Novell Developer Kit

www.novell.com

EDIRECTORY™ EVENT SERVICES

March 1, 2006

Novell®

Legal Notices

Novell, Inc.
404 Wyman Street, Suite 500
Waltham, MA 02451
U.S.A.
www.novell.com

Online Documentation: To access the online documentation for this and other Novell developer products, and to get updates, see developer.novell.com/ndk. To access online documentation for Novell products, see www.novell.com/documentation.

Novell Trademarks

Third-Party Materials

Contents

About This Guide

Novell® eDirectory™ Event Services provide a way for applications to monitor the activity of eDirectory on an individual server. Your application can specify which events to monitor and when it wants notification. The information about eDirectory Event Services is divided into the following sections:

Feedback

We want to hear your comments and suggestions about this manual and the other documentation included with this product. Please use the User Comments feature at the bottom of each page of the online documentation.

Documentation Updates

For the most recent version of this guide, see eDirectory Libraries for C (http://developer.novell.com/ndk/ndslib.htm).

Additional Information

For information about other eDirectory interfaces, see the following guides:

- eDirectory Iterator Services (http://developer.novell.com/ndk/doc/ndslib/skds_enu/data/front.html)
- eDirectory Backup Services (http://developer.novell.com/ndk/doc/ndslib/dsbk_enu/data/front.html)
- eDirectory Technical Overview (http://developer.novell.com/ndk/doc/ndslib/dsov_enu/data/h6tvg4z7.html)
- eDirectory Core Services (http://developer.novell.com/ndk/doc/ndslib/nds__enu/data/h2y7hdit.html)
- eDirectory Schema Reference (http://developer.novell.com/ndk/doc/ndslib/schm_enu/data/h4q1mn1i.html)

For help with eDirectory problems or questions, visit the eDirectory Libraries for C Developer Support Forum (http://developer.novell.com/ndk/devforums.htm).

For product information about eDirectory, see the eDirectory Documentation Site (http://www.novell.com/documentation/edirectory.html).

Documentation Conventions

In this documentation, a greater-than symbol (>) is used to separate actions within a step and items within a cross-reference path.

A trademark symbol (®, ™, etc.) denotes a Novell trademark. An asterisk (*) denotes a third-party trademark.

Concepts

1

This section provides an overview of Novell® eDirectory™ Event Services, its functions, and uses.

1.1 eDirectory Event Introduction

The eDirectory Event system provides a mechanism for monitoring eDirectory activity on an individual server. The event system generates events for local activities such as adding eDirectory objects, deleting eDirectory objects, and modifying attribute values. See "Values" on page 79 for a list of the events and the data types they use.

A monitoring application can decide which events it wants to monitor and then register for these events. Such registered applications become event handlers (see "eDirectory Event Handling" on page 13). The eDirectory event system can notify the event handler during the event or after the event. The event handler remains registered until the application requests that the handler be removed (unregistered).

eDirectory allocates an eDirectory Event Slot Table (page 13) to track the events.The table contains only the events that have one or more handlers registered for them. The table is dynamic and grows as handlers register for events not previously listed in the table. When a handler is the only handler registered for an event and that handler unregisters, the event is removed from the table. When events are removed, the table does not shrink. It grows to accommodate new events, but remains at its maximum size when events are removed.

Handlers use the NWDSERegisterForEvent (page 38) function to register for an event. They use the NWDSEUnRegisterForEvent (page 43) function to remove their handler from the Slot Table.

In addition to the registration events, eDirectory uses eDirectory Event Helper Functions (page 13) to help access and evaluate the data for the event. If successful, these functions return zero. If unsuccessful, they return a negative value that identifies the error.

When an eDirectory module generates an event, if no handler has registered for the event, the event is dropped. If a handler or handlers are registered, the handlers are notified according to their registered priority.

eDirectory Event Services can also be used for:

- External Synchronization to the Directory: The Directory provides a global database that can be used to store information about organizations. External databases can use the Directory as an information source. In this case, eDirectory Event notification can be used to help keep the external database synchronized with the Directory.

- Customized eDirectory Security: eDirectory Event notification allows a monitoring application to register to be called when specified eDirectory events occur. Being registered for such calls allows the callback to determine whether or not the event is allowable, thus providing you the ability to create customized eDirectory Security. For example, you could create an application that restricts the deletion of certain classes of objects from the Directory.
- eDirectory Performance Analysis: An application could watch for the replication of a specific object. The application could then time how long it takes for that object to appear on other replicas.

See Also

- "eDirectory Event Handling" on page 13
- "eDirectory Event Helper Functions" on page 13
- Section 1.3, "eDirectory Event Priorities," on page 14
- "eDirectory Event Registration Functions" on page 12
- "eDirectory Event Slot Table" on page 13
- "Values" on page 79

1.2 eDirectory Event Functions

eDirectory Event provides two types of functions: registration and helper. eDirectory Event Registration Functions (page 12) allow an NLM application to register and unregister callback functions when a specific event occurs. eDirectory Event Helper Functions (page 13) are for accessing and evaluating the event data.

