Inside Mac OS X

Directory Services Plug-ins

Draft. Preliminary. April 11, 2000

Contents

Figures, Tables, and Listings

P R E F A C E

About This Manual

This manual describes the programming interface for Directory Services plug-ins for Mac OS 8, Mac OS 9, and Mac OS X. Directory Services provides an abstraction layer that isolates Directory Services clients from the actual implementation of a directory system. Each Directory Services plug-in is responsible for responding to Directory Services clients that request service from the directory system that the plug-in represents.

You would want to write a Directory Services plug-in if you want to provide support for directory services that are not supported by Mac OS.

Conventions Used in This Manual

The Courier font is used to indicate server control calls, code, and text that you type. Terms that are defined in the glossary appear in boldface at first mention in the text. This guide includes special text elements to highlight important or supplemental information:

Note
Text set off in this manner presents sidelights or interesting points of information. ◆

IMPORTANT
Text set off in this manner—with the word Important—presents important information or instructions. ▲

▲ **WARNING**
Text set off in this manner—with the word Warning—indicates potentially serious problems. ▲

For More Information

The following books provide information that is important for Directory Services developers:

- *Directory Services.* Apple Computer, Inc.

About Directory Services Plug-ins

A Directory Services plug-in is a Mac OS X dynamically loaded library that responds to requests for directory service from applications that are clients of Directory Services.

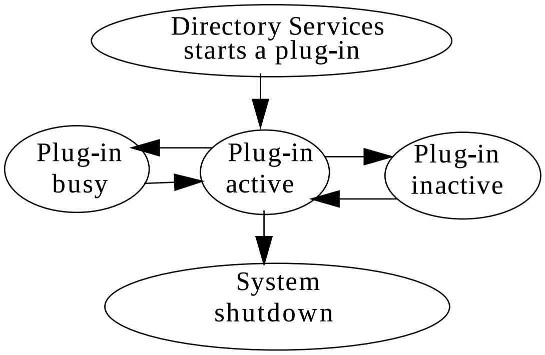
This chapter describes the runtime environment for Directory Services plug-ins, the entry points that a Directory Services plug-in must provide, the requests that a Directory Services plug-in must be prepared to respond to, the Directory Services callback routines that the plug-in can call to write entries in the Directory Services log file and to register and unregister nodes. This chapter also describes how to build and configure a Directory Services plug-in.

Runtime Environment

Plug-ins are loaded by Directory Services, which may instruct the plug-in to make itself active or inactive at any time in response to instructions entered by an administrator in the Directory Services control panel.

Figure 1-1 shows the state diagram for a Directory Services plug-in.

Figure 1-1 Directory Services plug-in state diagram



Before Directory Services starts a plug-in, the plug-in is in the unloaded state. After the plug-in loads, it is in the “loaded but not initialized” state. Until the plug-in successfully completes initialization, it is in the “attempting initialization” state. When the plug-in successfully initializes itself, it enters the active state.

As shown in Figure 1-1, a plug-in that is in the active state can enter the busy, inactive, and shutdown states. A plug-in that is in the busy state can only enter the active state, and a plug-in that is in the inactive state can only enter the active state. A plug-in that is in the shutdown state cannot enter any other state.

While in the active state, the plug-in should be prepared to be called through its periodic task, process request, shutdown, and set plug-in state entry points, as described in the next section, “Required Entry Points.”

While in the busy state, the plug-in should be prepared to be called through its periodic task and process request entry points (as described in the next section, “Required Entry Points”) and should schedule the task or request at the appropriate time.

While in the inactive state, the plug-in should be prepared to be called through its period task, set plug-in state, and shutdown entry points, as described in the next section, “Required Entry Points.”

Required Entry Points

Every Directory Services plug-in must provide the entry points described in this section. The entry points are

- `Initialize`, a routine that Directory Services calls so that the plug-in can initialize itself.
- `PeriodicTask`, a routine that Directory Services calls to perform periodic tasks.
- `ProcessRequest`, a routine that Directory Services calls to pass requests from Directory Services clients. This routine is described in detail in the next section, "Processing Directory Services Requests."
- `Shutdown`, a routine that Directory Services calls to tell the plug-in that Directory Services is shutting down. For example, this routine would be called when the system shuts down. The plug-in's shutdown routine should release memory and perform any other necessary tasks.
- `SetPluginState`, a routine that Directory Services calls to notify the plug-in of a change in state. For example, this routine would be called if the network administrator enabled or disabled this plug-in.

