



I n s i d e M a c O S X

Directory Services Plug-ins



Draft. Preliminary. July 2000

Technical Publications

© Apple Computer, Inc. 2000



Apple Computer, Inc.

© 2000 Apple Computer, Inc.

All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, mechanical, electronic, photocopying, recording, or otherwise, without prior written permission of Apple Computer, Inc., except to make a backup copy of any documentation provided on CD-ROM.

The Apple logo is a trademark of Apple Computer, Inc.

Use of the “keyboard” Apple logo (Option-Shift-K) for commercial purposes without the prior written consent of Apple may constitute trademark infringement and unfair competition in violation of federal and state laws.

No licenses, express or implied, are granted with respect to any of the technology described in this book. Apple retains all intellectual property rights associated with the technology described in this book. This book is intended to assist application developers to develop applications only for Apple-labeled or Apple-licensed computers.

Every effort has been made to ensure that the information in this manual is accurate. Apple is not responsible for typographical errors.

Apple Computer, Inc.

1 Infinite Loop

Cupertino, CA 95014

408-996-1010

Apple, the Apple logo, and Macintosh are trademarks of Apple Computer, Inc., registered in the United States and other countries.

Adobe, Acrobat, and PostScript are trademarks of Adobe Systems Incorporated or its subsidiaries and may be registered in certain jurisdictions.

Helvetica and Palatino are registered trademarks of Linotype-Hell AG and/or its subsidiaries.

ITC Zapf Dingbats is a registered trademark of International Typeface Corporation.

Simultaneously published in the United States and Canada.

Even though Apple has reviewed this manual, APPLE MAKES NO WARRANTY OR REPRESENTATION, EITHER EXPRESS OR IMPLIED, WITH RESPECT TO THIS MANUAL, ITS QUALITY, ACCURACY, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. AS A RESULT, THIS MANUAL IS SOLD “AS IS,” AND YOU, THE PURCHASER, ARE ASSUMING THE ENTIRE RISK AS TO ITS QUALITY AND ACCURACY.

IN NO EVENT WILL APPLE BE LIABLE FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES RESULTING FROM ANY DEFECT OR INACCURACY IN THIS MANUAL, even if advised of the possibility of such damages.

THE WARRANTY AND REMEDIES SET FORTH ABOVE ARE EXCLUSIVE AND IN LIEU OF ALL OTHERS, ORAL OR WRITTEN, EXPRESS OR IMPLIED. No Apple dealer, agent, or employee is authorized to make any modification, extension, or addition to this warranty.

Some states do not allow the exclusion or limitation of implied warranties or liability for incidental or consequential damages, so the above limitation or exclusion may not apply to you. This warranty gives you specific legal rights, and you may also have other rights which vary from state to state.

Contents

Figures, Tables, and Listings 5

Preface **About This Manual** 7

Conventions Used in This Manual 7
For More Information 8

Chapter 1 **About Directory Services Plug-ins** 9

Runtime Environment 9
Required Entry Points 11
Processing Directory Services Requests 11
Processing Concurrent Requests 13
Directory Services Callbacks 13
Calling Mac OS Functions 14
Managing References 14
Standard Record and Attribute Types 14
Authentication 15
Building a Directory Services Plug-in 15
Configuring a Directory Services Plug-in 16

Chapter 2 **Directory Services Plug-in Reference** 17

Directory Service Plug-in Entry Points 17
Directory Services Callback Routines 21
Request Structures 25
Result Codes 61

Appendix A	Attribute Schema	69
------------	------------------	----

	Index	77
--	-------	----

Figures, Tables, and Listings

Chapter 1	About Directory Services Plug-ins	9
	Figure 1-1	Directory Services plug-in state diagram 10
	Table 1-1	Directory Services functions that causes the ProcessRequest entry point to be called 12
	Listing 1-1	Property list for a sample plug-in 16
Appendix A	Attribute Schema	69
	Table A-1	Attribute schema 69

About This Manual

This manual describes the programming interface for Directory Services plug-ins for 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.

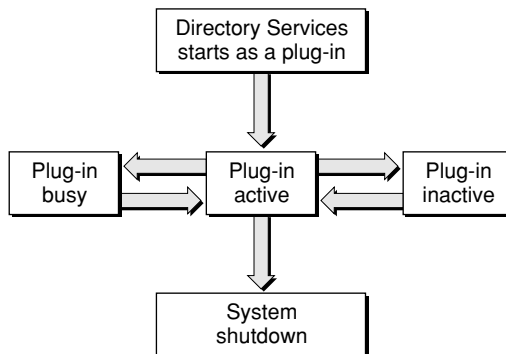
Runtime Environment

Plug-ins are loaded by Directory Services, which may instruct any 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.

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.

Figure 1-1 shows the state diagram for a Directory Services plug-in once it enters the active state.

Figure 1-1 Directory Services plug-in state diagram



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 listed below in the order in which they are typically called.

- **Initialize**, a routine that Directory Services calls so that the plug-in can initialize itself.
- **Validate**, a routine that Directory Services calls when Directory Services plug-ins are loaded in order to pass to each plug-in a unique signature that the plug-in later uses when it registers nodes with Directory Services or when it writes information in a log file.
- **Configure**, a routine that Directory Services calls when it receives a request (typically from a system administrator who has specified settings in the plug-in's control panel) for the plug-in to configure itself.
- **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 the plug-in.
- **PeriodicTask**, a routine that Directory Services calls on a regular basis so that the plug-in can 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.

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

About Directory Services Plug-ins

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 causes the ProcessRequest entry point to be called

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

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:

- `Log`. 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.
- `UnregisterAll`. Unregisters all nodes that the plug-in has previously registered.

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 and attribute types as they want.

Authentication

Directory Services plug-ins are not required to support any authentication type, but they should support at least one authentication type. The standard authentication types are

- Clear text
- Two-way random
- 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 directories:

`/System/Library/Frameworks/DirectoryServices/Resources/Plugins` (which may be read-only)
`/Library/Frameworks/DirectoryServices/Resources/Plugins` (which is always writable)

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

Directory Services loads Directory Services plug-ins using the CFLOAD load mechanism.

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

To build a Directory Services plug-in, you must provide 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" = "dsp";
    "CFBundlePackageSignature" = "adss";
    "CFPlugInDynamicRegistration" = "NO";
    "CFPlugInFactories" = {
        "D970D52E-D515-11D3-9FF9-000502C1C736" = "ModuleFactory";
    };
    "CFPlugInTypes" = { "697B5D6C-87A1-1226-89CA-000502C1C736" =
        ("D970D52E-D515-11D3-9FF9-000502C1C736");
    };
    "DSServerSignature" = "Samp";
}

```

In Listing 1-1, the plug-in is responsible for setting the values of `CFBundleExecutable`, `CFBundleIdentifier`, `CFBundleVersion`, `CFBundlePackageType`, `DSPlugInTypes`, and `DSServerSignature`.

In Listing 1-1, the first value set for `CFPlugInTypes` must always be the value shown in bold (**697B5D6C-87A1-1226-89CA-000502C1C736**).

The value of `DSPlugInFactories` must be a UNIX unique identifier (UUID) that is unique among all plug-ins. Use the `makeUUID` utility to generate the identifier for your plug-in. Your plug-in's UUID must also be the second value that appears in the `CFPlugInTypes` statement.

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

Configuring a Directory Services Plug-in

The Directory Services preference panel lists each Directory Services plug-in. When an administrator selects a plug-in and clicks the Configure button, Directory Services calls the plug-in's `Configure` entry point, which can, for example, launch an application that allows the administrator to configure the plug-in or reads a text file that contains information that configures 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. The entry points are

- `Configure` (page 17)
- `Initialize` (page 18)
- `PeriodicTask` (page 18)
- `ProcessRequest` (page 19)
- `SetPluginState` (page 20)
- `Shutdown` (page 20)

Configure

Configures the plug-in.

```
long Configure (void);
```

result A value of type `long`. If the `Configure` routine completes successfully, it should return `dsNoErr`. If it encounters an error, it should return `ePlugInConfigureError`.

DISCUSSION

A plug-in's `Configure` entry point is called when a request is made for the plug-in to configure itself. The request is typically made by a system administrator from the plug-in's control panel.

Initialize

Initializes the plug-in.

```
long Initialize (void);
```

result A value of type `long`. 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. The plug-in might, for example, open network ports and any files it requires. A Directory Services plug-in's `Initialize` routine is called just one time after all Directory Services plug-ins that can be loaded are loaded.

PeriodicTask

Performs a periodic task.

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

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

DISCUSSION

The `PeriodicTask` routine is called once every two minutes and gives the plug-in time to perform a periodic task. If a plug-in has no periodic tasks to perform, it should immediately return a result code of `dsNoErr`.

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. Plug-ins are responsible for keeping track of the time that elapses between the time the plug-in's `PeriodicTask` routine is called and when the plug-in's `PeriodicTask` routine returns.

ProcessRequest

Processes requests.

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

<i>inData</i>	A pointer to an arbitrary value containing the request that is to be processed.
<i>result</i>	A value of type <code>long</code> . If the <code>ProcessRequest</code> routine completes successfully, it should return <code>dsNoErr</code> . If it encounters an error, it should return <code>ePlugInProcessRequestError</code> .

DISCUSSION

The `ProcessRequest` routine performs the majority of work that a Directory Services plug-in is asked to do. This routine responds to all requests for directory service from applications that are clients of Directory Services.

The request is pointed to by the `inData` parameter and consists of a structure whose fields vary depending on the request type. The first byte of the structure always identifies the request type.

Each request type is described in the section “Request Structures” (page 25).

SetPluginState

Sets the plug-in's state.