- Section 1.2.1, "eDirectory Event Registration Functions," on page 12
- Section 1.2.2, "eDirectory Event Helper Functions," on page 13
- Section 1.2.3, "eDirectory Event Handling," on page 13
- Section 1.2.4, "eDirectory Event Slot Table," on page 13

1.2.1 eDirectory Event Registration Functions

The following table lists registration functions:

Name	Description
NWDSERegisterForEvent (page 38)	Registers a function to be used as a callback when a specific eDirectory event occurs.
NWDSEUnRegisterForEvent (page 43)	Unregisters a callback that has been registered to be called when a specified eDirectory event occurs.

See Also

- Section 1.2, "eDirectory Event Functions," on page 12

1.2.2 eDirectory Event Helper Functions

The functions listed in the following table are helper functions:

Name	Description
NWDSEConvertEntryName (page 24)	Converts object names returned in the DSEEntryInfo structure to a form that is consistent with the NWDS functions.
NWDSEGetLocalAttrID (page 26)	Retrieves the local ID of a specified eDirectory attribute.
NWDSEGetLocalAttrName (page 28)	Retrieves the name of the eDirectory attribute associated with the supplied local ID.
NWDSEGetLocalClassID (page 30)	Retrieves the local ID for the specified object class.
NWDSEGetLocalClassName (page 32)	Retrieves the name of the eDirectory object class associated with the supplied local ID.
NWDSEGetLocalEntryID (page 34)	Retrieves the local ID for the specified eDirectory object.
NWDSEGetLocalEntryName (page 36)	Retrieves the name of the eDirectory object that is associated with the supplied local ID.

See Also

- Section 1.2, “eDirectory Event Functions,” on page 12

1.2.3 eDirectory Event Handling

The handler parameter of NWDSERegisterForEvent (page 38) points to a function called when the event occurs. Separate functions can be registered for each event or a single function can be registered for multiple events. If a callback processes multiple events, it can use the type parameter to determine which event has occurred.

See Also

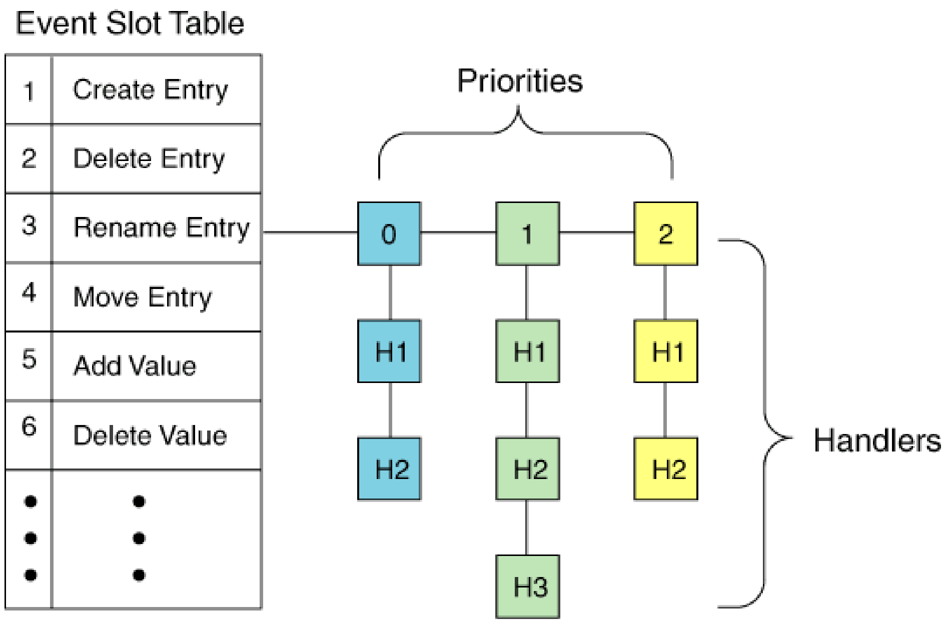
- Section 1.3, “eDirectory Event Priorities,” on page 14
- Section 1.5, “eDirectory Event Types,” on page 19

1.2.4 eDirectory Event Slot Table

Each event has an assigned number (see “Values” on page 79), which corresponds to the event slot. The Slot Table is dynamically extended if a new event is registered with an event number greater than those previously registered. The system does not currently do any validation on the event number, so the callers of the registration function need to be reasonable and not ask to register for event 1,000,000 when the system is handling only a few hundred events.

When an event is first registered for, an EventSlot structure is allocated containing fields to hold the number of handlers registered for the event, and fields to manage the individual handlers in priority order.

