordReference` (page 89) assigned by the DirectoryService daemon and that the plug-in associates with the internal structure the plug-in uses to maintain information about the reference.

Discussion

The DirectoryService daemon calls a plug-in's `ProcessRequest` entry point and passes an `sCreateRecord` structure when an Open Directory client calls `dsCreateRecord`or `dsCreateRecordAndOpen` to create a record.

The plug-in uses the `fInNodeRef` field of the `sCreateRecord` structure to determine the directory node in which the record is to be created, the `fInRecType` field to set the type of the record that is to be created, and the `fInRecName` field to set the name of the record that is to be created.

If the plug-in can create the new record, it sets `fResult` to `eDSNoErr`, and returns.

If the plug-in cannot create the new record, it sets `fResult` to an appropriate result code as described in "Result Codes" (page 169) and returns.

sDeleteRecord

Structure received when an Open Directory client calls `dsDeleteRecord`.

```
typedef struct {
    uInt32          fType
    sInt32          fResult;
    tRecordReferencefInRecRef;
} sDeleteRecord;
```

Fields

`fType`

Always `kDeleteRecord`.

`fResult`

Value of type `uInt32` that the plug-in sets to `eDSNoErr` before returning to indicate that it was able to delete the record. If an error occurs, the plug-in sets `fResult` to a value listed in "Result Codes" (page 169).

`fInRecRef`

Value of type `tRecordReference` (page 89) representing the record that is to be deleted. The plug-in created the value of `fInRecRef` when it was called to process a request to open the record.

Discussion

The DirectoryService daemon calls a plug-in's `ProcessRequest` entry point and passes an `sDeleteRecord` structure when an Open Directory client calls `dsDeleteRecord` to delete a record.

The plug-in uses the `fInRecRef` field of the `sDeleteRecord` structure to determine the record that is to be deleted. If the plug-in can delete the record, it invalidates the record reference specified by the `fInRecRef` field, invalidates any attribute list references and any attribute value list references associated with the record, sets `fResult` to `eDSNoErr`, and returns.

If the plug-in cannot delete the record, it sets `fResult` to an appropriate result code as described in "Result Codes" (page 169) and returns.

sDoAttrValueSearch

Structure received when an Open Directory client calls `dsDoAttributeValueSearch`.

```
typedef struct {
    uInt32          fType;
    sInt32          fResult;
    tDirNodeReferencefInNodeRef;
    tDataBufferPtr  fOutDataBuff;
    tDataListPtr    fInRecTypeList;
    tDataNodePtr    fInAttrType;
    tDirPatternMatchfInPattMatchType;
    tDataNodePtr    fInPatt2Match;
    unsigned long   fInOutMatchRecordCount;
    tContextData    fIOContinueData;
} sDoAttrValueSearch;
```

Fields

`fType`

Always `kDoAttributeValueSearch`.

`fResult`
: Value of type `sInt32` that the plug-in sets to `eDSNoErr` before returning to indicate that it was able to search for the attribute values. If an error occurs, the plug-in sets `fResult` to a value listed in "Result Codes" (page 169). If no matches are found, the plug-in should set `fResult` to `eDSNoErr`, `fInOutMatchRecordCount` to zero, and `fIOContinueData` to `NULL`.

`fInNodeRef`
: Value of type `tDirNodeReference` (page 89) that identifies the directory node for which the search is to be conducted. The directory node reference was created when the client application opened the directory node session.

`fOutDataBuff`
: Value of type `tDataBufferPtr` (page 87) that points to the `tDataBuffer` (page 84) structure in which the plug-in is to place search results.

`fInRecTypeList`
: Value of type `tDataListPtr` (page 87) pointing to a `tDataList` (page 84) structure containing the record types that are to be searched. See Standard Record Types (page 143) and Meta Record Type Constants (page 142) for possible values. If `NULL`, set `fResult` to `eDSEmptyRecordTypeList` and return.

`fInAttrType`
: Value of type `tDataNodePtr` (page 88) pointing to a `tDataNode` (page 88) structure containing the attribute types that are to be searched for. See the attribute constants described in the "Constants" (page 129) section for possible values. If `NULL`, set `fResult` to `eDSEmptyAttributeType` and return.

`fInPattMatchType`
: Value of type `tDirPatternMatch` that describes the way in which the pattern specified by `fInPatt2Match` is to be matched. The pattern match type can be a value that the plug-in and application agree upon or a constant defined by Open Directory, as described in the section Pattern Matching Constants (page 137).

`fInPatt2Match`
: Value of type `tDataNodePtr` (page 88) pointing to a `tDataNode` (page 88) structure containing the pattern that is to be matched.

`fInOutMatchRecordCount`
: Value of type `unsigned long`. The first time the client application calls `dsDoAttributeValueSearch`, `fInOutMatchRecordCount` is zero to receive all matching records or is a positive integer value that specifies the total number of records the client application wants to receive across what may be a series of `dsDoAttributeValueSearch` calls. If the latter, the plug-in should use the initial input value of `fInOutMatchRecordCount` to limit the total number of matching records it returns. Before returning, the plug-in should set `fInOutMatchRecordCount` to the number of records it has placed in the buffer pointed to by `fOutDataBuff`. The plug-in should ignore the input value of `fInOutMatchRecordCount` whenever it is processing a `sDoAttrValueSearch` structure that has an `fIOContinueData` field that is not `NULL`.

`fIOContinueData`
: Value of type `tContextData` (page 87) containing continuation data. For the first in a series of calls to `dsDoAttributeValueSearch`, the input value is `NULL`. If the plug-in can store all of the matching records in the buffer pointed to by `fOutDataBuff`, it sets `fIOContinueData` to `NULL` before returning. If there more records than can be stored in the buffer, the plug-in stores as much data as possible and sets `fIOContinueData` to a plug-in–defined value that the plug-in can use when the client application calls `dsDoAttributeValueSearch` again to get another buffer of data. You may want to include a timestamp in the continuation data and return an error if you determine that `fOutContinueData` is out of date.

Discussion
The DirectoryService daemon calls a plug-in's `ProcessRequest` entry point and passes an `sDoAttrValueSearch` structure when an Open Directory client calls `dsDoAttributeValueSearch` to search for records that have attributes whose values match a pattern.

The plug-in uses the `fInNodeRef` field of the `sDoAttrValueSearch` structure to determine the directory node in which the search is to be conducted, the `fInRecTypeList` field to determine the record types that are to be searched, the `fInAttrType` field to determine the attribute types that are to be searched, the `fInPatt2Match` field to get the pattern to match, and the `fInPattMatchType` field to determine the way in which the pattern is to be compared. If this is the first in what may be a series of calls to get the complete search results, the `fIOContinueData` field is `NULL`. Otherwise, `fIOContinueData` contains a plug-in–defined value that the plug-in uses to provide the context required to resume filling the buffer pointed to by `fOutDataBuff` with search results.

Depending on the size of the data buffer pointed to by `fOutDataBuff` and the length of the search results, the plug-in's routine for processing `sDoAttrValueSearch` structures may be called multiple times in order to return all of the search results. The first time the plug-in's routine for processing `sDoAttrValueSearch` structures is called, the input value of `fIOContinueData` is `NULL` and input value of `fInOutRecEntryCount` specifies the total number of records that the plug-in should return even if the plug-in's routine for processing `sDoAttrValueSearch` structures must be called more than once.

If there are records that match the search criteria specified by `fInRecTypeList`, `fInAttrType`, `fInPattMatchType`, and `fInPatt2Match`, the plug-in puts the record entries, attribute entries, and attribute values in the buffer pointed to by `fOutDataBuff`. It also sets `fInOutMatchRecordCount` to the number of records that have been placed in `fOutDataBuff` and sets `fResult` to `eDSNoErr`. If the buffer pointed to by `fOutDataBuff` is too small to hold all of the data, the plug-in sets `fIOContinueData` to a plug-in–defined value that the plug-in can use when the client application calls `dsDoAttributeValueSearch` again to get another buffer of data. If the buffer pointed to by `fOutDataBuff` contains all of the records or contains the last records in the record list, the plug-in sets `fIOContinueData` to `NULL`.

If the plug-in returns before it gets search results to place in the buffer pointed to by `fOutDataBuff`, it should set `fInOutMatchRecordCount` to zero, set `fResult` to `eDSNoErr`, and set `fIOContinueData` to a plug-in–defined value that is not `NULL`. These settings indicate to the client application that it should call `dsDoAttributeValueSearch` again to get the search results.

If there are no matching records, the plug-in sets `fInOutMatchRecordCount` to zero, `fIOContinueData` to `NULL`, sets `fResult` to `eDSNoErr`, and returns.

sDoAttrValueSearchWithData

Structure received when an Open Directory client calls `dsDoAttributeValueSearchWithData`.

```
typedef struct {
    uInt32          fType;
    sInt32          fResult;
    tDirNodeReferencefInNodeRef;
    tDataBufferPtr  fOutDataBuff;
    tDataListPtr    fInRecTypeList;
    tDataNodePtr    fInAttrType;
    tDirPatternMatchfInPattMatchType;
    tDataNodePtr    fInPatt2Match;
    unsigned long   fInOutMatchRecordCount;
    tContextData    fIOContinueData;
    tDataListPtr    fInAttrTypeRequestList;
    bool            fInAttrInfoOnly;
} sDoAttrValueSearchWithData;
```

Fields

`fType`

Always `kDoAttributeValueSearchWithData`.

`fResult`

Value of type `sInt32` that the plug-in sets to `eDSNoErr` before returning to indicate that it was able to search for the attribute values. If an error occurs, the plug-in sets `fResult` to a value listed in "Result Codes" (page 169). If no matches are found, the plug-in should set `fResult` to `eDSNoErr`, `fInOutMatchRecordCount` to zero, and `fIOContinueData` to `NULL`.

`fInNodeRef`

Value of type `tDirNodeReference` (page 89) that identifies the directory node for which the search is to be conducted. The directory node reference was created when the client application opened the directory node session.

`fOutDataBuff`

Value of type `tDataBufferPtr` (page 87) that points to the `tDataBuffer` (page 84) structure in which the plug-in is to place search results.

`fInRecTypeList`

Value of type `tDataListPtr` (page 87) pointing to a `tDataList` (page 84) structure containing the record types that are to be searched. See Standard Record Types (page 143) and Meta Record Type Constants (page 142) for possible values. If `NULL`, set `fResult` to `eDSEmptyRecordTypeList` and return.

`fInAttrType`

Value of type `tDataNodePtr` (page 88) pointing to a `tDataNode` (page 88) structure containing the attribute types that are to be searched for. See the attribute constants described in the "Constants" (page 129) section for possible values. If `NULL`, set `fResult` to `eDSEmptyAttributeType` and return.

`fInPattMatchType`

Value of type `tDirPatternMatch` that describes the way in which the pattern specified by `fInPatt2Match` is to be matched. The pattern match type can be a value that the plug-in and application agree upon or a constant defined by Open Directory, as described in the section `Pattern Matching Constants` (page 137).

`fInPatt2Match`

Value of type `tDataNodePtr` (page 88) pointing to a `tDataNode` (page 88) structure containing the pattern that is to be matched.

`fInOutMatchRecordCount`

Value of type `unsigned long`. The first time the client application calls `dsDoAttributeValueSearchWithData`, `fInOutMatchRecordCount` is zero to receive all matching records or is a positive integer value that specifies the total number of records the client application wants to receive across what may be a series of `dsDoAttributeValueSearchWithData` calls. If the latter, the plug-in should use the initial input value o `ffInOutMatchRecordCount` to limit the total number of matching records it returns. Before returning, the plug-in should set `fInOutMatchRecordCount` to the number of records it has placed in the buffer pointed to by `fOutDataBuff`. The plug-in should ignore the input value of `fInOutMatchRecordCount` whenever it is processing a `sDoAttributeValueSearchWithData` structure that has an `fIOContinueData` field that is not `NULL`.

`fIOContinueData`

Value of type `tContextData` (page 87) containing continuation data. For the first in a series of calls to `dsDoAttributeValueSearchWithData`, the input value is `NULL`. If the plug-in can store all of the matching records in the buffer pointed to by `fOutDataBuff`, it sets `fIOContinueData` to `NULL` before returning. If there more records than can be stored in the buffer, the plug-in stores as much data as possible and sets `fIOContinueData` to a plug-in–defined value that the plug-in can use when the client application calls `dsDoAttributeValueSearchWithData` again to get another buffer of data. You may want to include a timestamp in the continuation data and return an error if you determine that `fOutContinueData` is out of date.

`fInAttrTypeRequestList`

Value of type `tDataListPtr` (page 87) pointing to a `tDataList` (page 84) structure containing attribute types that are to be returned if matches are found. See the attribute constants described in the "Constants" (page 129) section for possible values. If `NULL`, set `fResult` to `eDSEmptyAttributeTypeList` and return.

`fInAttrInfoOnly`

Boolean value set to `TRUE` if the plug-in is only to provide information about attributes or set to `FALSE` if the plug-in is to provide the values of the attributes as well as information about the attributes.

Discussion

The DirectoryService daemon calls a plug-in's `ProcessRequest` entry point and passes an `sDoAttrValueSearchWithData` structure when an Open Directory client calls `dsDoAttributeValueSearchWithData` to search for records. Matches are based on the value of `fInAttrType`, `fInPattMatchType`, and `fInPatt2Match`. For records that match, the `fInAttrTypeRequestList` field determines which attributes to return.

The plug-in uses the `fInNodeRef` field of the `sDoAttrValueSearchWithData` structure to determine the directory node in which the search is to be conducted, the `fInRecTypeList` field to determine the record types that are to be searched, the `fInAttrType` field to determine the attribute types that are to be searched, the `fInPatt2Match` field to get the pattern to match, and the `fInPattMatchType` field to determine the way in which the pattern is to be compared. If this is the first in what may be a series of calls to get the complete search results, the `fIOContinueData` field is `NULL`. Otherwise, `fIOContinueData` contains a plug-in–defined value that the plug-in uses to provide the context required to resume filling the buffer pointed to by `fOutDataBuff` with search results.

The `sDoAttrValueSearchWithData` structure differs from the `sDoAttrValueSearch` structure in that the `sDoAttrValueSearchWithData` structure has two additional fields: `fInAttrTypeRequestList`, which specifies the type of attributes for which information is to be returned when a match is found, and `fInAttrInfoOnly`, which indicates whether attribute information or attribute information and attribute values are to be returned when a match is found.

Depending on the size of the data buffer pointed to by `fOutDataBuff` and the length of the search results, the plug-in's routine for processing `sDoAttrValueSearchWithData` structures may be called multiple times in order to return all of the search results. The first time the plug-in's routine for processing `sDoAttrValueSearchWithData` structures is called, the input value of `fIOContinueData` is `NULL` and input value of `fInOutRecEntryCount` specifies the total number of records that the plug-in should return even if the plug-in's routine for processing `sDoAttrValueSearchWithData` structures must be called more than once.

If there are records that match the search criteria specified by `fInRecTypeList`, `fInAttrType`, `fInPattMatchType`, and `fInPatt2Match`, plug-in puts the record entries, attribute entries, and attribute values in the buffer pointed to by `fOutDataBuff`. It also sets `fInOutMatchRecordCount` to the number of records that have been placed in `fOutDataBuff` and sets `fResult to eDSNoErr`. If the buffer pointed to by `fOutDataBuff` is too small to hold all of the data, the plug-in sets `fIOContinueData` to a plug-in–defined value that the plug-in can use when the client application calls `dsDoAttributeValueSearchWithData` again to get another buffer of data. If the buffer pointed to by `fOutDataBuff` contains all of the records or contains the last records in the record list, the plug-in sets `fIOContinueData` to `NULL`.

If the plug-in returns before it gets search results to place in the buffer pointed to by `fOutDataBuff`, it should set `fInOutMatchRecordCount` to zero, set `fResult` to `eDSNoErr`, and set `fIOContinueData` to a plug-in–defined value that is not `NULL`. These settings indicate to the client application that it should call `dsDoAttributeValueSearchWithData` again to get the search results.

If there are no matching records, the plug-in sets `fInOutMatchRecordCount` to zero, `fIOContinueData` to `NULL`, sets `fResult` to `eDSNoErr`, and returns.

sDoDirNodeAuth

Structure received when an Open Directory client calls `dsDoDirNodeAuth`.

```
typedef struct {
    uInt32              fType;
    sInt32              fResult;
    tDirNodeReference   fInNodeRef;
    tDataNodePtr        fInAuthMethod;
    bool                fInDirNodeAuthOnlyFlag;
    tDataBufferPtr      fInAuthStepData;
    tDataBufferPtr      fOutAuthStepDataResponse;
    tContextData        fIOContinueData;
} sDoDirNodeAuth;
```

Fields

`fType`
: Always `kDoDirNodeAuth`.

`fResult`
: Value of type `sInt32` that the plug-in sets to `eDSNoErr` before returning to indicate that it was able to authenticate the session. If an error occurs, the plug-in sets `fResult` to a value listed in "Result Codes" (page 169).

`fInNodeRef`
: Value of type `tDirNodeReference` (page 89) that identifies the directory node session that is to be authenticated. The directory node reference was created when the client application opened the session with the directory node.

`fInAuthMethod`
: Value of type `tDataNodePtr` (page 88) that points to a `tDataNode` (page 88) structure containing the authentication task that is to be performed.Examples include `kDSStdAuthSetPasswd`, `kDSStdAuthSetPasswdAsRoot`, and `kDSStdAuthChangePasswd` to set or change a password and `kDSStdAuthClearText` and `kDSStdAuth2WayRandom` to authenticate a user for a directory node session. See the attribute constants described in the "Constants" (page 129) section for possible values.

`fInDirNodeAuthOnlyFlag`
: Boolean value that is `TRUE` if the client application does not want the result of this authentication to be used to grant or deny access for subsequent operations pertaining to this node. When the value of `fInDirNodeAuthOnlyFlag` is `FALSE`, the client application wants the result of this authentication to be applied to other operations that pertain to this directory node.

`fInAuthStepData`
: Value of type `tDataBufferPtr` (page 87) pointing to a `tDataBuffer` (page 84) structure that contains a value that identifies the step in the authentication process for which the plug-in `ProcessRequest` routine has been called.

`fOutAuthStepDataResponse`
: Value of type `tDataBufferPtr` (page 87) that points to the `tDataBuffer` (page 84) structure in which the plug-in is to place its response.

`fIOContinueData`
: Value of type `tContextData` (page 87). If this the first step in the authentication process, `fIOContinueData` is `NULL`. If this is any other step, `fIOContinueData` should contain a value that the plug-in returned to the client application when the client previously called `dsDoDirNodeAuth`. The plug-in can use `fIOContinueData` to maintain context information about the authentication process as it progresses through the various steps required by the authentication method. You may want to include a timestamp in `fIOContinueData` and fail the next step in the authentication process if `fIOContinueData` is too old.

Discussion

The DirectoryService daemon calls a plug-in's `ProcessRequest` entry point and passes an `sDoDirNoteAuth` structure when an Open Directory client calls `dsDoDirNodeAuth` to perform an authentication task.

The plug-in uses the `fInNodeRef` field of the `sDoDirNodeAuth` structure to determine the directory node for which the authentication task is to be perform and the `fInAuthMethod` field to determine the authentication task. The plug-in also uses the `fInDirNodeAuthOnlyFlag` field to determine whether to apply the results of the authentication to other Open Directory calls the client application may make, and the `fInAuthStepData` field indicates the current step in the authentication process.

If this step in the authentication process is successful, the plug-in sets `fResult to eDSNoErr`. If there are additional steps in the authentication process, the plug-in sets `fOutAuthStepDataResponse` to a value that is appropriate for this authentication method and sets `fIOContinueData` to a plug-in–defined value before returning. If this is the last step in the authentication process, the plug-in sets `fIOContinueData` to `NULL`.

If this step in the authentication process was not successful, the plug-in sets `fResult` to an appropriate result code as described in "Result Codes" (page 169), sets `fIOContinueData` to `NULL`, and returns.

sDoMultiAttrValueSearch

Structure received when an Open Directory client calls `dsDoMultipleAttributeValueSearch`.

```
typedef struct {
    uInt32          fType;
    sInt32          fResult;
    tDirNodeReferencefInNodeRef;
    tDataBufferPtr  fOutDataBuff;
    tDataListPtr    fInRecTypeList;
    tDataNodePtr    fInAttrType;
    tDirPatternMatchfInPattMatchType;
    tDataListPtr    fInPatterns2MatchList;
    unsigned long   fInOutMatchRecordCount;
    tContextData    fIOContinueData;
} sDoMultiAttrValueSearch;
```

Fields

`fType`

Always `kDoMultipleAttributeValueSearch`.

`fResult`

Value of type `sInt32` that the plug-in sets to `eDSNoErr` before returning to indicate that it was able to search for the attribute values. If an error occurs, the plug-in sets `fResult` to a value listed in "Result Codes" (page 169). If no matches are found, the plug-in should set `fResult` to `eDSNoErr`, `fInOutMatchRecordCount` to zero, and `fIOContinueData` to `NULL`.

`fInNodeRef`

Value of type `tDirNodeReference` (page 89) that identifies the directory node for which the search is to be conducted. The directory node reference was created when the client application opened the directory node session.

`fOutDataBuff`

Value of type `tDataBufferPtr` (page 87) that points to the `tDataBuffer` (page 84) structure in which the plug-in is to place search results.

`fInRecTypeList`

Value of type `tDataListPtr` (page 87) pointing to a `tDataList` (page 84) structure containing the record types that are to be searched. See Standard Record Types (page 143) and Meta Record Type Constants (page 142) for possible values. If `NULL`, set `fResult` to `eDSEmptyRecordTypeList` and return.

`fInAttrType`

Value of type `tDataNodePtr` (page 88) pointing to a `tDataNode` (page 88) structure containing the attribute types that are to be searched for. See the attribute constants described in the "Constants" (page 129) section for possible values. If `NULL`, set `fResult` to `eDSEmptyAttributeType` and return.

`fInPattMatchType`

Value of type `tDirPatternMatch` that describes the way in which the pattern specified by `fInPatt2Match` is to be matched. The pattern match type can be a value that the plug-in and application agree upon or a constant defined by Open Directory, as described in the section Pattern Matching Constants (page 137).

`fInPatterns2MatchList`

Value of type `tDataListPtr` (page 87) pointing to a `tDataList` (page 84) structure containing a list of patterns to be matched.

`fInOutMatchRecordCount`

Value of type `unsigned long`. The first time the client application calls `dsDoAttributeValueSearch`, `fInOutMatchRecordCount` is zero to receive all matching records or is a positive integer value that specifies the total number of records the client application wants to receive across what may be a series of `dsDoAttributeValueSearch` calls. If the latter, the plug-in should use the initial input value of `fInOutMatchRecordCount` to limit the total number of matching records it returns. Before returning, the plug-in should set `fInOutMatchRecordCount` to the number of records it has placed in the buffer pointed to by `fOutDataBuff`. The plug-in should ignore the input value of `fInOutMatchRecordCount` whenever it is processing a `sDoMultiAttrValueSearch` structure that has an`fIOContinueData` field that is not `NULL`.

`fIOContinueData`

Value of type `tContextData` (page 87) containing continuation data. For the first in a series of calls to `dsDoAttributeValueSearch`, the input value is `NULL`. If the plug-in can store all of the matching records in the buffer pointed to by `fOutDataBuff`, it sets `fIOContinueData` to `NULL` before returning. If there more records than can be stored in the buffer, the plug-in stores as much data as possible and sets `fIOContinueData` to a plug-in–defined value that the plug-in can use when the client application calls `dsDoAttributeValueSearch` again to get another buffer of data. You may want to include a timestamp in the continuation data and return an error if you determine that `fOutContinueData` is out of date.

Discussion

The DirectoryService daemon calls a plug-in's `ProcessRequest` entry point and passes an `sDoMultiAttrValueSearch` structure when an Open Directory client calls `dsDoMultipleAttributeValueSearch` to search for records that have attributes whose values match one of multiple specified patterns.

The plug-in uses the `fInNodeRef` field of the `sDoMultiAttrValueSearch` structure to determine the directory node in which the search is to be conducted, the `fInRecTypeList` field to determine the type of records that are to be searched, the `fInAttrType` field to determine the attributes that are to be searched, the `fInPatterns2MatchList` field to get the patterns to match, and the `fInPattMatchType` field to determine the way in which the patterns are to be compared. If this is the first in what may be a series of calls to get the complete search results, the `fIOContinueData` field is `NULL`. Otherwise, `fIOContinueData` contains a plug-in–defined value that the plug-in uses to provide the context required to resume filling the buffer pointed to by `fOutDataBuff` with search results.

Depending on the size of the data buffer pointed to by `fOutDataBuff` and the length of the search results, the plug-in's routine for processing `sDoMultiAttrValueSearch` structures may be called multiple times in order to return all of the search results. The first time the plug-in's routine for processing the `sDoMultiAttrValueSearch` structure is called, the input value of `fIOContinueData` is `NULL` and input value of `fInOutRecEntryCount` specifies the total number of records that the plug-in should return even if the plug-in's routine for processing `sDoMultiAttrValueSearch` structures must be called more than once.