```
long 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 `long`. If the `SetPluginState` routine completes successfully, it should return `dsNoErr`. If it routine encounters an error, it should return `ePlugInSetPluginStateError`.

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 {
    kUninitialized    = 0x00000000,
    kActive           = 0x00000001,
    kInactive         = 0x00000002,
    kSleep            = 0x00000004
} ePluginState;
```

Shutdown

Prepares the plug-in for shut down.

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

result A value of type `long`. If the `Shutdown` routine completes successfully, it should return `dsNoErr`. If it routine encounters an error, it should return `ePlugInShutdownError`.

DISCUSSION

The `Shutdown` routine is called in order to allow the plug-in to prepare itself for shut down.

Validate

Validates the plug-in

```
long Initialize (unsigned long inSignature);
```

inSignature A value of type `unsigned long` that uniquely identifies the plug-in.

result A value of type `long`. If the `Validate` routine completes successfully, it should return `dsNoErr`. If it encounters an error, it should return `ePlugInValidateError`.

DISCUSSION

The `Validate` routine is called during plug-in load time. Directory Services passes to the plug-in a unique signature that the plug-in later provides when it calls the `RegisterNode` (page 23) callback routine to register nodes with Directory Services or when it calls the `Log` (page 22) callback routine to write information in a log file.

Directory Services Callback Routines

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

- `Log` (page 22)
- `RegisterNode` (page 23)
- `UnregisterAll` (page 24)
- `UnregisterNode` (page 24)

Log

Writes information in a log file.

```
uInt32 Log (const uInt32 inSignature,
            eLogCode inLogCode,
            const char *inFormat,
            va_list inArgs);
```

<i>inSignature</i>	A value of type <code>uInt32</code> that uniquely identifies the plug-in.
<i>inLogCode</i>	A value of type <code>eLogCode</code> that specifies the log file in which the date is to be written. For possible values, see the <code>eLogCode</code> enumeration in the Discussion section below.
<i>inFormat</i>	A pointer to a character array that specifies the format that is to be used to write the data. For additional information, see <code>sprintf(3)</code> .
<i>inArgs</i>	A value of type <code>va_list</code> that specifies the values that are to be inserted in the format specified by <i>inFormat</i> .
<i>result</i>	A value of type <code>uInt32</code> . If the <code>Log</code> callback routine completes successfully, it returns <code>dsNoErr</code> . If the <code>Log</code> callback routine cannot complete successfully, it returns an error.

DISCUSSION

The `Log` callback routine writes the data specified by *inArgs* using the format specified by *inFormat* in the Directory Services log specified by *inLogCode*.

The following enumeration defines values for *inLogCode*:

```
typedef enum {
    kErrorLog      = 0x00000001,
    kServerLog     = 0x00000002,
    kDebugLog      = 0x00000004
} eLogCode;
```

The log files are located in the `/Library/Logs` directory and are named `DirectoryServices.Error.Log`, `DirectoryServices.Server.Log`, and `DirectoryServices.Debug.Log`.

Note

Sending a `SIGUSR1` signal to the Directory Services process from the command line toggles debug logging off and on. Debug logging is off by default. ♦

RegisterNode

Registers a node.

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

<i>inSignature</i>	A value of type <code>uInt32</code> that uniquely identifies the plug-in.
<i>inNode</i>	A pointer to a value of <code>tDataList</code> that specifies the name of the node that is to be registered.
<i>inNodeType</i>	A value of type <code>eDirNodeType</code> that specifies the type of the node that is to be registered. See the Discussion section below for possible values.
<i>result</i>	A value of type <code>uInt32</code> . If the <code>RegisterNode</code> callback routine completes successfully, it returns <code>dsNoErr</code> . If the <code>RegisterNode</code> callback routine cannot complete successfully, it returns an error, for example, if the specified node is already registered or if <code>inNode</code> contains a node name that has invalid characters.

DISCUSSION

The `RegisterNode` callback 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;
```

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

The plug-in is responsible for keeping the list registered nodes accurate. It can use the `PeriodicTask` (page 18) entry point to update the list on a regular basis.

UnregisterAll

Unregisters all of the registered nodes that are associated with a Directory Services plug-in.

```
uInt32 UnregisterNode (const uInt32 inSignature);
```

inSignature A value of type `uInt32` that uniquely identifies the plug-in.

result A value of type `uInt32`. If the `UnregisterAll` callback routine completes successfully, it returns `dsNoErr`. If the `UnregisterAll` callback routine cannot complete successfully, it returns an error.

DISCUSSION

The `UnregisterAll` callback routine unregisters all of the nodes that have been registered by the plug-in identified by `inSignature`. When Directory Services tells a plug-in to become inactive, the plug-in can call the `UnregisterAll` callback routine to unregister all of its registered nodes. The nodes must be registered again when the plug-in is told to become active.

UnregisterNode

Unregisters a node.

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

inSignature A value of type `uInt32` that uniquely identifies the plug-in.

Directory Services Plug-in Reference

<i>inNode</i>	A pointer to a value of <code>tDataList</code> that specifies the name of the node that is to be unregistered.
<i>result</i>	A value of type <code>uInt32</code> . If the <code>UnregisterNode</code> callback routine completes successfully, it returns <code>dsNoErr</code> . If the <code>UnregisterNode</code> callback routine cannot complete successfully, it returns an error.

DISCUSSION

The `UnregisterNode` callback routine unregisters the specified node.

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;
    tDirStatus      fResult;
    tDirReference   fInDirRef;
    tDataListPtr    fInDirNodeName;
    tDirNodeReference fOutNodeRef;
} sOpenDirNode;
```

<i>fType</i>	Always <code>kOpenDirNode</code> .
<i>fResult</i>	A value of type <code>tDirStatus</code> that is <code>eDSNoErr</code> to indicate that the plug-in was able to open the directory node specified by <code>fInDirNodeName</code> or an appropriate value to indicate failure.

Directory Services Plug-in Reference

<code>fInDirRef</code>	A value of type <code>tDirReference</code> that was created when the calling application opened the Directory Services session for which this directory node is to be opened.
<code>fInDirNodeName</code>	A value of type <code>tDataListPtr</code> containing the name of the node that is to be opened.
<code>fOutNodeRef</code>	A value of type <code>tDirNodeReference</code> that uniquely identifies the opened directory 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 “Result Codes” (page 61) 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;
    tDirStatus  fResult;
    tDirReferenceInNodeRef;
} sCloseDirNode;
```

<code>fType</code>	Always <code>kCloseDirNode</code> .
<code>fResult</code>	A value of type <code>tDirStatus</code> that is <code>eDSNoErr</code> to indicate that the plug-in was able to close the directory node specified by <code>fInNodeRef</code> or an appropriate value to indicate failure.

Directory Services Plug-in Reference

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 `eDSNoErr` and returns. If the plug-in cannot open the node, it sets `fResult` to an appropriate result code as described in “Result Codes” (page 61) 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;
    tDirStatus      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 `tDirStatus` that is `eDSNoErr` to indicate that the plug-in was able to get information about the node identified by `fInNodeRef` or an appropriate value to indicate failure.

Directory Services Plug-in Reference

<code>fInNodeRef</code>	A value of type <code>tDirNodeReference</code> that identifies the directory node for which information is to be obtained. The directory node reference was created when the calling application opened the directory node.
<code>fInDirNodeInfoTypeList</code>	A value of type <code>tDataListPtr</code> that points to a data list containing the attribute types for which information is being requested.
<code>fOutDataBuff</code>	A value of type <code>tDataBufferPtr</code> pointing to a <code>tDataBuffer</code> structure. If the plug-in obtains the requested information, it puts the data in the data buffer pointed to by <code>fOutDataBuff</code> .
<code>inAttrInfoOnly</code>	A Boolean value set to <code>TRUE</code> if the plug-in is only to provide information about attributes or set to <code>FALSE</code> if the plug-in is to provide the values of the attributes as well as information about the attributes.
<code>fOutAttrInfoCount</code>	On output, <code>outAttributeInfoCount</code> points to the number of attribute types present in the buffer pointed to by <code>fOutDataBuff</code> .
<code>outAttrListRef</code>	A value of type <code>tAttributeListRef</code> that uniquely identifies a <code>tAttributeEntry</code> structure.
<code>fOutContinueData</code>	A value of type <code>tContextData</code> . If there is more information than can fit into <code>fOutDataBuff</code> , set <code>fOutContinueData</code> to a plug-in-defined value. Otherwise, set <code>fOutContinueData</code> to <code>NULL</code> . Plug-ins may choose to include a timestamp in <code>fOutContinueData</code> and fail subsequent calls to <code>dsGetDirNodeInfo</code> if it determines that <code>fOutContinueData</code> is too old.

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.