Processing Directory Services Requests

Directory Services passes to the appropriate Directory Services plug-in certain requests from Directory Services clients. The requests correspond to a subset of the Directory Services function calls described in *Inside Mac OS X: Directory Services*. Every Directory Services plug-in must be prepared to process each of the requests described in this section even if only to respond that the requested service is not available (`eDSServiceUnavailable`). To indicate the outcome of processing a request, the plug-in should return a result code from the list of result codes documented in *Inside Mac OS X: Directory Services*.

The plug-in must be prepared to process requests for each of the Directory Services functions described in this section.

Table 1-1 Directory Services functions that are passed to plug-ins

dsAddAttribute	dsGetAttributeValue	dsRemoveAttribute
dsAddAttributeValue	dsGetCustomAllocate	dsRemoveAttributeValue
dsCloseDirNode	dsGetCustomThread*	dsSetAttributeAccess
dsCloseRecord	dsGetDirNodeInfo	dsSetAttributeFlags
dsCreateRecord	dsGetRecordAttributeInfo	dsSetAttributeValue
dsCreateRecordAndOpen	dsGetRecordAttributeValueByID	dsSetRecordAccess
dsDeleteRecord	dsGetRecordEntry	dsSetRecordFlags
dsDoAttributeValueSearch	dsGetRecordList	dsSetRecordName
dsDoDirNodeAuth	dsGetRecordReferenceInfo	dsSetRecordType
dsDoSetPassword	dsOpenDirNode	dsUnRegisterCustomMemory*
dsDoPluginCustomCall	dsOpenRecord	dsUnRegisterCustomThread*
dsFlushRecord	dsRegisterCustomMemory*	
dsGetAttributeEntry	dsRegisterCustomThread*	

*Applies only to Directory Services plug-ins that run on Mac OS 8 or Mac OS 9.

Directory Services plug-ins must also be prepared to receive messages that Directory Services may send, such as notification of power management information, the ejection of a CD-ROM disc, or an IP address change.

Processing Concurrent Requests

Directory Services plug-ins may be called multiple times by multiple applications. For example, the following requests may occur at the same time:

- Application A makes a request that takes a long length of time to complete.
- Application B makes a request that takes a short length of time to complete.
- Application C makes a request that takes a medium length of time to complete.

Directory Services passes requests to the responsible plug-in as the requests come in and does not manage or serialize requests in any way. The plug-in is responsible for handling multiple concurrent requests in any way that it deems appropriate. It may choose to process Application A’s request first and Application A’s request last, process the requests serially, or use some other algorithm for determining the order in which to process concurrent requests.

Directory Services Callbacks

Directory Services plug-ins can call Directory Services callback routines. The callback routines are:

- `DebugLog`. Writes an entry in the Directory Services log file. All records written by all Directory Services plug-ins are written to the same log file in the order by which Directory Services receives them.
- `RegisterNode`. Registers a node so that it appears in the Directory Services control panel, thereby allowing an administrator to make the plug-in active.
- `UnregisterNode`. Unregisters a node that was previously registered. A node that is not registered does not appear in the Directory Services control panel.

Calling Mac OS Functions

Directory Services plug-ins can call any Mac OS functions that are safe to call.

Managing References

Directory Services allocates Directory Service references, such as open directory node references, open record references, and attribute list value references, and passes them to the appropriate plug-in as part of a process request. Plug-ins can use these references to keep track of their own data. When a reference becomes invalid, such as when an open directory node is closed, the plug-in must free any memory that is associated with the now invalid reference.

Standard Record and Attribute Types

Plug-ins must honor all standard record and attribute types as documented in *Inside Mac OS X: Directory Services* by mapping the standard record and attribute types to the plug-in's native record and attribute types. Plug-ins must also honor the meta types described in that document. (Meta types are types that are created dynamically, such as a user's current location.)

Plug-ins are free to support as many native record band attribute types as they want.

Authentication

Directory Services plug-ins should provide support for at least one authentication type. The standard authentication types are

- Clear text
- Two-way random
- APOP
- UNIX encryption
- SMB
- Node native

Plug-ins can also support as many native authentication types as desired.