If there are records that match the search criteria specified by `fInRecTypeList`, `fInAttrType`, `fInPattMatchType`, and `fInPattern2MatchList`, the plug-in puts the record entries, attribute entries, and attribute values in the buffer pointed to by `fOutDataBuff`. It also sets `fInOutMatchRecordCount` to the number of records that have been placed in `fOutDataBuff` and sets `fResult` to `eDSNoErr`. If the buffer pointed to by `fOutDataBuff` is too small to hold all of the data, the plug-in sets `fIOContinueData` to a plug-in–defined value that the plug-in can use when the client application calls `dsDoMultipleAttributeValueSearch` again to get another buffer of data. If the buffer pointed to by `fOutDataBuff` contains all of the records or contains the last records in the list of records, the plug-in sets `fIOContinueData` to `NULL`.

If the plug-in returns before it gets search results to place in the buffer pointed to by `fOutDataBuff`, it should set `fInOutMatchRecordCount` to zero, set `fResult` to `eDSNoErr`, and set `fIOContinueData` to a plug-in–defined value that is not `NULL`. These settings indicate to the client that it should call `dsDoMultipleAttributeValueSearch` again to get the search results.

If there are no matching records, the plug-in sets `fInOutMatchRecordCount` to zero, `fIOContinueData` to `NULL`, sets `fResult` to `eDSNoErr`, and returns.

sDoMultiAttrValueSearchWithData

Structure received when an Open Directory client calls `dsDoMultipleAttributeValueSearch`.

```
typedef struct {
    uInt32          fType;
    sInt32          fResult;
    tDirNodeReferencefInNodeRef;
    tDataBufferPtr  fOutDataBuff;
    tDataListPtr    fInRecTypeList;
    tDataNodePtr    fInAttrType;
    tDirPatternMatchfInPattMatchType;
    tDataListPtr    fInPatterns2MatchList;
    unsigned long   fInOutMatchRecordCount;
    tContextData    fIOContinueData;
    tDataListPtr    fInAttrTypeRequestList;
    bool            fInAttrInfoOnly;
} sDoMultiAttrValueSearchWithData;
```

Fields

`fType`

Always `kDoMultipleAttributeValueSearchWithData`.

`fResult`

Value of type `sInt32` that the plug-in sets to `eDSNoErr` before returning to indicate that it was able to search for the attribute values. If an error occurs, the plug-in sets `fResult` to a value listed in "Result Codes" (page 169). If no matches are found, the plug-in should set `fResult` to `eDSNoErr`, `fInOutMatchRecordCount` to zero, and `fIOContinueData` to `NULL`.

`fInNodeRef`

Value of type `tDirNodeReference` (page 89) that identifies the directory node for which the search is to be conducted. The directory node reference was created when the client application opened the directory node session.

`fOutDataBuff`

Value of type `tDataBufferPtr` (page 87) that points to the `tDataBuffer` (page 84) structure in which the plug-in is to place search results.

`fInRecTypeList`

Value of type `tDataListPtr` (page 87) pointing to a `tDataList` (page 84) structure containing the record types that are to be searched. See Standard Record Types (page 143) and Meta Record Type Constants (page 142) for possible values. If `NULL`, set `fResult` to `eDSEmptyRecordTypeList` and return.

`fInAttrType`
: Value of type `tDataNodePtr` (page 88) pointing to a `tDataNode` (page 88) structure containing the attribute types that are to be searched for. See the attribute constants described in the "Constants" (page 129) section for possible values. If `NULL`, set `fResult` to `eDSEmptyAttributeType` and return.

`fInPattMatchType`
: Value of type `tDirPatternMatch` that describes the way in which the pattern specified by `fInPatt2Match` is to be matched. The pattern match type can be a value that the plug-in and application agree upon or a constant defined by Open Directory, as described in the section Pattern Matching Constants (page 137).

`fInPatterns2MatchList`
: Value of type `tDataListPtr` (page 87) pointing to a `tDataList` (page 84) structure containing a list of patterns to be matched.

`fInOutMatchRecordCount`
: Value of type `unsigned long`. The first time the client application calls `dsDoAttributeValueSearch`, `fInOutMatchRecordCount` is zero to receive all matching records or is a positive integer value that specifies the total number of records the client application wants to receive across what may be a series of `dsDoAttributeValueSearch` calls. If the latter, the plug-in should use the initial input value of `fInOutMatchRecordCount` to limit the total number of matching records it returns. Before returning, the plug-in should set `fInOutMatchRecordCount` to the number of records it has placed in the buffer pointed to by `fOutDataBuff`. The plug-in should ignore the input value of `fInOutMatchRecordCount` whenever it is processing a `sDoMultiAttrValueSearchWithData` structure that has an `fIOContinueData` field that is not `NULL`.

`fIOContinueData`
: Value of type `tContextData` (page 87) containing continuation data. For the first in a series of calls to `dsDoAttributeValueSearch`, the input value is `NULL`. If the plug-in can store all of the matching records in the buffer pointed to by `fOutDataBuff`, it sets `fIOContinueData` to `NULL` before returning. If there more records than can be stored in the buffer, the plug-in stores as much data as possible and sets `fIOContinueData` to a plug-in–defined value that the plug-in can use when the client application calls `dsDoAttributeValueSearch` again to get another buffer of data. You may want to include a timestamp in the continuation data and return an error if you determine that `fOutContinueData` is out of date.

`fInAttrTypeRequestList`
: Value of type `tDataListPtr` (page 87) pointing to a `tDataList` (page 84) structure containing the types of attribute that are to be returned if matches are found. See the attribute constants described in the "Constants" (page 129) section for possible values. If `NULL`, set `fResult` to `eDSEmptyAttributeTypeList` and return.

`fInAttrInfoOnly`
: Boolean value set to `TRUE` if the plug-in is only to provide information about attributes or set to `FALSE` if the plug-in is to provide the values of the attributes as well as information about the attributes.

Discussion

The DirectoryService daemon calls a plug-in's `ProcessRequest` entry point and passes an `sDoMultiAttrValueSearchWithData` structure when an Open Directory client calls `dsDoMultipleAttributeValueSearchWithData` to search for records that have attributes whose values match one of multiple specified patterns and return the values of the attributes specified by `fInAttrTypeRequestList` when a match occurs.

The plug-in uses the `fInNodeRef` field of the `sDoMultiAttrValueSearch` structure to determine the directory node in which the search is to be conducted, the `fInRecTypeList` field to determine the type of records that are to be searched, the `fInAttrType` field to determine the attributes that are to be searched, the fInPatterns2MatchList field to get the patterns to match, and the fInPattMatchType field to determine the way in which the patterns are to be compared. If this is the first in what may be a series of calls to get the complete search results, the fIOContinueData field is NULL. Otherwise, fIOContinueData contains a plug-in–defined value that the plug-in uses to provide the context required to resume filling the buffer pointed to by fOutDataBuff with search results.

The sDoMultiAttrValueSearchWithData structure differs from the sDoMultiAttrValueSearch structure in that the sDoMultiAttrValueSearchWithData structure has two additional fields: fInAttrTypeRequestList, which specifies the type of attributes for which information is to be returned when a match is found, and fInAttrInfoOnly, which indicates whether attribute information or attribute information and attribute values are to be returned when a match is found.

Depending on the size of the data buffer pointed to by fOutDataBuff and the length of the search results, the plug-in's routine for processing sDoMultiAttrValueSearchWithData structures may be called multiple times in order to return all of the search results. The first time the plug-in's routine for processing the sDoMultiAttrValueSearchWithData structure is called, the input value of fIOContinueData is NULL and input value of fInOutRecEntryCount specifies the total number of records that the plug-in should return even if the plug-in's routine for processing sDoMultiAttrValueSearchWithData structures must be called more than once.

If there are records that match the search criteria specified by fInRecTypeList, fInAttrType, fInPattMatchType, and fInPattern2MatchList, the plug-in puts the record entries, attribute entries, and attribute values in the buffer pointed to by fOutDataBuff. It also sets fInOutMatchRecordCount to the number of records that have been placed in fOutDataBuff and sets fResult to eDSNoErr. If the buffer pointed to by fOutDataBuff is too small to hold all of the data, the plug-in sets fIOContinueData to a plug-in–defined value that the plug-in can use when the client application calls dsDoMultipleAttributeValueSearch again to get another buffer of data. If the buffer pointed to by fOutDataBuff contains all of the records or contains the last records in the list of records, the plug-in sets fIOContinueData to NULL.

If the plug-in returns before it gets search results to place in the buffer pointed to by fOutDataBuff, it should set fInOutMatchRecordCount to zero, set fResult to eDSNoErr, and set fIOContinueData to a plug-in–defined value that is not NULL. These settings indicate to the client that it should call dsDoMultipleAttributeValueSearch again to get the search results.

If there are no matching records, the plug-in sets fInOutMatchRecordCount to zero, fIOContinueData to NULL, sets fResult to eDSNoErr, and returns.

sDoPluginCustomCall

Structure received when an Open Directory client calls dsDoPluginCustomCall.

```
typedef struct {
    uInt32          fType;
    sInt32          fResult;
    tDirNodeReferencefInNodeRef;
    unsigned long   fInRequestCode;
    tDataBufferPtr  fInRequestData;
    tDataBufferPtr  fOutRequestResponse;
} sDoPlugInCustomCall;
```

Fields

fType

Always kDoPlugInCustomCall.

`fResult`

Value of type `sInt32` that the plug-in sets to `eDSNoErr` before returning to indicate that the plug-in responded without error when it processed the `sDoPluginCustomCall` structure. If an error occurs, the plug-in sets `fResult` to a value listed in "Result Codes" (page 169).

`fInNodeRef`

Value of type `tDirNodeReference` (page 89) that identifies the directory node session to which `fInRequestCode` and `fInRequestData` apply.

`fInRequestCode`

Value of type `unsigned long` that contains a request code that has significance to the plug-in.

`fInRequestData`

Value of type `tDataBufferPtr` (page 87) that points to a `tDataBuffer` (page 84) structure containing data sent by the client application to the plug-in.

`fOutRequestResponse`

Value of type `tDataBufferPtr` (page 87) that points to a `tDataBuffer` (page 84) structure in which the plug-in places data that is to be returned to the client application.

Discussion

The DirectoryService daemon calls a plug-in's `ProcessRequest` entry point and passes an `sDoPluginCustomCall` structure when an Open Directory client calls `dsDoPluginCustomCall` to send custom data to the plug-in.

The plug-in verifies that the directory node reference stored in `fInNodeRef` is valid. It then interprets the value of the `fInRequestCode` field, parses the value pointed to by the `fInRequestData` field, and performs an action that is appropriate for the request code. If the plug-in needs to return data to the client application, it stores the data in the `tDatabuffer` structure pointed to by `fOutRequestResponse`.

If the plug-in performs the action without error, it sets `fResult` to `eDSNoErr`; otherwise, it sets `fResult` to an appropriate result code as described in "Result Codes" (page 169).

sFlushRecord

Structure received when an Open Directory client calls `dsFlushRecord`.

```
typedef struct {
    uInt32          fType;
    sInt32          fResult;
    tRecordReferencefInRecRef;
} sFlushRecord;
```

Fields

`fType`

Always `kFlushRecord`.

`fResult`

Value of type `sInt32` that the plug-in sets to `eDSNoErr` before returning to indicate that it was able to flush the record. If an error occurs, the plug-in sets `fResult` to a value listed in "Result Codes" (page 169).

`fInRecRef`

Value of type `tRecordReference` (page 89) representing the record that is to be flushed. The plug-in created the value of `fInRecRef` when it was called to process a request to open the record.

Discussion

The DirectoryService daemon calls a plug-in's `ProcessRequest` entry point and passes an `sFlushRecord` structure when an Open Directory client calls `dsFlushRecord`.

The plug-in uses the `fInRecRef` field of the `sFlushRecord` structure to determine the record that is to be flushed. If the plug-in can write the record, it does so and sets `fResult` to `eDSNoErr`, and returns. If the plug-in cannot flush the record, it sets `fResult` to an appropriate result code as described in "Result Codes" (page 169) and returns.

sGetAttributeEntry

Structure received when an Open Directory client calls `dsGetAttributeEntry`.

```
typedef struct {
    uInt32                  fType;
    sInt32                  fResult;
    tDirNodeReference       fInNodeRef;
    tDataBufferPtr          fInOutDataBuff;
    tAttributeListRef       fInAttrListRef;
    unsigned long           fInAttrInfoIndex;
    tAttributeValueListRef  fOutAttrValueListRef;
    tAttributeEntryPtr      fOutAttrInfoPtr;
} sGetAttributeEntry;
```

Fields

`fType`

Always `kGetAttributeEntry`.

`fResult`

Value of type `sInt32` that the plug-in sets to `eDSNoErr` before returning to indicate that it was able to get the requested attribute information. If an error occurs, the plug-in sets `fResult` to a value listed in "Result Codes" (page 169).

`fInNodeRef`

Value of type `tDirNodeReference` (page 89) that identifies the directory node of the record whose attribute information is to be obtained. The directory node reference was created when the client application opened the directory node session.

`fInOutDataBuff`

Value of type `tDataBufferPtr` (page 87) that points to the `tDataBuffer` (page 84) structure from which the attribute information is to be obtained.

`fInAttrListRef`

Value of type `tAttributeListRef` (page 86) that refers to an attribute list that the plug-in returned to the client application when it processed a `sGetDirNodeInfo` (page 111) or a `sGetRecordEntry` (page 116) structure or that the plug-in returned to the client application when it previously called `dsGetAttributeEntry`. The plug-in uses the attribute list reference to locate the attribute information in the buffer pointed to by `fInOutDataBuff`.

`fInAttrInfoIndex`

Value of type `unsigned long` that specifies the one-based index number of the attribute whose information is to be obtained from the buffer pointed to by `fInOutDataBuff`.

`fOutAttrValueListRef`

Value of type `tAttributeValueListRef` (page 86) assigned by the DirectoryService daemon and that the plug-in associates with the internal structure the plug-in uses to maintain information about the reference.

`fOutAttrInfoPtr`
: Pointer to a value of type `tAttributeValueEntryPtr` (page 86) that points to a `tAttributeEntry` (page 82) structure in which the plug-in is to place the attribute information.

Discussion

The DirectoryService daemon calls a plug-in's `ProcessRequest` entry point and passes an `sGetAttributeEntry` structure when an Open Directory client calls `dsGetAttributeEntry` to get information about an attribute from the buffer pointed to by `fInOutDataBuff`.

The plug-in uses the `fInNodeRef` field to determine the directory node of the record for which attribute information is requested and the `fInAttrInfoIndex` field to determine the attribute for which attribute information is requested. The information includes the number of values the attribute has, the total number of bytes the values use, the maximum size of a value for the specified attribute, and the attribute's unique signature.

If the plug-in can get the requested information from `fInOutDataBuff`, it puts the attribute information in the attribute entry structure pointed to by `fOutAttrInfoPtr`, sets `fOutAttrValueListRef` to a value that the plug-in can use to locate the attribute's value if its routine for processing `sGetAttributeValue` (page 110) structures is called, sets `fResult` to `eDSNoErr`, and returns.

If the plug-in cannot provide the requested attribute information, it sets `fOutAttrValueListRef` to `NULL`, sets `fResult` to an appropriate result code as described in "Result Codes" (page 169) and returns.

For information on parsing the data buffer, see the section "Client Side Buffer Parsing" in Chapter 1.

sGetAttributeValue

Structure received when an Open Directory client calls `dsGetAttributeValue`.

```
typedef struct {
    uInt32                  fType;
    sInt32                  fResult;
    tDirNodeReference       fInNodeRef;
    tDataBufferPtr          fInOutDataBuff;
    unsigned long           fInAttrValueIndex;
    tAttributeValueListRef  fInAttrValueListRef;
    tAttributeValueEntryPtr fOutAttrValue;
} sGetAttributeValue;
```

Fields

`fType`
: Always `kGetAttributeValue`.

`fResult`
: Value of type `sInt32` that the plug-in sets to `eDSNoErr` before returning to indicate that it was able to get the requested attribute value. If an error occurs, the plug-in sets `fResult` to a value listed in "Result Codes" (page 169).

`fInNodeRef`
: Value of type `tDirNodeReference` (page 89) that identifies the directory node of the record whose attribute value is to be obtained. The directory node reference was created when the client application opened the directory node session.

`fInOutDataBuff`
: Value of type `tDataBufferPtr` (page 87) pointing to the `tDataBuffer` (page 84) structure containing information previously obtained when the plug-in responded to an `sGetRecordList`, `sDoAttrValueSearch`, `sDoAttrValueSearchWithData`, `sDoMultiAttrValueSearch`, or `sDOMultiAttrValueSearchWithData` request from the client application.

`fInAttrValueIndex`
: Value of type `unsigned long` containing a one-based index that specifies which attribute value to get. A value of 1 specifies the first value, a value of 2 specifies the second value, and so on.

`fInAttrValueListRef`
: Value of type `tAttributeValueListRef` (page 86) created by the plug-in when its routine for processing `sGetAttributeEntry` (page 109) structures was called. The reference contains information that the plug-in uses to locate the attribute value in the data buffer pointed to by `fInOutDataBuff`.

`fOutAttrValue`
: Value of type `tAttributeValueEntryPtr` (page 86) pointing to the `tAttributeValueEntry` (page 83) structure in which the plug-in is to place the value of the attribute specified by the `fInAttrValueIndex` field.

Discussion

The DirectoryService daemon calls a plug-in's `ProcessRequest` entry point and passes an `sGetAttributeValue` structure when an Open Directory client calls `dsGetAttributeValue`.

The plug-in uses the `fInNodeRef` field of the `sGetAttributeValue` structure to determine the directory node of the record for which an attribute value is being obtained.

If the plug-in can get the requested value from the data buffer pointed to by `fInOutDataBuff`, it puts the value in the attribute value entry structure pointed to by `fOutAttrValue`, sets `fResult` to `eDSNoErr`, and returns.

If the plug-in cannot get the requested value, it sets `fResult` to an appropriate result code as described in "Result Codes" (page 169) and returns.

For information on parsing the data buffer, see the section "Client Side Buffer Parsing" in Chapter 1.

sGetDirNodeInfo

Structure received when an Open Directory client application calls `dsGetDirNodeInfo`.

```
typedef struct {
    uInt32              fType;
    sInt32              fResult;
    tDirNodeReference   fInNodeRef;
    tDataListPtr        fInDirNodeInfoTypeList;
    tDataBufferPtr      fOutDataBuff;
    bool                fInAttrInfoOnly;
    unsigned long       fOutAttrInfoCount;
    tAttributeListRef   fOutAttrListRef;
    tContextData        fOutContinueData;
} sGetDirNodeInfo;
```

Fields

`fType`
: Always `kGetDirNodeInfo`.

`fResult`
: Value of type `sInt32` that the plug-in sets to `eDSNoErr` before returning to indicate that it was able to get information about the directory node identified by `fInNodeRef`. If an error occurs, the plug-in sets `fResult` to a value listed in "Result Codes" (page 169).

`fInNodeRef`
: Value of type `tDirNodeReference` (page 89) that identifies the directory node for which information is to be obtained. The directory node reference was created when the client application opened the directory node.

`fInDirNodeInfoTypeList`
: Value of type `tDataListPtr` (page 87) that points to a `tDataList` (page 84) structure containing the attribute types for which information is being requested.

`fOutDataBuff`
: Value of type `tDataBufferPtr` (page 87) pointing to a `tDataBuffer` (page 84) structure. If the plug-in obtains the requested information, it puts the information in the data buffer pointed to by `fOutDataBuff`.

`fInAttrInfoOnly`
: Boolean value set to `TRUE` if the plug-in is only to provide information about attributes or set to `FALSE` if the plug-in is to provide the values of the attributes as well as information about the attributes.

`fOutAttrInfoCount`
: On return, `fOutAttrInfoCount` contains the number of attribute types the plug-in has placed in the buffer pointed to by `fOutDataBuff`.

`fOutAttrListRef`
: Value of type `tAttributeListRef` (page 86) assigned by the DirectoryService daemon and that the plug-in associates with the internal structure the plug-in uses to maintain information about the reference.

`fOutContinueData`
: Value of type `tContextData` (page 87) that represents continuation data. If this is the first call in what may be a series of calls for this value of fInNodeRef, the input value of `fOutContinueData` is `NULL`. If all of the directory node information fits in the buffer pointed to by `fOutDataBuff`, the plug-in sets `fOutContinueData` to `NULL`. If there is more information than can fit in the buffer, set `fOutContinueData` to a plug-in–defined value.Your routine for processing `sGetDirNodeInfo` structures will be called again, and the `fOutContinueData` field will contain the continuation data that you previously returned to the client application. Therefore, the continuation data should be a value that you can use to determine which directory node information to place in the data buffer the next time your routine for processing `sGetDirNodeInfo` structures is called for this value of `fInNodeRef`. You may want to include a timestamp in the continuation data and return an error if you determine that `fOutContinueData` is out of date.

Discussion

The DirectoryService daemon calls a plug-in's `ProcessRequest` entry point and passes an `sGetDirNodeInfo` structure when an Open Directory client calls `dsGetDirNodeInfo`.

The plug-in uses the `fInNodeRef` field of the `sGetDirNodeInfo` structure to determine the directory node for which information is requested, the data list pointed to by `fInDirNodeInfoTypeList` to determine the type of information that is requested, and `fInAttrInfoOnly` to determine whether to also return attribute values.

If the plug-in can get attribute information for the specified directory node, it puts the requested information in the buffer pointed to by `fOutDataBuff`. If `fOutDataBuff` is too small to hold all of the information, the plug-in sets `fOutContinueData` to a plug-in–defined value. If all of the information fits in the buffer, the plug-in sets `fOutDataBuff` to `NULL`. Before returning, the plug-in sets `fOutAttrInfoCount` to the number of attributes types that have been placed in the buffer.

If the plug-in cannot get the requested information, it sets `fResult` to an appropriate result code as described in "Result Codes" (page 169), sets `fOutContinueData` to `NULL`, sets `fOutAttrInfoCount` to zero, and returns.

sGetRecAttribInfo

Structure received when an Open Directory client calls `dsGetRecordAttributeInfo`.

```
typedef struct {
    uInt32              fType;
    sInt32              fResult;
    tRecordReference    fInRecRef;
    tDataNodePtr        fInAttrType;
    tAttributeEntryPtr  fOutAttrInfoPtr;
} sGetRecAttribInfo;
```

Fields

`fType`

Always `kGetRecordAttributeInfo`.

`fResult`

Value of type `sInt32` that the plug-in sets to `eDSNoErr` before returning to indicate that it was able to get information about the attribute of the record referred to by `fInRecRef`. If an error occurs, the plug-in sets `fResult` to a value listed in "Result Codes" (page 169).

`fInRecRef`

Value of type `tRecordReference` (page 89) that represents the record for which information about an attribute is to be obtained. The plug-in created the value of `fInRecRef` when it was called to process a request to open the record.

`fInAttrType`

Value of type `tDataNodePtr` (page 88) that points to a `tDataNode` (page 88) structure containing the attribute type for which information is requested.

`fOutAttrInfoPtr`

Value of type `tAttributeValueEntryPtr` (page 86) that points to an `tAttributeEntry` (page 82) structure containing the requested attribute information.

Discussion

The DirectoryService daemon calls a plug-in's `ProcessRequest` entry point and passes an `sGetRecAttribInfo` structure when an Open Directory client calls `dsGetRecordAttributeInfo`.

The plug-in uses the `fInRecRef` field of the `sGetRecAttribInfo` structure to determine the record for which information about an attribute is to be obtained and the `fInAttrType` field to determine the attribute type for which attribute information is to be obtained. The information includes the number of values the attribute has, the total number of bytes the values use, the maximum size of a value for the specified attribute, and the attribute's unique signature.

If the plug-in can get the attribute information, it places the information in the attribute entry structure pointed to by `fOutAttrInfoPtr`, sets `fResult` to `eDSNoErr`, and returns.

If the plug-in cannot get the attribute's information, it sets `fResult` to an appropriate result code as described in "Result Codes" (page 169) and returns.

sGetRecordAttributeValueByID

Structure received when an Open Directory client calls `dsGetRecordAttributeValueByID`.

```
typedef struct {
    uInt32                   fType;
    sInt32                   fResult;
    tRecordReference         fInRecRef;
    tDataNodePtr             fInAttrType;
    unsigned long            fInAttrValueID;
    tAttributeValueEntryPtr  fOutEntryPtr;
} sGetRecordAttributeValueByID;
```

Fields

`fType`

Always `kGetRecordAttributeValueByID`.

`fResult`

Value of type `sInt32` that the plug-in sets to `eDSNoErr` before returning to indicate that it was able to get the value of the attribute. If an error occurs, the plug-in sets `fResult` to a value listed in "Result Codes" (page 169).

`fInRecRef`

Value of type `tRecordReference` (page 89) that represents the record for which an attribute value is to be obtained. The plug-in created the value of `fInRecRef` when it was called to process a request to open the record.

`fInAttrType`

Value of type `tDataNodePtr` (page 88) that points to a `tDataNode` (page 88) structure containing the type of attribute for which an attribute value is requested.

`fInAttrValueID`

Value of type `unsigned long` that specifies the attribute value ID of the attribute value that is to be obtained.

`fOutEntryPtr`

Value of type `tAttributeValueEntryPtr` (page 86) that points to an `tAttributeValueEntry` (page 83) structure in which the plug-in places the requested attribute value.

Discussion

The DirectoryService daemon calls a plug-in's `ProcessRequest` entry point and passes an `sGetRecordAttributeValueByID` structure when an Open Directory client calls `dsGetRecordAttributeValueByID` to get the value of an attribute by it attribute value ID.