Directory Services Plug-in Reference

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 “Result Codes” (page 61) 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;
    tDirStatus      fResult;
    tDirNodeReference fInNodeRef;
    tDataBufferPtr  fInDataBuff;
    tDataListPtr    fInRecNameList;
    tDirPatternMatch fInPatternMatch;
    tDataListPtr    fInRecTypeList;
    tDataListPtr    fInAttribTypeList;
    bool            fInAttribInfoOnly;
    unsigned long   fOutRecEntryCount;
    tContextData    fIOContinueData;
} sGetRecordList;
```

`fType` Always `kGetRecordList`.

`fResult` A value of type `tDirStatus` that is `eDSNoErr` to indicate that the plug-in was able to get the requested record information for the node identified by `fInNodeRef` or an appropriate value to indicate failure.

Directory Services Plug-in Reference

<code>fInNodeRef</code>	A value of type <code>tDirNodeReference</code> that identifies the directory node for which record information is to be obtained. The directory node reference was created when the calling application opened the directory node.
<code>fInDataBuff</code>	A value of type <code>tDataBufferPtr</code> pointing to the buffer in which the plug-in is to return record information.
<code>fInRecNameList</code>	A value of type <code>tDataBufferPtr</code> that points to a data list containing the names of records for which information is to be obtained. If <code>fInRecNameList</code> is <code>NULL</code> , the plug-in should return information about all records in the node identified by <code>fInNodeRef</code> .
<code>fInPatternMatch</code>	A value of type <code>tDirPatternMatch</code> containing the pattern that is to be matched. If <code>fInPatternMatch</code> is <code>NULL</code> , the plug-in should return information about all records in the node identified by <code>fInNodeRef</code> .
<code>fInRecTypeList</code>	A value of type <code>tDataListPtr</code> that points to a data list containing the types of records for which information is to be returned. If <code>fInRecTypeList</code> is <code>NULL</code> , the plug-in should return information about all records in the node identified by <code>fInNodeRef</code> .
<code>fInAttributeTypeList</code>	A value of type <code>tDataListPtr</code> that points to a data list structure containing the attribute types that are to be returned. If <code>fInAttributeTypeList</code> is <code>NULL</code> , the plug-in should return information about all of the record's attributes.
<code>fInAttributeInfoOnly</code>	A value of type <code>bool</code> that is <code>TRUE</code> if the calling application wants information about the record's attributes and <code>FALSE</code> if the calling application was the value of the record's attributes in addition to information about the attributes.
<code>fOutRecEntryCount</code>	A value of type <code>unsigned long</code> that specifies the number of records the plug-in has returned in the buffer pointed to by <code>fInDataBuff</code> .

`fIOContinueData`

A value of type `tContextData`. If there is more information than can fit into `fIOContinueData`, set `fIOContinueData` to a plug-in-defined value. Otherwise, set `fIOContinueData` to `NULL`. Plug-ins may choose to include a timestamp in `fOutContinueData` and fail subsequent calls to `dsGetDirNodeInfo` if it determines that `fOutContinueData` is too old.

DISCUSSION

When a Directory Services plug-in receives a request to get a list of records in a directory node, it uses the `fInNodeRef` field of the `sGetRecordList` structure to determine the node for which the record list is requested, the data list pointed to by `fInRecNameList` to determine the names of records for 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. If `fInPatternMatch` is not `NULL`, the plug-in should return only those records that match the specified pattern. The value of the `fInAttributeInfoOnly` field determines whether the plug-in should also return attribute values.

If the plug-in can get the requested information, it sets `fResult` to `eDSNoErr`, puts the requested information in the buffer pointed to by `fInDataBuff`, and returns. If `fInDataBuff` is not large enough to hold all of the information, the plug-in sets `fIOContinueData` 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 “Result Codes” (page 61) and returns.

`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;
    tDirStatus      fResult;
```

Directory Services Plug-in Reference

```

    tDirNodeReference    fInNodeRef;
    tDataBufferPtr       fInOutDataBuff;
    unsigned long        fInRecEntryIndex;
    tAttributeListRef     fOutAttrListRef;
    tRecordEntryPtr       fOutRecEntryPtr;
} sGetRecordEntry;

```

fType **Always** kGetRecordEntry.

fResult A value of type `tDirStatus` that is `eDSNoErr` to indicate that the plug-in was able to get the record entries for the node identified by `fInNodeRef` or an appropriate value to indicate failure.

fInNodeRef A value of type `tDirNodeReference` that identifies the directory node for which record information is to be obtained. The directory node reference was created when the calling application opened the directory node.

fInOutDataBuff A value of type `tDataBufferPtr` that points to the data buffer from which the record is to be obtained.

fInRecEntryIndex A value of type `unsigned long` that specifies the next record to get. The `fInRecEntryIndex` field contains a value that is a one-based index.

fOutAttrListRef A value of type `tAttributeListRef` that refers to an attribute list containing attribute information for the record pointed to by `fOutRecEntryPtr`. Later, the calling application may call `dsGetAttributeEntry` and pass the attribute list reference to the plug-in's routine for processing `sGetAttributeEntry` requests to get information about the attribute.

fOutRecEntryPtr A value of type `tRecordEntryPtr` that points to a `tRecordEntry` structure containing the requested record.

DISCUSSION

When a Directory Services plug-in receives a request to get a record from a `fInOutDataBuff` (which was filled in by the plug's routine that responds to

Directory Services Plug-in Reference

`kGetRecordList` requests), uses the `fInNodeRef` field to determine the record's node, and uses the `fInRecEntryIndex` field to determine which record to get.

If the plug-in can get the requested information, it sets `fResult` to `eDSNoErr`, puts the requested record in the record entry structure pointed to by `fOutRecEntryPtr`, puts the record's attribute information in the attribute list referred to by `tOutAttrListRef`, and returns.

If the plug-in cannot get the requested information, it sets `fResult` to an appropriate result code as described in "Result Codes" (page 61) and returns.

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 information about an attribute. The `sGetAttributeEntry` structure is defined as follows:

```
typedef struct {
    UInt32          fType;
    tDirStatus      fResult;
    tDirNodeReference fInNodeRef;
    tDataBufferPtr  fInOutDataBuff;
    tAttributeListRef fInAttrListRef;
    unsigned long   fInAttrInfoIndex;
    tAttributeValueListRef fOutAttrValueListRef;
    tAttributeEntryPtr fOutAttrInfoPtr;
} sGetAttributeEntry;
```

`fType` Always `kGetAttributeEntry`.

`fResult` A value of type `tDirStatus` that is `eDSNoErr` to indicate that the plug-in was able to get the requested attribute information or an appropriate value to indicate failure.

`fInNodeRef` A value of type `tDirNodeReference` that identifies the directory node of the record whose attribute information is to be obtained. The directory node reference was created when the calling application opened the directory node.

Directory Services Plug-in Reference

`fInOutDataBuff`

A value of type `tDataBufferPtr` that points to the data buffer from which the attribute entry is to be obtained

`fInAttrListRef`

A value of type `tAttributeListRef` that refers to an attribute list previously created when the plug-in's routine for responding to `sGetRecordEntry` requests received an `sGetRecordEntry` request from the calling application.

`fInAttrInfoIndex`

A value of type `unsigned long` that specifies the one-based index number of the attribute entry whose value is to be obtained.

`fOutAttrValueListRef`

A value of type `tAttributeValueListRef` that refers to an attribute value list containing the attribute value that was obtained. Later, the calling application may call `dsGetAttributeValue` and pass a pointer to `fOutAttrValueListRef` to get the attribute's value.

DISCUSSION

When a Directory Services plug-in receives a request to get a information about an attribute from a `fInOutDataBuff` (which was filled in by the plug's routine that responds to `kGetRecordList` requests), the plug-in's routine for processing `sGetAttributeEntry` requests uses the `fInNodeRef` field to determine the node of the record for which attribute information is requested, and uses the `fInAttrInfoIndex` field to determine for which attribute to get information.

If the plug-in can get the requested information from `fInOutDataBuff`, it sets `fResult` to `eDSNoErr`, puts information about the attribute and the attribute's value(s) in the attribute value list referred to by `fOutAttrValueListRef`, and returns.

If the plug-in cannot get the requested information or if the calling application is not authorized to receive the requested information, it sets `fResult` to an appropriate result code as described in "Result Codes" (page 61) and returns.

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;
    tDirStatus      fResult;
    tDirNodeReference fInNodeRef;
    tDataBufferPtr  fInOutDataBuff;
    unsigned long   fInAttrValueIndex;
    tAttributeValueListReffInAttrValueListRef;
    tAttributeValueEntryPtrfOutAttrValue;
} sGetAttributeValue;
```

`fType` Always `kGetAttributeValue`.

`fResult` A value of type `tDirStatus` that is `eDSNoErr` to indicate that the plug-in was able to get the requested attribute value or an appropriate value to indicate failure.

`fInNodeRef` A value of type `tDirNodeReference` that identifies the directory node of the record whose attribute information is to be obtained. The directory node reference was created when the calling application opened the directory node.

`fInOutDataBuff` A value of type `tDataBufferPtr` pointing to a data buffer containing record information previously obtained with the plug-in responded to an `sGetRecordEntry` request from the calling application.

`fInAttrValueIndex` A value of type `unsigned long` containing a one-based index that specifies which attribute value to get.

`fInAttrValueListRef` A value of type `tAttributeValueListRef` that references an attribute list containing all of the attribute's values.

Directory Services Plug-in Reference

fOutAttrValue A value of type `tAttributeValueEntryPtr` pointing to the attribute value list in which the plug-in is to place the value of the attribute specified by the `fInAttrValueIndex` field.

DISCUSSION

When a Directory Services plug-in receives a request to get information about an attribute, it uses the `fInNodeRef` field of the `sGetAttributeValue` structure to determine the node of the record for which an attribute value is being obtained.

If the plug-in can get the requested information, it sets `fResult` to `eDSNoErr`, puts the requested record in the record entry structure pointed to by `fOutRecEntryPtr`, puts the record's attribute information in the attribute list referred to by `tOutAttrListRef`, and returns.

If the plug-in cannot get the requested value or if the calling application is not authorized to receive the requested value, it sets `fResult` to an appropriate result code as described in "Result Codes" (page 61) and returns.

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;
    tDirStatus      fResult;
    tDirNodeReference fInNodeRef;
    tDataNodePtr    fInRecType;
    tDataNodePtr    fInRecName;
    tRecordReference fOutRecRef;
} sOpenRecord;
```

fType Always `kOpenRecord`.

fResult A value of type `tDirStatus` that is `eDSNoErr` to indicate that the plug-in was able to open the specified record or an appropriate value to indicate failure.