The Slot Table initially allocates enough handler space to hold two handlers for each of the three priorities. Handler space can be dynamically expanded if more handlers are needed within a priority. The following figure illustrates this design.



The figure above illustrates a possible configuration for slot number 3. The other slots would have similar configurations. Slot 3 has the default configuration with space for two handles for priorities 0 and 2, and a space for an extra handler for priority 1. Just as the Slot Table can grow dynamically as events are added, the list of handlers can grow as handles register for an event.

See Also

- “EP_INLINE” on page 16
- “EP_JOURNAL” on page 16
- “EP_WORK” on page 17
- Section 1.2, “eDirectory Event Functions,” on page 12

1.3 eDirectory Event Priorities

The priority parameter of NWDSERegisterForEvent (page 38) specifies the registered priority of a callback.The behavior of a callback must respond partly to its registered priority.

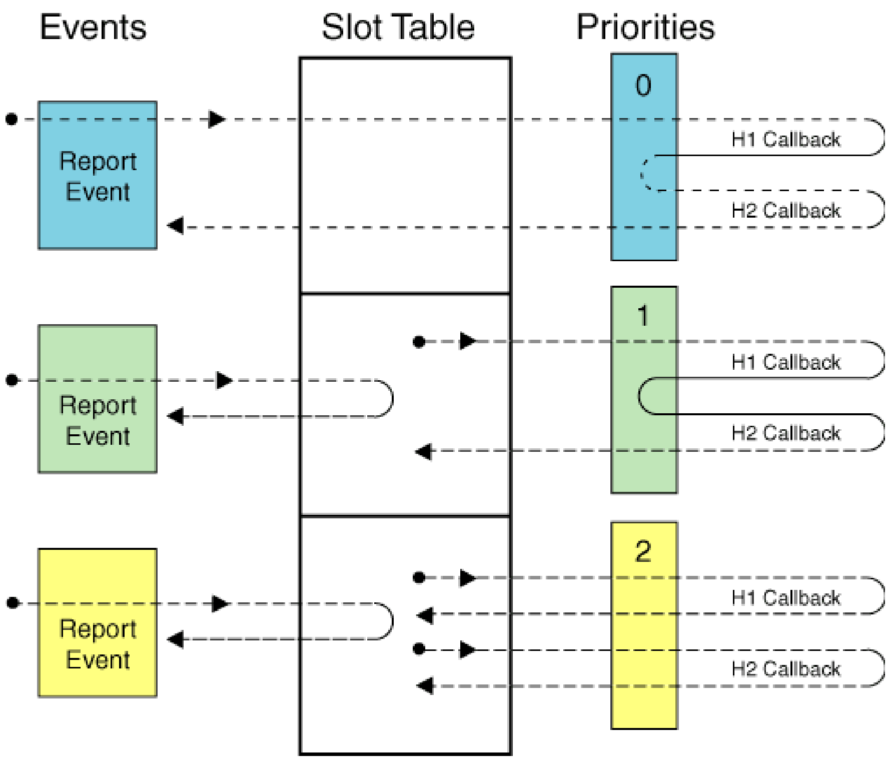
This section covers the following topics:

- Section 1.3.1, “EP_INLINE,” on page 16
- Section 1.3.2, “EP_JOURNAL,” on page 16
- Section 1.3.3, “EP_WORK,” on page 17
- Section 1.3.4, “Priority 0,” on page 17
- Section 1.3.5, “Priority 1,” on page 17
- Section 1.3.6, “Priority 2,” on page 18

The priority flags determine the order in which handlers are notified when an event is generated. When an event is generated, the module reports the event to the Slot Table. Handlers are notified in the following order:

- Priority 0: EP_INLINE. Notified first. All callback processing is completed before handlers registered for priority 1 are notified.
- Priority 1: EP_JOURNAL. Notified second. All callback processing is completed before handlers registered for priority 2 are notified.
- Priority 2: EP_WORK. Notified last.

The following figure illustrates how the handlers are notified. It also traces which thread is used to process the handler's callback function.



The graphic shows the thread paths for three events. To simplify the graphic, each event has handlers registered for only one priority. The dots (·) indicate the beginning of a thread, or at least where the thread's processing starts as it enters the eDirectory event system.

See Also

- Synchronous pre-event reporting, discussed in "EP_INLINE" on page 16
- Synchronous post-event reporting, discussed in "EP_JOURNAL" on page 16
- Asynchronous post-event reporting, discussed in "EP_WORK" on page 17

1.3.1 EP_INLINE

The EP_INLINE priority provides synchronous pre-event reporting, as follows:

- The callback can determine whether or not the event is allowable. If the callback returns a nonzero value, the transaction is aborted, and the return value of the callback is returned to the client.
- The client waits for a response while the callback processes. If the callback takes too long, the client could time out. Callbacks need to return as quickly as possible.
- The callback cannot call any of the NWDS functions because the local database is locked. In addition, the only function it can use from eDirectory Event is NWDSEConvertEntryName (page 24).
- The callback can sleep (normally only to allocate memory).
- This priority faces the most difficult issues when using chained event handlers. You cannot assume that an eDirectory event will complete if your callback returns zero. This is because the next callback in the chain could abort the transaction. To verify that changes occurred, register a callback for the EP_JOURNAL or EP_WORK priorities.