The plug-in uses the `fInRecRef` field of the `sGetRecordAttributeValueByID` structure to determine the record for which the value of an attribute is to be obtained, the `fInAttrType` field to determine the type of the attribute whose value is to be obtained, and the `fInAttrValueID` field to determine the ID of the attribute value to get.

If the plug-in can get the specified attribute value, it places the value in the attribute value entry structure pointed to by `fOutEntryPtr`, sets `fResult` to `eDSNoErr`, and returns.

If the plug-in cannot get the attribute's value, it sets `fResult` to an appropriate result code as described in "Result Codes" (page 169) and returns.

sGetRecordAttributeValueByIndex

Structure received when an Open Directory client calls `dsGetRecordAttributeValueByIndex`.

```
typedef struct {
    uInt32                  fType;
    sInt32                  fResult;
    tRecordReference        fInRecRef;
    tDataNodePtr            fInAttrType;
    unsigned long           fInAttrValueIndex;
    tAttributeValueEntryPtr fOutEntryPtr;
} sGetRecordAttributeValueByIndex;
```

Fields

`fType`

Always `kGetRecordAttributeValueByIndex`.

`fResult`

Value of type `sInt32` that the plug-in sets to `eDSNoErr` before returning to indicate that it was able to get the value of the attribute. If an error occurs, the plug-in sets `fResult` to a value listed in "Result Codes" (page 169).

`fInRecRef`

Value of type `tRecordReference` (page 89) that represents the record whose attribute value is to be obtained. The plug-in created the value of `fInRecRef` when it was called to process a request to open the record.

`fInAttrType`

Value of type `tDataNodePtr` (page 88) that points to a `tDataNode` (page 88) structure containing the type of the attribute whose value is requested.

`fInAttrValueIndex`

Value of type `unsigned long` that specifies the attribute for which information is to be obtained, using a one-based index.

`fOutEntryPtr`

Value of type `tAttributeValueEntryPtr` (page 86) that points to an `tAttributeValueEntry` (page 83) in which the plug-in is to place the attribute's value.

Discussion

The DirectoryService daemon calls a plug-in's `ProcessRequest` entry point and passes an `sGetRecordAttributeValueByIndex` structure when an Open Directory client calls `dsGetRecordAttributeValueByIndex` to get the value of an attribute by its attribute index.

The plug-in uses the `fInRecRef` field of the `sGetRecordAttributeValueByIndex` structure to determine the record for which the value of an attribute is to be obtained, the `fInAttrType` field to determine the type of the attribute whose value is to be obtained, and the `fInAttrValueIndex` field to determine which attribute value to obtain.

If the plug-in can get the specified attribute value, it places the value in the attribute value entry structure pointed to by `fOutEntryPtr`, sets `fResult` to `eDSNoErr`, and returns.

If the plug-in cannot get the attribute's value, it sets `fResult` to an appropriate result code as described in "Result Codes" (page 169) and returns.

sGetRecordAttributeValueByValue

Structure received when an Open Directory client calls `dsGetRecordAttributeValueByValue`.

```
typedef struct {
    uInt32                  fType;
    sInt32                  fResult;
    tRecordReference        fInRecRef;
    tDataNodePtr            fInAttrType;
    tDataNodePtr            fInAttrValue;
    tAttributeValueEntryPtr fOutEntryPtr;
} sGetRecordAttributeValueByValue;
```

Fields

`fType`

Always `kGetRecordAttributeValueByValue`.

`fResult`

Value of type `sInt32` that the plug-in sets to `eDSNoErr` before returning to indicate that it was able to get the value of the attribute. If an error occurs, the plug-in sets `fResult` to a value listed in "Result Codes" (page 169).

`fInRecRef`

Value of type `tRecordReference` (page 89) that represents the record whose attribute value is to be obtained. The plug-in created the value of `fInRecRef` when it was called to process a request to open the record.

`fInAttrType`

Value of type `tDataNodePtr` (page 88) that points to a `tDataNode` (page 88) structure containing the type of the attribute whose value is requested.

`fInAttrValue`

Value of type `tDataNodePtr` (page 88) that specifies the value that is to be obtained.

`fOutEntryPtr`

Value of type `tAttributeValueEntryPtr` (page 86) that points to an `tAttributeValueEntry` (page 83) in which the plug-in is to place the attribute's value.

Discussion

The DirectoryService daemon calls a plug-in's `ProcessRequest` entry point and passes an `sGetRecordAttributeValueByValue` structure when an Open Directory client calls `dsGetRecordAttributeValueByValue` to get the value of an attribute by its value.

The plug-in uses the `fInRecRef` field of the `sGetRecordAttributeValueByValue` structure to determine the record for which the value of an attribute is to be obtained, the `fInAttrType` field to determine the type of the attribute whose value is to be obtained, and the `fInAttrValue` field to determine which attribute value to obtain.

If the plug-in can get the specified attribute value, it places the value in the attribute value entry structure pointed to by `fOutEntryPtr`, sets `fResult` to `eDSNoErr`, and returns.

If the plug-in cannot get the attribute's value, it sets `fResult` to an appropriate result code as described in "Result Codes" (page 169) and returns.

sGetRecordEntry

Structure received when an Open Directory client calls `dsGetRecordEntry`.

```
typedef struct {
    uInt32              fType;
    sInt32              fResult;
    tDirNodeReference   fInNodeRef;
    tDataBufferPtr      fInOutDataBuff;
    unsigned long       fInRecEntryIndex;
    tAttributeListRef   fOutAttrListRef;
    tRecordEntryPtr     fOutRecEntryPtr;
} sGetRecordEntry;
```

Fields

`fType`
: Always `kGetRecordEntry`.

`fResult`
: Value of type `sInt32` that the plug-in sets to `eDSNoErr` before returning to indicate that it was able to get the record entries for the directory node identified by `fInNodeRef`. If an error occurs, the plug-in sets `fResult` to a value listed in "Result Codes" (page 169).

`fInNodeRef`
: Value of type `tDirNodeReference` (page 89) that identifies the directory node for which the record entry is to be obtained. The directory node reference was created when the client application opened the directory node.

`fInOutDataBuff`
: Value of type `tDataBufferPtr` (page 87) that points to the `tDataBuffer` (page 84) structure from which the record entry is to be obtained.

`fInRecEntryIndex`
: Value of type unsigned long that specifies the record to get. The `fInRecEntryIndex` field contains a value that is a one-based index.

`fOutAttrListRef`
: Value of type `tAttributeListRef` (page 86) assigned by the DirectoryService daemon and that the plug-in associates with the internal structure the plug-in uses to maintain information about the reference.

`fOutRecEntryPtr`
: Value of type `tRecordEntryPtr` (page 89) that points to a `tRecordEntry` (page 85) structure containing the requested record.

Discussion

The DirectoryService daemon calls a plug-in's `ProcessRequest` entry point and passes an `sGetRecordEntry` structure when an Open Directory client calls `dsGetRecordEntry` to get information about a record.

The plug-in gets information about the record from the buffer pointed to by `fInOutDataBuff`. The record information consists of the record's name, type, and number of attributes. The buffer pointed to by `fInOutDataBuff` was previously filled in by the plug-in when the plug-in's `ProcessRequest` routine responded to the receipt of an `sGetRecordList` structure.

The plug-in verifies that the directory node reference provided in the `fInNodeRef` field is valid. If the directory node reference is valid, the plug-in uses the `fInRecEntryIndex` field to determine the record for which record information is to be obtained, places the information in the record entry structure pointed to by the `fOutRecEntryPtr` field, and places the record's attribute information in the attribute list referred to by `tOutAttrListRef`. Before returning, the plug-in sets `fResult` to `eDSNoErr`.

If the plug-in cannot get the requested information, it sets `fResult` to an appropriate result code as described in "Result Codes" (page 169) and returns.

For information on parsing the data buffer, see the section “Client Side Buffer Parsing” in Chapter 1.

sGetRecordList

Structure called when an Open Directory client calls `dsGetRecordList`.

```
typedef struct {
    uInt32              fType;
    sInt32              fResult;
    tDirNodeReference   fInNodeRef;
    tDataBufferPtr      fInDataBuff;
    tDataListPtr        fInRecNameList;
    tDirPatternMatch    fInPatternMatch;
    tDataListPtr        fInRecTypeList;
    tDataListPtr        fInAttribTypeList;
    bool                fInAttribInfoOnly;
    unsigned long       fOutRecEntryCount;
    tContextData        fIOContinueData;
} sGetRecordList;
```

Fields

`fType`

Always `kGetRecordList`.

`fResult`

Value of type `sInt32` that the plug-in sets to `eDSNoErr` before returning to indicate that it was able to get the requested list of records for the node identified by `fInNodeRef`. If an error occurs, the plug-in sets `fResult` to a value listed in “Result Codes” (page 169). If no matches are found, the plug-in should set `fResult` to `eDSNoErr`, fOutRecEntryCount to zero, and `fIOContinueData` to `NULL`.

`fInNodeRef`

Value of type `tDirNodeReference` (page 89) that identifies the directory node for which the record list is to be obtained. The directory node reference was created when the client application opened the directory node.

`fInDataBuff`

Value of type `tDataBufferPtr` (page 87) pointing to the `tDataBuffer` (page 84) structure in which the plug-in is to return the record list.

`fInRecNameList`

Value of type `tDataListPtr` (page 87) that points to a `tDataList` (page 84) structure containing patterns in UTF-8 encoding that are to be compared with record names. If `fInRecNameList` is `kDSRecordsAll`, the plug-in should ignore `fInPatternMatch` and include all records for the directory node identified by `fInNodeRef`.

`fInPatternMatch`

Value of type `tDirPatternMatch` that describes the way in which the patterns specified by `fInRecNameList` are to be compared. See Pattern Matching Constants (page 137) for possible constants. The pattern match type may also be a type defined by the Open Directory plug-in that handles the directory system represented by `inDirReference`.

`fInRecTypeList`

Value of type `tDataListPtr` (page 87) that points to a`tDataList` (page 84) structure containing the types of records to get. See Standard Record Types (page 143) and Meta Record Type Constants (page 142) for possible values.

`fInAttribTypeList`
: Value of type `tDataListPtr` (page 87) that points to a `tDataList` (page 84) structure containing the attribute types of records to get. See the attribute constants described in the "Constants" (page 129) section for possible values.

`fInAttribInfoOnly`
: Value of type `bool`. If `fInAttribInfoOnly` is `TRUE`, the plug-in should include in the buffer pointed to by `fInDataBuff` attribute information for matching records. If `fInAttribInfoOnly` is `FALSE`, the plug-in should include in the buffer pointed to by `fInDataBuff` attribute information as well as attribute values for matching records.

`fOutRecEntryCount`
: Value of type `unsigned long.` The first time the client application calls `dsGetRecordList`, `fOutRecEntryCount` is zero to receive all matching records or is a positive integer value that specifies the total number of records the client application wants to receive across what may be a series of `dsGetRecordList` calls. If the latter, the plug-in should use the initial input value of `fOutRecEntryCount` to limit the total number of matching records it returns. Before returning, the plug-in should set `fOutRecEntryCount` to the number of records it has placed in the buffer pointed to by `fInDataBuff`. The plug-in should ignore the input value of `fOutRecEntryCount` whenever it is processing a `sGetRecordList` structure that has an `fIOContinueData` field that is not `NULL`.

`fIOContinueData`
: Value of type `tContextData` (page 87) containing continuation data. For the first in a series of calls to `dsGetRecordList`, the input value is `NULL`. If the plug-in can store all of the matching records in the buffer pointed to by `fInDataBuff`, it sets `fIOContinueData` to `NULL` before returning. If there more records than can be stored in the buffer, the plug-in stores as much data as possible and sets `fIOContinueData` to a plug-in–defined value that the plug-in can use when the client application calls `dsGetRecordList` again to get another buffer of data. You may want to include a timestamp in the continuation data and return an error if you determine that `fOutContinueData` is out of date.

Discussion

The DirectoryService daemon calls a plug-in's `ProcessRequest` entry point and passes an `sGetRecordList` structure when an Open Directory client calls `dsGetRecordList` to get a list of records for a directory node.

The plug-in uses the `fInNodeRef` field of the `sGetRecordList` structure to determine the directory node for which the record list is requested, the data list pointed to by fInRecNameList to get the names of records for which information is requested, the data list pointed to by `fInRecTypeList` to determine the types of records for which information is requested, and the data list pointed to by `fInAttributeTypeList` to determine the attributes for which information is requested. The plug-in should return only those records whose names match the pattern specified by fInRecNameList. The value of the `fInAttributeInfoOnly` field determines whether the plug-in should also return attribute values.

Depending on the size of the data buffer pointed to by `fInDataBuff` and the length of the list of records, the plug-in's routine for processing `sGetRecordList` structures may be called multiple times in order to return the complete list. The first time the plug-in's routine for processing `sGetRecordList` structures is called, the input value of `fIOContinueData` is `NULL` and input value of `fInOutRecEntryCount` specifies the total number of records that the plug-in should return even if the plug-in's routine for processing `sGetRecordList` structures must be called more than once .

If there are records that match the criteria specified by `fInRecNameList`, `fInPatternMatch`, `fInRecTypeList`, and `fInAttributeTypeList`, plug-in puts the record entries, attribute entries, and attribute values (if `fInAttributeInfoOnly` is `FALSE`) in the buffer pointed to by `fInDataBuff`. It also sets `fInOutRecEntryCount` to the number of records that have been placed in `fInDataBuff` and sets `fResult to eDSNoErr`. If the buffer pointed to by `fInDataBuff` is too small to hold all of the records, the plug-in sets fIOContinueData to a plug-in–defined value that the plug-in can use when the client application calls dsGetRecordList again to get another buffer of data. If the buffer pointed to by fInDataBuff contains all of the records or contains the last records in the record list, the plug-in sets fIOContinueData to NULL.

If the plug-in returns before it can get records to place in the buffer pointed to by fInDataBuff, it should set fOutRecEntryCount to zero, set fResult to eDSNoErr, set fIOContinueData to a plug-in–defined value that is not NULL. These settings indicate to the client application that it should call dsGetRecordList again to get the records.

If there are no matching records, the plug-in sets fOutRecEntryCount to zero, fIOContinueData to NULL, and fResult to eDSNoErr, and returns.

sGetRecRefInfo

Structure received when an Open Directory client calls dsGetRecordReferenceInfo.

```
typedef struct {
    uInt32              fType;
    sInt32              fResult;
    tRecordReference    fInRecRef;
    tRecordEntryPtr     fOutRecInfo;
} sGetRecRefInfo;
```

Fields

fType

Always kGetRecordRefInfo.

fResult

Value of type sInt32 that the plug-in sets to eDSNoErr before returning to indicate that it was able to get information for the record reference specified by fInRecRef. If an error occurs, the plug-in sets fResult to a value listed in "Result Codes" (page 169).

fInRecRef

Value of type tRecordReference (page 89) that specifies the record for which information is to be obtained. The plug-in created the value of fInRecRef when it was called to process a request to open the record.

fOutRecInfo

Value of type tRecordEntryPtr (page 89) that points to a tRecordEntry (page 85) structure containing the requested information.

Discussion

The DirectoryService daemon calls a plug-in's ProcessRequest entry point and passes an sGetRecRefInfo structure when an Open Directory client calls dsGetRecordReferenceInfo to get information about a record.

The plug-in uses the fInRecRef field of the sGetRecRefInfo structure to determine the record for information is to be obtained. The information consists of the record's name, type, and the number of attributes the record has.

If the plug-in can get the record's information, it places the information in the record entry structure pointed to by fOutRecInfo, sets fResult to eDSnoErr, and returns.

If the plug-in cannot get the record's information, it sets fResult to an appropriate result code as described in "Result Codes" (page 169) and returns.

sHeader

Structure for passing the DirectoryService daemon's run loop and the Kerberos mutex.

```
typedef struct {
    uInt32              fType;
    sInt32              fResult;
    tContextData        fContextData;
} sHeader;
```

Fields

`fType`
: `kServerRunLoop` or `kKerberosMutex`.

`fResult`
: Value of type `sInt32` that the plug-in sets to `eDSNoErr` before returning. If an error occurs, the plug-in sets `fResult` to a value listed in "Result Codes" (page 169).

`fContextData`
: Value of type `tContextData` (page 87) containing the run loop or the Kerberos mutex.

Discussion

The DirectoryService daemon calls the plug-in's `ProcessRequest` entry point once after the plug-in has been loaded and initialized to pass in the `fContextData` field the CFRunloop for the currently executing process. You can use the run loop to set up timers as an alternative to using the `PeriodicTask` entry point for setting timers.

Here is an example that gets the run loop from the `fContextData` field:

```
if ( ((sHeader *)inData)->fType == kServerRunLoop)
{
    if ( (((sHeader *)inData)->fContextData) != nil )
    {
        fServerRunLoop = (CFRunLoopRef)(((sHeader *)inData)->fContextData);
    }
}
```

The `sHeader` structure is also used to pass the Kerberos mutex, a value of type `DSMutexSempaphore`, immediately after the run loop is passed.

sOpenDirNode

Structure received when an Open Directory client calls `dsOpenDirNode`.

```
typedef struct {
    uInt32          fType;
    sInt32          fResult;
    tDirReference   fInDirRef;
    tDataListPtr    fInDirNodeName;
    tDirNodeReferencefOutNodeRef;
    uid_t           fInUID;
    uid_t           fInEffectiveUID;
} sOpenDirNode;
```

Fields

`fType`
: Always `kOpenDirNode`.

`fResult`
: Value of type `sInt32` that the plug-in sets to `eDSNoErr` before returning to indicate that it was able to open the directory node specified by `fInDirNodeName`. If an error occurs, the plug-in sets `fResult` to a value listed in "Result Codes" (page 169).

`fInDirRef`
: Value of type `tDirReference` (page 89) that was created when the client application opened the Open Directory session for which this directory node is to be opened.

`fInDirNodeName`
: Value of type `tDataListPtr` (page 87) pointing to a `tDataList` (page 84) structure containing the name of the directory node that is to be opened.

`fOutNodeRef`
: Value of type `tDirNodeReference` (page 89) assigned by the DirectoryService daemon and that the plug-in associates with the internal structure the plug-in uses to maintain information about the reference.

`fInUID`
: Value of type `uid_t` containing the UID of the calling process. Your plug-in can use the value of `fInUID` and `fInEffectiveUID` to determine whether to allow a process to perform certain activities without requiring authentication.

`fInEffectiveUID`
: Value of type `uid_t` containing the effective UID of the calling process. Your plug-in can use the value of `fInEffectiveUID` and `fInUID` to determine whether to allow a process to perform certain activities without requiring authentication.

Discussion

The DirectoryService daemon calls a plug-in's `ProcessRequest` entry point and passes an `sOpenDirNode` structure when an Open Directory client calls `dsOpenDirNode` to open a directory node.

The plug-in uses the `fInDirNodeName` field to determine the name of the directory node to open.

If the plug-in can open the specified directory node, it sets `fResult` to `eDSNoErr` and returns.

If the plug-in cannot open the directory node or if the Open Directory reference is invalid, the plug-in sets `fResult` to an appropriate result code as described in "Result Codes" (page 169) and returns.

sOpenRecord

Structure received when an Open Directory client calls `dsOpenRecord`.

```
typedef struct {
    uInt32              fType;
    sInt32              fResult;
    tDirNodeReference   fInNodeRef;
    tDataNodePtr        fInRecType;
    tDataNodePtr        fInRecName;
    tRecordReference    fOutRecRef;
} sOpenRecord;
```

Fields

`fType`
: Always `kOpenRecord`.

`fResult`
: Value of type `sInt32` that the plug-in sets to `eDSNoErr` before returning to indicate that it was able to open the specified record. If an error occurs, the plug-in sets `fResult` to a value listed in "Result Codes" (page 169).

`fInNodeRef`
: Value of type `tDirNodeReference` (page 89) that identifies the directory node of the record that is to be opened. The directory node reference was created when the client application opened the directory node.

`fInRecType`
: Value of type `tDataNodePtr` (page 88) that points to a `tDataNode` (page 88) structure containing the type of the record that is to be opened.

`fInRecName`
: Value of type `tDataNodePtr` (page 88) that points to a `tDataNode` (page 88) structure containing the name of the record that is to be opened.

`fOutRecRef`
: Value of type `tRecordReference` (page 89) assigned by the DirectoryService daemon and that the plug-in associates with the internal structure the plug-in uses to maintain information about the reference.

Discussion

The DirectoryService daemon calls a plug-in's `ProcessRequest` entry point and passes an `sOpenRecord` structure when an Open Directory client calls `dsOpenRecord`.

The plug-in uses the `fInNodeRef` field of the `sOpenRecord` structure to determine the directory node of the record that is to be opened, the `fInRecType` field to determine the type of the record that this is to be opened, and the `fInRecName` field to determine the name of the record that is to be opened.

If the plug-in can open the record, it sets `fResult` to `eDSNoErr`, and returns. Later, when the client application calls Open Directory functions that operate on the opened record, the record reference will be passed to the plug-in, which should use the record reference to identify the record.

If the plug-in cannot open the record, it should set `fResult` to an appropriate result code as described in "Result Codes" (page 169) and return.

sReleaseContinueData

Structure received when an Open Directory client calls `dsReleaseContinueData`.

```
typedef struct {
    uInt32          fType;
    sInt32          fResult;
    tDirReference   fInDirReference;
    tContextData    fInContinueData;
} sReleaseContinueData;
```

Fields

`fType`
: Always `kReleaseContinueData`.

`fResult`
: Value of type `sInt32` that the plug-in sets to `eDSNoErr` before returning to indicate that it was able to deallocate the memory associated with `fInContinueData`. If an error occurs, the plug-in set `fResult` to a value listed in "Result Codes" (page 169).

`fInDirReference`
: Value of type `tDirReference` (page 89) or of type `tDirNodeReference` (page 89), depending on the type of reference that was used in the call that created the continue data that is to be released.

`fInContinueData`
: Value of type `tContextData` (page 87) that points to memory that is to be released.

Discussion

The DirectoryService daemon calls a plug-in's `ProcessRequest` entry point and passes an `sReleaseContinueData` structure when an Open Directory client calls `dsReleaseContinueData`.

The plug-in deallocates the memory associated with `fInContinueData`, sets `fResult` to `eDSNoErr`, and returns.

If the plug-in cannot deallocate the memory associated with `fInContinueData`, it sets `fResult` to an appropriate result code as described in "Result Codes" (page 169) and returns.

sRemoveAttribute

Structure received when an Open Directory client calls `dsRemoveAttribute`.

```
typedef struct {
    uInt32          fType;
    sInt32          fResult;
    tRecordReferencefInRecRef;
    tDataNodePtr    fInAttribute;
} sRemoveAttribute;
```

Fields

`fType`
: Always `kRemoveAttribute`.

`fResult`
: Value of type `sInt32` that the plug-in sets to `eDSNoErr` before returning to indicate that it was able to remove the attribute. If an error occurs, the plug-in sets `fResult` to a value listed in "Result Codes" (page 169).

`fInRecRef`
: Value of type `tRecordReference` (page 89) representing the record from which the attribute is to be removed.

`fInAttribute`
: Value of type `tDataNodePtr` (page 88) that points to a `tDataNode` (page 88) structure containing the name of the attribute that is to be removed.

Discussion

The DirectoryService daemon calls a plug-in's `ProcessRequest` entry point and passes an `sRemoveAttribute` structure when an Open Directory client calls `dsRemoveAttribute` to remove an attribute from a record.

The plug-in uses the `fInRecRef` field of the `sRemoveAttribute` structure to determine the record from which an attribute is to be removed and the `fInAttribute` field to determine the name of the attribute that is to be removed.

If the plug-in can remove the attribute, it removes the attribute and all of its values, invalidates any attribute list references that may be active for this attribute, sets `fResult` to `eDSNoErr`, and returns. After returning, the plug-in responds with an error to any calls of its `ProcessRequest` entry point that provide a pointers to an attribute entry structure or an attribute value entry structure for the removed attribute.

If the plug-in cannot remove the attribute, it sets `fResult` to an appropriate result code as described in "Result Codes" (page 169) and returns.

sRemoveAttributeValue

Structure received when an Open Directory client calls `dsRemoveAttributeValue`.

```
typedef struct {
    uInt32              fType;
    sInt32              fResult;
    tRecordReference    fInRecRef;
    tDataNodePtr        fInAttrType;
    unsigned long       fInAttrValueID;
} sRemoveAttributeValue;
```

Fields

`fType`

Always `kRemoveAttributeValue`.

`fResult`

Value of type `sInt32` that the plug-in sets to `eDSNoErr` before returning to indicate that it was able to remove the value from the attribute. If an error occurs, the plug-in sets `fResult` to a value listed in "Result Codes" (page 169).

`fInRecRef`

Value of type `tRecordReference` (page 89) representing the record for which a value is to be removed to an attribute.

`fInAttrType`

Value of type `tDataNodePtr` (page 88) that points to a `tDataNode` (page 88) structure containing the type the attribute from which a value is to be removed.

`fInAttrValueID`

Value of type `unsigned long` that specifies the ID of the value that is to be removed.

Discussion

The DirectoryService daemon calls a plug-in's `ProcessRequest` entry point and passes an `sRemoveAttributeValue` structure when an Open Directory client calls `dsRemoveAttributeValue` to remove a value from an attribute.