Building a Directory Services Plug-in

A Directory Services plug-in is a standard Mac OS X “package” and follows the guidelines defined for Mac OS X packages.

Directory Services plug-ins are loaded from the following directory:

```
/System/Library/Frameworks/DirectoryServices/Resources/Plugins
```

or from other directories that may be defined later by Mac OS X.

Directory Services plug-ins use CFLOAD.

No special linker commands required to build a plug-in.

To build a Directory Services plug-in, you must include a property list file. Here is the property list file for a plug-in named `SamplePlugin`:

Listing 1-1 Property list for a sample plug-in

```
{
    "CFBundleExecutable" = "SamplePlugin";
    "CFBundleIdentifier" = "com.apple.iServers.SamplePlugin";
    "CFBundleVersion" = "1.0.0d1";
    "CFBundlePackageType" = "dspi";
    "CFBundlePackageSignature" = "adss";
    "CFPlugInDynamicRegistration" = "NO";
    "CFPlugInFactories" = {
        "D970D52E-D515-11D3-9FF9-000502C1C736" = "ModuleFactory";
    };
    "CFPlugInTypes" = {
        "697B5D6C-87A1-1226-89CA-000502C1C736" =
("D970D52E-D515-11D3-9FF9-000502C1C736");
    };
    "DSPluginPrefix" = "SamplePlugin";
    "DSServerSignature" = "Samp";
}
```

In Listing 1-1, the plug-in is responsible for setting the values of `CFBundleExecutable`, `CFBundleIdentifier`, `CFBundleVersion`, `CFBundlePackageType`, `DSPlugInTypes`, `DSPluginPrefix`, `and DSServerSignature`. The value of `CFPlugInFactories` must always be the value shown in Listing 1-1.

The value of `DSPlugInTypes` must be a UNIX unique identifier (UUID). Use the `makeUUID` utility to generate the identifier for your plug-in.

The value of `CFPluginDynamicRegistration` must be `NO`.

Configuring a Directory Services Plug-in

Developers must provide a control panel for the system administrator to use to configure the plug-in.

Directory Services Plug-in Reference

This chapter describes the entry points and requests that a Directory Services plug-in must support as well as the callback routines that a Directory Services plug-in can call.

Directory Service Plug-in Entry Points

This section describes the entry points that a Directory Services plug-in must provide.

Initialize

Initializes the plug-in

```
sInt32 Initialize (uInt32 inSignature);
```

`inSignature`	A value of type `uInt32` that uniquely identifies the plug-in.
result	A value of type `sInt32`. If the `Initialize` routine completes successfully, it should return `dsNoErr`. If it encounters an error, it should return `ePlugInInitError`.

DISCUSSION

The `Initialize` routine initializes the plug-in and prepares it to run.

PeriodicTask

Performs a periodic task.

```
sInt32 PeriodicTask (void);
```

result A value of type `sInt32`. If the `PeriodicTask` routine completes successfully, it should return `dsNoErr`. If it encounters an error, it should return `ePlugInError`.

DISCUSSION

The `PeriodicTask` routine performs a periodic task.

ProcessRequest

Processes requests.

```
sInt32 ProcessRequest (void *inData);
```

`inData` A pointer to an arbitrary value containing the request that is to be processed.

result A value of type `sInt32`. If the `ProcessRequest` routine completes successfully, it should return `dsNoErr`. If it encounters an error, it should returns an appropriate result code from the list of result codes described in *Inside Mac OS X: Directory Services*.

DISCUSSION

The `ProcessRequest` routine processes the request pointed to by `inData`. The request consists of a structure whose fields vary depending on the request type, which is always identified by the first byte of the request.

SetPluginState

Processes requests.

```
sInt32 SetPluginState (ePluginState inNewState);
```

`inNewState`	A value of type `ePluginState` containing the new plug-in state. See the Discussion section below for possible values.
result	A value of type `sInt32`. If the `SetPluginState` routine completes successfully, it should return `dsNoErr`. If it routine encounters an error, it should returns an appropriate result code from the list of result codes described in *Inside Mac OS X: Directory Services*.

DISCUSSION

The `SetPluginState` routine sets the plug-in's state to the state specified by `inNewState`.