See Also

- Section 1.3, "eDirectory Event Priorities," on page 14
- "EP_JOURNAL" on page 16
- "EP_WORK" on page 17

1.3.2 EP_JOURNAL

The EP_JOURNAL priority provides synchronous post-event reporting, as follows:

- Synchronous post-event reporting because event information is stored in a journal queue that records the events in the order they occurred.
- A single thread services all of the callbacks for the events, so the callback's execution time should be minimized. (The callback can determine if data should be used If it should be used, it can store the data in a list that another thread processes.)
- If multiple callbacks are registered for the same event, the current callback must be processed before the next callback is called.
- The callback can sleep.
- The callback can use any of the NWDS and NWDSE functions.

IMPORTANT: While inside this callback, use discretion in calling NWDS functions that create more eDirectory events. This is a closed loop where the growth of the journal queue could be uncontrollable.

See Also

- Section 1.3, "eDirectory Event Priorities," on page 14
- "EP_INLINE" on page 16
- "EP_WORK" on page 17

1.3.3 EP_WORK

The EP_WORK priority provides asynchronous postevent reporting, as follows:

- The events are reported after they have occurred, but not necessarily in the order that they occurred. They are reported only after all of the event's callbacks registered for the EP_JOURNAL priority have completed.
- Each callback is run on a separate thread. This frees the event handler from the time constraints of the other two priorities.
- The callback can use any of the NWDS and NWDSE functions.
- The callback can sleep.
- Time is not a critical issue.

See Also

- Section 1.3, "eDirectory Event Priorities," on page 14
- "EP_INLINE" on page 16
- "EP_JOURNAL" on page 16
- "EP_WORK" on page 17

1.3.4 Priority 0

For handlers registered for priority 0, the thread that generates the event is used to process all the handler callback functions. When they are finished processing, the thread returns to the module that generated the event. Since the same thread is used to process all the callback functions, callbacks need to return as quickly as possible.

Because the thread that generates the event is the same thread that processes the handlers' callback functions, the callback functions can influence the outcome of the event. However, the last handler called has the final say. The value that is reported by the last handler is used for handlers that have registered for the other priorities and is returned the module that generated the event.

If more than one handler registers for priority 0, the handler cannot specify its position in the list. Handlers are added in the order they register. However, one handler can register to be the auditor handler (DSHF_AUDIT) with a notify flag. This places this handler last in the list and allows the handler's callback function to have the final say in whether the event fails or succeeds. Only one handler can register as the auditor handler.

See Also

- "EP_INLINE" on page 16
- "EP_JOURNAL" on page 16
- "EP_WORK" on page 17

1.3.5 Priority 1

For handlers registered for priority 1, the thread that generates the event reports the event to the eDirectory Event Slot Table (page 13) and immediately returns to the module that generated the event. The Slot Table then assigns a different thread to process all the callback functions registered for priority 1. Since the same thread is used to process all the callback functions, callbacks need to return as quickly as possible.

See Also

- Section 1.3, “eDirectory Event Priorities,” on page 14

1.3.6 Priority 2

For handlers registered for priority 2, the thread that generates the event reports the event to the Slot Table and immediately returns to the module that generated the event. The eDirectory Event Slot Table (page 13) then assigns a different thread to each registered handler. Since each callback function has its own thread, the callback function can be scheduled to do work that is time consuming. The results of the callbacks are asynchronous because the finishing order is indeterminate.

Each callback thread consumes a service process that is a limited resource on NetWare.

See Also

- Section 1.3, “eDirectory Event Priorities,” on page 14

1.4 eDirectory Event Data Filtering

When a callback is called, it must determine if the data (pointed to by the data parameter) contains information the NLM™ application requires. For example, if the NLM application is only concerned with changes to telephone numbers, it would use only data containing Telephone Number attribute information. Otherwise, the callback would simply return.

Data can be filtered in two ways:

- By use of local IDs, as described in Section 1.4.1, “Filtering eDirectory Events by Local ID,” on page 18
- By use of DSTraceEvents, as described in Section 1.4.2, “Filtering eDirectory Events by DSTrace Events,” on page 19

1.4.1 Filtering eDirectory Events by Local ID

When examining the data structures passed in as the data parameter of your callback, you will see that the structures use IDs rather than names. For example, the DSEValueInfo structure contains the following IDs:

perpetratorID
entryID
attrID
syntaxID
classID

While the object names in the Directory are global, the local IDs for objects on individual servers are not. Each object on a server is identified by a local ID that is relevant only on that server. The object’s local ID on another server is probably not the same.