The plug-in uses the `fInRecRef` field of the `sRemoveAttributeValue` structure to determine the record for which a value is to be removed from an attribute, the `fInAttrType` field to determine the type of the attribute from which a value is to be removed, and the `fInAttrValueID` field to determine which attribute value to remove.

If the plug-in can remove the specified value from the specified attribute, it removes the attribute, invalidates any attribute value list references for the removed value, sets `fResult` to `eDSNoErr`, and returns. After returning, the plug-in responds with an error to any calls of its `ProcessRequest` entry point that provide a pointer to an attribute value entry structure for the removed attribute value.

If the plug-in cannot add the attribute value, it sets `fResult` to an appropriate result code as described in "Result Codes" (page 169) and returns.

sSetAttributeValue

Structure received when an Open Directory client calls dsSetAttributeValue.

```
typedef struct {
    uInt32                  fType;
    sInt32                  fResult;
    tRecordReference        fInRecRef;
    tDataNodePtr            fInAttrType;
    tAttributeValueEntryPtr fInAttrValueEntry;
} sSetAttributeValue;
```

Fields

fType

Always kSetAttributeValue.

fResult

Value of type sInt32 that the plug-in sets to eDSNoErr before returning to indicate that it was able to set the specified value in the attribute. If an error occurs, the plug-in sets fResult to a value listed in "Result Codes" (page 169).

fInRecRef

Value of type tRecordReference (page 89) representing the record for which a value is to be set in an attribute. The record reference was created when the plug-in processed an sOpenRecord (page 122) structure.

fInAttrType

Value of type tDataNodePtr (page 88) that points to a tDataNode (page 88) structure containing the type the attribute whose value is to be set.

fInAttrValueEntry

Value of type tAttributeValueEntryPtr (page 86) that points to a tAttributeValueEntry (page 83) structure containing the value that is to be set and its attribute value ID.

Discussion

The DirectoryService daemon calls a plug-in's ProcessRequest entry point and passes an sSetAttributeValue structure when an Open Directory client calls dsSetAttributeValue to set an attribute's value.

The plug-in uses the fInRecRef field of the sSetAttributeValue structure to determine the record for which an attribute value is to be set and the fInAttrType field to determine the type of the attribute whose value is to be set. The fInAttrValueEntry field contains a pointer to a tAttributeValueEntry (page 83) structure whose fAttributeValueID field identifies which value is to be replaced and whose fAttributeValueData field contains the new value.

If the plug-in can set the attribute value, it sets the value, sets fResult to eDSNoErr, and returns.

If the plug-in cannot set the attribute value, it sets fResult to an appropriate result code as described in "Result Codes" (page 169) and returns.

sSetAttributeValues

Structure received when an Open Directory client calls dsSetAttributeValues.

```
typedef struct {
    uInt32              fType;
    sInt32              fResult;
    tRecordReference    fInRecRef;
    tDataNodePtr        fInAttrType;
    tDataListPtr        fInAttrValueList;
} sSetAttributeValues;
```

Fields

`fType`
: Always `kSetAttributeValues`.

`fResult`
: Value of type `sInt32` that the plug-in sets to `eDSNoErr` before returning to indicate that it was able to set the specified value in the attribute. If an error occurs, the plug-in sets `fResult` to a value listed in "Result Codes" (page 169).

`fInRecRef`
: Value of type `tRecordReference` (page 89) representing the record for which a value is to be set in an attribute. The record reference was created when the plug-in processed an `sOpenRecord` (page 122) structure.

`fInAttrType`
: Value of type `tDataNodePtr` (page 88) that points to a `tDataNode` (page 88) structure containing the type the attribute whose value is to be set.

`fInAttrValueList`
: Value of type `tAttributeValueEntryPtr` (page 86) that points to a `tAttributeValueEntry` (page 83) structure containing the attribute ID of the attribute whose values are to be replaced and a list of the replacement values.

Discussion

The DirectoryService daemon calls a plug-in's `ProcessRequest` entry point and passes an `sSetAttributeValues` structure when an Open Directory client calls `dsSetAttributeValues` to replace multiple values for the specified attribute. If the attribute does not exist, the plug-in creates the attribute and sets it to the values specified by `fInAttrValueList`.

The plug-in uses the `fInRecRef` field of the `sSetAttributeValues` structure to determine the record for which an attribute value is to be set and the `fInAttrType` field to determine the type of the attribute for which values are to be set. The `fInAttrValueList` field points to a `tDataList` (page 84) structure containing a list of values that are to be set for the attribute.

If the plug-in can set the attribute values, it sets the values, sets `fResult` to `eDSNoErr`, and returns.

If the plug-in cannot set the attribute values, it sets `fResult` to an appropriate result code as described in "Result Codes" (page 169) and returns.

sSetRecordName

Structure received when an Open Directory client calls `dsSetRecordName`.

```
typedef struct {
    uInt32          fType;
    sInt32          fResult;
    tRecordReferencefInRecRef;
    tDataNodePtr    fInNewRecName;
} sSetRecordName;
```

Fields

`fType`

Always `kSetRecordName`.

`fResult`

Value of type `sInt32` that the plug-in sets to `eDSNoErr` before returning to indicate that it was able to set the record's name. If an error occurs, the plug-in sets `fResult` to a value listed in "Result Codes" (page 169).

`fInRecRef`

Value of type `tRecordReference` (page 89) representing the record whose name is to be set. The plug-in created the value of `fInRecRef` when it was called to process a request to open the record.

`fInNewRecName`

Value of type `tDataNodePtr` (page 88) that points to a `tDataNode` (page 88) structure containing the name in UTF-8 encoding that is to be set.

Discussion

The DirectoryService daemon calls a plug-in's `ProcessRequest` entry point and passes an `sSetRecordName` structure when an Open Directory client calls `dsSetRecordName` to set the name of a record.

The plug-in uses the `fInRecRef` field of the `sSetRecordName` structure to determine the record whose name is to be set.

If the plug-in can set the new name, it sets the new name, sets `fResult` to `eDSNoErr`, and returns.

If the plug-in cannot set the new name, it sets `fResult` to an appropriate result code as described in "Result Codes" (page 169) and returns.

sSetRecordType

Structure received when an Open Directory client application calls `dsSetRecordType`.

```
typedef struct {
    uInt32              fType;
    sInt32              fResult;
    tRecordReference    fInRecRef;
    tDataNodePtr        fInNewRecType;
} sSetRecordType;
```

Fields

`fType`

Always `kSetRecordType`.

`fResult`

Value of type `sInt32` that the plug-in sets to `eDSNoErr` before returning to indicate that it was able to set the record's type. If an error occurs, the plug-in sets `fResult` to a value listed in "Result Codes" (page 169).

`fInRecRef`
: Value of type `tRecordReference` (page 89) representing the record whose type is to be set. The plug-in created the value of `fInRecRef` when it was called to process a request to open the record.

`fInNewRecType`
: Value of type `tDataNodePtr` (page 88) that points to a `tDataNode` (page 88) structure containing the type that is to be set.

Discussion

The DirectoryService daemon calls a plug-in's `ProcessRequest` entry point and passes an `sSetRecordType` structure when an Open Directory client calls `dsSetRecordType` to set a record's type.

The plug-in uses the `fInRecRef` field of the `sSetRecordType` structure to determine the record whose type is to be set.

If the plug-in can set the new type, it sets the record to the new type, sets `fResult to eDSNoErr`, and returns.

If the plug-in cannot set the new type, it sets `fResult` to an appropriate result code as described in "Result Codes" (page 169) and returns.

Note that this structure is deprecated in Mac OS X 10.3 and later and is not supported on LDAP.

Constants

Authentication Constants

Authentication constants.

```
#define kDSStdAuthMethodPrefix "dsAuthMethodStandard:"
#define kDSNativeAuthMethodPrefix "dsAuthMethodNative:"
#define kDSSetPasswordBestOf "dsSetPasswdBestOf"
#define kDSValueAuthAuthorityDefault "kDSValueAuthAuthorityBasic"
#define kDSValueAuthAuthorityBasic ";basic;"
#define kDSTagAuthorityBasic ";basic;"
#define kDSValueAuthAuthorityLocalWindowsHash ";LocalWindowsHash;"
#define kDSTagAuthAuthorityLocalWindowsHash "LocalWindowsHash"
#define kDSValueAuthAuthorityShadowHash ";ShadowHash;"
#define kDSTagAuthAuthoridyShadowHash "ShadowHash"
#define kDSTagAuthAuthorityBetterHashOnly "BetterHashOnly"
#define kDSValueAuthAuthorityPasswordServerPrefix ";ApplePasswordServer;"
#define kDSTagAuthAuthorityPasswordServer "ApplePasswordServer"
#define kDSValueAuthAuthorityKerberosv5 ";Kerberosv5;"
#define kDSTagAuthAuthorityKerberosv5 "Kerberosvr5"
#define kDSValueAuthAuthorityLocalCachedUser ";LocalCachedUser;"
#define kDSTagAuthAuthorityLocalCachedUser "LocalCachedUser"
#define kDSValueAuthAuthorityDisabledUser ";DisabledUser;"
#define kDSTagAuthAuthorityDisabledUser "DisabledUser"
#define kDSValueNonCryptPasswordMarker "********"
```

Constants

`kDSStdAuthMethodPrefix`
: Prefix defined for standard authentication methods.

`kDSNativeAuthMethodPrefix`
: Prefix defined for native authentication methods.

`kDSSetPasswdBestOf`
: Not used; retained for backward compatibility only.

`kDSValueAuthAuthorityDefault`
: The default value to use for the `kDSNAttrAuthenticationAuthority` attribute. Set this attribute before creating a user record. By default, the value of this attribute is `kDSValueAuthAuthorityBasic`.

`kDSValueAuthAuthorityBasic`
: Standard authentication authority value for basic (crypt) authentication.

`kDSTagAuthAuthorityBasic`
: Standard center tag data of the authentication authority value for basic (crypt) authentication.

  Available in Mac OS X v10.2 and later.

`kDSValueAuthAuthorityLocalWindowsHash`
: Standard authentication authority value for Local Windows Hash authentication; retained for backward compatibility only.

`kDSTagAuthAuthorityLocalWindowsHash`
: Standard center tag data of the authentication authority value for Local Windows Hash authentication. Available in Mac OS X v10.2 and later but retained for backward compatibility only in Mac OS X v10.3 and later.

`kDSValueAuthAuthorityShadowHash`
: Standard authentication authority value for ShadowHash authentication.

  Available in Mac OS X v10.2 and later.

`kDSTagAuthAuthorityShadowHash`
: Standard center tag data of the authentication authority value for ShadowHash authentication.

  Available in Mac OS X v10.3 and later.

`kDSTagAuthAuthorityBetterHashOnly`
: Used as authentication authority data with Shadow Hash authentication authority. Available in Mac OS X v10.3 and later. Superseded in Mac OX X version 10.4 by specifying customized hash lists. For details, see the section "Shadow Hash Authentication" in Chapter 1, "Concepts."

`kDSValueAuthAuthorityPasswordServerPrefix`
: Standard authentication authority value for Apple Password Server authentication.

`kDSTagAuthAuthorityPasswordServer`
: Standard center tag data of the authentication authority value for Apple Password Server authentication.

  Available in Mac OS X v10.3 and later.

`kDSValueAuthAuthorityKerberosv5`
: Standard authentication authority value for Kerberos version 5 authentication.

  Available in Mac OS X v10.3 and later.

`kDSTagAuthAuthorityKerberosv5`
: Tag form of the Kerberos version 5 authentication type.

  Available in Mac OS X v10.3 and later.

`kDSValueAuthAuthorityLocalCachedUser`
: Standard authentication authority value for Local Cached User authentication.

  Available in Mac OS X v10.3 and later.

`kDSTagAuthAuthorityLocalCachedUser`
: Standard center tag data of the authentication authority value for Local Cached User authentication.

  Available in Mac OS X v10.3 and later.

`kDSValueAuthAuthorityDisabledUser`
: Standard authentication authority value for Disabled User authentication.

  Available in Mac OS X v10.2 and later.

`kDSTagAuthAuthorityDisabledUser`
: Standard center tag data of the authentication authority value for Disabled User authentication.

  Available in Mac OS X v10.2 and later.

`kDSValueNonCryptPasswordMarker`
: Marker used for password attribute value to indicate non-crypt authentication.

Declared In
`DirectoryService/DirServicesConst.h`

Authentication Methods

Constants defined for authentication methods.

```
#define kDSStdAuth2WayRandom "dsAuthMethodStandard:dsAuth2WayRandom"
#define kDSStdAuth2WayRandomChangePasswd
"dsAuthMethodStandard:dsAuth2WayRandomChangePasswd"
#define kDSStdAuthAPOP "dsAuthMethodStandard:dsAuthAPOP"
#define kDSStdAuthCHAP "dsAuthMethodStandard:dsAuthCHAP"
#define kDSStdAuthCRAM_MD5 "dsAuthMethodStandard:dsAuthNodeCRAM-MD5"
#define kDSStdAuthChangePasswd "dsAuthMethodStandard:dsAuthChangePasswd"
#define kDSStdAuthClearText "dsAuthMethodStandard:dsAuthClearText"
#define kDSStdAuthCrypt "dsAuthMethodStandard:dsAuthCrypt"
#define kDSStdAuthDIGEST_MD5 "dsAuthMethodStandard:dsAuthNodeDIGEST-MD5"
#define kDSStdAuthDeleteUser "dsAuthMethodStandard:dsAuthDeleteUser"
#define kDSStdAuthGetEffectivePolicy  "dsAuthMethodStandard:dsAuthGetEffectivePolicy"
#define kDSStdAuthGetGlobalPolicy "dsAuthMethodStandard:dsAuthGetGlobalPolicy"
#define kDSStdAuthGetKerberosPrincipal
"dsAuthMethodStandard:dsAuthGetKerberosPrincipal"
#define kDSStdAuthGetPolicy "dsAuthMethodStandard:dsAuthGetPolicy"
#define kDSStdAuthGetUserData "dsAuthMethodStandard:dsAuthGetUserData"
#define kDSStdAuthGetUserName "dsAuthMethodStandard:dsAuthGetUserName"
#define kDSStdAuthMASKE_A "dsAuthMethodStandard:dsAuthMASKE-A"
#define kDSStdAuthMASKE_B "dsAuthMethodStandard:dsAuthMASKE-B"
#define kDSStdAuthMPPEMasterKeys "dsAuthMethodsStandard:dsAuthMPPEMasterKeys"
#define kDSStdAuthMSCHAP1 "dsAuthMethodStandard:dsAuthMSCHAP1"
#define kDSStdAuthMSCHAP2 "dsAuthMethodStandard:dsAuthMSCHAP2"
#define kDSStdAuthNTLMv2 "dsAuthMethodsStandard:dsAuthNodeNTLMv2"
#define kDSStdAuthNewUser "dsAuthMethodStandard:dsAuthNewUser"
#define kDSStdAuthNewUserWithPolicy  "dsAuthMethodsStandard:dsAuthNewUserWithPolicy"
#define kDSStdAuthNodeNativeClearTextOK
"dsAuthMethodStandard:dsAuthNodeNativeCanUseClearText"
#define kDSStdAuthNodeNativeNoClearText
"dsAuthMethodStandard:dsAuthNodeNativeCannotUseClearText"
#define kDSStdAuthReadSecureHash "dsAuthMethodStandard:dsAuthReadSecureHash"
#define kDSStdAuthSMBWorkStationCredentialSessionKey
"dsAuthMethodStandard:dsAuthSMBWorkStationCredentialSessionKey"
#define kDSStdAuthSMB_LM_Key "dsAuthMethodStandard:dsAuthSMBLMKey"
#define kDSStdAuthSMB_NT_Key "dsAuthMethodStandard:dsAuthSMBNTKey"
#define kDSStdAuthSMB_NT_UserSessionKey
"dsAuthMethodStandard:dsAuthSMBNTUserSessionKey"
#define kDSStdAuthSecureHash "dsAuthMethodStandard:dsAuthSecureHash"
#define kDSStdAuthSetGlobalPolicy "dsAuthMethodStandard:dsAuthSetGlobalPolicy"
#define kDSStdAuthSetLMHash "dsAuthMethodsStandard:dsAuthSetLMHash"
#define kDSStdAuthSetNTHash "dsAuthMethodsStandard:dsAuthSetNTHash"
#define kDSStdAuthSetPasswd "dsAuthMethodStandard:dsAuthSetPasswd"
#define kDSStdAuthSetPasswdAsRoot "dsAuthMethodStandard:dsAuthSetPasswdAsRoot"
#define kDSStdAuthSetPolicy "dsAuthMethodStandard:dsAuthSetPolicy"
#define kDSStdAuthSetPolicyAsRoot "dsAuthMethodStandard:dsAuthSetPolicyAsRoot"
#define kDSStdAuthSetUserData "dsAuthMethodStandard:dsAuthSetUserData"
#define kDSStdAuthSetUserName "dsAuthMethodStandard:dsAuthSetUserName"
#define kDSStdAuthSetWorkStationPasswd
"dsAuthMethodStandard:dsAuthSetWorkstationPasswd"
#define kDSStdAuthWithAuthorizationRef
"dsAuthMethodStandard:dsAuthWithAuthorizationRef"
#define kDSStdAuthWriteSecureHash "dsAuthMethodStandard:dsAuthWriteSecureHash"
```

Constants

`kDSStdAuth2WayRandom`

Two-way random authentication method. Deprecated in Mac OS X v10.3.

`kDSStdAuth2WayRandomChangePasswd`
: Authentication method for changing the password of a user using the two-way random authentication method. Use of this authentication method does not require prior authentication. The packed buffer consists of a four byte length of username, the UTF-8 encoded user name, followed by four bytes specifying the length of the old password that follows, followed by the old password encrypted with the new password, followed by four bytes specifying the length of the new password, followed by the new password encrypted by the old password. Deprecated in Mac OS X v10.3.

`kDSStdAuthAPOP`
: APOP authentication method.

`kDSStdAuthCHAP`
: CHAP authentication.

  Available in Mac OS X v10.3 and later.

`kDSStdAuthCRAM_MD5`
: CRAM MD5 authentication method.

`kDSStdAuthChangePasswd`
: Authentication method for changing passwords. When changing a password, send the following information in a single buffer: four bytes containing the length of the user name, the user name in UTF-8 encoding, four bytes containing the length of the old password, the old password in UTF-8 encoding, four bytes containing the length of the new password, and the new password in UTF-8 encoding.

`kDSStdAuthClearText`
: Clear text authentication method.

`kDSStdAuthCrypt`
: Crypt password authentication method. When performing crypt authentication, send the following information in a single buffer: four bytes containing the length of the user name, the user name in UTF-8 encoding, four bytes containing the length of the password, and the password in UTF-8 encoding. Open Directory plug-ins are not required to support this authentication method.

`kDSStdAuthDIGEST_MD5`
: Digest MD5 authentication method.

`kDSStdAuthDeleteUser`
: Authentication method used by the Apple Password Server for deleting a user.

`kDSStdAuthGetEffectivePolicy`
: Used to extract from an Apple Password Server the combination of global and user policies that will be applied to a user.

  Available in Mac OS X v10.3 and later.

`kDSStdAuthGetGlobalPolicy`
: Authentication method used by the Apple Password Server plug-in for getting the global password policy.

`kDSStdAuthGetKerberosPrincipal`
: Authentication method for getting the Kerberos Principal name.

  Available in Mac OS X v10.3 and later.

`kDSStdAuthGetPolicy`
: The Open Directory plug-in determines which authentication method to use. Prior to Mac OS X v10.4, this authentication method was used only by the Apple Password Server, which does not require authentication to use this authentication method. Starting with Mac OX X version 10.4, Shadow Hash authentication supports password policies, so use of `kDSStdAuthGetPolicy` is no longer limited to the Apple Password Server. Send the following items in a single buffer: four bytes containing the length of the authenticator's UserID, the authenticator's UserID in UTF-8 encoding, four bytes containing the length of the authenticator's password, the authenticator's password in UTF-8 encoding, four bytes containing the length of the UserID that follows, and the UserID in UTF-8 encoding of the account for which policies are to be obtained. The first and second items can be empty strings and the third item can be a username if calling a directory node. This authentication method is used by the Apple Password Server, which does not require authentication to use this authentication method.

`kDSStdAuthGetUserData`
: Authentication method used by the Apple Password Server for getting a user's data.

`kDSStdAuthGetUserName`
: Authentication method used by the Apple Password Server for getting a user's name.

`kDSStdAuthMASKE_A`
: Retained for backward compatibility only.

`kDSStdAuthMASKE_B`
: Retained for backward compatibility only.

`kDSStdAuthMPPEMasterKeys`
: 40- or 128-bit master key generated from MS-CHAPv2 credentials (RFC 3079).

  Available in Mac OS X v10.4 and later.

`kDSStdAuthMSCHAP1`
: MS-CHAP1 authentication method.

  Available in Mac OS X v10.3 and later.

`kDSStdAuthMSCHAP2`
: MS-CHAP2, a mutual authentication method. The Open Directory plug-in generates the data and sends it back to the client. The input buffer format consists of a four byte value specifying the length of the user name that follows, the user name, a four byte value specifying the length of the server challenge that follows, the server challenge, a four byte value specifying the length of the peer challenge that follows, the peer challenge, a four byte value specifying the length of the client's digest that follows, and the client's digest. The output buffer consists of a four byte value specifying the length of the return digest for the client's challenge.

`kDSStdAuthNTLMv2`
: NTLMv2 session key packed as follows: 4 byte length of username, username in UTF-8 encoding, four byte length of the Samba server challenge, the Samba server challenge, four byte length of the NTLMv2 client data, the client data (which includes 16 bytes of client digest prefixed to the client data), four byte length of the user name used to calculate the digest, the user name used to calculate the digest in UTF-8 encoding, four byte length of the Samba domain, and the Samba domain in UTF-8 encoding. If the NTLMv2 session key is supported, it is returned in the output buffer.

  Available in Mac OS X v10.4 and later.

`kDSStdAuthNewUser`
: Create a new user record with an authentication authority. Send the following information in a single buffer: four bytes containing the length of the authenticator's UserID, the authenticator's UserID in UTF-8 encoding, four bytes containing the length of the authenticator's password, the authenticator's password in UTF-8 encoding, four bytes containing the new user's Short Name, the user's Short Name, four byte length of the new user's password, and the new user's password. This authentication type is used by the Apple Password Server.

`kDSStdAuthNewUserWithPolicy`
Create a new user record with an authentication authority and initial policy settings. Send the following information in a single buffer: four bytes containing the length of the authenticator's UserID, the authenticator's UserID in UTF-8 encoding, four bytes containing the length of the authenticator's password, the authenticator's password in UTF-8 encoding, four bytes containing the new user's Short Name, the user's Short Name, four byte length of the user's password, the new user's password, four byte length of the policy string, and the policy string in UTF-8 encoding. This authentication type is used by the Apple Password Server.

`kDSStdAuthNodeNativeClearTextOK`
Native authentication method that allows clear text passwords. The Open Directory plug-in determines which authentication method to use and may decide to use clear text. When using this authentication method, send the following information in a single buffer: four bytes containing the length of the user name, the user name in UTF-8 encoding, four bytes containing the password, and the password in UTF-8 encoding.

`kDSStdAuthNodeNativeNoClearText`
Native authentication method that does not allow clear text passwords. The Open Directory plug-in determines which authentication method to use but must not use clear text. When using this authentication method, send the following information in a single buffer: four bytes containing the length of the user name, the user name in UTF-8 encoding, four bytes containing the password, and the password in UTF-8 encoding.

`kDSStdAuthReadSecureHash`
Allows a root process to read the secure hash attribute of a user record directly.

Available in Mac OS X v10.3 and later.

`kDSStdAuthSMBNTv2UserSessionKey`
Used to generate an NTLMv2 user session key; requires prior authentication using a trusted authentication method. The buffer is packed as follows: four byte length of the directory services name, the directory services name in UTF-8 encoding, four byte length of the server challenge, eight byte server challenge, four byte length of the client response, and the client response buffer.

Available in Mac OS X v10.4 and later.

`kDSStdAuthSMBWorkstationCredentialSessionKey`
SMB workstation credential session key authentication; used to support PDC SMB iteration with Open Directory.

Available in Mac OS X v10.3 and later.

`kDSStdAuthSMB_LM_Key`
SMB LAN Manager authentication method that uses DES.

`kDSStdAuthSMB_NT_Key`
MD5 hash-based SMB authentication method.

`kDSStdAuthSMB_NT_UserSessionKey`
SMB NT session key authentication; used to support PDC SMB iteration with Open Directory.

Available in Mac OS X v10.3 and later.

`kDSStdAuthSecureHash`
Secure Hash authentication method.

Available in Mac OS X v10.3 and later.

`kDSStdAuthSetGlobalPolicy`
Authentication method used by the Apple Password Server plug-in for setting the global password policy, such as the minimum password length, time before a password expires, and maximum number of failed logins allowed. Starting with Mac OS X verion 10.4, this authentication method can also be used with ShadowHash on local NetInfo data.

`kDSStdAuthSetLMHash`
: Used to set the LAN Manager hash for a user; requires prior authentication using a trusted authentication method. The buffer is packed as follows: four byte length of the user name, the user name in UTF-8 encoding, four byte length of the LAN Manager hash, and the LAN Manager hash buffer (24 bytes).

  Available in Mac OS X v10.4 and later.