The following enumeration defines values for `inNewState`:

```
typedef enum {
    kUninitalized   = 0x00000000,
    kActive         = 0x00000001,
    kInactive       = 0x00000002,
    kSleep          = 0x00000004
} ePluginState;x
```

Shutdown

Prepares the plug-in for shut down.

```
sInt32 Shutdown (void);
```

result	A value of type `sInt32`. If the `Shutdown` routine completes successfully, it should return `dsNoErr`. If it routine encounters an error, it should returns an appropriate result code from the list of result codes described in *Inside Mac OS X: Directory Services*.

DISCUSSION

The `Shutdown` routine is called when prepares the plug-in for shut down.

Directory Services Callback Routines

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

RegisterNode

```
uInt32 RegisterNode (const uInt32 inSignature,
                     tDataList *inNode,
                     eDirNodeType inNodeType );
```

`inSignature`	A value of type `uInt32` that uniquely identifies the plug-in.
`inNode`	A pointer to a value of `tDataList` that specifies the name of the node that is to be registered.
`inNodeType`	A value of type `eDirNodeType` that specifies the type of the node that is to be registered. See the Discussion section below for possible values.
result	A value of type `uInt32` that indicates success or failure. A value of `dsNoErr` indicates success.

DISCUSSION

The `RegisterNode` routine registers the specified node.

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

```
typedef enum {
    kUnknownNodeType= 0x00000000,
    kDirNodeType    = 0x00000001,
    kLocalNodeType  = 0x00000002,
    kSearchNodeType = 0x00000004
} eDirNodeType;
```

UnregisterNode

```
uInt32 UnregisterNode (const uInt32 inSignature,
                       tDataList *inNode );
```

`inSignature`	A value of type `uInt32` that uniquely identifies the plug-in.
`inNode`	A pointer to a value of `tDataList` that specifies the name of the node that is to be unregistered.
result	A value of type `uInt32` that indicates success or failure. A value of `dsNoErr` indicates success.

DISCUSSION

The `UnregisterNode` routine unregisters the specified node.

DebugLog

```
uInt32 DebugLog (const uInt32 inSignature,
                       const char *inFormat,
                       va_list inArgs );
```

`inSignature`	A value of type `uInt32` that uniquely identifies the plug-in.
`inFormat`	A pointer to a character array.
`inArgs`	A value of type `va_list`.

DISCUSSION

The `DebugLog` routine writes the data specified by into the Directory Services log using the format specified by `inFormat`.

Request Structures

This section describes the structures that Directory Services passes to the plug-in's `ProcessRequest` entry point.

sOpenDirNode

Directory Services calls a plug-in's `ProcessRequest` entry point and passes an `sOpenDirNode` structure when a Directory Services client application calls the `dsOpenDirNode` function to open a directory node. The `sOpenDirNode` structure is defined as follows:

```
typedef struct {
    uInt32          fType;
    sInt32          fResult;
    tDirReference   fInDirRef;
    tDataListPtr    fInDirNodeName;
    tDirNodeReferencefOutNodeRef;
} sOpenDirNode;
```

`fType`	Always `kOpenDirNode`.
`fResult`	A value of type `sInt32` that indicates whether the plug-in was able to open the directory node specified by `fInDirNodeName`.
`fInDirRef`	A value of type `tDirReference` that was created when the calling application opened the Directory Services session for which this directory node is to be opened.
`fInDirNodeName`	A value of type `tDataListPtr` containing the name of the node that is to be opened.
`fOutNodeRef`	A value of type `tDirNodeReference` that uniquely identifies the opened node if the open was successful.

DISCUSSION

When a Directory Services plug-in receives a request to open a directory node, it uses the `fInDirNodeName` field to determine the name of the node to open.

If the plug-in can open the specified node, it sets `fResult` to `dsNoErr` and `fOutNodeRef` to a value that uniquely identifies the opened node and returns. If the plug-in cannot open the node, it sets `fResult` to an appropriate result code as described in I*nside Mac OS X: Directory Services* and returns.

sCloseDirNode

Directory Services calls a plug-in's `ProcessRequest` entry point and passes an `sCloseDirNode` structure when a Directory Services client application calls the `dsCloseDirNode` function to close a directory node. The `sCloseDirNode` structure is defined as follows:

```
typedef struct {
    uInt32      fType;
    sInt32      fResult;
    tDirReferencefInNodeRef;
} sCloseDirNode;
```

`fType`	Always `kCloseDirNode`.
`fResult`	A value of type `sInt32` that indicates whether the plug-in was able to close the directory node specified by `fInNodeRef`.
`fInNodeRef`	A value of type `tDirReference` that identifies the node that is to be closed. The node reference was created when the calling application opened the node that is to be closed.