Directory Services Plug-in Reference

<code>fInNodeRef</code>	A value of type <code>tDirNodeReference</code> that identifies the directory node of the record that is to be opened. The directory node reference was created when the calling application opened the directory node.
<code>fInRecType</code>	A value of type <code>tDataNodePtr</code> that points to a data node containing the type of the record that is to be opened.
<code>fInRecName</code>	A value of type <code>tDataNodePtr</code> that points to a data node containing the name of the record that is to be opened.
<code>fOutRecRef</code>	A value of type <code>tRecordReference</code> in which the plug-in places a value that uniquely identifies the record that has been opened.

DISCUSSION

When a Directory Services plug-in receives a request to open a record, it 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 creates a record reference and places it in `fOutRecRef`, sets `fResult` to `eDSNoErr`, and returns. Later, when the calling application calls Directory Services functions that operate on the opened record, it will pass the record reference to the plug-in, which should use the record reference to identify the opened record.

If the plug-in cannot open the record, it sets `fResult` to an appropriate result code as described in “Result Codes” (page 61) and returns.

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;
    tDirStatus      fResult;
```

Directory Services Plug-in Reference

```

        tRecordReference    fInRecRef;
        tRecordEntryPtr     fOutRecInfo;
    } sGetRecRefInfo;

```

<code>fType</code>	Always <code>kGetRecRefInfo</code> .
<code>fResult</code>	A value of type <code>tDirStatus</code> that is <code>eDSNoErr</code> to indicate that the plug-in was able to get information about the record referred to by <code>fInRecRef</code> or an appropriate value to indicate failure.
<code>fInRecRef</code>	A value of type <code>tRecordReference</code> that represents the record for which name and type information is to be obtained. The plug-in created the value of <code>fInRecRef</code> when it was called to process a request to open the record.
<code>fOutRecInfo</code>	A value of type <code>tRecordEntryPtr</code> that points to a record entry containing the requested information.

DISCUSSION

When a Directory Services plug-in receives a request to get a record's name and type, it uses the `fInRecRef` field of the `sGetRecRefInfo` structure to determine the record for which the name and type are to be obtained.

If the plug-in can get the 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 open the record, it sets `fResult` to an appropriate result code as described in "Result Codes" (page 61) and returns.

sGetRecAttrInfo

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

```

typedef struct {
    UInt32          fType;
    tDirStatus      fResult;
}

```

Directory Services Plug-in Reference

```

    tRecordReference    fInRecRef;
    tDataNodePtr        fInAttrType;
    tAttributeEntryPtr  fOutAttrInfoPtr;
} sGetRecAttribInfo;

```

fType	Always kGetRecAttribInfo.
fResult	A value of type tDirStatus that is eDSNoErr to indicate that the plug-in was able to get information about the attributes of the record referred to by fInRecRef or an appropriate value to indicate failure.
fInRecRef	A value of type tRecordReference 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	A value of type tDataNodePtr that points to a data node containing the type of attribute for which information is requested.
fOutAttrInfoPtr	A value of type tAttributeEntryPtr that points to an attribute entry containing the requested attribute information.

DISCUSSION

When a Directory Services plug-in receives a request to get information about an attribute, it 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 for which information is to be obtained.

If the plug-in can get the attribute's 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 61) and returns.

sGetRecAttrValueByIndex

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

```
typedef struct {
    UInt32          fType;
    tDirStatus      fResult;
    tRecordReference fInRecRef;
    tDataNodePtr    fInAttrType;
    unsigned long    fInAttrValueIndex;
    tAttributeValueEntryPtr fOutEntryPtr;
} sGetRecAttrValueByIndex;
```

<code>fType</code>	Always <code>kGetRecAttrValueByIndex</code> .
<code>fResult</code>	A value of type <code>tDirStatus</code> that is <code>eDSNoErr</code> to indicate that the plug-in was able to get the value of the attribute or an appropriate value to indicate failure.
<code>fInRecRef</code>	A value of type <code>tRecordReference</code> that represents the record whose attribute value is to be obtained. The plug-in created the value of <code>fInRecRef</code> when it was called to process a request to open the record.
<code>fInAttrType</code>	A value of type <code>tDataNodePtr</code> that points to a data node containing the type of attribute whose value is requested.
<code>fInAttrValueIndex</code>	A value of type <code>unsigned long</code> that specifies the attribute for which information is to be obtained, using a one-based index.
<code>fOutEntryPtr</code>	A value of type <code>tAttributeValueEntryPtr</code> that points to the attribute's value.

DISCUSSION

When a Directory Services plug-in receives a request to get the value of an attribute by index, it uses the `fInRecRef` field of the `sGetRecAttrValueByIndex` structure to determine the record for which the value of an attribute is to be

Directory Services Plug-in Reference

obtained, the `fInAttrType` field to determine 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 61) and returns.

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;
    tDirStatus   fResult;
    tRecordReference fInRecRef;
} sFlushRecord;
```

`fType` Always `kFlushRecord`.

`fResult` A value of type `tDirStatus` that is `eDSNoErr` to indicate that the plug-in was able to flush the record or an appropriate value to indicate failure.

`fInRecRef` A value of type `tRecordReference` 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

When a Directory Services plug-in receives a request to flush a record, it 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 61) and returns.

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;
    tDirStatus      fResult;
    tRecordReference fInRecRef;
} sCloseRecord;
```

`fType` Always `kCloseRecord`.

`fResult` A value of type `tDirStatus` that is `eDSNoErr` to indicate that the plug-in was able to close the record or an appropriate value to indicate failure.

`fInRecRef` A value of type `tRecordReference` 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

When a Directory Services plug-in receives a request to close a record, it uses the `fInRecRef` field of the `sCloseRecord` structure to determine the record that is to be closed.

If the plug-in can close the record, it does so, 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 61) and returns.

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;
    tDirStatus   fResult;
    tRecordReference fInRecRef;
    tDataNodePtr fInNewRecName;
} sSetRecordName;
```

`fType` Always `kSetRecordName`.

`fResult` A value of type `tDirStatus` that is `eDSNoErr` to indicate that the plug-in was able to set the record's name or an appropriate value to indicate failure.

`fInRecRef` A value of type `tRecordReference` 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` A value of type `tDataNodePtr` that points to a data node containing the name that is to be set.

DISCUSSION

When a Directory Services plug-in receives a request to set the name of a record, it 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 does so, 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 61) and returns.

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;
    tDirStatus      fResult;
    tRecordReference fInRecRef;
    tDataNodePtr    fInNewRecType;
} sSetRecordType;
```

<code>fType</code>	Always <code>kSetRecordType</code> .
<code>fResult</code>	A value of type <code>tDirStatus</code> that is <code>eDSNoErr</code> to indicate that the plug-in was able to set the record's type or an appropriate value to indicate failure.
<code>fInRecRef</code>	A value of type <code>tRecordReference</code> representing the record whose type is to be set. The plug-in created the value of <code>fInRecRef</code> when it was called to process a request to open the record.
<code>fInNewRecType</code>	A value of type <code>tDataNodePtr</code> that points to a data node containing the type that is to be set.

DISCUSSION

When a Directory Services plug-in receives a request to set the type of a record, it 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 does so, 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 61) and returns.

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
    tDirStatus      fResult;
    tRecordReference fInRecRef;
} sDeleteRecord;
```

`fType` Always `kDeleteRecord`.

`fResult` A value of type `tDirStatus` that is `eDSNoErr` to indicate that the plug-in was able to delete the record or an appropriate value to indicate failure.

`fInRecRef` A value of type `tRecordReference` 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

When a Directory Services plug-in receives a request to delete a record, it 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 does so, 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 61) and returns.

sCreateRecord

Directory Services calls a plug-in's `ProcessRequest` entry point and passes an `sCreateRecord` structure when a Directory Services client application calls the

Directory Services Plug-in Reference

`dsCreateRecord` function or `dsCreateRecordAndOpen` to create a record. The `sCreateRecord` structure is defined as follows:

```
typedef struct {
    UInt32          fType;
    tDirStatus      fResult;
    tDirNodeReference fInNodeRef;
    tDataNodePtr    fInRecType;
    tDataNodePtr    fInRecName;
    bool            fInOpen;
    tRecordReference fOutRecRef;
} sCreateRecord;
```

<code>fType</code>	Always <code>kCreateRecord</code> or <code>kCreateRecordAndOpen</code> .
<code>fResult</code>	A value of type <code>tDirStatus</code> that is <code>eDSNoErr</code> to indicate that the plug-in was able to create the record or an appropriate value to indicate failure.
<code>fInNodeRef</code>	A value of type <code>tDirNodeReference</code> that identifies the directory node in which the record is to be created. The directory node reference was created when the calling application opened the directory node.
<code>fInRecType</code>	A value of type <code>tDataNodePtr</code> that points to a data node containing the type of the record that is to be created.
<code>fInRecName</code>	A value of type <code>tDataNodePtr</code> that points to a data node containing the name of the record that is to be created.
<code>fInOpen</code>	A Boolean whose value is <code>TRUE</code> if the calling application wants to create the record and open it. Otherwise, the value of <code>fInOpen</code> is <code>FALSE</code> to indicate that the calling application wants only to create the record.
<code>fOutRecRef</code>	A value of type <code>tRecordReference</code> in which the plug-in places a value that uniquely identifies the record that has been opened.

DISCUSSION

When a Directory Services plug-in receives a request to create a record, it 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

Directory Services Plug-in Reference

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 does so, sets `fResult` to `eDSNoErr`, and returns. If the value of `fInOpen` is `TRUE`, the plug-in creates a record reference for the new record and places it in `fOutRecRef` before returning.