`kDSStdAuthSetNTHash`
: Used to set the NT hash for a user; requires prior authentication using a trusted authentication method. The buffer is packed as follows: four byte length of the user name, the user name in UTF-8 encoding, four byte length of the NT hash, and the NT hash buffer (24 bytes).

  Available in Mac OS X v10.4 and later.

`kDSStdAuthSetPasswd`
: Authentication method for setting passwords. The buffer is packed as follows: four byte length of the authenticator username, an authenticator username in UTF-8 encoding, four byte length of the authenticator password, authenticator password in UTF-8 encoding, four byte length of the target username, target username in UTF-8 encoding, four byte length of the new password, and the new password in UTF-8 encoding. The authenticator is usually an administrator that has permission to change the target user's password.

`kDSStdAuthSetPasswdAsRoot`
: Authentication method used by root processes that allow the setting of passwords using Basic or Shadow Hash authentication on local domains. This authentication method also works if you previously called `dsDoDirNodeAuth` (page 41) or `dsDoDirNodeAuthOnRecordType` (page 43) and set the `inDirNodeAuthOnly` or `inDirNodeAuthOnlyFlag` parameter, respectively, to `FALSE`. In this case, your previous credentials determine whether the set password operation succeeds. For example, administrators can usually set any user's password because their credentials have saved by setting the `inDirNodeAuthOnly` or `inDirNoeAuthOnlyFlag` parameter to `FALSE`.

`kDSStdAuthSetPolicy`
: The Open Directory plug-in determines which authentication method to use. Send the following information in a single buffer: four bytes containing the length of authenticator's UserID, the authenticator's UserID in UTF-8 encoding, four bytes containing the length of the authenticator's password, the authenticator's password in UTF-8 encoding, four bytes containing the length of the UserID of the account that is setting policies, and the UserID of the account that is setting policies in UTF-8 encoding. This authentication type is used by the Apple Password Server and, starting with Mac OS X v10.4, can be used with ShadowHash on local NetInfo data.

`kDSStdAuthSetPolicyAsRoot`
: A two-item buffer version of set policy for the Apple Password Server. Available in Mac OS X v10.3 and later. Starting with Mac OS X v10.4, this authentication method can be used with ShadowHash on local NetInfo data.

`kDSStdAuthSetUserData`
: Authentication method used by the Apple Password Server for setting user's data.

`kDSStdAuthSetUserName`
: Authentication method used by the Apple Password Server for setting a user's name.

`kDSStdAuthSetWorkstationPasswd`
: Authentication method used to set the workstation password; used to support PDC SMB iteration with Open Directory.

  Available in Mac OS X v10.3 and later.

`kDSStdAuthWithAuthorizationRef`
: Allows access to local directories as root with a valid AuthorizationRef. Input buffer format consists of an externalized AuthorizationRef.

  Available in Mac OS X v10.3 and later.

`kDSStdAuthWriteSecureHash`
: Allows a root process to write the secure hash attribute of a user record directly.

  Available in Mac OS X v10.3 and later.

Declared In
`DirectoryService/DirServicesConst.h`

Neighborhood Types

Constants defined for neighbor types.

```
#define kDSValueNSLTopLevelNeighborhoodType "NSLTopLevelNeighborhoodType"
#define kDSValueNSLStaticNeighborhoodType "NSLStaticNeighborhoodType"
#define kDSValueNSLDynamicNeighborhoodType "NSLDynamicNeighborhoodType"
#define kDSValueNSLLocalNeighborhoodType "NSLLocalNeighborhoodType"
```

Constants
`kDSValueNSLTopLevelNeighborhoodType`
: Top level value type for records of type `kDSStdRecordTypeNeighborhoods`.

  Available in Mac OS X v10.4 and later.

`kDSValueNSLStaticNeighboodType`
: Static neighborhood value type for records of type `kDSStdRecordTypeNeighborhoods`.

  Available in Mac OS X v10.4 and later.

`kDSValueNSLDynamicNeighboodType`
: Dynamic neighborhood value type for records of type `kDSStdRecordTypeNeighborhoods`.

  Available in Mac OS X v10.4 and later.

`kDSValueNSLLocalNeighboodType`
: Local neighborhood value type for records of type `kDSStdRecordTypeNeighborhoods`.

  Available in Mac OS X v10.4 and later.

Declared In
`DirectoryService/DirServicesConst.h`

Pattern Matching Constants

Constants defined for pattern matching.

```
typedef enum {
eDSNoMatch1 = 0x0000,
eDSAnyMatch = 0x0001,
eDSBeginAppleReserve1 = 0x0002,
eDSEndAppleReserve1 = 0x1fff,
eDSExact = 0x2001,
eDSStartsWith = 0x2002,
eDSEndsWith = 0x2003,
eDSContains = 0x2004,
eDSLessThan = 0x2005,
eDSGreaterThan = 0x2006,
eDSLessEqual = 0x2007,
eDSGreaterEqual = 0x2008,
eDSWildCardPattern = 0x2009,
eDSRegularExpression = 0x200A,
eDSCompoundExpression = 0x200B,
eDSiExact = 0x2101,
eDSiStartsWith = 0x2102,
eDSiEndsWith = 0x2103,
eDSiContains = 0x2104,
eDSiLessThan = 0x2105,
eDSiGreaterThan = 0x2106,
eDSiLessEqual = 0x2107,
eDSiGreaterEqual = 0x2108,
eDSiWildCardPattern = 0x2109,
eDSiRegularExpression = 0x210A,
eDSiCompoundExpression = 0x210B,
eDSLocalNodeNames = 0x2200,
eDSSearchNodeName = 0x2201,
eDSConfigNodeName = 0x2202,
eDSLocalHostedNodes = 0x2203,
eDSAuthenticationSearchNodeName = 0x2201,
eDSContactsSearchNodeName = 0x2204,
eDSNetworkSearchNodeName = 0x2205,
eDSDefaultNetworkNodes = 0x2206,
dDSBeginPlugInCustom = 0x3000,
eDSEndPlugInCustom = 0x4fff,
eDSBeginAppleReserve2 = 0x5000,
eDSEndAppleReserve2 = 0xfffe,
eDSNoMatch2 = 0xffff
} tDirPatternMatch;
```

Constants

`eDSNoMatch1`

Reserved.

Available in Mac OS X v10.0 and later.

Declared in `DirServicesTypes.h`.

`eDSAnyMatch`

Matches any value.

Available in Mac OS X v10.0 and later.

Declared in `DirServicesTypes.h`.

`eDSBeginAppleReserve1`
: Beginning of a range of values reserved for use by Apple Computer.

  Available in Mac OS X v10.0 and later.

  Declared in `DirServicesTypes.h`.

`eDSEndAppleReserve1`
: End of a range of values reserved for use by Apple Computer.

  Available in Mac OS X v10.0 and later.

  Declared in `DirServicesTypes.h`.

`eDSExact`
: Matches the specified value exactly (case sensitive).

  Available in Mac OS X v10.0 and later.

  Declared in `DirServicesTypes.h`.

`eDSStartsWith`
: Matches values that start with the specified value (case sensitive).

  Available in Mac OS X v10.0 and later.

  Declared in `DirServicesTypes.h`.

`eDSEndsWith`
: Matches values that end with the specified value (case sensitive).

  Available in Mac OS X v10.0 and later.

  Declared in `DirServicesTypes.h`.

`eDSContains`
: Matches values that contain the specified value (case sensitive).

  Available in Mac OS X v10.0 and later.

  Declared in `DirServicesTypes.h`.

`eDSLessThan`
: Matches values that are less than the specified value (case sensitive).

  Available in Mac OS X v10.0 and later.

  Declared in `DirServicesTypes.h`.

`eDSGreaterThan`
: Matches values that are greater than the specified value (case sensitive).

  Available in Mac OS X v10.0 and later.

  Declared in `DirServicesTypes.h`.

`eDSLessEqual`
: Matches values that are less than or equal to the specified value (case sensitive).

  Available in Mac OS X v10.0 and later.

  Declared in `DirServicesTypes.h`.

`eDSGreaterEqual`
: Matches values that are greater than or equal to the specified value (case sensitive).

  Available in Mac OS X v10.0 and later.

  Declared in `DirServicesTypes.h`.

`eDSWildCardPattern`
: Matches values using the specified wild card pattern (case sensitive).

  Available in Mac OS X v10.0 and later.

  Declared in `DirServicesTypes.h`.

`eDSRegularExpression`
: Matches values using the specified regular expression (case sensitive).

  Available in Mac OS X v10.0 and later.

  Declared in `DirServicesTypes.h`.

`eDSCompoundExpression`
: Accommodates an attribute search based on a pre-built compound expression.

  Available in Mac OS X v10.2 and later.

  Declared in `DirServicesTypes.h`.

`eDSiExact`
: Matches the specified value exactly (case insensitive).

  Available in Mac OS X v10.0 and later.

  Declared in `DirServicesTypes.h`.

`eDSiStartsWith`
: Matches values that start with the specified value (case insensitive).

  Available in Mac OS X v10.0 and later.

  Declared in `DirServicesTypes.h`.

`eDSiEndsWith`
: Matches values that end with the specified value (case insensitive).

  Available in Mac OS X v10.0 and later.

  Declared in `DirServicesTypes.h`.

`eDSiContains`
: Matches values that contain the specified value (case insensitive).

  Available in Mac OS X v10.0 and later.

  Declared in `DirServicesTypes.h`.

`eDSiLessThan`
: Matches values that are less than the specified value (case insensitive).

  Available in Mac OS X v10.0 and later.

  Declared in `DirServicesTypes.h`.

`eDSiGreaterThan`
: Matches values that are greater than the specified value (case insensitive).

  Available in Mac OS X v10.0 and later.

  Declared in `DirServicesTypes.h`.

`eDSiLessEqual`
: Matches values that are less than or equal to the specified value (case insensitive).

  Available in Mac OS X v10.0 and later.

  Declared in `DirServicesTypes.h`.

`eDSiGreaterEqual`
: Matches values that are greater than or equal to the specified value (case insensitive).

  Available in Mac OS X v10.0 and later.

  Declared in `DirServicesTypes.h`.

`eDSiWildCardPattern`
: Matches values using the specified wild card pattern (case insensitive).

  Available in Mac OS X v10.0 and later.

  Declared in `DirServicesTypes.h`.

`eDSiRegularExpression`
: Matches values using the specified regular expression (case insensitive).

  Available in Mac OS X v10.0 and later.

  Declared in `DirServicesTypes.h`.

`eDSiCompoundExpression`
: Accommodates an attribute search based on a pre-built compound expression (case insensitive).

  Available in Mac OS X v10.2 and later.

  Declared in `DirServicesTypes.h`.

`eDSLocalNodeNames`
: Matches the local node name.

  Available in Mac OS X v10.0 and later.

  Declared in `DirServicesTypes.h`.

`eDSSearchNodeName`
: Matches the node name that is to be used to authenticate the Open Directory client.

  Available in Mac OS X v10.0 and later.

  Declared in `DirServicesTypes.h`.

`eDSConfigNodeName`
: Matches the configuration node. Used primarily by the Directory Access application for configuration purposes; not intended for use by developers.

  Available in Mac OS X v10.0 and later.

  Declared in `DirServicesTypes.h`.

`eDSLocalHostedNodes`
: Matches NetInfo domains stored on this machine.

  Available in Mac OS X v10.0 and later.

  Declared in `DirServicesTypes.h`.

`eDSAuthenticationSearchNodeName`
: Matches the node name that is to be used to authenticate an Open Directory client. (This is another name for `eDSSearchNodeName`.)

  Available in Mac OS X v10.0 and later.

  Declared in `DirServicesTypes.h`.

`eDSContactsSearchNodeName`
: Matches the node name that is to be used for searching when authentication is not required; used by Address Book and Mail applications.

  Available in Mac OS X v10.0 and later.

  Declared in `DirServicesTypes.h`.

`eDSNetworkSearchNodeName`
: Searches across all the nodes returned by `eDSDefaultNetworkNodes`.

  Available in Mac OS X v10.2 and later.

  Declared in `DirServicesTypes.h`.

`eDSDefaultNetworkNodes`
: Matches the default network node.

  Available in Mac OS X v10.2 and later.

  Declared in `DirServicesTypes.h`.

`dDSBeginPlugInCustom`
: Beginning of a range of values reserved for use by Open Directory plug-ins.

  Available in Mac OS X v10.0 and later.

  Declared in `DirServicesTypes.h`.

`eDSEndPlugInCustom`
: End of a range of values reserved for use by Open Directory plug-ins.

  Available in Mac OS X v10.0 and later.

  Declared in `DirServicesTypes.h`.

`eDSBeginAppleReserve2`
: Beginning of a range of values reserved for use by Apple Computer.

  Available in Mac OS X v10.0 and later.

  Declared in `DirServicesTypes.h`.

`eDSEndAppleReserve2`
: End of a range of values reserved for use by Apple Computer.

  Available in Mac OS X v10.0 and later.

  Declared in `DirServicesTypes.h`.

`eDSNoMatch2`
: Reserved.

  Available in Mac OS X v10.0 and later.

  Declared in `DirServicesTypes.h`.

Discussion
The `tDirPatternMatch` enumeration defines constants for use with Open Directory functions that look for pattern matches. A directory service is not required to support all types of pattern matching.

Declared In
`DirectoryService/DirServicesTypes.h`

Meta Record Type Constants

Constants defined to work with all records, standard records, or native records.

```
#define kDSRecordsAll "dsRecordsAll"
#define kDSRecordsStandardAll "dsRecordsStandardAll"
#define kDSRecordsNativeAll "dsRecordsNativeAll"
#define kDSNativeRecordTypePrefix "dsRecTypeNative:"
#define kDSStdRecordTypeAll "dsRecTypeStandard:All
#define kDSStdUserNamesMeta "dsRecTypeStandard:MetaUserNames"
```

Constants

`kDSRecordsAll`
: Used to indicate that all records should be returned (instead of returning records that match a pattern).

`kDSRecordsStandardAll`
: Retained for backward compatibility.

`kDSRecordsNativeAll`
: Retained for backward compatibility.

`kDSStdRecordTypePrefix`
: Used as the prefix for all standard record types.

`kDSNativeRecordTypePrefix`
: Prefix used to identify a native record type.

`kDSStdRecordTypeAll`
: Used to indicate that all record types need to be searched.

  Available in Mac OS X v10.4 and later.

`kDSStdUserNamesMeta`
: Retained for backward compatibility.

Declared In

`DirectoryService/DirServicesConst.h`

Standard Record Types

Constants defined for standard record types.

```
#define kDSStdRecordTypeAccessControls "dsRecTypeStandard:AccessControls
#define kDSStdRecordTypeAFPServer "dsRecTypeStandard:AFPServer"
#define kDSStdRecordTypeAFPUserAliases "dsRecTypeStandard:AFPUserAliases"
#define kDSStdRecordTypeAliases "dsRecTypeStandard:Aliases"
#define kDSStdRecordTypeAutoServerSetup "dsRecTypeStandard:AutoServerSetup"
#define kDSStdRecordTypeBootp "dsRecTypeStandard:Bootp"
#define kDSStdRecordTypeCertificateAuthorities
"dsRecTypeStandard:CertificateAuthorities
#define kDSStdRecordTypeComputerLists "dsRecTypeStandard:ComputerLists"
#define kDSStdRecordTypeComputers "dsRecTypeStandard:Computers"
#define kDSStdRecordTypeConfig "dsRecTypeStandard:Config"
#define kDSStdRecordTypeEthernets "dsRecTypeStandard:Ethernets"
#define kDSStdRecordTypeFileMakerServers "dsRecTypeStandard:FileMakerServers"
#define kDSStdRecordTypeFTPServer "dsRecTypeStandard:FTPServer"
#define kDSStdRecordTypeGroupAliases "dsRecTypeStandard:GroupAliases"
#define kDSStdRecordTypeGroups "dsRecTypeStandard:Groups"
#define kDSStdRecordTypeHostServices "dsRecTypeStandard:HostServices"
#define kDSStdRecordTypeHosts "dsRecTypeStandard:Hosts"
#define kDSStdRecordTypeLDAPServer "dsRecTypeStandard:LDAPServer"
#define kDSStdRecordTypeLocations "dsRecTypeStandard:Locations"
#define kDSStdRecordTypeMachines "dsRecTypeStandard:Machines"
#define kDSStdRecordTypeMeta "dsRecTypeStandard:AppleMetaRecord"
#define kDSStdRecordTypeMounts "dsRecTypeStandard:Mounts"
#define kDSStdRecordTypeNeighborhoods "dsRecTypeStandard:Neighborhoods"
#define kDSStdRecordTypeNFS "dsRecTypeStandard:NFS"
#define kDSStdRecordTypeNetDomains "dsRecTypeStandard:NetDomains"
#define kDSStdRecordTypeNetGroups "dsRecTypeStandard:NetGroups"
#define kDSStdRecordTypeNetworks "dsRecTypeStandard:Networks"
#define kDSStdRecordTypePasswordServer "dsRecTypeStandard:PasswordServer"
#define kDSStdRecordTypePeople "dsRecTypeStandard:People"
#define kDSStdRecordTypePresetComputerLists  "dsRecTypeStandard:PresetComputerLists"
#define kDSStdRecordTypePresetGroups "dsRecTypeStandard:PresetGroups"
#define kDSStdRecordTypePresetUsers "dsRecTypeStandard:PresetUsers"
#define kDSStdRecordTypePrintService "dsRecTypeStandard:PrintService"
#define kDSStdRecordTypePrintServiceUser "dsRecTypeStandard:PrintServiceUser"
#define kDSStdRecordTypePrinters "dsRecTypeStandard:Printers"
#define kDSStdRecordTypeProtocols "dsRecTypeStandard:Protocols"
#define kDSStdRecordTypeQTSServer "dsRecTypeStandard:QTSServer"
#define kDSStdRecordTypeRPC "dsRecTypeStandard:RPC"
#define kDSStdRecordTypeSMBServer "dsRecTypeStandard:SMBServer"
#define kDSStdRecordTypeServer "dsRecTypeStandard:Server"
#define kDSStdRecordTypeServices "dsRecTypeStandard:Services"
#define kDSStdRecordTypeSharePoints "dsRecTypeStandard:SharePoints"
#define kDSStdRecordTypeUserAliases "dsRecTypeStandard:UserAliases"
#define kDSStdRecordTypeUsers "dsRecTypeStandard:Users"
#define kDSStdRecordTypeWebServer "dsRecTypeStandard:WebServer"
```

Constants

`kDSStdRecordTypeAccessControls`

Record type for storing directory access control directives.

Available in Mac OS X v10.4 and later.

`kDSStdRecordTypeAFPServer`

Record type for storing Apple Filing Protocol (AFP) server records.

`kDSStdRecordTypeAFPUserAliases`

Record type for storing AFP user aliases records used exclusively by AFP processes. Not used in Mac OS X v10.4 and later.

`kDSStdRecordTypeAliases`
Record type for representing alias records.

`kDSStdRecordTypeAutoServerSetup`
Type for locating automated server set up information.

Available in Mac OS X v10.3 and later.

`kDSStdRecordTypeBootp`
Record in the local node for storing `bootp` information.

`kDSStdRecordTypeCertificateAuthorities`
Record type for storing certificate authority information.

Available in Mac OS X v10.4 and later.

`kDSStdRecordTypeComputerLists`
Record type for identifying computer record lists.

Available in Mac OS X v10.2 and later.

`kDSStdRecordTypeComputers`
Record type for identifying computer records.

Available in Mac OS X v10.2 and later.

`kDSStdRecordTypeConfig`
Record type for identifying configuration records.

`kDSStdRecordTypeEthernets`
Record type in the local node for storing Ethernets.

`kDSStdRecordTypeFileMakerServers`
Record type for storing FileMaker server records that describe FileMaker servers.

Available in Mac OS X v10.4 and later.

`kDSStdRecordTypeFTPServer`
Record type for storing File Transfer Protocol (FTP) server records.

`kDSStdRecordTypeGroupAliases`
Record type for group aliases records. Not supported in Mac OS X v10.4 or later.

`kDSStdRecordTypeGroups`
Record type for identifying group records.

`kDSStdRecordTypeHostServices`
Record in the local node for storing host services.

`kDSStdRecordTypeHosts`
Record type for storing host records.

`kDSStdRecordTypeLDAPServer`
Record type for storing Lightweight Directory Access Protocol (LDAP) server records.

`kDSStdRecordTypeLocations`
Record type for storing location information.

Available in Mac OS X v10.4 and later.

`kDSStdRecordTypeMachines`
Record type for storing machine records.

`kDSStdRecordTypeMeta`
Record type for identifying meta records. Not used as of Mac OS X v10.4.

`kDSStdRecordTypeMounts`
Record type for identifying mount records.

`kDSStdRecordTypeNeighborhoods`
: Record type for identifying records that contain a list of computers and other neighborhoods; used for network browsing.

  Available in Mac OS X v10.4 and later, and used for Managed Network Views.

`kDSStdRecordTypeNFS`
: Record type for identifying Network File System (NFS) records.

`kDSStdRecordTypeNetDomains`
: Record type in the local node for storing net domains.

`kDSStdRecordTypeNetGroups`
: Record type in the local node for storing net groups.

`kDSStdRecordTypeNetworks`
: Record type for identifying network records.

`kDSStdRecordTypePasswordServer`
: Record type for discovering password servers via Bonjour.

  Available in Mac OS X v10.3 and later.

`kDSStdRecordTypePeople`
: Record type for identifying “people” records containing contact information.

  Available in Mac OS X v10.3 and later.

`kDSStdRecordTypePresetComputerLists`
: Record type for identifying preset computer list records used in record creation.

  Available in Mac OS X v10.2 and later.

`kDSStdRecordTypePresetGroups`
: Record type for identifying preset group records used in record creation.

  Available in Mac OS X v10.2 and later.

`kDSStdRecordTypePresetUsers`
: Record type for identifying preset user records used in record creation.

  Available in Mac OS X v10.2 and later.

`kDSStdRecordTypePrintService`
: Record type for identifying print service records.

`kDSStdRecordTypePrintServiceUser`
: Record type in the local node for storing quota usage for a user.

  Available in Mac OS X v10.3 and later.

`kDSStdRecordTypePrinters`
: Record type for identifying printer records.

`kDSStdRecordTypeProtocols`
: Record type for identifying protocol records.

`kDSStdRecordTypeQTSServer`
: Record type for identifying QuickTime Streaming Server (QTSS) records.

`kDSStdRecordTypeRPC`
: Record type for identifying RPC records.

`kDSStdRecordTypeSMBServer`
: Record type for identifying SMB server records.

`kDSStdRecordTypeServer`
: Record type for identifying generic server records.

`kDSStdRecordTypeServices`
: Record type for identifying directory-based service records.

`kDSStdRecordTypeSharePoints`
: Record type for identifying share point records.

  Available in Mac OS X v10.3 and later.

`kDSStdRecordTypeUserAliases`
: Record type for storing user aliases records. Not supported in Mac OS X v10.4 or later.

`kDSStdRecordTypeUsers`
: Record type for identifying user records.

`kDSStdRecordTypeWebServer`
: Record type for identifying Web server records.

Declared In
`DirectoryService/DirServicesConst.h`

Meta Attribute Type Constants

Constants defined to get all attributes, standard attributes, or native attributes.

```
#define kDSAttributesAll "dsAttributesAll"
#define kDSAttributesStandardAll "dsAttributesStandardAll"
#define kDSAttributesNativeAll "dsAttributesNativeAll"
#define kDSStdAttrTypePrefix "dsAttrTypeStandard:"
#define kDSNativeAttrTypePrefix "dsAttrTypeNative:"
#define kDSAttrNone "dsNone"
```

Constants

`kDSAttributesAll`
: Indicates that all attribute types should be searched or returned.

`kDSAttributesStandardAll`
: Indicates that all standard attribute types should be searched or returned.

`kDSAttributesNativeAll`
: Indicates that all native attribute types should be searched or returned.

`kDSStdAttrTypePrefix`
: Prefix used to identify all standard attribute types.

`kDSNativeAttrTypePrefix`
: Prefix used to identify directory-native attribute types.

`kDSAttrNone`
: Retained for backward compatibility.

Declared In
`DirectoryService/DirServicesConst.h`

Alias Attribute Constants

Constants for accessing alias information.

```
#define kDS1AttrAlias "dsAttrTypeStandard:Alias"
#define kDS1AttrAliasData "dsAttrTypeStandard:AppleAliasData"
#define kDSNAttrRecordAlias "dsAttrTypeStandard:RecordAlias"
#define kStandardTargetAlias "dsAttrTypeStandard:AppleMetaAliasTarget"
#define kStandardSourceAlias "dsAttrTypeStandard:AppleMetaAliasSource"
```

Constants

`kDS1AttrAlias`
: Single-value attribute for storing a pointer to another node, record, or attribute.

`kDS1AttrAliasData`
: Single-value attribute for storing alias data.

`kDSNAttrRecordAlias`
: Multi-value attribute for storing record aliases. Not supported in Mac OS X v10.4 or later.

`kStandardTargetAlias`
: Single-value attribute for storing a target alias. Not supported in Mac OS X v10.4 or later.

`kStandardSourceAlias`
: Single-value attribute for storing a source alias. Not supported in Mac OS X v10.4 or later.

Declared In

`DirectoryService/DirServicesConst.h`

Boot Attribute Constants

Constants for accessing boot information.

```
#define kDS1AttrBootFile "dsAttrTypeStandard:BootFile"
#define kDSNAttrBootParams "dsAttrTypeStandard:BootParams"
```

Constants

`kDS1AttrBootFile`
: Single-value attribute for storing the name of the kernel that this machine uses by default when performing a netboot. This attribute is available in Mac OS X v10.4 and later.