DISCUSSION

When a Directory Services plug-in receives a request to close a directory node, it uses the `fInNodeRef` field to determine the node to close.

If the plug-in can close the specified node, it sets `fResult` to e`DSNoErr` and returns. If the plug-in cannot open the node, it sets `fResult` to an appropriate result code as described in I*nside Mac OS X: Directory Services* and returns.

sGetDirNodeInfo

Directory Services calls a plug-in's `ProcessRequest` entry point and passes an `sGetDirNodeInfo` structure when a Directory Services client application calls the `dsGetDirNodeInfo` function to get information about a directory node. The `sGetDirNodeInfo` structure is defined as follows:

```
typedef struct {
    uInt32              fType;
    sInt32              fResult;
    tDirNodeReference   fInNodeRef;
    tDataListPtr        fInDirNodeInfoTypeList;
    tDataBufferPtr      fOutDataBuff;
    bool                fInAttrInfoOnly;
    unsigned long       fOutAttrInfoCount;
    tAttributeListRef   fOutAttrListRef;
    tContextData        fOutContinueData;
} sGetDirNodeInfo;
```

`fType`	Always `kGetDirNodeInfo`.
`fResult`	A value of type `sInt32` that indicates whether the plug-in was able to get information about the node identified by `fInNodeRef`.
`fInNodeRef`	A value of type `tDirNodeReference` that identifies the node for which information is to be obtained. The node reference was created when the calling application opened the node.
`fInDirNodeInfoTypeList`	A value of type tDataListPtr that points to a data list containing the attribute types for which information is being requested.
`fOutDataBuff`	A value of type `tDataBufferPtr` pointing to a `tDataBuffer` structure. If the plug-in obtains the requested information, it puts the data in the data buffer pointed to by `fOutDataBuff`.
`inAttrInfoOnly`	A 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 wells as information about the attributes.

fOutAttrInfoCount
: On output, outAttributeInfoCount points to the number of attribute types present in the buffer pointed to by fOutDataBuff.

outAttrListRef
: A value of type tAttributeListRef that uniquely identifies a tAttributeEntry structure.

fOutContinueData
: A value of type tContextData. If there is more information than can fit into fOutDataBuff, set fOutContinueData to a plug-in–defined value. Otherwise, set fOutContinueData to NULL.

DISCUSSION

When a Directory Services plug-in receives a request to get information about a directory node, it uses the fInNodeRef field of the sGetDirNodeInfo structure to determine the 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 it should also return attribute values.

If the plug-in can get the specified information about the specified node, it sets fResult to eDSNoErr, puts the requested information in the buffer pointed to by fOutDataBuff, and returns. If fOutDataBuff is not large enough to hold all of the information, the plug-in sets fOutContinueData to a plug-in–defined value before it returns.

If the plug-in cannot get the requested information, it sets fResult to an appropriate result code as described in I*nside Mac OS X: Directory Services* and returns.

sGetRecordList

Directory Services calls a plug-in's ProcessRequest entry point and passes an sGetRecordList structure when a Directory Services client application calls the dsGetRecordList function to get a list of records. The sGetRecordList structure is defined as follows:

```
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;
```

DISCUSSION

(To be delivered)

sGetRecordEntry

Directory Services calls a plug-in's `ProcessRequest` entry point and passes an `sGetRecordEntry` structure when a Directory Services client application calls the `dsGetRecordEntry` function to get a record. The `sGetRecordEntry` structure is defined as follows:

```
typedef struct {
    uInt32              fType;
    sInt32              fResult;
    tDirNodeReference   fInNodeRef;
    tDataBufferPtr      fInOutDataBuff;
    unsigned long       fInRecEntryIndex;
    tAttributeListRef   fOutAttrListRef;
    tRecordEntryPtr     fOutRecEntryPtr;
} sGetRecordEntry;
```