If the plug-in cannot create the new record, it sets `fResult` to an appropriate result code as described in “Result Codes” (page 61) and returns.

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;
    tDirStatus      fResult;
    tRecordReference fInRecRef;
    tAccessControlEntryPtr fInNewRecAccess;
} sSetRecordAccess;
```

`fType` Always `kSetRecordAccess`.

`fResult` A value of type `tDirStatus` that is `eDSNoErr` to indicate that the plug-in was able to set the record’s access or an appropriate value to indicate failure.

`fInRecRef` A value of type `tRecordReference` 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.

`fInNewRecAccess` A value of type `tAccessControlEntryPtr` that points to an access control entry structure containing the new access value that is to be set.

DISCUSSION

When a Directory Services plug-in receives a request to set a record's access controls, it uses the `fInRecRef` field of the `sSetRecordAccess` structure to determine the record whose access controls are to be set and the `fInNewRecAccess` field as the value to which the access controls are to be set. Before setting the new access controls, the plug-in should examine the value of `fInNewRecAccess` to be sure it contains a valid value.

If the plug-in can set the record's access controls, it does so, sets `fResult` to `eDSNoErr`, and returns.

If the plug-in cannot set the record's access controls, it sets `fResult` to an appropriate result code as described in "Result Codes" (page 61) and returns.

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;
    tDirStatus      fResult;
    tRecordReference fInRecRef;
    unsigned long   fInRecFlags;
} sSetRecordFlags;
```

`fType` Always `kSetRecordFlags`.

`fResult` A value of type `tDirStatus` that is `eDSNoErr` to indicate that the plug-in was able to set the record's flags or an appropriate value to indicate failure.

`fInRecRef` A value of type `tRecordReference` representing the record whose flags are to be set. The plug-in created the value of `fInRecRef` when it was called to process a request to open the record.

`fInNewRecFlags` A value of type `unsigned long` containing the flags that are to be set.

DISCUSSION

When a Directory Services plug-in receives a request to set a record's flags, it uses the `fInRecRef` field of the `sSetRecordAccess` structure to determine the record whose access flags are to be set and the `fInNewRecFlags` field as the value to which the flags are to be set. Before setting the new access controls, the plug-in should examine the value of `fInNewRecFlags` to be sure it contains a valid value.

If the plug-in can set the record's flags, it does so, sets `fResult` to `eDSNoErr`, and returns.

If the plug-in cannot set the record's flags, it sets `fResult` to an appropriate result code as described in "Result Codes" (page 61) and returns.

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;
    tDirStatus      fResult;
    tRecordReference fInRecRef;
    tDataNodePtr    fInNewAttr;
    tAccessControlEntryPtr fInNewAttrAccess;
    tDataNodePtr    fInFirstAttrValue;
} sAddAttribute;
```

<code>fType</code>	Always <code>kAddAttribute</code> .
<code>fResult</code>	A value of type <code>tDirStatus</code> that is <code>eDSNoErr</code> to indicate that the plug-in was able to add the attribute or an appropriate value to indicate failure.
<code>fInRecRef</code>	A value of type <code>tRecordReference</code> representing the record to which the attribute is to be added.
<code>fInNewAttr</code>	A value of type <code>tDataNodePtr</code> that points to a data node containing the name of the attribute that is to be added.

Directory Services Plug-in Reference

`fInNewAttrAccess`

A value of type `tAccessControlEntryPtr` that points to a `tAccessControlEntry` structure containing the access controls for the new attribute.

`fInFirstAttrValue`

A value of type `tDataNodePtr` that points to data node containing the first value of the new attribute.

DISCUSSION

When a Directory Services plug-in receives a request to add an attribute to a record, 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, the `fInNewAttrAccess` field to obtain the value that is to be set as the access controls for the new attribute, and the `fInFirstAttrValue` field as the new attribute's first value.

If the plug-in can add the attribute, it does so, sets `fResult` to `eDSNoErr`, and returns.

If the plug-in cannot set the record's flags, it sets `fResult` to an appropriate result code as described in "Result Codes" (page 61) and returns.

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;
    tDirStatus      fResult;
    tRecordReferenceInRecRef;
    tDataNodePtr    fInAttribute;
} sRemoveAttribute;
```

`fType` Always `kRemoveAttribute`.

Directory Services Plug-in Reference

<code>fResult</code>	A value of type <code>tDirStatus</code> that is <code>eDSNoErr</code> to indicate that the plug-in was able to remove the attribute or an appropriate value to indicate failure.
<code>fInRecRef</code>	A value of type <code>tRecordReference</code> representing the record from which the attribute is to be removed.
<code>fInAttribute</code>	A value of type <code>tDataNodePtr</code> that points to a data node containing the name of the attribute that is to be removed.

DISCUSSION

When a Directory Services plug-in receives a request to remove an attribute from a record, it 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 does so, sets `fResult` to `eDSNoErr`, and returns.

If the plug-in cannot remove the attribute, it sets `fResult` to an appropriate result code as described in “Result Codes” (page 61) and returns.

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;
    tDirStatus      fResult;
    tRecordReference fInRecRef;
    tDataNodePtr    fInAttrType;
    tAccessControlEntryPtr fInAttrAccess;
} sSetAttributeAccess;
```

`fType` Always `kSetAttributeAccess`.

Directory Services Plug-in Reference

<code>fResult</code>	A value of type <code>tDirStatus</code> that is <code>eDSNoErr</code> to indicate that the plug-in was able to set the attribute's access controls or an appropriate value to indicate failure.
<code>fInRecRef</code>	A value of type <code>tRecordReference</code> representing the record for which an attribute's access controls are to be set.
<code>fInAttrType</code>	A value of type <code>tDataNodePtr</code> that points to a data node containing the type the attribute whose access controls are to be set.
<code>fInAttrAccess</code>	A value of type <code>tAccessControlEntryPtr</code> that points to a <code>tAccessControlEntry</code> structure containing the access controls to be set.

DISCUSSION

When a Directory Services plug-in receives a request to set a attribute's access controls, it uses the `fInRecRef` field of the `sSetAttributeAccess` structure to determine the record for which an attribute's access controls are to be set, the `fInAttrType` field to determine the type of the attribute whose access controls are to be set, and the `fInAttrAccess` field as the value to which the access controls are to be set. Before setting the new access controls, the plug-in should examine the value of `fInAttrAccess` to be sure it contains a valid value.

If the plug-in can set the attribute's access controls, it does so, sets `fResult` to `eDSNoErr`, and returns.

If the plug-in cannot set the attribute's access controls, it sets `fResult` to an appropriate result code as described in "Result Codes" (page 61) and returns.

`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;
    tDirStatus      fResult;
```

Directory Services Plug-in Reference

```

    tRecordReference fInRecRef;
    tDataNodePtr    fInAttrType;
    unsigned long   fInAttrFlags;
} sSetAttributeFlags;

```

<code>fType</code>	Always <code>kSetAttributeFlags</code> .
<code>fResult</code>	A value of type <code>tDirStatus</code> that is <code>eDSNoErr</code> to indicate that the plug-in was able to set the attribute's flags or an appropriate value to indicate failure.
<code>fInRecRef</code>	A value of type <code>tRecordReference</code> representing the record for which an attribute's flags are to be set.
<code>fInAttrType</code>	A value of type <code>tDataNodePtr</code> that points to a data node containing the type of the attribute whose flags are to be set.
<code>fInAttrFlags</code>	A value of type <code>unsigned long</code> containing the flags that are to be set.

DISCUSSION

When a Directory Services plug-in receives a request to set an attribute's flags, it uses the `fInRecRef` field of the `sSetAttributeFlags` structure to determine the record for which an attribute's flags are to be set, the `fInAttrType` field to determine the type of the attribute whose flags are to be set, and the `fInAttrFlags` field as the value to which the flags are to be set. Before setting the new access controls, the plug-in should examine the value of `fInAttrFlags` to be sure it contains a valid value.

If the plug-in can set the attribute's flags, it does so, sets `fResult` to `eDSNoErr`, and returns.

If the plug-in cannot set the attribute's flags, it sets `fResult` to an appropriate result code as described in "Result Codes" (page 61) and returns.

sAddAttributeValue

Directory Services calls a plug-in's `ProcessRequest` entry point and passes an `sAddAttributeValue` structure when a Directory Services client application calls

Directory Services Plug-in Reference

the `dsAddAttributeValue` function to add a value to an attribute. The `sAddAttributeValue` structure is defined as follows:

```
typedef struct {
    UInt32          fType;
    tDirStatus      fResult;
    tRecordReference fInRecRef;
    tDataNodePtr    fInAttrType;
    tDataNodePtr    fInAttrValue;
} sAddAttributeValue;
```

<code>fType</code>	Always <code>kAddAttributeValue</code> .
<code>fResult</code>	A value of type <code>tDirStatus</code> that is <code>eDSNoErr</code> to indicate that the plug-in was able to add the value to the attribute or an appropriate value to indicate failure.
<code>fInRecRef</code>	A value of type <code>tRecordReference</code> representing the record for which a value is to be added to an attribute.
<code>fInAttrType</code>	A value of type <code>tDataNodePtr</code> that points to a data node containing the type the attribute to which a value is to be added.
<code>fInAttrValue</code>	A value of type <code>tDataNodePtr</code> that points to a data node containing the value that is to be added to the attribute.

DISCUSSION

When a Directory Services plug-in receives a request to add a value to an attribute, it uses the `fInRecRef` field of the `sAddAttributeValue` structure to determine the record for which a value is to be added to an attribute, the `fInAttrType` field to determine the type of the attribute to which an attribute is to be added, and the `fInAttrValue` field as the value to that is to be added.