`kDSNAttrBootParams`
: Multi-value attribute for storing boot parameters. This attribute is found in records of type `kDSStdRecordTypeHosts` or `kDSStdRecordTypeMachines`.

Declared In

`DirectoryService/DirServicesConst.h`

Certificate Attribute Constants

Certificate attribute constants.

```
#define kDS1AttrAuthorityRevocationList  "dsAttrTypeStandard:AuthorityRevocationList"
#define kDS1AttrCACertificate "dsAttrTypeStandard:CACertificate"
#define kDS1AttrCertificateRevocationList
"dsAttrTypeStandard:CertificateRevocationList"
#define kDS1AttrCrossCertificatePair "dsAttrTypeStandard:CrossCertificatePair"
#define kDS1AttrUserCertificate "dsAttrTypeStandard:UserCertificate"
#define kDS1AttrUserPKCS12Data "dsAttrTypeStandard:UserPKCS12Data"
#define kDS1AttrUserSMIMECertificate "dsAttrTypeStandard:UserSMIMECertificate"
```

Constants

`kDS1AttrAuthorityRevocationList`
: Single-value attribute for storing a list of binary certificate authority certificates that are no longer trusted. No user certificates are included in this list. This attribute is usually found in records of type `kDSStdRecordTypeCertificateAuthorities`. This attribute is available in Mac OS X v10.4 and later.

`kDS1AttrCACertificate`
: Single-value attribute for storing the binary of a certificate of a certificate authority. The corresponding private key is used to sign certificates. This attribute is usually found in records of type `kDSStdRecordTypeCertificateAuthorities`. This attribute is available in Mac OS X v10.4 and later.

`kDS1AttrCertificateRevocationList`
: Single-value attribute for storing the list of binary certificates that are no longer trusted. This attribute is usually found in records of type `kDSStdRecordTypeCertificateAuthorities`. This attribute is available in Mac OS X v10.4 and later.

`kDS1AttrCrossCertificatePair`
: Single-value attribute for storing the binary of a pair of certificates that verify each other. Both certificates have the same level of authority. This attribute is usually found in records of type `kDSStdRecordTypeCertificateAuthorities`. This attribute is available in Mac OS X v10.4 and later.

`kDS1AttrUserCertificate`
: Single-value attribute for storing the binary of a user's certificate, where a certificate is data that identifies the user and that is attested to by a known party and that can be independently verified by a third party. This attribute is usually found in user records. This attribute is available in Mac OS X v10.4 and later.

`kDS1AttrUserPKCS12Data`
: Single-value attribute for storing binary data usually encrypted with a passphrase, such as keys, certificates and other related information, in PKCS #12 format. This attribute is available in Mac OS X v10.4 and later.

`kDS1AttrUserSMIMECertificate`
: Single-value attribute containing the binary of the user's SMIME certificate and usually found in records of type `kDSStdRecordTypeUsers`. The certificate is data that identifies a user, is attested to by a known third party, and can be independently verified by a third party. SMIME certificates are often used for signed or encrypted e-mail. This attribute is available in Mac OS X v10.4 and later.

Declared In

`DirectoryService/DirServicesConst.h`

DNS Attribute Constants

Constants defined for attributes that store DNS information.

```
#define kDS1AttrDNSDomain "dsAttrTypeStandard:DNSDomain"
#define kDS1AttrDNSNameServer "dsAttrTypeStandard:DNSNameServer"
#define kDSNAttrDNSName "dsAttrTypeStandard:DNSName"
```

Constants

`kDS1AttrDNSDomain`
Single-value attribute for storing a DNS Resolver domain. This attribute is available in Mac OS X v10.4 and later.

`kDS1AttrDNSNameServer`
Single-value attribute for storing a DNS Resolver name server. This attribute is available in Mac OS X v10.4 and later.

`kDSNAttrDNSName`
Multi-value attribute for storing DNS names.

Declared In

`DirectoryService/DirServicesConst.h`

Kerberos Attribute Constants

Constants for accessing to Kerberos attributes.

```
#define kDSNAttrKDCAuthKey "dsAttrTypeStandard:KDCAuthKey"
#define kDS1AttrKDCConfigData "dsAttrTypeStandard:KDCConfigData"
#define kDS1AttrKerberosRealm "dsAttrTypeStandard:KerberosRealm"
```

Constants

`kDSNAttrKDCAuthKey`
Multi-value attribute for storing KDC master keys. Each key is RSA-encrypted with the realm public key.

Available in Mac OS X v10.3 and later.

`kDS1AttrKDCConfigData`
Single-value attribute for storing the contents of the Kerberos Key Distribution Center (KDC) file, `kdc.conf`.

Available in Mac OS X v10.3 and later.

`kDS1AttrKerberosRealm`
Attribute for storing the Kerberos realm; used with the Open Directory `dsGetDirNodeInfo` function in support of Kerberos SMB server services.

Available in Mac OS X v10.4 and later.

Declared In

`DirectoryService/DirServicesConst.h`

LDAP Attribute Constants

Constants for accessing LDAP attributes.

```
#define kDS1AttrRelativeDNPrefix "dsAttrTypeStandard:RelativeDNPrefix"
#define kDSNAttrLDAPReadReplicas "dsAttrTypeStandard:LDAPReadReplicas"
#define kDSNAttrLDAPWriteReplicas "dsAttrTypeStandard:LDAPWriteReplicas"
```

Constants

`kDS1AttrRelativeDNPrefix`
: Single-value attribute for storing information needed to map the first native LDAP attribute type. This is required to build the Relative Distinguished Name for creating LDAP records.

  Available in Mac OS X v10.3 and later.

`kDSNAttrLDAPReadReplicas`
: Attribute for storing LDAP server URLs that can be used to read directory data.

  Available in Mac OS X v10.3 and later.

`kDSNAttrLDAPWriteReplicas`
: Attribute for storing LDAP server URLs that can be used to write directory data.

  Available in Mac OS X v10.3 and later.

Declared In

`DirectoryService/DirServicesConst.h`

Network Address Attribute Constants

Constants for accessing network address attributes.

```
#define kDS1AttrENetAddress "dsAttrTypeStandard:ENetAddress'
#define kDSNAttrIPAddress "dsAttrTypeStandard:IPAddress"
#define kDSNAttrNBPEntry "dsAttrTypeStandard:NBPEntry"
```

Constants

`kDS1AttrENetAddress`
: Single-value attribute for storing a hardware Ethernet (MAC) address. This attribute is found in records of type `kDSStdRecordTypeComputers` and `kDSStdRecordTypeMachines`.

`kDSNAttrIPAddress`
: Multi-value attribute for storing IP addresses. This attribute is found in records of type `kDSStdRecordTypeComputers` and `kDSStdRecordTypeMachines`.

`kDSNAttrNBPEntry`
: Multi-value attribute for storing Name Binding Protocol (NBP) data; retained for backward compatibility only.

Declared In

`DirectoryService/DirServicesConst.h`

Machine and Host Record Attribute Constants

Constants for accessing certain attributes typically found in records of type `kDSStdRecordTypeHosts` and `kDSStdRecordTypeMachines`.

```
#define kDS1AttrContactPerson "dsAttrTypeStandard:ContactPerson"
#define kDSNAttrMachineServes "dsAttrTypeStandard:MachineServes"
```

Constants

`kDS1AttrContactPerson`
: Single-value attribute for storing the name of the contact person for the machine. This attribute is available in Mac OS X v10.4 and later.

`kDSNAttrMachineServes`
: Multi-value attribute for storing the NetInfo domains that a machine or host serves. This attribute is supported in Mac OS X v10.4 and later.

Declared In

`DirectoryService/DirServicesConst.h`

Managed Clients for Mac OS X Attribute Constants

Constants for Managed Clients for Mac OS X (MCX) attributes.

```
#define kDS1AttrMCXFlags "dsAttrTypeStandard:MCXFlags"
#define kDS1AttrMCXSettings "dsAttrTypeStandard:MCXSettings"
#define kDSNAttrMCXSettings "dsAttrTypeStandard:MCXSettings"
```

Constants

`kDS1AttrMCXFlags`
: Single-value attribute for storing MCX flags.

`kDS1AttrMCXSettings`
: Single-value attribute for storing MCX settings.

`kDSNAttrMCXSettings`
: Multi-value attribute for storing MCX settings.

  Available in Mac OS X v10.3 and later.

Declared In

`DirectoryService/DirServicesConst.h`

Miscellaneous Attribute Constants

Constants for accessing miscellaneous attributes.

```
#define kDS1AttrCopyTimestamp "dsAttrTypeStandard:CopyTimestamp"
#define kDS1AttrDataStamp "dsAttrTypeStandard:DataStamp"
#define kDS1AttrPresetUserIsAdmin "dsAttrTypeStandard:PresetUserIsAdmin"
#define kDS1AttrRARA "dsAttrTypeStandard:RARA"
#define kDS1AttrTimePackage "dsAttrTypeStandard:TimePackage"
#define kDSNAttrAccessControlEntry "dsAttrTypeStandard:AccessControlEntry"
#define kDSNAttrAuthMethod "dsAttrTypeStandard:AuthMethod"
#define kDSNAttrComputers "dsAttrTypeStandard:Computers"
#define kDSNAttrGroup "dsAttrTypeStandard:Group"
#define kDSNAttrHTML "dsAttrTypeStandard:HTML"
#define kDSNAttrKeywords "dsAttrTypeStandard:Keywords"
#define kDSNAttrMember "dsAttrTypeStandard:Member"
#define kDSNAttrMIME "dsAttrTypeStandard:MIME"
#define kDSNAttrNetworkView "dsAttrTypeStandard:NetworkView"
#define kDSNAttrPGPPublicKey "dsAttrTypeStandard:PGPPublicKey"
#define kDSNAttrProtocols "dsAttrTypeStandard:Protocols"
#define kDSNAttrSchema "dsAttrTypeStandard:Schema"
#define kDSNAttrURL "dsAttrTypeStandard:URL"
#define kDSNAttrURLForNSL "dsAttrTypeStandard:URLForNSL"
#define kDSStdMachPortName "com.apple.DirectoryService"
```

Constants

`kDS1AttrCopyTimestamp`
: Single-value attribute for storing a timestamp used in local account caching.

  Available in Mac OS X v10.3 and later.

`kDS1AttrDataStamp`
: Single-value attribute for storing checksum meta data.

`kDS1AttrPresetUserIsAdmin`
: Single-value attribute whose value indicates whether users created using this preset are administrators by default. This attribute is found in records of type `kDSStdRecordTypePresetUsers`.

`kDS1AttrRARA`
: Retained for backward compatibility.

`kDS1AttrTimePackage`
: Single-value attribute for storing creation, modification, and backup dates in UTC.

`kDSNAttrAccessControlEntry`
: Multi-value attribute for storing directory access control directives. This attribute is supported in Mac OS X v10.4 and later.

`kDSNAttrAuthMethod`
: Multi-value attribute for storing authentication methods for an authentication-capable record.

`kDSNAttrComputers`
: Multi-value attribute for storing names of records of type `kDSStdRecordTypeComputers` that are members of a computer list. Used by records of type `kDSStdRecordTypeComputerLists`; maps to "computers" in NetInfo.

`kDSNAttrGroup`
: Multi-value attribute for storing group records.

`kDSNAttrHTML`
: Multi-value attribute for storing HTML locations.

`kDSNAttrKeywords`
: Multi-value attribute for storing search keywords.

  Available in Mac OS X v10.3 and later.

`kDSNAttrMember`
: Multi-value attribute for storing member records.

`kDSNAttrMIME`
: Multi-value attribute for storing fully qualified MIME types.

`kDS1AttrNetworkView`
: Single-value attribute for storing the name of the managed network view a computer should use for browsing.

`kDSNAttrPGPPublicKey`
: Multi-value attribute for storing Pretty Good Privacy (PGP) public keys.

`kDSNAttrProtocols`
: Multi-value attribute for storing the names of protocols.

`kDSNAttrSchema`
: Multi-value attribute for storing attribute types.

`kDSNAttrURL`
: Multi-value attribute for storing URLs.

`kDSNAttrURLForNSL`
: Multi-value attribute for storing URLs used by the Network Services Location Manager; not used.

`kDSStdMachPortName`
: Registered name used with `mach_init` for DirectoryService daemon.

Declared In
`DirectoryService/DirServicesConst.h`

Neighborhood Attribute Constants

Neighborhood attribute constants.

```
#define kDS1AttrComputerAlias "dsAttrTypeStandard:ComputerAlias"
#define kDS1AttrNeighborhoodAlias "dsAttrTypeStandard:NeighborhoodAlias"
#define kDS1AttrNeighborhoodType "dsAttrTypeStandard:NeighborhoodType"
#define kDS1AttrNodePathXMLPlist "dsAttrTypeStandard:NodePathXMLPlist"
```

Constants
`kDS1AttrComputerAlias`
: Single-value attribute found in records of type `kDSStdRecordTypeNeighborhoods`; used to describe computer records pointed to by this neighborhood. This attribute is available in Mac OS X v10.4 and later.

`kDS1AttrNeighborhoodAlias`
: Single-value attribute found in records of type `kDSStdRecordTypeNeighborhoods`; used to describe sub-neighborhood records. This attribute is available in Mac OS X v10.4 and later.

`kDS1AttrNeighborhoodType`
: Single-value attribute for storing a description of the function of a record of type `kDSStdRecordTypeNeighborhoods`. This attribute is available in Mac OS X v10.4 and later.

`kDS1AttrNodePathXMLPlist`
: Single-value attribute found in records of type `kDSStdRecordTypeNeighborhoods`; used to describe the Open Directory node to search when looking for aliases in this neighborhood. This attribute is available in Mac OS X v10.4 and later.

Declared In
`DirectoryService/DirServicesConst.h`

Node Attribute Constants

Constants defined for storing information about nodes.

```
#define kDS1AttrAuthCredential "dsAttrTypeStandard:AuthCredential"
#define kDS1AttrCapabilities "dsAttrTypeStandard:Capabilities"
#define kDS1AttrOriginalNodeName "dsAttrTypeStandard:OriginalNodeName"
#define kDS1AttrReadOnlyNode "dsAttrTypeStandard:ReadOnlyNode"
#define kDSNAttrMetaNodeLocation "dsAttrTypeStandard:AppleMetaNodeLocation"
#define kDSNAttrNodePath "dsAttrTypeStandard:NodePath"
#define kDSNAttrPlugInInfo "dsAttrTypeStandard:PlugInInfo"
#define kDSNAttrSubNodes "dsAttrTypeStandard:SubNodes"
```

Constants

`kDS1AttrAuthCredential`
Single-value attribute for storing an authentication credential used to authenticate to other directory nodes.

`kDS1AttrCapabilities`
Single-value attribute used to store information about the API capabilities of a directory node.

`kDS1AttrOriginalNodeName`
Single-value attribute for storing the node name used in local account caching.

Available in Mac OS X v10.3 and later.

`kDS1AttrReadOnlyNode`
Single-value attribute for storing the read/write status of a node, which can be one of `ReadOnly`, `ReadWrite`, or `WriteOnly`. Attributes of this type can be found by calling `dsGetDirNodeInfo`. Note that `ReadWrite` does not imply fully readable or fully writable.

`kDSNAttrMetaNodeLocation`
Multi-value attribute for storing the registered node name returned by an Open Directory plug-in.

`kDSNAttrNodePath`
Multi-value attribute for storing, in order, plug-in defined sub-strings of an Open Directory node.

`kDSNAttrPlugInInfo`
Multi-value attribute for storing information provided by the plug-in that services a particular directory node. Clients can use this attribute to get information about an Open Directory plug-in, such as its version, signature, “about” information, and credits. As of Mac OX X version 10.4, this attribute is not used.

`kDSNAttrSubNodes`
Multi-value attribute for storing a list of a node’s subnodes. This attribute is supported in Mac OS X v10.4 and later.

Declared In
`DirectoryService/DirServicesConst.h`

Password Attribute Constants

Constants for accessing password policy and password setting method attributes.

```
#define kDS1AttrPasswordPolicyOptions "dsAttrTypeStandard:PasswordPolicyOptions"
#define kDS1AttrPwdAgingPolicy "dsAttrTypeStandard:PwdAgingPolicy"
#define kDSNAttrSetPasswdMethod "dsAttrTypeStandard:SetPasswdMethod"
```

Constants

`kDS1AttrPasswordPolicyOptions`

Single-value attribute for storing the collection of password policy options; used in records of type `kDSStdRecordTypePresetUsers`.

Available in Mac OS X v10.3 and later.

`kDS1AttrPwdAgingPolicy`

Single-value attribute for storing password aging policy data for an authentication-capable record. Not implemented and not used.

`kDSNAttrSetPasswdMethod`

Multi-value attribute for storing password-setting methods. Not implemented and not used.

Declared In

`DirectoryService/DirServicesConst.h`

Password Server Attribute Constants

Constants for accessing Password Server attributes.

```
#define kDS1AttrPasswordServerList "dsAttrTypeStandard:PasswordServerList"
#define kDS1AttrPasswordServerLocation  "dsAttrTypeStandard:PasswordServerLocation"
```

Constants

`kDS1AttrPasswordServerList`

Single-value attribute for storing an Apple Password Server's replication information.

Available in Mac OS X v10.3 and later.

`kDS1AttrPasswordServerLocation`

Single-value attribute for storing the IP address or domain name of the Password Server associated with a given directory node. This attribute is found in configuration records named "passwordserver".

Available in Mac OS X v10.2 and later.

Declared In

`DirectoryService/DirServicesConst.h`

Print Attribute Constants

Constants for accessing print-related attributes.

```
#define kDS1AttrNote "dsAttrTypeStandard:Note"
#define kDS1AttrPrinter1284DeviceID "dsAttrTypeStandard:Printer1284DeviceID"
#define kDS1AttrPrinterLPRHost "dsAttrTypeStandard:PrinterLPRHost"
#define kDS1AttrPrinterLPRQueue "dsAttrTypeStandard:PrinterLPRQueue"
#define kDS1AttrPrinterMakeAndModel "dsAttrTypeStandard:PrinterMakeAndModel"
#define kDS1AttrPrinterType "dsAttrTypeStandard:PrinterType"
#define kDS1AttrPrinterURI "dsAttrTypeStandard:PrinterURI"
#define kDS1AttrPrintServiceInfoText "dsAttrTypeStandard:PrintServiceInfoText"
#define kDS1AttrPrintServiceInfoXML "dsAttrTypeStandard:PrintServiceInfoXML"
#define kDS1AttrPrintServiceUserData "dsAttrTypeStandard:PrintServiceUserData"
#define kDSNAttrPrinterXRISupported "dsAttrTypeStandard:PrinterXRISupported"
```

Constants

`kDS1AttrNote`
Single-value attribute for storing a note; commonly used in printer records. This attribute is available in Mac OS X v10.4 and later.

`kDS1AttrPrinter1284DeviceID`
Single-value attribute for storing a printer's IEEE 1284 DeviceID, which is used when configuring a printer. This attribute is available in Mac OS X v10.4 and later.

`kDS1AttrPrinterLPRHost`
Single-value attribute for storing the name of the host for an LPR printer in records of type `kDSStdRecordTypePrinters`.

Available in Mac OS X v10.3 and later.

`kDS1AttrPrinterLPRQueue`
Single-value attribute for storing the name of the queue for an LPR printer in records of type `kDSStdRecordTypePrinters`.

Available in Mac OS X v10.3 and later.

`kDS1AttrPrinterMakeAndModel`
Single-value attribute for storing a printer's make and model; for example, "HP LaserJet 2200". The value of this attribute is used to determine the PPD file to use when configuring a printer and is based on RFC 3712, Lightweight Directory Access Protocol (LDAP) Schema for Printer Services and RFC 2911, Internet Printing Protocol/1.1 (IPP), and the IETF IPP-LDAP Printer Record. This attribute is available in Mac OS X v10.4 and later.

`kDS1AttrPrinterType`
Single-value attribute for storing the printer type in records of type `kDSStdRecordTypePrinters`.

Available in Mac OS X v10.3 and later.

`kDS1AttrPrinterURI`
Single-value attribute for storing a printer's URI; for example, "ipp://*address*" or "smb://*server/queue*". The value of this attribute is used when configuring a printer and is based on RFC 3712, Lightweight Directory Access Protocol (LDAP) Schema for Printer Services and RFC 2911, Internet Printing Protocol/1.1 (IPP), and the IETF IPP-LDAP Printer Record. This attribute is available in Mac OS X v10.4 and later.

`kDS1AttrPrintServiceInfoText`
Single-value attribute for storing text print service information.

`kDS1AttrPrintServiceInfoXML`
Single-value attribute for storing XML print service information.

`kDS1AttrPrintServiceUserData`
Single-value attribute for storing print quota configuration or statistics (XML data). This attribute is found in records of type `kDSStdRecordTypeUsers` and `kDSStdRecordTypePrintServiceUser`.

`kDSNAttrPrinterXRISupported`
: Multi-value attribute for storing additional URIs that a printer supports. This attribute is used when configuring a printer and is based on the RFC 3712, Lightweight Directory Access Protocol (LDAP) Schema for Printer Services and RFC 2911, Internet Printing Protocol/1.1 (IPP), and the IETF IPP-LDAP Printer Record. This attribute is available in Mac OS X v10.4 and later.

Declared In
`DirectoryService/DirServicesConst.h`

Record Attribute Constants

Constants for accessing attributes that store information about records.

```
#define kDS1AttrCreationTimestamp "dsAttrTypeStandard:CreationTimeStamp"
#define kDS1AttrDateRecordCreated "dsAttrTypeStandard:DateRecordCreated"
#define kDS1AttrModificationTimestamp  "dsAttrTypeStandard:ModificationTimestamp"
#define kDS1AttrOwner "dsAttrTypeStandard:Owner"
#define kDS1AttrRecordImage "dsAttrTypeStandard:RecordImage"
#define kDS1AttrTimeToLive "dsAttrTypeStandard:TimeToLive"
#define kDS1AttrTotalSize "dsAttrTypeStandard:TotalSize"
#define kDSNAttrAllNames "dsAttrTypeStandard:AllNames"
#define kDSNAttrRecordName "dsAttrTypeStandard:RecordName"
#define kDSNAttrRecordType "dsAttrTypeStandard:RecordType"
```

Constants

`kDS1AttrCreationTimestamp`
: Single-value attribute for storing the date and time the record was created. The date and time are stored in x.208 format (YYYYMMDDHHMMSSZ) where "Z" is required to be Greenwich Mean Time (GMT). This attribute is available in Mac OS X v10.4 and later.

`kDS1AttrDateRecordCreated`
: Single-value attribute for storing the date the record was created.

  Available in Mac OS X v10.4 and later.

`kDS1AttrModificationTimestamp`
: Single-value attribute for storing the date and time the record was modified. The date and time are stored in x.208 format (YYYYMMDDHHMMSSZ) where "Z" is required to be GMT. This attribute is available in Mac OS X v10.4 and later.

`kDS1AttrOwner`
: Single-value attribute for storing the owner of a record; typically the value is an LDAP distinguished name. This attribute is available in Mac OS X v10.4 and later.

`kDS1AttrRecordImage`
: Single-value attribute for storing a record image; clients can use this attribute to force a directory service to generate a binary image of the record and all of its attributes. Not used or implemented.

`kDS1AttrTimeToLive`
: Single-value attribute for storing the recommended amount of time to cache the record's attribute values. The time is stored as an unsigned 32-bit value representing the number of seconds. For example, 300 is five minutes. This attribute is available in Mac OS X v10.4 and later.

`kDS1AttrTotalSize`
: Single-value attribute for storing checksum or meta data. Not used or implemented.

`kDSNAttrAllNames`
: Multi-value attribute for all possible names for a record; retained for backward compatibility but has never been supported.

`kDSNAttrRecordName`
: Multi-value attribute for storing a list of names and keys for a record.

`kDSNAttrRecordType`
: Multi-value attribute for storing record types; a single value is allowed for records and multiple values are allowed for directory nodes.

Declared In
`DirectoryService/DirServicesConst.h`

Search Attribute Constants

Constants for accessing search-related attributes.

```
#define kDS1AttrCSPSearchPath "dsAttrTypeStandard:CSPSearchPath"
#define kDS1AttrLSPSearchPath "dsAttrTypeStandard:LSPSearchPath"
#define kDS1AttrNSPSearchPath "dsAttrTypeStandard:NSPSearchPath"
#define kDS1AttrSearchPath "dsAttrTypeStandard:SearchPath"
#define kDS1AttrSearchPolicy "dsAttrTypeStandard:SearchPolicy"
#define kDSNAttrCSPSearchPath "dsAttrTypeStandard:CSPSearchPath"
#define kDSNAttrLSPSearchPath "dsAttrTypeStandard:LSPSearchPath"
#define kDSNAttrNSPSearchPath "dsAttrTypeStandard:NSPSearchPath"
#define kDSNAttrSearchPath "dsAttrTypeStandard:SearchPath"
```

Constants

`kDS1AttrCSPSearchPath`
: Retained for backward compatibility only.

`kDS1AttrLSPSearchPath`
: Retained for backward compatibility only.

`kDS1AttrNSPSearchPath`
: Retained for backward compatibility only.

`kDS1AttrSearchPath`
: Retained for backward compatibility only.