DISCUSSION

(To be delivered)

sGetAttributeEntry

Directory Services calls a plug-in's `ProcessRequest` entry point and passes an `sGetAttributeEntry` structure when a Directory Services client application calls the `dsGetAttributeEntry` function to get an attribute. The `sGetAttributeEntry` structure is defined as follows:

```
typedef struct {
    uInt32              fType;
    sInt32              fResult;
    tDirNodeReference   fInNodeRef;
    tDataBufferPtr      fInOutDataBuff;
    tAttributeListRef   fInAttrListRef;
    unsigned long       fInAttrInfoIndex;
    tAttributeValueListReffOutAttrValueListRef;
    tAttributeEntryPtr  fOutAttrInfoPtr;
} sGetAttributeEntry;
```

DISCUSSION

(To be delivered)

sGetAttributeValue

Directory Services calls a plug-in's `ProcessRequest` entry point and passes an `sGetAttributeValue` structure when a Directory Services client application calls the `dsGetAttributeValue` function to get an attribute value. The `sGetAttributeValue` structure is defined as follows:

```
typedef struct {
    uInt32              fType;
    sInt32              fResult;
    tDirNodeReference   fInNodeRef;
    tDataBufferPtr      fInOutDataBuff;
    unsigned long       fInAttrValueIndex;
    tAttributeValueListReffInAttrValueListRef;
    tAttributeValueEntryPtrfOutAttrValue;
} sGetAttributeValue;
```

DISCUSSION

(To be delivered)

sOpenRecord

Directory Services calls a plug-in's `ProcessRequest` entry point and passes an `sOpenRecord` structure when a Directory Services client application calls the `dsOpenRecord` function to open a record. The `sOpenRecord` structure is defined as follows:

```
typedef struct {
    uInt32              fType;
    sInt32              fResult;
    tDirNodeReference   fInNodeRef;
    tDataNodePtr        fInRecType;
    tDataNodePtr        fInRecName;
    tRecordReference    fOutRecRef;
} sOpenRecord;
```

DISCUSSION

(To be delivered)

sGetRecRefInfo

Directory Services calls a plug-in's `ProcessRequest` entry point and passes an `sGetRecRefInfo` structure when a Directory Services client application calls the `dsGetRecRefInfo` function to get the name and type of a record. The `sGetRecRefInfo` structure is defined as follows:

```
typedef struct {
    uInt32              fType;
    sInt32              fResult;
    tRecordReference    fInRecRef;
    tRecordEntryPtr     fOutRecInfo;
} sGetRecRefInfo;
```

DISCUSSION

(To be delivered)

sGetRecAttribInfo

Directory Services calls a plug-in's `ProcessRequest` entry point and passes an `sGetRecAttribInfo` structure when a Directory Services client application calls the `dsGetRecordAttributeInfo` function to get information about an attribute using an attribute type. The `sGetRecAttribInfo` structure is defined as follows:

```
typedef struct {
    uInt32              fType;
    sInt32              fResult;
    tRecordReference    fInRecRef;
    tDataNodePtr        fInAttrType;
    tAttributeEntryPtr  fOutAttrInfoPtr;
} sGetRecAttribInfo;
```

DISCUSSION

(To be delivered)

sGetRecAttribValueByIndex

Directory Services calls a plug-in's `ProcessRequest` entry point and passes an `sGetRecAttribValueByIndex` structure when a Directory Services client application calls the `dsGetRecordAttributeValueByIndex` function to get the value of an attribute using an index. The `sGetRecAttribValueByIndex` structure is defined as follows:

```
typedef struct {
    uInt32              fType;
    sInt32              fResult;
    tRecordReference    fInRecRef;
    tDataNodePtr        fInAttrType;
    unsigned long       fInAttrValueIndex;
    tAttributeValueEntryPtrfOutEntryPtr;
} sGetRecAttrValueByIndex;
```

DISCUSSION

(To be delivered)

sFlushRecord

Directory Services calls a plug-in's `ProcessRequest` entry point and passes an `sFlushRecord` structure when a Directory Services client application calls the `dsFlushRecord` function to write a record. The `sFlushRecord` structure is defined as follows:

```
typedef struct {
    uInt32          fType;
    sInt32          fResult;
    tRecordReferencefInRecRef;
} sFlushRecord;
```