The use of local IDs is not limited to object names. These IDs are also used to identify attributes and object classes. (The IDs for syntaxes are defined in NWDSDEFS.H.)

eDirectory events are reported by ID to enhance speed. IDs are 32-bit values; comparing for equality is faster with two IDs than with two strings.

In most cases, you can use these IDs as your filter.

For example, if an organization has an external telephone directory that needs to be kept current, it could create an NLM™ application that registers for DSE_ADD_VALUE to determine when any object's phone number changes. It would then get the attribute ID by calling NWDSEGetLocalAttrID (page 26) and filter on the attrID field.

See Also

- Section 1.4, "eDirectory Event Data Filtering," on page 18
- "Filtering eDirectory Events by DSTrace Events" on page 19

1.4.2 Filtering eDirectory Events by DSTrace Events

eDirectory Event allows NLM™ applications to register to the DSTrace events. These events are the same events used to report the DSTrace information when the `SET DSTRACE=ON` command is issued at the server console.

IMPORTANT: Your NLM application should not rely upon the text strings supplied with the DSTrace event. These strings are for internal debugging purposes and are not guaranteed to remain the same in future OS versions.

See Also

- Section 1.4, "eDirectory Event Data Filtering," on page 18
- "Filtering eDirectory Events by Local ID" on page 18

1.5 eDirectory Event Types

The type parameter of NWDSERegisterForEvent (page 38) specifies the type of event with which to associate the callback. See the "Values" on page 79 for information about eDirectory Event types.

See Also

- "eDirectory Event Handling" on page 13
- Section 1.3, "eDirectory Event Priorities," on page 14

1.6 Global Network Monitoring

eDirectory Event does not provide a global solution to monitoring eDirectory events. Instead, it provides information that is local to each server. If your application is to provide a global solution it must do the following.

- Provide an eDirectory event handler on each server being monitored.

- Provide a method of sorting out duplicate events. For example, if there are three replicas on three servers, each having an eDirectory event handler registered, then deleting an object will show up as a separate event on each server. In this case the global monitor must either reject all events from two of the servers or deal with receiving multiple copies of the same event.

In some implementations, it might be advantageous to obtain events from all servers that hold an instance of a partition. Such an application might be one that measures replication time in a network.

See Also

- "eDirectory Event Handling" on page 13

Tasks

2

This section describes the most common tasks associated with an application's use of Novell® eDirectory™ Event Services.

- Section 2.1, "Monitoring eDirectory Events," on page 21
- Section 2.2, "Registering for eDirectory Events," on page 21
- Section 2.3, "Unregistering for eDirectory Events," on page 22

2.1 Monitoring eDirectory Events

The following list is a high-level view of the steps an eDirectory event-monitoring application must take.

1 Register for events. Follow the steps outlined in Section 2.2, "Registering for eDirectory Events," on page 21

2 Ensure that your application conforms to the following:

   - When the specified event occurs, such as the creation of an object, the callback is called and given data about the event.
   - The callback determines whether to use the data. If it uses the data, the callback either immediately processes the data or makes and stores a local copy of the data. Then the callback returns.
   - If the callback saved data locally so another thread can process it, that thread runs.

3 Unregister for events as explained in Section 2.3, "Unregistering for eDirectory Events," on page 22.

2.2 Registering for eDirectory Events

The following list is a high-level view of the steps an application must take to register for eDirectory events.

1 Using the helper functions, determine the IDs of the desired objects, object classes, or attributes.

2 Call NWDSERegisterForEvent (page 38) to register a function you want used as a callback when a specific event occurs. Call NWDSERegisterForEvent (page 38) once for each event you want monitored.

See Also

- Section 2.1, "Monitoring eDirectory Events," on page 21
- Section 2.3, "Unregistering for eDirectory Events," on page 22

2.3 Unregistering for eDirectory Events

The following list is a high-level view of the steps an application must take to unregister for eDirectory events.

1. When information about an event is no longer needed, call NWDSEUnRegisterForEvent (page 43) to remove its callbacks from the notification lists.

IMPORTANT: You must call NWDSEUnRegisterForEvent once for each registered event.

See Also

- Section 2.1, "Monitoring eDirectory Events," on page 21
- Section 2.2, "Registering for eDirectory Events," on page 21

Functions

3

This section describes the functions used in Novell® eDirectory™ Event Services.

NWDSEConvertEntryName

Converts the object names returned in the DSEEntryInfo structure to a form that is consistent with the functions whose names begin with NWDS.