`kDS1AttrSearchPolicy`
: Single-value attribute for storing the search policy of a search node.

`kDSNAttrCSPSearchPath`
: Single-value attribute for storing a custom search path configured by an administrator.

`kDSNAttrLSPSearchPath`
: Single-value attribute for storing the local-only search path defined by the search node.

`kDSNAttrNSPSearchPath`
: Single-value attribute for storing the automatic search path defined by the search node.

`kDSNAttrSearchPath`
: Single-value attribute for storing the search path used by the search node.

Declared In
`DirectoryService/DirServicesConst.h`

Server Attribute Constants

Constants defined for server attributes.

```
#define kDS1AttrLocation "dsAttrTypeStandard:Location"
#define kDS1AttrPort "dsAttrTypeStandard:Port"
#define kDS1AttrServiceType "dsAttrTypeStandard:ServiceType"
#define kDS1AttrXMLPlist "dsAttrTypeStandard:XMLPlist"
```

Constants

`kDS1AttrLocation`

Single-value attribute for storing the location at which a service is available. The location is usually a domain name. This attribute is found in records of type `kDSStdRecordTypeAFPServer`, `kDSStdRecordTypeLDAPServer`, and `kDSStdRecordTypeWebServer`.

`kDS1AttrPort`

Single-value attribute for storing the port number at which a service is available. This attribute is typically found in records of type `kDSStdRecordTypeAFPServer`, `kDSStdRecordTypeLDAPServer`, and `kDSStdRecordTypeWebServer`.

`kDS1AttrServiceType`

Single-value attribute for storing the service type for a service. For example, a record of type `kDSStdRecordTypeWebserver` would have a `kDS1AttrServiceType` attribute whose value is `http` or `https`.

`kDS1AttrXMLPlist`

Single-value attribute for storing AutoServer configuration settings. Also used for storing encrypted Kerberos information in computer records when using the Open Directory delegated administration feature for adding a Kerberized server to the network.

Available in Mac OS X v10.3 and later.

Declared In

`DirectoryService/DirServicesConst.h`

Setup Assistant Attribute Constants

Constants for accessing Setup Assistant attributes.

```
#define kDS1AttrSetupAdvertising "dsAttrTypeStandard:SetupAssistantAdvertising"
#define kDS1AttrSetupAutoRegister  "dsAttrTypeStandard:SetupAssistantAutoRegister"
#define kDS1AttrSetupLocation "dsAttrTypeStandard:SetupAssistantLocation"
#define kDS1AttrSetupOccupation "dsAttrTypeStandard:Occupation"
```

Constants

`kDS1AttrSetupAdvertising`

Single-value attribute used by Setup Assistant to store advertising information.

`kDS1AttrSetupAutoRegister`

Single-value attribute used by Setup Assistant to store automatic registration information.

`kDS1AttrSetupLocation`

Single-value attribute used by Setup Assistant to store a location.

`kDS1AttrSetupOccupation`

Single-value attribute used by Setup Assistant to store an occupation.

Declared In

`DirectoryService/DirServicesConst.h`

SMB Attribute Constants

Constants for accessing SMB attributes.

```
#define kDS1AttrPrimaryNTDomain "dsAttrTypeStandard:PrimaryNTDomain"
#define kDS1AttrNTDomainComputerAccount  "dsAttrTypeStandard:NTDomainComputerAccount"
#define kDS1AttrSMBAcctFlags "dsAttrTypeStandard:SMBAccountFlags"
#define kDS1AttrSMBGroupRID "dsAttrTypeStandard:SMBGroupRID"
#define kDS1AttrSMBHome "dsAttrTypeStandard:SMBHome"
#define kDS1AttrSMBHomeDrive "dsAttrTypeStandard:SMBHomeDrive"
#define kDS1AttrSMBKickoffTime "dsAttrTypeStandard:SMBKickoffTime"
#define kDS1AttrSMBLogoffTime "dsAttrTypeStandard:SMBLogoffTime"
#define kDS1AttrSMBLogonTime "dsAttrTypeStandard:SMBLogonTime"
#define kDS1AttrSMBPrimaryGroupSID "dsAttrTypeStandard:SMBPrimaryGroupSID"
#define kDS1AttrSMBProfilePath "dsAttrTypeStandard:SMBProfilePath"
#define kDS1AttrSMBPWDLastSet "dsAttrTypeStandard:SMBPWDLastSet"
#define kDS1AttrSMBRID "dsAttrTypeStandard:SMBRID"
#define kDS1AttrSMBScriptPath "dsAttrTypeStandard:SMBScriptPath"
#define kDS1AttrSMBSID "dsAttrTypeStandard:SMBSID"
#define kDS1AttrSMBUserWorkstations "dsAttrTypeStandard:SMBUserWorkstations"
```

Constants

`kDS1AttrPrimaryNTDomain`
: Single-value attribute for storing the primary NT domain; used with the Open Directory `dsGetDirNodeInfo` function in support of Kerberos SMB server services.

  Available in Mac OS X v10.4 and later.

`kDS1AttrPrimaryNTDomainComputerAccount`
: Single-value attribute for storing the primary NT domain computer account; used with the Open Directory `dsGetDirNodeInfo` function in support of Kerberos SMB server services.

  Available in Mac OS X v10.4 and later.

`kDS1AttrSMBAcctFlags`
: Single-value attribute for storing account flags.

  Available in Mac OS X v10.3 and later.

`kDS1AttrSMBGroupRID`
: Single-value attribute used for storing information about PDC SMB interaction with Open Directory.

  Available in Mac OS X v10.3 and later.

`kDS1AttrSMBHome`
: Single-value attribute for storing the Universal Naming Convention (UNC) address of a Windows home directory mount point (\\server\\sharepoint).

  Available in Mac OS X v10.3 and later.

`kDS1AttrSMBHomeDrive`
: Single-value attribute for storing the drive letter for the home directory mount point.

  Available in Mac OS X v10.3 and later.

`kDS1AttrSMBKickoffTime`
: Single-value attribute for storing the kickoff time. Authentications before the kick off time will fail.

  Available in Mac OS X v10.3 and later.

`kDS1AttrSMBLogoffTime`
: Single-value attribute for storing the time the user last logged off.

  Available in Mac OS X v10.3 and later.

`kDS1AttrSMBLogonTime`
: Single-value attribute for storing the current log on time.

  Available in Mac OS X v10.3 and later.

`kDS1AttrSMBPrimaryGroupSID`
: Single-value attribute for storing an SMB Primary Group Security ID stored as a string of up to 64 bytes. Found in records of type `kDSStdRecordTypeUsers`, `kDSStdRecordTypeGroups`, and `kDSStdRecordTypeComputers`.

  Available in Mac OS X v10.4 and later.

`kDS1AttrSMBProfilePath`
: Single-value attribute for storing desktop management information, such as desktop links and docking information.

  Available in Mac OS X v10.3 and later.

`kDS1AttrSMBPWDLastSet`
: Single-value attribute for storing the last time the password was set.

  Available in Mac OS X v10.3 and later.

`kDS1AttrSMBRID`
: Single-value attribute used for storing information about PDC SMB interaction with Open Directory.

  Available in Mac OS X v10.3 and later.

`kDS1AttrSMBScriptPath`
: Single-value attribute for storing the login script path.

  Available in Mac OS X v10.3 and later.

`kDS1AttrSMBSID`
: Single-value attribute for storing an SMB Security ID stored as a string of up to 64 bytes. Found in records of type `kDSStdRecordTypeUsers`, `kDSStdRecordTypeGroups`, and `kDSStdRecordTypeComputers`.

  Available in Mac OS X v10.4 and later.

`kDS1AttrSMBUserWorkstations`
: Single-value attribute for storing the list of workstations user can log in from.

  Available in Mac OS X v10.3 and later.

Declared In
`DirectoryService/DirServicesConst.h`

User and Group Record Attribute Constants

Constants for accessing attributes typically found in records of type `kDSStdRecordTypeUsers` and `kDSStdRecordTypeGroups`.

```
#define kDS1AttrAdminLimits "dsAttrTypeStandard:AdminLimits"
#define kDS1AttrAdminStatus "dsAttrTypeStandard:AdminStatus"
#define kDS1AttrAlternateDatastoreLocation
"dsAttrTypeStandard:AlternateDatastoreLocation"
#define kDS1AttrAuthenticationHint "dsAttrTypeStandard:AuthenticationHint"
#define kDS1AttrChange "dsAttrTypeStandard:Change"
#define kDS1AttrComment "dsAttrTypeStandard:Comment"
#define kDS1AttrDistinguishedName "dsAttrTypeStandard:RealName"
#define kDS1AttrExpire "dsAttrTypeStandard:Expire"
#define kDS1AttrFirstName "dsAttrTypeStandard:FirstName"
#define kDS1AttrGeneratedUID "dsAttrTypeStandard:GeneratedUID"
#define kDS1AttrHomeDirectorySoftQuota  "dsAttrTypeStandard:HomeDirectorySoftQuota"
#define kDS1AttrHomeDirectoryQuota "dsAttrTypeStandard:HomeDirectoryQuota"
#define kDS1AttrHomeLocOwner "dsAttrTypeStandard:HomeLocOwner"
#define kDS1AttrInternetAlias "dsAttrTypeStandard:InetAlias"
#define kDS1AttrLastName "dsAttrTypeStandard:LastName"
#define kDS1AttrMailAttribute "dsAttrTypeStandard:MailAttribute"
#define kDS1AttrMiddleName "dsAttrTypeStandard:MiddleName"
#define kDS1AttrNFSHomeDirectory "dsAttrTypeStandard:NFSHomeDirectory"
#define kDS1AttrOriginalNFSHomeDirectory
"dsAttrTypeStandard:OriginalNFSHomeDirectory"
#define kDS1AttrPassword "dsAttrTypeStandard:Password"
#define kDS1AttrPasswordPlus "dsAttrTypeStandard:PasswordPlus"
#define kDS1AttrPicture "dsAttrTypeStandard:Picture"
#define kDS1AttrPrimaryGroupID "dsAttrTypeStandard:PrimaryGroupID"
#define kDS1AttrRealuserID "dsAttrTypeStandard:RealUserID"
#define kDS1AttrUniqueID "dsAttrTypeStandard:UniqueID"
#define kDS1AttrUserShell "dsAttrTypeStandard:UserShell"
#define kDSNAttrAddressLine1 "dsAttrTypeStandard:AddressLine1"
#define kDS1StandardAttrHomeLocOwner "DS1AttrHomeLocOwner"
#define kDSNAttrAddressLine2 "dsAttrTypeStandard:AddressLine2"
#define kDSNAttrAddressLine3 "dsAttrTypeStandard:AddressLine3"
#define kDSNAttrAreaCode "dsAttrTypeStandard:AreaCode"
#define kDSNAttrAuthenticationAuthority  "dsAttrTypeStandard:AuthenticationAuthority"
#define kDSNAttrBuilding "dsAttrTypeStandard:Building"
#define kDSNAttrCity "dsAttrTypeStandard:City"
#define kDSNAttrCountry "dsAttrTypeStandard:Country"
#define kDSNAttrDepartment "dsAttrTypeStandard:Department"
#define kDSNAttrEMailAddress "dsAttrTypeStandard:EMailAddress"
#define kDSNAttrFaxNumber "dsAttrTypeStandard:FAXNumber"
#define kDSNAttrGroupMembers "dsAttrTypeStandard:GroupMembers
#define kDSNAttrGroupMembership "dsAttrTypeStandard:GroupMembership"
#define kDSNAttrHomeDirectory "dsAttrTypeStandard:HomeDirectory"
#define kDSNAttrIMHandle "dsAttrTypeStandard:IMHandle"
#define kDSNAttrJobTitle "dsAttrTypeStandard:JobTitle"
#define kDSNAttrMobileNumber "dsAttrTypeStandard:MobileNumber"
#define kDSNAttrNamePrefix "dsAttrTypeStandard:NamePrefix"
#define kDSNAttrNameSuffix "dsAttrTypeStandard:NameSuffix"
#define kDSNAttrNestedGroups "dsAttrTypeStandard:NestedGroups"
#define kDSNAttrNetGroups "dsAttrTypeStandard:NetGroups"
#define kDSNAttrNickName "dsAttrTypeStandard:NickName"
#define kDSNAttrOrganizationName "dsAttrTypeStandard:OrganizationName"
#define kDSNAttrOriginalHomeDirectory  "dsAttrTypeStandard:OriginalHomeDirectory"
#define kDSNAttrPagerNumber "dsAttrTypeStandard:PagerNumber"
#define kDSNAttrPhoneNumber "dsAttrTypeStandard:PhoneNumber"
#define kDSNAttrPostalAddress "dsAttrTypeStandard:PostalAddress"
#define kDSNAttrPostalCode "dsAttrTypeStandard:PostalCode"
#define kDSNAttrState "dsAttrTypeStandard:State"
```

```
#define kDSNAttrStreet "dsAttrTypeStandard:Street"
```

Constants

`kDS1AttrAdminLimits`
: Single-value attribute for storing an XML plist indicating what the user can edit as an administrator.

`kDS1AttrAdminStatus`
: Single-value attribute for storing an administrator status; retained for backward compatibility.

`kDS1AttrAlternateDatastoreLocation`
: Single-value attribute for storing the UNIX path to the location at which a user's e-mail is stored.

  Available in Mac OS X v10.3 and later.

`kDS1AttrAuthenticationHint`
: Single-value attribute for storing the authentication hint that is displayed when an incorrect password is entered several times at `loginwindow`.

`kDS1AttrChange`
: Single-value attribute whose value indicates whether a password needs to be changed. Currently not used and usually set to zero; the Password Server and ShadowHash provide this functionality now.

`kDS1AttrComment`
: Single-value attribute for storing an unformatted comment.

`kDSNAttrDepartment`
: Multi-value attribute for storing the department name of a user or group.

  Available in Mac OS X v10.3 and later.

`kDS1AttrDistinguishedName`
: Single-value attribute for storing a user's real name.

`kDS1AttrExpire`
: Single-value attribute used for storing an expiration date or time, depending on the context. Currently not used and usually set to zero; the Password Server and ShadowHash provide this functionality now.

`kDS1AttrFirstName`
: Single-value attribute for storing a user's first name.

`kDS1AttrGeneratedUID`
: Single-value attribute for storing a universal unique identifier (UUID) consisting of 32 characters containing hexadecimal data, plus four dash (-) characters, for a total of 36 characters, or 128 bits.

`kDS1AttrHomeDirectorySoftQuota`
: Single-value attribute for storing the home directory size limit in bytes at which the user is notified that the hard limit has nearly been reached.

  Available in Mac OS X v10.3 and later.

`kDS1AttrHomeDirectoryQuota`
: Single-value attribute for storing the allowed usage in bytes for a user's home directory.

`kDS1AttrHomeLocOwner`
: Single-value attribute for storing the owner of a workgroup's shared home directory.

`kDS1AttrInternetAlias`
: Single-value attribute used to track Internet aliases.

`kDS1AttrLastName`
: Single-value attribute for storing a user's last name.

`kDS1AttrMailAttribute`
: Single-value attribute for storing mail account configuration information.

`kDS1AttrMiddleName`
Single-value attribute for storing a user's middle name.

`kDS1AttrNFSHomeDirectory`
Single-value attribute for storing a user's home directory path on the local machine.

`kDS1AttrOriginalNFSHomeDirectory`
Single-value attribute used in local account caching for storing the user's original NFS home directory path.

Available in Mac OS X v10.3 and later.

`kDS1AttrPassword`
Single-value attribute for storing a password or credential value.

`kDS1AttrPasswordPlus`
Single-value attribute for storing marker data to indicate possible authentication redirection.

`kDS1AttrPicture`
Single-value attribute for storing the path to the picture of each user displayed in the login window.

`kDS1AttrPrimaryGroupID`
Single-value attribute for storing the signed 32-bit unique ID representing the primary group of which the user is a member, stored in string format.

`kDS1AttrRealUserID`
Single-value attribute for storing the user's real user ID; used to support managed desktop features.

Available in Mac OS X v10.3 and later.

`kDS1AttrUniqueID`
Single-value attribute for storing a 32-bit unique ID representing the user in the legacy manner and stored in string format.

`kDS1AttrUserShell`
Single-value attribute for storing the user's shell setting.

`kDSNAttrAddressLine1`
Multi-value attribute for storing the first line of an address.

`kDSNAttrAddressLine2`
Multi-value attribute for storing the second line of an address.

`kDSNAttrAddressLine3`
Multi-value attribute for storing the third line of an address.

`kDSNAttrAreaCode`
Multi-value attribute for storing area codes.

`kDSNAttrAuthenticationAuthority`
Multi-value attribute for storing the mechanism to use when verifying or setting a user's password. If this attribute has multiple values, the first attribute returned takes precedence. This attribute is typically found in records of type `kDSStdRecordTypeUsers` and `kDSStdRecordTypeComputers`.

`kDSNAttrBuilding`
Multi-value attribute for storing the building name of a user or person.

Available in Mac OS X v10.3 and later.

`kDSNAttrCity`
Multi-value attribute for storing the names of cities; usually found in a record of type `kDSStdRecordTypeUsers`.

`kDSNAttrCountry`
: Multi-value attribute for storing the country of a user or person; usually found in records of type `kDSStdRecordTypeUsers`.

  Available in Mac OS X v10.3 and later.

`kDSNAttrEMailAddress`
: Multi-value attribute for storing e-mail addresses; usually found in records of type `kDSStdRecordTypeUsers`.

`kDSNAttrFaxNumber`
: Multi-value attribute for storing the fax numbers of a user or person; usually found in records of type `kDSStdRecordTypeUsers`.

  Available in Mac OS X v10.3 and later.

`kDSNAttrGroupMembers`
: Multi-value attribute listing member user records by record name. Found in records of type `kDSStdRecordTypeGroups`.

  Available in Mac OS X v10.3 and later.

`kDSNAttrGroupMembership`
: Multi-value attribute for storing the users that belong to a given group record.

`kDSNAttrHomeDirectory`
: Multi-value attribute for storing network home directory URLs.

`kDSNAttrIMHandle`
: Multi-value attribute for storing the Instant Messaging handles of a user. Values should be prefixed with the appropriate IM type, such as `AIM:`, `Jabber:`, `MSN:`, `Yahoo:`, and `ICQ:`.

  Available in Mac OS X v10.3 and later.

`kDSNAttrJobTitle`
: Multi-value attribute for storing the job title of a user; usually found in records of type `kDSStdRecordTypeUsers`.

  Available in Mac OS X v10.3 and later.

`kDSNAttrMobileNumber`
: Multi-value attribute for storing the mobile numbers of a user or person; usually found in records of type `kDSStdRecordTypeUsers`.

  Available in Mac OS X v10.3 and later.

`kDSNAttrNamePrefix`
: Multi-value attribute for storing the name prefix of a user, such as `Mr.`, `Ms.`, `Mrs.`, or `Dr`.

  Available in Mac OS X v10.3 and later.

`kDSNAttrNameSuffix`
: Multi-value attribute for storing the name suffix of a user, such as `Jr.`, or `Sr`.

  Available in Mac OS X v10.3 and later.

`kDSNAttrNestedGroups`
: Multi-value attribute for storing GUID values for nested groups; found in records of type `kDSStdRecordTypeGroups`. This attribute is supported in Mac OS X v10.4 and later.

`kDSNAttrNetGroups`
: Multi-value attribute for storing the net groups in which the record is a member. This attribute is found in records of type `kDSStdRecordTypeUsers`, `kDSStdRecordTypeHosts`, and `kDSStdRecordTypeNetDomains`.

`kDSNAttrNickName`
: Multi-value attribute for storing the nickname of a user or group.

  Available in Mac OS X v10.3 and later.

`kDSNAttrOrganizationName`
: Multi-value attribute for storing organization names.

`kDSNAttrOriginalHomeDirectory`
: Multi-value attribute for storing home directory URL used in local account caching.

  Available in Mac OS X v10.3 and later.

`kDSNAttrPagerNumber`
: Multi-value attribute for storing the pager numbers of a user or person; usually found in records of type `kDSStdRecordTypeUsers`.

  Available in Mac OS X v10.3 and later.

`kDSNAttrPhoneNumber`
: Multi-value attribute for storing phone numbers.

`kDSNAttrPostalAddress`
: Multi-value attribute for storing postal addresses; usually excludes the postal code.

`kDSNAttrPostalCode`
: Multi-value attribute for storing postal codes such as zip codes.

`kDSNAttrState`
: Multi-value attribute for storing the names of states or provinces.

`kDSNAttrStreet`
: Multi-value attribute for storing the street address of a user or person; usually found in records of type `kDSStdRecordTypeUsers`.

  Available in Mac OS X v10.3 and later.

Declared In
`DirectoryService/DirServicesConst.h`

VFS Attribute Constants

Constants for accessing virtual file system (VFS) attributes.

```
#define kDS1AttrVFSDumpFreq "dsAttrTypeStandard:VFSDumpFreq"
#define kDS1AttrVFSLinkDir "dsAttrTypeStandard:VFSLinkDir"
#define kDS1AttrVFSPassNo "dsAttrTypeStandard:VFSPassNo"
#define kDS1AttrVFSType "dsAttrTypeStandard:VFSType"
#define kDSNAttrVFSOpts "dsAttrTypeStandard:VFSOpts"
```

Constants

`kDS1AttrVFSDumpFreq`
: Single-value attribute for storing a dump frequency.

`kDS1AttrVFSLinkDir`
: Single-value attribute for storing the beginning of a path in a mounts record; usually is set to `/Network/Servers`. A record name is appended to the value of this attribute to create the path to mount. Maps to “dir” in NetInfo.

`kDS1AttrVFSPassNo`
: Single-value attribute for storing mount record information; usually set to zero.

`kDS1AttrVFSType`
: Single-value attribute for storing a VFS type.

`kDSNAttrVFSOpts`
: Multi-value attribute for storing VFS options.

Declared In
`DirectoryService/DirServicesConst.h`

eAttribute Flags

Constants for getting and setting an attribute's read/write status.

```
typedef enum {
keAttrReadOnly = 0x00000001,
keAttrReadWrite = 0x00000002
} eAttributeFlags;
```

Constants

`keAttrReadOnly`
: Attribute is a read-only attribute.

`keAttrReadWrite`
: Attribute can be read and written.

Discussion
This enumeration is not currently used or supported.

ePluginState Constants

Constants for setting a plug-in's state.

```
typedef enum {
kUnknownState = 0x00000000,
kActive = 0x00000001,
kInactive = 0x00000002,
kInitialized = 0x00000004,
kUninitialized = 0x00000008,
kFailedToInit = 0x00000010,
} ePluginState;
```

Constants

`kUnknownState`
: Plug-in has not yet been loaded.

`kActive`
: Plug-in is loaded, initialized, and active.

`kInactive`
: Plug-in is loaded and initialized but is not active.

`kUninitialized`
: Plug-in is loaded but not initialized.

`kFailedToInit`
: Plug-in is loaded but inactive because it failed to initialize.

Result Codes

The result codes for Open Directory are listed here. Note that some errors, such as system errors, do not appear in this list.

Result Code	Value	Description
`eDSNoErr`	0	No error occurred. Available in Mac OS X v10.0 and later.
`eDSOpenFailed`	-14000	Attempt to open an Open Directory session failed. Available in Mac OS X v10.0 and later.
`eDSCloseFailed`	-14001	Attempt to close an Open Directory session failed. Available in Mac OS X v10.0 and later.
`eDSOpenNodeFailed`	-14002	Attempt to open a node failed. Available in Mac OS X v10.0 and later.
`eDSBadDirReferences`	-14003	Specified Open Directory reference is invalid.
`eDSNullRecordReference`	-14004	Specified record reference is empty. Available in Mac OS X v10.0 and later.
`eDSMaxSessionsOpen`	-14005	Session limit has been reached. Available in Mac OS X v10.0 and later.
`eDSCannotAccessSession`	-14006	Specified session is not valid. Available in Mac OS X v10.0 and later.
`eDSDirSrvcNotOpened`	-14007	No Open Directory session has been opened. Available in Mac OS X v10.0 and later.
`eDSNodeNotFound`	-14008	Specified node could not be found. Available in Mac OS X v10.0 and later.
`eDSUnknownNodeName`	-14009	Node of the specified name is unknown. Available in Mac OS X v10.0 and later.
`eDSRegisterCustomFailed`	-14010	Registration of a custom routine failed. Available in Mac OS X v10.0 and later.
`eDSGetCustomFailed`	-14011	Unable to get a custom routine. Available in Mac OS X v10.0 and later.
`eDSUnRegisterFailed`	-14012	Deregistration of a custom routine failed. Available in Mac OS X v10.0 and later.