If the plug-in can add the specified value to the specified attribute, it does so, sets `fResult` to `eDSNoErr`, and returns.

If the plug-in cannot add the attribute value, it sets `fResult` to an appropriate result code as described in “Result Codes” (page 61) and returns.

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;
    tDirStatus   fResult;
    tRecordReference fInRecRef;
    tDataNodePtr fInAttrType;
    unsigned long fInAttrValueID;
} sRemoveAttributeValue;
```

`fType` Always `kRemoveAttributeValue`.

`fResult` A value of type `tDirStatus` that is `eDSNoErr` to indicate that the plug-in was able to remove the value from the attribute or an appropriate value to indicate failure.

`fInRecRef` A value of type `tRecordReference` representing the record for which a value is to be removed to an attribute.

`fInAttrType` A value of type `tDataNodePtr` that points to a data node containing the type the attribute from which a value is to be removed.

`fInAttrValueID` A value of type `unsigned long` that specifies the attribute ID of the value that is to be removed.

DISCUSSION

When a Directory Services plug-in receives a request to remove a value from an attribute, it 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 is to be removed.

If the plug-in can remove the specified value from the specified attribute, it does so, sets `fResult` to `eDSNoErr`, and returns.

If the plug-in cannot add the attribute value, it sets `fResult` to an appropriate result code as described in “Result Codes” (page 61) and returns.

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;
    tDirStatus      fResult;
    tRecordReference fInRecRef;
    tDataNodePtr    fInAttrType;
    tAttributeValueEntryPtr fInAttrValueEntry;
} sSetAttributeValue;
```

<code>fType</code>	Always <code>kSetAttributeValue</code> .
<code>fResult</code>	A value of type <code>tDirStatus</code> that is <code>eDSNoErr</code> to indicate that the plug-in was able to set the specified value in the attribute or an appropriate value to indicate failure.
<code>fInRecRef</code>	A value of type <code>tRecordReference</code> representing the record for which a value is to be set in an attribute.
<code>fInAttrType</code>	A value of type <code>tDataNodePtr</code> that points to a data node containing the type the attribute whose value is to be set.
<code>fInAttrValueEntry</code>	A value of type <code>fInAttrValueEntryPtr</code> that points to a <code>tAttributeValueEntry</code> structure containing the value that is to be set and specifying the number of the value that is to be set.

DISCUSSION

When a Directory Services plug-in receives a request to set the value of an attribute, it uses the `fInRecRef` field of the `sSetAttributeValue` structure to determine the record for which an attribute value is to be set and the

Directory Services Plug-in Reference

`fInAttrType` field to determine the type of the attribute whose value is to be set. The `fInAttrValueEntry` field contains a pointer to an `fInAttrValueEntry` structure whose `fAttributeValueID` field identifies which value is to be set and whose `fAttributeValueData` field contains the value that is to be set.

If the plug-in can set the attribute value, it does so, 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 61) and returns.

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 session with a directory node. The `sDoDirNodeAuth` structure is defined as follows:

```
typedef struct {
    UInt32                fType;
    tDirStatus             fResult;
    tDirNodeReference      fInNodeRef;
    tDataNodePtr           fInAuthMethod;
    bool                   fInDirNodeAuthOnlyFlag;
    tDataBufferPtr         fInAuthStepData;
    tDataBufferPtr         fOutAuthStepDataResponse;
    tContextData           fIOContinueData;
} sDoDirNodeAuth;
```

`fType` Always `kDoAuthentication`.

`fResult` A value of type `tDirStatus` that is `eDSNoErr` to indicate that the plug-in was able to successfully authenticate the user or an appropriate value to indicate failure.

`fInNodeRef` A value of type `tDirNodeReference` that identifies the directory node for which the user is to be authenticated. The directory node reference was created when the calling application opened the directory node.

Directory Services Plug-in Reference

fInAuthMethod A value of type `tDataNodePtr` that points to a data node containing the name of the authentication method that is to be used.

fInDirNodeAuthOnlyFlag A Boolean value that is `TRUE` if the calling application does not want to use 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 calling application wants the result of this authentication to be applied to other operations that pertain to this directory node.

fInAuthStepData A value of type `tDataBufferPtr` that points to a data buffer containing a value that identifies the step in the authentication process for which the plug-in has been called.

fOutAuthStepDataResponse A value of type `tDataBufferPtr` that points to the data buffer in which the plug-in is to place its response.

fIOContinueData A value of type `tContext` containing data passed by the calling application. If the plug-in is processing the first step in the authentication process, `fIOContinueData` is `NULL`. If the plug-in is processing any other step, `fIOContinueData` should contain a value that the plug-in previously returned to the calling application. The plug-in can use `fIOContinueData` to maintain context information about the authentication as it progresses through the various steps required by the authentication method. Plug-ins may choose to include a timestamp in `fOutContinueData` and fail subsequent calls to `dsGetDirNodeInfo` if it determines that `fOutContinueData` is too old.

DISCUSSION

When a Directory Services plug-in receives a request to authenticate a user to a directory node, it uses the `fInNodeRef` field of the `sDoDirNodeAuth` structure to determine the node that is to be authenticated, the `fInAuthMethod` field to determine the authentication method, and the `fInDirNodeAuthOnlyFlag` field as an indicator of whether the calling application wants the results of the authentication to apply to subsequent Directory Services calls. The `fInAuthStepData` field indicates the current step in the authentication process.

Directory Services Plug-in Reference

If this step in the authentication process was successful, the plug-in sets `fResult` to `eDSNoErr`, and returns. 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 it returns.

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 61) and returns.

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 `sDoAttrValueSearch` structure is defined as follows:

```
typedef struct {
    UInt32          fType;
    tDirStatus      fResult;
    tDirNodeReference fInNodeRef;
    tDataBufferPtr  fOutDataBuff;
    tDataListPtr    fInRecTypeList;
    tDataNodePtr    fInAttrType;
    tDirPatternMatch fInPattMatchType;
    tDataNodePtr    fInPatt2Match;
    unsigned long   fOutMatchRecordCount;
    tContextData    fIOContinueData;
} sDoAttrValueSearch;
```

`fType` Always `kDoAttributeValueSearch`.

`fResult` A value of type `tDirStatus` that is `eDSNoErr` to indicate that the plug-in was able to search for the attribute values or an appropriate value to indicate failure.

Directory Services Plug-in Reference

<code>fInNodeRef</code>	A value of type <code>tDirNodeReference</code> that identifies the directory node for in which the search is to be conducted. The directory node reference was created when the calling application opened the directory node.
<code>fOutDataBuff</code>	A value of type <code>tDataBufferPtr</code> that points to the buffer in which the plug-in is to place search results.
<code>fInRecTypeList</code>	A value of type <code>tDataListPtr</code> pointing to a list of record types that are to be searched.
<code>fInAttrType</code>	A value of type <code>tDataNodePtr</code> pointing to a data node containing the attribute types that are to be searched for.
<code>fInPattMatchType</code>	A value of type <code>tDirPatternMatch</code> containing the pattern match type. The pattern match type can be a value that the plug-in and application agree upon or a constant defined by Directory Services, as described in the section “Pattern Matching Constants” in Chapter 3 of <i>“Inside Mac OS X: Directory Services.”</i> for constants defined by Directory Services.
<code>fInPatt2Match</code>	A value of type <code>tDataNodePtr</code> pointing to a data node containing the pattern that is to be matched.
<code>fOutMatchRecordCount</code>	A value of type <code>unsigned long</code> representing the number of records that matched the condition specified by <code>fInPattMatchType</code> and <code>fInPatt2Match</code> .
<code>fIOContinueData</code>	A value of type <code>tContext</code> containing context information. If this is the first time the calling application has requested this particular search, the value of <code>fIOContinueData</code> is <code>NULL</code> . If the search is successful and there is more data than can fit in <code>fOutDataBuff</code> , the plug-in sets <code>fIOContinueData</code> to a plug-in-defined value and returns. When the calling application has processed the data in <code>fOutDataBuff</code> , it can call <code>dsDoAttributeValueSearch</code> again and pass the context data value it received when it previously called <code>dsDoAttributeValueSearch</code> . When the plug-in receives the context data value in <code>fIOContinueData</code> , it can use that value to determine the next set of search results to put in <code>fOutDataBuff</code> . Plug-ins may choose to

include a timestamp in `fOutContinueData` and fail subsequent calls to `dsGetDirNodeInfo` if it determines that `fOutContinueData` is too old.

DISCUSSION

When a Directory Services plug-in receives a search for attributes whose values match a specific pattern, it uses the `fInNodeRef` field of the `sDoDirNodeAuth` structure to determine the directory node in which the search is to be conducted, the `fInRecTypeList` field to determine record types that are to be searched, the `fInAttrType` field to determine the types of attributes that are to be searched, the `fInPattMatchType` field to determine the way in which the pattern specified by `fInPatt2Match` is to be applied, the `fInPatt2Match` to obtain the pattern to match, and the `fIOContinueData` field to determine whether this is the first or subsequent call to search for attribute values.

If this is the first attempt to locate attributes whose values match the pattern and if matches are found, the plug-in places the records, attribute information, and attribute values for the matching attribute values in `fOutDataBuff` and returns. If there is more data than can fit in `fOutDataBuff`, the plug-in sets `fIOContinueData` to a plug-in-defined value before it returns.

If this the calling application has previously called `dsDoAttributeValueSearch` for the directory node specified by `fInNodeRef`, the plug-in uses `fIOContinueData` to determine what data to place into `fOutDataBuff` and returns. If there is more data than can fit in `fOutDataBuff`, the plug-in sets `fIOContinueData` to another plug-in-defined value before it returns.