DISCUSSION

(To be delivered)

sCloseRecord

Directory Services calls a plug-in's `ProcessRequest` entry point and passes an `sCloseRecord` structure when a Directory Services client application calls the `dsCloseRecord` function to close a record. The `sCloseRecord` structure is defined as follows:

```
typedef struct {
    uInt32          fType;
    sInt32          fResult;
    tRecordReferencefInRecRef;
} sCloseRecord;
```

DISCUSSION

(To be delivered)

sSetRecordName

Directory Services calls a plug-in's `ProcessRequest` entry point and passes an `sSetRecordName` structure when a Directory Services client application calls the `dsSetRecordName` function to set the name of a record. The `sSetRecordName` structure is defined as follows:

```
typedef struct {
    uInt32          fType;
    sInt32          fResult;
    tRecordReferencefInRecRef;
    tDataNodePtr    fInNewRecName;
} sSetRecordName;
```

DISCUSSION

(To be delivered)

sSetRecordType

Directory Services calls a plug-in's `ProcessRequest` entry point and passes an `sSetRecordType` structure when a Directory Services client application calls the `dsSetRecordType` function to set the type of a record. The `sSetRecordType` structure is defined as follows:

```
typedef struct {
    uInt32          fType;
    sInt32          fResult;
    tRecordReferencefInRecRef;
    tDataNodePtr    fInNewRecType;
} sSetRecordType;
```

DISCUSSION

(To be delivered)

sDeleteRecord

Directory Services calls a plug-in's `ProcessRequest` entry point and passes an `sDeleteRecord` structure when a Directory Services client application calls the `dsDeleteRecord` function to delete a record. The `sDeleteRecord` structure is defined as follows:

```
typedef struct {
    uInt32          fType
    sInt32          fResult;
    tRecordReferencefInRecRef;
} sDeleteRecord;
```

DISCUSSION

(To be delivered)

sCreateRecord

Directory Services calls a plug-in's `ProcessRequest` entry point and passes an `sCreateRecord` structure when a Directory Services client application calls the `dsCreateRecord` function or `dsCreateRecordAndOpen` to create a record. The `sCreateRecord` structure is defined as follows:

```
typedef struct {
    uInt32              fType;
    sInt32              fResult;
    tDirNodeReference   fInNodeRef;
    tDataNodePtr        fInRecType;
    tDataNodePtr        fInRecName;
    bool                fInOpen;
    tRecordReferencefOutRecRef;
} sCreateRecord;
```

DISCUSSION

(To be delivered)

sSetRecordAccess

Directory Services calls a plug-in's `ProcessRequest` entry point and passes an `sSetRecordAccess` structure when a Directory Services client application calls the `dsSetRecordAccess` function to set access controls for a record. The `sSetRecordAccess` structure is defined as follows:

```
typedef struct {
    uInt32              fType;
    sInt32              fResult;
    tRecordReference    fInRecRef;
    tAccessControlEntryPtrfInNewRecAccess;
} sSetRecordAccess;
```

DISCUSSION

(To be delivered)

sSetRecordFlags

Directory Services calls a plug-in's `ProcessRequest` entry point and passes an `sSetRecordFlags` structure when a Directory Services client application calls the `dsSetRecordFlags` function to set a record's flags. The `sSetRecordFlags` structure is defined as follows:

```
typedef struct {
    uInt32              fType;
    sInt32              fResult;
    tRecordReference    fInRecRef;
    unsigned long       fInRecFlags;
} sSetRecordFlags;
```

DISCUSSION

(To be delivered)

sAddAttribute

Directory Services calls a plug-in's `ProcessRequest` entry point and passes an `sAddAttribute` structure when a Directory Services client application calls the `dsAddAttribute` function to add an attribute to a record. The `sAddAttribute` structure is defined as follows:

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

DISCUSSION

(To be delivered)

sRemoveAttribute

Directory Services calls a plug-in's `ProcessRequest` entry point and passes an `sRemoveAttribute` structure when a Directory Services client application calls the `dsRemoveAttribute` function to remove an attribute from a record. The `sRemoveAttribute` structure is defined as follows:

```
typedef struct {
    uInt32          fType;
    sInt32          fResult;
    tRecordReferencefInRecRef;
    tDataNodePtr    fInAttribute;
} sRemoveAttribute;
```