Local Servers: blocking

Remote Servers: N/A

Classification: 4.x, 5.x, 6.x

Platform: NLM

Service: eDirectory Event

Syntax

```
#include <nwdsdsa.h>
#include <nwdsevnt.h>

N_EXTERN_LIBRARY (NWDSCCODE)  NWDSEConvertEntryName  (
   NWDSContextHandle   context,
   const punicode      DSEventName,
   pnstr               objectName);
```

Parameters

context

(IN) Specifies the Directory context for the request.

DSEventName

(IN) Points to the object name to be converted.

objectName

(OUT) Points to the object's name in a form consistent with the eDirectory functions.

Return Values

0x0000	Successful
Negative value	Negative values indicate errors. For error values, see "NDS Return Values" (*NDK: Novell eDirectory Core Services*).

Remarks

The form of the object names returned in the dn and newDN fields of the DSEEntryInfo structure is not consistent with the form used by the eDirectory functions. These names must be converted by NWDSEConvertEntryName before you use them with eDirectory functions.

The format of the name returned in newDN is determined by the settings in the eDirectory context.

The caller must allocate space for the object name to be returned. The size of the allocated memory is ((MAX_DN_CHARS)+1)*sizeof(character size) where character size is 1 for single-byte characters, and 2 for double-byte characters (Unicode characters are double-byte). One character is used for the NULL terminator.

NWDSEGetLocalAttrID

Retrieves the local ID of a specified eDirectory attribute.

Local Servers: blocking

Remote Servers: N/A

Classification: 4.x, 5.x, 6.x

Platform: NLM

Service: eDirectory Event

Syntax

```
#include <nwdsdsa.h>
#include <nwdsevnt.h>

N_EXTERN_LIBRARY (NWDSCCODE)  NWDSEGetLocalAttrID  (
   NWDSContextHandle   context,
   const pnstr         name,
   pnuint32            id);
```

Parameters

context

(IN) Specifes the eDirectory context for the request.

name

(IN) Points to the name of the eDirectory attribute whose local ID is to be returned.

id

(OUT) Points to the local ID for the eDirectory attribute.

Return Values

0x0000	Successful
Negative Value	Negative values indicate errors. For error values, see "NDS Return Values" (*NDK: Novell eDirectory Core Services*).

Remarks

An attribute's local ID is valid only for the server on which NWDSEGetLocalAttrID is called. For this reason, this ID is called a local ID.

The data structures returned for eDirectory events do not contain attribute names. Instead, these structures use local IDs to identify the attribute that is associated with the event. NWDSEGetLocalAttrID is used to map an attribute name, such as User, and convert it to a local ID that can be used to compare with the local ID in an event structure.

Comparisons of IDs are faster than comparisons of text strings. Therefore, to avoid unnecessary processing time, your application should filter on IDs when possible.

See Also

NWDSEGetLocalAttrName (page 28)

NWDSEGetLocalAttrName

Retrieves the name of the eDirectory attribute associated with the supplied local ID.

Local Servers: blocking

Remote Servers: N/A

Classification: 4.x, 5.x, 6.x

Platform: NLM

Service: eDirectory Event

Syntax

```
#include <nwdsdsa.h>
#include <nwdsevnt.h>

N_EXTERN_LIBRARY (NWDSCCODE)  NWDSEGetLocalAttrName  (
   NWDSContextHandle   context,
   nuint32             attrID,
   pnstr               name);
```

Parameters

context

(IN) Specifies the eDirectory context for the request.

attrID

(IN) Specifies the local ID for the schema attribute.

name

(OUT) Points to the name of the attribute associated with the local ID.

Return Values

0x0000	Successful
Negative Value	Negative values indicate errors. For error values, see “NDS Return Values” (*NDK: Novell eDirectory Core Services*).

Remarks

The data structures returned for eDirectory events do not contain attribute names. Instead, these structures use local IDs to identify the attribute associated with the event. NWDSEGetLocalAttrName is used to map the local attribute ID found in the structures, to a text form of the name, such as "Telephone Number."

Comparisons of IDs is faster than comparisons of text strings. Therefore, to avoid unnecessary processing time, your application should filter on IDs when possible.

The caller must allocate space for the attribute name pointed to by name. The size of the allocated memory is ((MAX_SCHEMA_NAME_CHARS)+1)*sizeof(character size) where character size is 1 for single-byte characters, and 2 for double-byte characters (Unicode is double-byte). One character is used for NULL termination.

See Also

NWDSEGetLocalAttrID (page 26)

NWDSEGetLocalClassID

Retrieves the local ID for the specified object class.

Local Servers: blocking

Remote Servers: N/A

Classification: 4.x, 5.x, 6.x

Platform: NLM

Service: eDirectory Event

Syntax

```
#include <nwdsdsa.h>
#include <nwdsevnt.h>

N_EXTERN_LIBRARY (NWDSCCODE)  NWDSEGetLocalClassID  (
   NWDSContextHandle   context,
   const pnstr         name,
   pnuint32            id);
```

Parameters

context

(IN) Specifies the eDirectory context for the request.

name

(IN) Points to the name of the object class whose local ID is to be returned.

id

(OUT) Points to the local class ID for the specified object class.