If no matches are found or if there are no more matches, the plug-in sets `fResult` to an appropriate result code as described in “Result Codes” (page 61) and returns.

Result Codes

The result codes that Directory Services plug-ins should return are listed here.

<code>eDSNoErr</code>	0	No error
<code>eDSOpenFailed</code>	-14000	Attempt to open a Directory Services session failed
<code>eDSCloseFailed</code>	-14001	Attempt to close a Directory Services session failed

Directory Services Plug-in Reference

eDSOpenNodeFailed	-14002	Attempt to open a directory node failed
EDSBadDirReferences	-14003	Invalid directory reference
eDSNullRecordReference	-14004	Attempt to perform an operation failed because the record reference is null
eDSMaxSessionsOpen	-14005	The session limit has been reached
eDSCannotAccessSession	-14006	The specified session is not valid
eDSDirSrvNotOpen	-14007	A Directory Services session has not been opened
eDSNodeNotFound	-14008	The specified node could not be found
eDSUnknownNodeName	-14009	The specified node name is unknown
eDSRegisterCustomFailed	-14010	Registration of a custom routine failed
eDSGetCustomFailed	-14011	Unable to get a custom routine
eDSUnRegisterFailed	-14012	Deregistration of a custom routine failed
eDSAllocationFailed	-14050	The requested data type could not be allocated
eDSDeAllocateFailed	-14051	The requested deallocation failed.
eDSCustomBlockFailed	-14052	A custom thread block routine failed
eDSCustomUnblockFailed	-14053	A custom thread unblock routine failed
eDSCustomYieldFailed	-14054	Custom yield routine failed
eDSCorruptBuffer	-14060	A buffer provided as a parameter to a Directory Services function has been corrupted
eDSInvalidIndex	-14061	The specified index is invalid
eDSIndexOutOfRange	-14062	The specified index could not be found
eDSIndexNotFound	-14063	The specified index could not be found
eDSCorruptRecEntryData	-14065	The data in a record entry structure is invalid
eDSRefSpaceFull	-14069	No space is available to accommodate the request
eDSRefTableAllocationError	-14070	The reference could not be allocated
eDSInvalidReference	-14071	The reference is invalid
eDSInvalidRefType	-14072	The reference type is invalid
eDSInvalidDirRef	-14073	Invalid Directory Services reference

Directory Services Plug-in Reference

eDSInvalidNodeRef	-14074	Invalid node reference
eDSInvalidRecordRef	-14075	Invalid record reference
eDSInvalidAttrListRef	-14076	Invalid attribute list reference
eDSInvalidAttrValueRef	-14077	Invalid attribute value reference
eDSInvalidContinueData	-14078	Invalid continue data value
eDSInvalidBuffFormat	-14079	The format of a buffer is invalid
eDSAuthFailed	-14090	Authentication failed
eDSAuthMethodNotSupported	-14091	The specified authentication is not supported
eDSAuthRespBufTooSmall	-14092	The provided response buffer is too small for the response data
eDSAuthParameterErr	-14093	An authentication parameter is invalid
eDSAuthInBuffFormatError	-14094	A format of a buffer provided for authentication is not correct
eDSAuthNoSuchEntity	-14095	The specified password is invalid
eDSAuthBadPassword	-14096	The specified continuation data is invalid
eDSAuthContinueDataBad	-14097	The specified user name does not exist
eDSAuthUnknownUser	-14098	The specified user name is invalid
eDSAuthInvalidUserName	-14099	The password could not be obtained
eDSAuthCannotRecoverPassword	-14100	Clear-text authentication failed
eDSAuthFailedClearTextOnly	-14101	An authentication server could not be found
eDSAuthNoAuthServerFound	-14102	The authentication server reported an error
eDSAuthServerError	-14103	The provided context data is invalid
eDSInvalidContext	-14104	The provided context data is invalid
eDSBadContextData	-14105	The user is not authorized to perform this operation
eDSPermissionError	-14120	The user is not authorized to make modifications
eDSReadOnly	-14121	The specified domain is invalid
eDSInvalidDomain	-14122	A NetInfo error occurred
eNetInfo	-14123	Invalid record type
eDSInvalidRecordType	-14130	Invalid attribute type
eDSInvalidAttributeType	-14131	The record name is invalid
eDSInvalidRecordName	-14133	The requested attribute could not be found
eDSAttributeNotFound	-14134	

Directory Services Plug-in Reference

eDSRecordAlreadyExists	-14135	A record of the specified name and type already exists
eDSRecordNotFound	-14136	The specified record could not be found
eDSNullParameter	-14200	A required parameter is NULL
eDSNullDataBuff	-14201	A required data buffer parameter is NULL
eDSNullNodeName	-14202	The node name is NULL
eDSNullRecEntryPtr	-14203	The record entry pointer is NULL
eDSNullRecName	-14204	The record name is NULL
eDSNullRecNameList	-14205	The list of record names is NULL
eDSNullRecType	-14206	The record type is NULL
eDSNullRecTypeList	-14207	The list of record types is NULL
eDSNullAttribute	-14208	The attribute is NULL
eDSNullAttributeAccess	-14209	The attribute's access flags are NULL
eDSNullAttributeValue	-14210	The attribute's value is NULL
eDSNullAttributeType	-14211	The attribute's type is NULL
eDSNullAttributeTypeList	-14212	The list of attribute types is NULL
eDSNullAttributeControlPtr	-14213	
eDSNullDataList	-14214	The data list is NULL
eDSNullDirNodeTypeList	-14215	The list of directory node types is NULL
eDSNullAuthMethod	-14216	The authentication method is NULL
eDSNullAuthStepData	-14217	The authentication step data is NULL
eDSNullAuthStepDataResp	-14218	The authentication step response data is NULL
eDSNullNodeInfoTypeList	-14219	The list of node type information is NULL
eDSNullPatternMatch	-14220	No match was found
eDSNullNodeNamePattern	-14221	The node name parameter that was specified as a pattern was NULL
eDSEmptyParameter	-14230	A required parameter was empty
eDSEmptyBuffer	-14231	A buffer that should have contained data was empty
eDSEmptyNodeName	-14232	A parameter that should have contained a node name was empty
eDSEmptyRecordName	-14233	A parameter that should have contained a record name was empty
eDSEmptyRecordNameList	-14234	A parameter that should have contained a list of record names was empty
eDSEmptyRecordType	-14235	A parameter that should have contained a record type was empty

Directory Services Plug-in Reference

eDSEmptyRecordTypeList	-14236	A parameter that should have contained a list of record types was empty
eDSEmptyRecordEntry	-14237	A parameter that should have contained a record entry was empty
eDSEmptyPatternMatch	-14238	A parameter that should have contained a pattern to match was empty.
eDSEmptyNodeNamePattern	-14239	A parameter that should have contained an attribute was empty
eDSEmptyAttribute	-14240	
eDSEmptyAttributeType	-14241	A parameter that should have contained an attribute type was empty
eDSEmptyAttributeTypeList	-14242	A parameter that should have contained a list of attribute types was empty
eDSEmptyAttributeValue	-14243	A parameter that should have contained an attribute value was empty
eDSEmptyDataList	-14244	A parameter that should have contained a data list was empty
eDSEmptyNodeInfoTypeList	-14245	A parameter that should have contained a list of node information types was empty
eDSEmptyAuthMethod	-14246	A parameter that should have contained an authentication method was empty
eDSEmptyAuthStepData	-14247	A parameter that should have contained information required for the next step in the authentication process was empty
eDSEmptyAuthStepDataResp	-14248	A parameter that should have contained the response from a previous authentication call was empty
eDSEmptyPattern2Match	-14249	A parameter that should have contain the pattern to match was empty
eDSBufferTooSmall	-14260	A buffer that was supplied as a parameter is too small to contain the results of the call.
eDSUnknownMatchType	-14261	The pattern to match was unknown

Directory Services Plug-in Reference

eDSUnsupportedMatchType	-14262	The pattern to match is not supported
eDSInvalidDataList	-14263	The data list that was provided as a parameter is invalid
eDSAttrListError	-14264	An error occurred for an attribute list
eServerNotRunning	-14270	Directory Services is not running
eUnknownAPICall	-14271	The function is not known
eUnknownServerError	-14272	An unknown server error occurred
eUnknownPlugIn	-14273	An unknown plug-in error occurred
ePlugInDataError	-14274	A plug-in data error occurred
ePlugInNotFound	-14275	The requested plug-in could not be found
ePlugInError	-14276	A plug-in error occurred
ePlugInInitError	-14277	A plug-in initialization error occurred
ePlugInNotActive	-14278	The requested plug-in is loaded but is not active
ePlugInFailedToInitialize	-14279	The plug-in could not initialize itself
ePlugInCallTimedOut	-14280	An attempt to communicate with a plug-in failed
eNoSearchNodesFound	-14290	No search node could be found
eNotHandledByThisNode	-14291	A native call to the wrong plug-in was made
eIPCSendError	-14300	A communication error occurred between Directory Services and a Directory Services plug-in
eIPCReceiveError	-14331	A communication error occurred between Directory Services and a Directory Services plug-in
eServerReplyError	-14332	
eNoPluginsLoaded	-14404	No Directory Services plug-ins are loaded
ePluginAlreadyLoaded	-14406	The requested plug-in is already loaded
ePluginPrefixNotFound	-14404	
eDuplicatePluginPrefix	-14408	
eNoPluginFactoriesFound	-14410	
eCFMGetFileSysRepErr	-14450	
eCFPlugInGetBundleErr	-14452	
eCFBndleGetInfoDictErr	-14454	
eCFDictGetValueErr	-14456	
eDSServerTimeout	-14458	

Directory Services Plug-in Reference

eDSContinue	-14470	
eDSInvalidHandle	-14472	
eDSSendFailed	-14473	
eDSReceiveFailed	-14474	
eDSBadPacket	-14475	A corrupt packet was received
eDSInvalidTag	-14476	The specified tag is invalid
eDSInvalidSession	-14477	The specified Directory Service reference is not valid
eDSInvalidName	-14478	The specified name is not valid
eDSUserUnknown	-14479	The specified user is not a valid user
eDSUnrecoverablePassword	-14480	The password could not be recovered
eDSAuthenticationFailed	-14481	The authentication failed
eDSBogusServer	-14482	
eDSOperationFailed	-14483	The operation failed
eDSNotAuthorized	-14484	The user is not authorized to perform the request operation
eDSNetInfoError	-14485	A NetInfo error occurred
eDSContactMaster	-14486	
eDSServiceUnavailable	-14487	The request service is not available
eMemoryError	-14900	A memory error occurred
eMemoryAllocError	-14901	A memory allocation error occurred
eServerError	-14902	A server error occurred
eUndefinedError	-14987	An undefined error occurred
eNotYetImplemented	-14988	The specifid function is not implemented yet.