DISCUSSION

(To be delivered)

sSetAttributeAccess

Directory Services calls a plug-in's `ProcessRequest` entry point and passes an `sSetAttributeAccess` structure when a Directory Services client application calls the `dsSetAttributeAccess` function to an attribute's access controls. The `sSetAttributeAccess` structure is defined as follows:

```
typedef struct {
    uInt32              fType;
    sInt32              fResult;
    tRecordReference    fInRecRef;
    tDataNodePtr        fInAttrType;
    tAccessControlEntryPtrfInAttrAccess;
} sSetAttributeAccess;
```

DISCUSSION

(To be delivered)

sSetAttributeFlags

Directory Services calls a plug-in's `ProcessRequest` entry point and passes an `sSetAttributeFlags` structure when a Directory Services client application calls the `dsSetAttributeFlags` function to set an attribute's flags. The `sSetAttributeFlags` structure is defined as follows:

```
typedef struct {
    uInt32          fType;
    sInt32          fResult;
    tRecordReferencefInRecRef;
    tDataNodePtr    fInAttrType;
    unsigned long   fInAttrFlags;
} sSetAttributeFlags;
```

DISCUSSION

(To be delivered)

sAddAttributeValue

Directory Services calls a plug-in's `ProcessRequest` entry point and passes an `sAddAttributeValue` structure when a Directory Services client application calls the `dsAddAttributeValue` function to add a value to an attribute. The `sAddAttributeValue` structure is defined as follows:

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

DISCUSSION

(To be delivered)

sRemoveAttributeValue

Directory Services calls a plug-in's `ProcessRequest` entry point and passes an `sRemoveAttributeValue` structure when a Directory Services client application calls the `dsRemoveAttributeValue` function to remove a value from an attribute. The `sRemoveAttributeValue` structure is defined as follows:

```
typedef struct {
    uInt32          fType;
    sInt32          fResult;
    tRecordReferencefInRecRef;
    tDataNodePtr    fInAttrType;
    unsigned long   fInAttrValueID;
} sRemoveAttributeValue;
```

DISCUSSION

(To be delivered)

sSetAttributeValue

Directory Services calls a plug-in's `ProcessRequest` entry point and passes an `sSetAttributeValue` structure when a Directory Services client application calls the `dsSetAttributeValue` function to set the value of an attribute. The `sSetAttributeValue` structure is defined as follows:

```
typedef struct {
    uInt32              fType;
    sInt32              fResult;
    tRecordReference    fInRecRef;
    tDataNodePtr        fInAttrType;
    tAttributeValueEntryPtrfInAttrValueEntry;
} sSetAttributeValue;
```

DISCUSSION

(To be delivered)

sDoDirNodeAuth

Directory Services calls a plug-in's `ProcessRequest` entry point and passes an `sDoDirNodeAuth` structure when a Directory Services client application calls the `dsDoDirNodeAuth` function to authenticate a user to a directory node. The `sSDoDirNodeAuth` structure is defined as follows:

```
typedef struct {
    uInt32              fType;
    sInt32              fResult;
    tDirNodeReference   fInNodeRef;
    tDataNodePtr        fInAuthMethod;
    bool                fInDirNodeAuthOnlyFlag;
    tDataBufferPtr      fInAuthStepData;
    tDataBufferPtr      fOutAuthStepDataResponse;
    tContextData        fIOContinueData;
} sDoDirNodeAuth;
```

DISCUSSION

(To be delivered)

sDoAttrValueSearch

Directory Services calls a plug-in's `ProcessRequest` entry point and passes an `sDoAttrValueSearch` structure when a Directory Services client application calls the `dsDoAttributeValueSearch` function to search for attributes whose values match a specified pattern. The `sSDoAttrValueSearch` structure is defined as follows:

```
typedef struct {
    uInt32              fType;
    sInt32              fResult;
    tDirNodeReference   fInNodeRef;
    tDataBufferPtr      fOutDataBuff;
    tDataListPtr        fInRecTypeList;
    tDataNodePtr        fInAttrType;
    tDirPatternMatch    fInPattMatchType;
    tDataNodePtr        fInPatt2Match;
    unsigned long       fOutMatchRecordCount;
    tContextData        fIOContinueData;
} sDoAttrValueSearch;
```

DISCUSSION

(To be delivered)

Index