Return Values

0x0000	Successful
Negative Value	Negative values indicate errors. For error values, see "NDS Return Values" (*NDK: Novell eDirectory Core Services*).

Remarks

The local ID of an object class is valid only on the server on which NWDSEGetLocalClassID is called. For this reason, this ID is called a local ID.

The data structures returned for DS events do not contain object class names. Instead, these structures use local IDs to identify the object class associated with an event. NWDSEGetLocalClassID is used to determine the local ID for an object class, such as User, so the ID can be used for comparison operations.

Comparisons of IDs are faster than comparisons of text strings. Therefore, to avoid unnecessary processing time, your application should filter on IDs when possible.

See Also

NWDSEGetLocalClassName (page 32)

NWDSEGetLocalClassName

Retrieves the name of the eDirectory object class associated with the supplied local ID.

Local Servers: blocking

Remote Servers: N/A

Classification: 4.x, 5.x, 6.x

Platform: NLM

Service: eDirectory Event

Syntax

```
#include <nwdsdsa.h>
#include <nwdsevnt.h>

N_EXTERN_LIBRARY (NWDSCCODE)  NWDSEGetLocalClassName  (
   NWDSContextHandle   context,
   nuint32             classID,
   pnstr               name);
```

Parameters

context

(IN) Specifies the eDirectory context for the request.

classID

(IN) Specifies the local ID for the eDirectory object class.

name

(OUT) Points to the name of the object class associated with the local ID.

Return Values

0x0000	Successful
Negative Value	Negative values indicate errors. For error values, see "NDS Return Values" (*NDK: Novell eDirectory Core Services*).

Remarks

The data structures returned for eDirectory events do not contain object class names. Instead, these structures use local IDs to identify the object class associated with an event. NWDSEGetLocalClassName is used to determine the name of the object class, such as User,that is associated with the object class ID.

Comparisons of IDs are faster than comparisons of text strings. Therefore, to avoid unnecessary processing time, your application should filter on IDs when possible.

The caller must allocate space for the object-class name that is returned. The size of the allocated memory is ((MAX_SCHEMA_NAME_CHARS)+1)*sizeof(character size) where character size is 1 for single-byte characters, and 2 for double-byte characters (Unicode is double-byte). One character is used for NULL termination.

See Also

NWDSEGetLocalClassID (page 30)

NWDSEGetLocalEntryID

Retrieves the local ID for the specified eDirectory object.

Local Servers: blocking

Remote Servers: N/A

Classification: 4.x, 5.x, 6.x

Platform: NLM

Service: eDirectory Event

Syntax

```
#include <nwdsdsa.h>
#include <nwdsevnt.h>

N_EXTERN_LIBRARY (NWDSCCODE)  NWDSEGetLocalEntryID  (
   NWDSContextHandle   context,
   const pnstr         objectName,
   pnuint32            id);
```

Parameters

context

(IN) Specifies the eDirectory context for the request.

objectName

(IN) Points to the name of the eDirectory object whose local ID is to be returned.

id

(OUT) Points to the local ID for the object.

Return Values

0x0000	Successful
Negative Value	Negative values indicate errors. For error values, see “NDS Return Values” (*NDK: Novell eDirectory Core Services*).

Remarks

The name specified by objectName is relative to the context specified by context.

An object’s local ID is valid only for the server on which NWDSEGetLocalClassID is called. For this reason, this ID is called a local ID.

Comparisons of IDs is faster than comparisons of text strings. Therefore, to avoid unnecessary processing time, your application should filter on IDs when possible.

See Also

NWDSEGetLocalEntryName (page 36)

NWDSEGetLocalEntryName

Retrieves the name of the eDirectory object associated with the supplied local ID.

Local Servers: blocking

Remote Servers: N/A

Classification: 4.x, 5.x, 6.x

Platform: NLM

Service: eDirectory Event

Syntax

```
#include <nwdsdsa.h>
#include <nwdsevnt.h>

N_EXTERN_LIBRARY (NWDSCCODE)  NWDSEGetLocalEntryName  (
   NWDSContextHandle   context,
   nuint32             entryID,
   pnstr               objectName);
```

Parameters

context

(IN) Specifies the eDirectory context for the request.

entryID

(IN) Specifies the local ID for the eDirectory object.

objectName

(OUT) Points to the name of the eDirectory object associated with the local ID specified by entryID.

Return Values

0x0000	Successful
Negative Value	Negative values indicate errors. For error values, see “NDS Return Values” (*NDK: Novell eDirectory Core Services*).

Remarks

The form of the name returned by NWDSEGetLocalEntryName is dependant upon the settings of the flags associated with the eDirectory context specified by context.