Attribute Schema

Table A-1 lists attributes that Directory Services plug-ins are required to support, attributes that are recommended, and attributes that are optional. The name of the attribute contains a “1” to indicate that the attribute can have only one value or contains an “N” to indicate that the attribute can have multiple values.

Table A-1 Attribute schema

Status	Attribute Name	Description	LDAP mapping	NetInfo mapping
Required	kDS1AttrCapabilities	Stores read-only meta data that describes a node’s capabilities.	meta data	meta data
Required	kDS1AttrMailAttribute	Stores XML text that describes a mail user’s configuration for the Mac OS X Apple Mail Server.	applemail	applemail
Required	kDSNAttrPlugInInfo	Stores read-only plug-in-specific data.	meta data	meta data

Table A-1 Attribute schema (continued)

Status	Attribute Name	Description	LDAP mapping	NetInfo mapping
Required	kDSNAttrSchema	Stores a list of read-only strings returned by the plug-in that represent the actual or potential set of attributes that can be requested by a Directory Services client for a node or record. This list should contain standard and native attribute type strings.	meta data	meta data
Required	kDS1AttrBSDetcPasswdLine	Stores a read-only collection of fields that represent the fields specified in a UNIX <code>/etc/passwd</code> file.	meta data	meta data
Required	kDS1AttrUniqueID	Stores an ASCII string of digits 0 through 9 whose value is an unsigned 32-bit number.	unixid	uid

Table A-1 Attribute schema (continued)

Status	Attribute Name	Description	LDAP mapping	NetInfo mapping
Required	kDS1AttrAuthMethod	Stores a read-only list of strings that represent the types of authentication methods supported by a given directory node.	meta data	metadata
Required	kDSNAttrGroupMembership	Stores a list of comma-separated record names (assumed to be of type kDSStandardUser) representing members of a group.	userlist	users
Required	kDSNAttrRecordName	Stores a set of Unicode record names.	cn, dn, sn	name, realname
Required	kDSNAttrRecordType	Stores a read-only set of strings representing all of the standard and native record RecordType strings this record would match.	meta data	meta data

Table A-1 Attribute schema (continued)

Status	Attribute Name	Description	LDAP mapping	NetInfo mapping
Required	kDSNAttrRecordAlias	Stores XML text that represents a Directory Services alias structure.	aliasdata	alias_data
Recommended	kDS1AttrInternetAlias	Stores the one, unique 7-bit ASCII name that can be used for legacy system compatibility.	meta data	meta data
Recommended	kDS1AttrPassword	Stores a string that represents the standard UNIX <code>crypt</code> password value for legacy purposes.	passwd	passwd
Recommended	kDSNAttrHomeDirectory	Stores in XML format the fully qualified set of home directory locations.	home, homeloc	home, homeloc

Table A-1 Attribute schema (continued)

Status	Attribute Name	Description	LDAP mapping	NetInfo mapping
Recommended	kDSNAttrURLforNSL	Stores a set of valid URLs that specify an object's network location. The Mac OS X Directory Services plug-in requests this attribute from Directory Services.	network-locURL	nslurl
Search policy only	kDSNAttrMetaNodeLocation		meta data	meta data
Optional	kDS1AttrComment	Stores a Unicode string	no mapping	no mapping

Table A-1 Attribute schema (continued)

Status	Attribute Name	Description	LDAP mapping	NetInfo mapping
Optional	kDS1AttrDataStamp	Stores the record's checksum. If the plug-in supports this attribute, this attribute contains a sequence of 7-bit ASCII characters that represent a checksum of all of a record's contents. If the record's contents change, the checksum should change.	no mapping	meta data
Optional	kDSNAttrDNSName	Stores a set of legal DNS names for a given record.	dnsdata	dnsdata
Optional	kDSNAttrEmailAddress	Stores a null-terminated list of comma-delimited e-mail addresses for a user record.	mail, email	mail, email
Optional	kDSNAttrGroup	Stores a set of Unicode group names that a user record is a member of.	grouplist	groups

Table A-1 Attribute schema (continued)

Status	Attribute Name	Description	LDAP mapping	NetInfo mapping
Optional	kDSNAttrHTML	Stores a set of valid HTML pages associated with a record	htmldata	htmldata
Optional	kDSNAttrIPAddress	Stores a set of IP addresses in dotted decimal format for a given record.	ipdata	ipdata
Optional	kDSNAttrNBPEnter	Stores a set of 8-bit Mac OS Roman NBP string tuples that represent the network locations of a given record.	nbpdata	nbpdata
Optional	kDSNAttrPGPPublicKey	Stores a standard Pretty Good Privacy (PGP) public key certificate	pgppublic-data	pgppublic-key
Optional	kDSNAttrPhoneNumber	No specification.	phone, telephone	phone, telephone
Optional	kDSNAttrURL	Stores a set of URLs associated with a record.	urldata	urldata

Index

A

active state 9
attempting initialization state 9
authentication 15

B

building a plug-in 15–16
busy state 10

C, D

callback routines
 Log 13, 22–23
 RegisterNode 13, 23
 UnregisterAll 13, 24
 UnregisterNode 13, 24
CFLOAD mechanism 15
clear text authentication 15
concurrency 13
Configure entry point 11, 17

E, F, G, H

entry points
 Configure 11, 17
 Initialize 18
 Inititalize 11
 PeriodicTask 11, 18
 ProcessRequest 11, 19, 25–61
 SetPluginState 11, 20
 Shutdown 11, 20
 Validate 11, 21

I, J, K

inactive state 10
Initialize entry point 11, 18

L

loaded but not initialized state 9
Log callback routine 13, 22–23

M

managing references 14

N, O

node native authentication 15

P, Q

PeriodicTask entry point 11, 18
ProcessRequest entry point 11, 19, 25–61
property list file 15

R

record types 14
references, managing 14
RegisterNode callback routine 13, 23
runtime environment 9–10

S

sAddAttribute **structure** 49–50
 sAddAttributeValue **structure** 53–54
 sCloseDirNode **structure** 26–27
 sCloseRecord **structure** 42
 sCreateRecord **structure** 45–47
 sDeleteRecord **structure** 45
 sDoAttrValueSearch **structure** 59–61
 sDoDirNodeAuth **structure** 57–59
 SetPluginState **entry point** 11, 20
 sFlushRecord **structure** 41
 sGetAttributeEntry **structure** 33–34
 sGetAttributeValue **structure** 35–36
 sGetDirNode **structure** 27–29
 sGetRecAttribInfo **structure** 38–39
 sGetRecAttribValueByIndex **structure** 40–41
 sGetRecordEntry **structure** 31–33
 sGetRecordList **structure** 29–31
 sGetRecRefInfo **structure** 37–38
 Shutdown **entry point** 11, 20
 shutdown state 10
 SMB authentication 15
 sOpenDirNode **structure** 25–26
 sOpenRecord **structure** 36–37
 sRemoveAttribute **structure** 50–51
 sRemoveAttributeValue **structure** 55–56
 sSetAttributeAccess **structure** 51–52
 sSetAttributeFlags **structure** 52–53
 sSetAttributeValue **structure** 56–57
 sSetRecordAccess **structure** 47–48
 sSetRecordFlags **structure** 48
 sSetRecordName **structure** 43
 sSetRecordType **structure** 44
 standard record types 14
 state diagram 10
 structure
 sAddAttribute 49–50
 sAddAttributeValue 53–54
 sCloseDirNode 26–27
 sCloseRecord 42
 sCreateRecord 45–47
 sDeleteRecord 45
 sDoAttrValueSearch 59–61
 sDoDirNodeAuth 57–59

sFlushRecord 41
 sGetAttributeEntry 33–34
 sGetAttributeValue 35–36
 sGetDirNode 27–29
 sGetRecAttribInfo 38–39
 sGetRecAttribValueByIndex 40–41
 sGetRecordEntry 31–33
 sGetRecordList 29–31
 sGetRecRefInfo 37–38
 sOpenDirNode 25–26
 sOpenRecord 36–37
 sRemoveAttribute 50–51
 sRemoveAttributeValue 55–56
 sSetAttributeAccess 51–52
 sSetAttributeFlags 52–53
 sSetAttributeValue 56–57
 sSetRecordAccess 47–48
 sSetRecordFlags 48
 sSetRecordName 43
 sSetRecordType 44

T

two-way random authentication 15

U

UNIX authentication 15
 unloaded state 9
 UnregisterAll **callback routine** 13, 24
 UnregisterNode **callback routine** 13, 24

V, W, X, Y, Z

Validate **entry point** 11, 21