Result Code	Value	Description
`eDSAllocationFailed`	-14050	Requested data type could not be allocated. Available in Mac OS X v10.0 and later.
`eDSDeAllocateFailed`	-14051	Requested deallocation failed. Available in Mac OS X v10.0 and later.
`eDSCustomBlockFailed`	-14052	Custom thread block routine failed. Available in Mac OS X v10.0 and later.
`eDSCustomUnblockFailed`	-14053	Custom thread unblock routine failed. Available in Mac OS X v10.0 and later.
`eDSCustomYieldFailed`	-14054	Custom yield routine failed. Available in Mac OS X v10.0 and later.
`eDSCorruptBuffer`	-14060	Contents of buffer provided as a parameter to an Open Directory function have been corrupted. Available in Mac OS X v10.0 and later.
`eDSInvalidIndex`	-14061	Specified index is invalid. Available in Mac OS X v10.0 and later.
`eDSIndexOutOfRange`	-14062	Specified index is out of range. Available in Mac OS X v10.0 and later.
`eDSIndexNotFound`	-14063	Specified index could not be found. Available in Mac OS X v10.0 and later.
`eDSCorruptRecEntryData`	-14065	Data in a record entry structure is corrupt. Available in Mac OS X v10.0 and later.
`eDSRefSpaceFull`	-14069	Reference table is full, so a new reference could not be allocated. Available in Mac OS X v10.0 and later.
`eDSRefTableAllocError`	-14070	Allocation error occurred; the new reference could not be allocated. Available in Mac OS X v10.0 and later.
`eDSInvalidReference`	-14071	Specified reference is invalid. Available in Mac OS X v10.0 and later.
`eDSInvalidRefType`	-14072	Specified reference has an invalid type. Available in Mac OS X v10.0 and later.

Result Code	Value	Description
`eDSInvalidDirRef`	-14073	Specified Open Directory reference is invalid. Available in Mac OS X v10.0 and later.
`eDSInvalidNodeRef`	-14074	Specified node reference is invalid. Available in Mac OS X v10.0 and later.
`eDSInvalidRecordRef`	-14075	Specified record reference is invalid. Available in Mac OS X v10.0 and later.
`eDSInvalidAttrListRef`	-14076	Specified attribute list reference is invalid. Available in Mac OS X v10.0 and later.
`eDSInvalidAttrValueRef`	-14077	Specified attribute value reference is invalid. Available in Mac OS X v10.0 and later.
`eDSInvalidContinueData`	-14078	Specified continuation date is invalid. Available in Mac OS X v10.0 and later.
`eDSInvalidBuffFormat`	-14079	Specified buffer format is invalid. Available in Mac OS X v10.0 and later.
`eDSInvalidPatternMatchType`	-14080	Specified pattern match type is invalid. Available in Mac OS X v10.0 and later.
`eDSRefTableError`	-14081	Reference table error occurred. Available in Mac OS X v10.2 and later.
`eDSRefTableNilError`	-14082	Reference table error occurred. Available in Mac OS X v10.3 and later.
`eDSRefTableIndexOfBoundsError`	-14083	Reference table index error occurred.
`eDSRefTableEntryNilError`	-14084	Reference table error occurred. Available in Mac OS X v10.3 and later.
`eDSRefTableCSBPAllocError`	-14085	Reference table client side buffer parsing (CSBP) error occurred. Available in Mac OS X v10.3 and later.
`eDSRefTableFWAllocError`	-14086	Reference table allocation error occurred. Available in Mac OS X v10.3 and later.
`eDSAuthFailed`	-14090	Authentication failed. Available in Mac OS X v10.0 and later.

Result Code	Value	Description
`eDSAuthMethodNotSupported`	-14091	Specified authentication is not supported. Available in Mac OS X v10.0 and later.
`eDSAuthResponseBufTooSmall`	-14092	Response buffer is too small for the response data. Available in Mac OS X v10.0 and later.
`eDSAuthParameterErr`	-14093	Authentication parameter is invalid.
`eDSAuthInBuffFormatError`	-14094	Buffer format error occurred during the authentication process. Available in Mac OS X v10.0 and later.
`eDSAuthNoSuchEntity`	-14095	Specified entity does not exist. Available in Mac OS X v10.0 and later.
`eDSAuthBadPassword`	-14096	Specified password is invalid. Available in Mac OS X v10.0 and later.
`eDSAuthContinueDataBad`	-14097	Specified authentication continuation data is invalid. Available in Mac OS X v10.0 and later.
`eDSAuthUnknownUser`	-14098	User is not known. Available in Mac OS X v10.0 and later.
`eDSAuthInvalidUserName`	-14099	User name is not valid. Available in Mac OS X v10.0 and later.
`eDSAuthCannotRecoverPasswd`	-14100	User's password could not be read. Available in Mac OS X v10.0 and later.
`eDSAuthFailedClearTextOnly`	-14101	Authentication failed because the specified authentication method requested that clear text authentication not be used (`kDSStdAuthNodeNativeNoClearText`), but clear text authentication is the only available method. Available in Mac OS X v10.0 and later.
`eDSAuthNoAuthServerFound`	-14102	No authentication server was found. Available in Mac OS X v10.0 and later.
`eDSAuthServerError`	-14103	Authentication server reported an error. Available in Mac OS X v10.0 and later.

Result Code	Value	Description
`eDSInvalidContext`	-14104	Continuation data does not fit with the other parameters. Available in Mac OS X v10.0 and later.
`eDSBadContextData`	-14105	Specified continuation data is bad. Available in Mac OS X v10.0 and later.
`eDSPermissionError`	-14120	Permission error occurred. Available in Mac OS X v10.0 and later.
`eDSReadOnly`	-14121	Write operation was attempted on data that is read only. Available in Mac OS X v10.0 and later.
`eDSInvalidDomain`	-14122	Specified domain is invalid. Available in Mac OS X v10.0 and later.
`eNetInfoError`	-14123	NetInfo error occurred. Available in Mac OS X v10.0 and later.
`eDSInvalidRecordType`	-14130	Specified record type is invalid. Available in Mac OS X v10.0 and later.
`eDSInvalidAttributeType`	-14131	Specified attribute type is invalid. Available in Mac OS X v10.0 and later.
`eDSInvalidRecordName`	-14133	Specified record name is invalid. Available in Mac OS X v10.0 and later.
`eDSAttributeNotFound`	-14134	Specified attribute could not be found. Available in Mac OS X v10.0 and later.
`eDSRecordAlreadyExists`	-14135	Specified record already exists. Available in Mac OS X v10.0 and later.
`eDSRecordNotFound`	-14136	Requested record was not found. Available in Mac OS X v10.0 and later.
`eDSAttributeDoesNotExist`	-14137	Specified attribute does not exist. Available in Mac OS X v10.0 and later.
`eDSNoStdMappingAvailable`	-14140	Standard mapping is not available for the specified attribute. Available in Mac OS X v10.0 and later.

Result Code	Value	Description
`eDSInvalidNativeMapping`	-14141	Native mapping is not available for the specified attribute. Available in Mac OS X v10.0 and later.
`eDSSchemaError`	-14142	Write operation failed because the result would conflict with the server's schema, such as trying to remove a required attribute. Available in Mac OS X v10.2 and later.
`eDSAttributeValueNotFound`	-14143	Specified attribute value could not be obtained, set, or removed. Available in Mac OS X v10.2 and later.
`eDSVersionMismatch`	-14149	Configuration file version is not compatible with this version of Open Directory or with the plug-in that loaded it. Available in Mac OS X v10.2 and later.
`eDSPlugInConfigFileError`	-14150	Error occurred with the plug-in’s configuration file. Available in Mac OS X v10.0 and later.
`eDSInvalidPlugInConfigData`	-14151	Specified plug-in’s configuration data is invalid. Available in Mac OS X v10.0 and later.
`eDSAuthNewPasswordRequired`	-14161	Password must be reset now. This result code is returned when authentication is successful and account settings require the setting of a new password. Available in Mac OS X v10.2 and later.
`eDSAuthPasswordExpired`	-14162	Password has expired and must be reset now. This result code is returned when authentication is successful and account settings indicate that the password has expired. Available in Mac OS X v10.2 and later.
`eDSAuthPasswordQualityCheckFailed`	-14165	New password does not meet security standards. This error only occurs when setting or changing a password, not when authenticating. Available in Mac OS X v10.2 and later.
`eDSAuthAccountDisabled`	-14167	Account is disabled. Available in Mac OS X v10.2 and later.
`eDSAuthAccountExpired`	-14168	Account has been automatically disabled because its expiration time has passed. Available in Mac OS X v10.2 and later.

Result Code	Value	Description
`eDSAuthAccountInactive`	-14169	Account has been automatically disabled because it was not used for a preset amount of time. Available in Mac OS X v10.2 and later.
`eDSAuthPasswordTooShort`	-14170	New password does not meet the password server's minimum length requirements; this result is returned when changing or setting a password, not when authenticating. Available in Mac OS X v10.2 and later.
`eDSAuthPasswordTooLong`	-14171	New password does not meet the password server's maximum length limit; this result is returned when changing or setting a password, not when authenticating. Available in Mac OS X v10.2 and later.
`eDSAuthPasswordNeedsLetter`	-14172	New password does contain a letter; this result is returned when changing or setting a password, not when authenticating. Available in Mac OS X v10.2 and later.
`eDSAuthPasswordNeedsDigit`	-14173	New password does not contain a number; this result is returned when changing or setting a password, not when authenticating. Available in Mac OS X v10.2 and later.
`eDSAuthPasswordChangeTooSoon`	-14174	Attempt to change a password occurred too soon. Available in Mac OS X v10.3 and later.
`eDSAuthInvalidLogonHours`	-14175	Attempt to log in at an inappropriate time. Available in Mac OS X v10.3 and later.
`eDSAuthInvalidComputer`	-14176	Attempt to log in from the wrong computer. Available in Mac OS X v10.3 and later.
`eDSAuthMasterUnreachabe`	-14177	Authentication could not be completed because a writable Open Directory replica could not be reached.
`eDSNullParameter`	-14200	Required parameter is null. Available in Mac OS X v10.0 and later.
`eDSNullDataBuff`	-14201	Specified data buffer is null. Available in Mac OS X v10.0 and later.
`eDSNullNodeName`	-14202	Specified node name is null. Available in Mac OS X v10.0 and later.

Result Code	Value	Description
`eDSNullRecEntryPtr`	-14203	Specified record entry pointer is null. Available in Mac OS X v10.0 and later.
`eDSNullRecName`	-14204	Specified record name is null. Available in Mac OS X v10.0 and later.
`eDSNullRecNameList`	-14205	Specified record name list is null. Available in Mac OS X v10.0 and later.
`eDSNullRecType`	-14206	Specified record type is null. Available in Mac OS X v10.0 and later.
`eDSNullRecTypeList`	-14207	Specified record type list is null. Available in Mac OS X v10.0 and later.
`eDSNullAttribute`	-14208	Specified attribute is null. Available in Mac OS X v10.0 and later.
`eDSNullAttributeAccess`	-14209	Reserved. Available in Mac OS X v10.0 and later.
`eDSNullAttributeValue`	-14210	Specified attribute value is null. Available in Mac OS X v10.0 and later.
`eDSNullAttributeType`	-14211	Specified attribute type is null. Available in Mac OS X v10.0 and later.
`eDSNullAttributeTypeList`	-14212	Specified attribute type list is null. Available in Mac OS X v10.0 and later.
`eDSNullAttributeControlPtr`	-14213	Reserved. Available in Mac OS X v10.0 and later.
`eDSNullAttributeRequestList`	-14214	Specified attribute request list is null. Available in Mac OS X v10.0 and later.
`eDSNullDataList`	-14215	Specified data list is empty. Available in Mac OS X v10.0 and later.
`eDSNullDirNodeTypeList`	-14216	Specified node type list is empty. Available in Mac OS X v10.0 and later.
`eDSNullAutMethod`	-14217	Specified authentication method is null. Available in Mac OS X v10.0 and later.

Result Code	Value	Description
`eDSNullAuthStepData`	-14218	Specified authentication step data is null. Available in Mac OS X v10.0 and later.
`eDSNullAuthStepDataResp`	-14219	Specified authentication step data response is null. Available in Mac OS X v10.0 and later.
`eDSNullNodeInfoTypeList`	-14220	Specified node information type list is null. Available in Mac OS X v10.0 and later.
`eDSNullPatternMatch`	-14221	Specified pattern to match is null. Available in Mac OS X v10.0 and later.
`eDSNullNodeNamePattern`	-14222	Specified node name pattern is null. Available in Mac OS X v10.0 and later.
`eDSNullTargetArgument`	-14223	Specified target argument is null. Available in Mac OS X v10.0 and later.
`eDSEmptyParameter`	-14230	Parameter is empty. Available in Mac OS X v10.0 and later.
`eDSEmptyBuffer`	-14231	Buffer is empty. Available in Mac OS X v10.0 and later.
`eDSEmptyNodeName`	-14232	Specified node name is empty. Available in Mac OS X v10.0 and later.
`eDSEmptyRecordName`	-14233	Specified record name is empty. Available in Mac OS X v10.0 and later.
`eDSEmptyRecordNameList`	-14234	Specified list of record names is empty. Available in Mac OS X v10.0 and later.
`eDSEmptyRecordType`	-14235	Specified record type is empty. Available in Mac OS X v10.0 and later.
`eDSEmptyRecordTypeList`	-14236	Specified list of record types is empty. Available in Mac OS X v10.0 and later.
`eDSEmptyRecordEntry`	-14237	Specified record entry is empty. Available in Mac OS X v10.0 and later.
`eDSEmptyPatternMatch`	-14238	Specified pattern is empty. Available in Mac OS X v10.0 and later.

Result Code	Value	Description
eDSEmptyNodeNamePattern	-14239	Specified node name pattern is empty. Available in Mac OS X v10.0 and later.
eDSEmptyAttribute	-14240	Specified attribute is empty. Available in Mac OS X v10.0 and later.
eDSEmptyAttributeType	-14241	Specified attribute type is empty. Available in Mac OS X v10.0 and later.
eDSEmptyAttributeTypeList	-14242	Specified list of attribute types is empty. Available in Mac OS X v10.0 and later.
eDSEmptyAttributeValue	-14243	Specified attribute value is empty. Available in Mac OS X v10.0 and later.
eDSEmptyAttributeRequestList	-14244	Specified list of attribute requests is empty. Available in Mac OS X v10.0 and later.
eDSEmptyDataList	-14245	Specified data list is empty. Available in Mac OS X v10.0 and later.
eDSEmptyNodeInfoTypeList	-14246	Specified node information type list is empty. Available in Mac OS X v10.0 and later.
eDSEmptyAuthMethod	-14247	Specified authentication method is empty. Available in Mac OS X v10.0 and later.
eDSEmptyAuthStepData	-14248	Specified authentication step data is empty. Available in Mac OS X v10.0 and later.
eDSEmptyAuthStepDataResp	-14249	Response to an authentication step is empty. Available in Mac OS X v10.0 and later.
eDSEmptyPattern2Match	-14250	Specified pattern to match is empty. Available in Mac OS X v10.0 and later.
eDSBadDataNodeLength	-14255	Specified data not length is invalid. Available in Mac OS X v10.0 and later.
eDSBadDataNodeFormat	-14256	Specified data not format is invalid. Available in Mac OS X v10.0 and later.
eDSBadSourceDataNode	-14257	Specified source data node is invalid. Available in Mac OS X v10.0 and later.

Result Code	Value	Description
`eDSBadTargetDataNode`	-14258	Specified target data node is invalid. Available in Mac OS X v10.0 and later.
`eDSBufferTooSmall`	-14260	Attempt was made to place information in a buffer that is too small. Nothing has been placed in the buffer. Available in Mac OS X v10.0 and later.
`eDSUnknownMatchType`	-14261	Specified match type is not known. Available in Mac OS X v10.0 and later.
`eDSUnSupportedMatchType`	-14262	Specified match type is not supported. Available in Mac OS X v10.0 and later.
`eDSInvalDataList`	-14263	Specified data list is invalid. Available in Mac OS X v10.0 and later.
`eDSAttrListError`	-14264	Attribute list error occurred. Available in Mac OS X v10.0 and later.
`eServerNotRunning`	-14270	Server is not running. Available in Mac OS X v10.0 and later.
`eUnknownAPICall`	-14271	Unknown call was attempted. Available in Mac OS X v10.0 and later.
`eUnknownServerError`	-14272	Server error occurred. Available in Mac OS X v10.0 and later.
`eUnknownPlugIn`	-14273	Specified plug-in is unknown. Available in Mac OS X v10.0 and later.
`ePlugInDataError`	-14274	Plug-in data error occurred. Available in Mac OS X v10.0 and later.
`ePlugInNotFound`	-14275	Specified plug-in was not found. Available in Mac OS X v10.0 and later.
`ePlugInError`	-14276	Plug-in error occurred. Available in Mac OS X v10.0 and later.
`ePlugInInitError`	-14277	Plug-in initialization error occurred. Available in Mac OS X v10.0 and later.
`ePlugInNotActive`	-14278	Specified plug-in is not active. Available in Mac OS X v10.0 and later.

Result Code	Value	Description
`ePlugInFailedToInitialize`	-14279	Specified plug-in could not initialize itself. Available in Mac OS X v10.0 and later.
`ePlugInCallTimedOut`	-14280	Call to a plug-in timed out. Available in Mac OS X v10.0 and later.
`eNoSearchNodesFound`	-14290	No search nodes were found. Available in Mac OS X v10.0 and later.
`eSearchPathNotDefined`	-14291	No search path is defined. Available in Mac OS X v10.0 and later.
`eNotHandledByThisNode`	-14292	Requested operation is not handled by the specified node. Available in Mac OS X v10.0 and later.
`eIPCSendError`	-14330	Send error occurred. Available in Mac OS X v10.0 and later.
`eIPCReceiveError`	-14331	Receive error occurred. Available in Mac OS X v10.0 and later.
`eServerReplyError`	-14332	Server reply error occurred. Available in Mac OS X v10.0 and later.
`eDSTCPSendError`	-14350	Error occurred when sending data to a remote server. Available in Mac OS X v10.2 and later.
`eDSTCPReceiveError`	-14351	Error occured when receiving data from a remote server. Available in Mac OS X v10.2 and later.
`eDSTCPVersionMismatch`	-14352	Mismatch of TCP versions occurred. Available in Mac OS X v10.2 and later.
`eDSIPUnreachable`	-14353	No response from the server at the specified IP address. Available in Mac OS X v10.2 and later.
`eDSUnknownHost`	-14354	Specified server could not be found. Available in Mac OS X v10.2 and later.
`ePluginHandlerNotLoaded`	-14400	Plug-in handler is not loaded. Available in Mac OS X v10.0 and later.

Result Code	Value	Description
`eNoPluginsLoaded`	-14402	No plug-ins loaded. Available in Mac OS X v10.0 and later.
`ePluginAlreadyLoaded`	-14404	Specified plug-in is already loaded. Available in Mac OS X v10.0 and later.
`ePluginVersionNotFound`	-14406	Plug-in's version is not specified in its configuration file. Available in Mac OS X v10.0 and later.
`ePluginNameNotFound`	-14408	Plug-in's name is not specified in its configuration file. Available in Mac OS X v10.0 and later.
`eNoPluginFactoriesFound`	-14410	Plug-in factories were not found. Available in Mac OS X v10.0 and later.
`ePluginConfigAvailNotFound`	-14412	Plug-in's property list does not contain a CFBundleConfigAvail statement. Available in Mac OS X v10.0 and later.
`ePluginConfigFileNotFound`	-14414	Plug-in's property list does not contain a CFBundleConfigFile statement. Available in Mac OS X v10.0 and later.
`eCFMGetFileSysRepErr`	-14450	Error occurred getting a file system report from the Code Fragment Manager. Available in Mac OS X v10.0 and later.
`eCFPlugInGetBundleErr`	-14452	Error occurred getting the bundle for a plug-in from the Code Fragment Manager. Available in Mac OS X v10.0 and later.
`eCFBndleGetInfoDictErr`	-14454	Error occurred getting the information dictionary for a plug-in from the Code Fragment Manager. Available in Mac OS X v10.0 and later.
`eCFDictGetValueErr`	-14456	Error occurred getting a value from the information dictionary. Available in Mac OS X v10.0 and later.
`eDSServerTimeout`	-14470	Server timeout occurred during authentication. Available in Mac OS X v10.0 and later.
`eDSContinue`	-14471	Authentication can continue. Available in Mac OS X v10.0 and later.

Result Code	Value	Description
`eDSInvalidHandle`	-14472	Invalid handle error occurred during authentication. Available in Mac OS X v10.0 and later.
`eDSSendFailed`	-14473	Send error occurred during authentication. Available in Mac OS X v10.0 and later.
`eDSReceiveFailed`	-14474	Receive error occurred during authentication. Available in Mac OS X v10.0 and later.
`eDSBadPacket`	-14475	Bad packet error occurred during authentication. Available in Mac OS X v10.0 and later.
`eDSInvalidTag`	-14476	Invalid tag error occurred during authentication. Available in Mac OS X v10.0 and later.
`eDSInvalidSession`	-14477	Session invalid error occurred during authentication. Available in Mac OS X v10.0 and later.
`eDSInvalidName`	-14478	Specified name is invalid. Available in Mac OS X v10.0 and later.
`eDSUserUnknown`	-14479	Specified user is unknown. Available in Mac OS X v10.0 and later.
`eDSUnrecoverablePassword`	-14480	Password could not be obtained. Available in Mac OS X v10.0 and later.
`eDSAuthenticationFailed`	-14481	Authentication failed. Available in Mac OS X v10.0 and later.
`eDSBogusServer`	-14482	Specified server is invalid. Available in Mac OS X v10.0 and later.
`eDSOperationFailed`	-14483	Specified operation failed during authentication. Available in Mac OS X v10.0 and later.
`eDSNotAuthorized`	-14484	Unauthorized operation was attempted. Available in Mac OS X v10.0 and later.
`eDSNetInfoError`	-14485	NetInfo error occurred during authentication. Available in Mac OS X v10.0 and later.

Result Code	Value	Description
`eDSContactMaster`	-14486	Changes must be written to the master rather than a replica. Available in Mac OS X v10.0 and later.
`eDSServiceUnavailable`	-14487	Authentication service is not available. Available in Mac OS X v10.0 and later.
`eFWGetDirNodeNameErr1`	-14501	Open Directory error occurred. Available in Mac OS X v10.0 and later.
`eFWGetDirNodeNameErr2`	-14502	Open Directory error occurred. Available in Mac OS X v10.0 and later.
`eFWGetDirNodeNameErr3`	-14503	Open Directory error occurred. Available in Mac OS X v10.0 and later.
`eFWGetDirNodeNameErr4`	-14504	Open Directory error occurred. Available in Mac OS X v10.0 and later.
`eParameterSendError`	-14700	Open Directory error occurred when sending a parameter to a plug-in. Available in Mac OS X v10.0 and later.
`eParameterReceiveError`	-14720	Open Directory error occurred when receiving a parameter from a plug-in. Available in Mac OS X v10.0 and later.
`eServerSendError`	-14740	Open Directory send error occurred. Available in Mac OS X v10.0 and later.
`eServerReceiveError`	-14760	Open Directory receive error occurred. Available in Mac OS X v10.0 and later.
`eMemoryError`	-14900	Open Directory experienced a memory error. Available in Mac OS X v10.0 and later.
`eMemoryAllocError`	-14901	Open Directory experienced a memory allocation error. Available in Mac OS X v10.0 and later.
`eServerError`	-14910	Server error occurred. Available in Mac OS X v10.0 and later.
`eParameterError`	-14915	Parameter error occurred. Available in Mac OS X v10.0 and later.

Result Code	Value	Description
`eDataReceiveErr_NoDirRef`	-14950	No Open Directory reference was received. Available in Mac OS X v10.0 and later.
`eDataReceiveErr_NoRecRef`	-14951	No record reference was received. Available in Mac OS X v10.0 and later.
`eDataReceiveErr_NoAttrListRef`	-14952	No attribute list reference was received. Available in Mac OS X v10.0 and later.
`eDataReceiveErr_NoAttrValueListRef`	-14953	No attribute list reference was received. Available in Mac OS X v10.0 and later.
`eDataReceiveErr_NoAttrEntry`	-14954	Plug-in did not return an attribute entry. Available in Mac OS X v10.0 and later.
`eDataReceiveErr_NoAttrValueEntry`	-14955	Plug-in did not return an attribute value entry. Available in Mac OS X v10.0 and later.
`eDataReceiveErr_NoNodeCount`	-14956	Plug-in did not return a node count. Available in Mac OS X v10.0 and later.
`eDataReceiveErr_NoAttrCount`	-14957	Plug-in did not return an attribute count. Available in Mac OS X v10.0 and later.
`eDataReceiveErr_NoRecEntry`	-14958	Plug-in did not return a record entry. Available in Mac OS X v10.0 and later.
`eDataReceiveErr_NoRecEntryCount`	-14959	Plug-in did not return a count of record entries. Available in Mac OS X v10.0 and later.
`eDataReceiveErr_NoRecMatchCount`	-14960	Plug-in did not return a count of record matches. Available in Mac OS X v10.0 and later.
`eDataReceiveErr_NoDataBuff`	-14961	Plug-in did not return a data buffer. Available in Mac OS X v10.0 and later.
`eDataReceiveErr_NoContinueData`	-14962	Plug-in did not return continuation data. Available in Mac OS X v10.0 and later.
`eDataReceiveErr_NoNodeChangeToken`	-14963	Plug-in did not provide a change token value. Available in Mac OS X v10.0 and later.
`eNoLongerSupported`	-14986	Specified call is not supported. Available in Mac OS X v10.0 and later.

Result Code	Value	Description
`eUndefinedError`	-14987	Undefined error occurred. Available in Mac OS X v10.0 and later.
`eNotYetImplemented`	-14988	Specified operation is not yet implemented. Available in Mac OS X v10.0 and later.

Document Revision History

This table describes the changes to *Open Directory Reference*.

Date	Notes
2006-05-23	Added Open Directory Plug-in reference information.
2006-04-04	New document that describes the Carbon API for using Open Directory.

Index