The caller must allocate memory to receive the object name that is returned. The size of the allocated memory is ((MAX_DN_CHARS)+1)*sizeof(character size) where character size is 1 for single-byte characters, and 2 for double-byte characters (Unicode is double-byte). One character is used for the NULL terminator.

See Also

NWDSEGetLocalEntryID (page 34)

NWDSERegisterForEvent

Registers a function to be used as a callback when a specific eDirectory event occurs.

Local Servers: blocking

Remote Servers: N/A

Classification: 4.x, 5.x, 6.x

Platform: NLM

Service: eDirectory Event

Syntax

```
#include <nwdsdsa.h>
#include <nwdsevnt.h>

N_EXTERN_LIBRARY (NWDSCCODE) NWDSERegisterForEvent  (
   nint       priority,
   nuint32    type,
   nint       (*handler) (
        nuint32    type,
        nuint      size,
        nptr      *data));
```

Parameters

priority

(IN) Specifies the state the eDirectory event will be in when it is reported (see Section 5.1, "Event Priorities," on page 79).

type

(IN) Specifies the type of the event for which the callback is being registered (see Section 5.2, "Event Types," on page 80).

handler

(IN) Points to a function to be used as a callback when the event specified by type occurs.

Return Values

0x0000	Successful
Negative Value	Negative values indicate errors. For error values, see "NDS Return Values" (*NDK: Novell eDirectory Core Services*).

Remarks

The handler parameter is a pointer to a function that is to be called when the specified eDirectory event occurs. The function is defined as follows:

type

(IN) Identifies the type of the event that has occurred. (See the type parameter above.

size

(IN) Specifies the size of the data that is returned for the event.

data

(IN) Points to the location of the data that contains information related to the event. See Section 5.2, "Event Types," on page 80 for a list of the events and the structures associated with them.

The value returned by the callback must be 0 for success and any other value for failure. If the callback returns a nonzero value during a EP_INLINE priority event, the event will be aborted. The callback's return values for the EP_JOURNAL and EP_WORK priority events are ignored.

WARNING: Your application must not modify the data at the location pointed to by data. Multiple callbacks can be registered for each event, and all of the callbacks receive that same data. When a callback returns, the information pointed to by data is passed into the next callback, if one is registered. Changing the information pointed to by data can produce unpredictable behavior in other callbacks. If you are going to modify the information pointed to by data, make a local copy of the information.

The callbacks are run on threads that do not have CLIB context. If you are using functions that need CLIB context, you must establish the context by calling SetThreadGroupID (http://developer.novell.com/ndk/doc/clib/thmp_enu/data/sdk347.html#sdk347) (NLM Threads Management (http://developer.novell.com/ndk/doc/clib/thmp_enu/data/h7g6q8vc.html#bktitle)).

See Also

NWDSEUnRegisterForEvent (page 43), NWDSERegisterForEventWithResult (page 40)

NWDSERegisterForEventWithResult

Registers a function to be used as a callback when a specific eDirectory event occurs. Passes in the current state of the event it is registering for.

Local Servers: blocking

Remote Servers: N/A

Classification: 4.x, 5.x, 6.x

Platform: NLM

Service: eDirectory Event

Syntax

```
#include <nwdsdsa.h>
#include <nwdsevnt.h>

N_EXTERN_LIBRARY (NWDSCCODE)  NWDSERegisterForEventResult  (
   nint      priority,
   nuint32   type,
   nint      (*handler)(
       nuint32   type,
       nuint     size,
       nptr     *data,
       nint     *result),
   nint     *flags);
```

Parameters

priority

(IN) Specifies the state the eDirectory event will be in when it is reported (see Section 5.1, “Event Priorities,” on page 79).

type

(IN) Specifies the type of the event for which the callback is being registered (see Section 5.2, “Event Types,” on page 80).

handler

(IN) Points to a function to be used as a callback when the event specified by type occurs.

flags

(IN) Points to a function to be used as a callback when the event specified by type occurs.

Return Values

0x0000	Successful

Negative Value	Negative values indicate errors. For error values, see "NDS Return Values" (*NDK: Novell eDirectory Core Services*).

Remarks

The handler parameter is a pointer to a function that is to be called when the specified eDirectory event occurs. The function is defined as follows:

type

(IN) Identifies the type of the event that has occurred. (See the type parameter above.)

size

(IN) Specifies the size of the data that is returned for the event.

data

(IN) Points to the location of the data that contains information related to the event. See Section 5.2, "Event Types," on page 80 for a list of the events and the structures associated with them.

result

(IN) Points to the location of the data that contains information related to the result of the event, whether an error code or success.

flags can be one of the following values:

HF_ALL 0x0000	Invoke handler regardless of event status.
HF_DEAD 0x0001	This handler is dead (not passed by the user).
HF_AUDIT 0x0002	This handler is the audit handler.
HF_SUCCESS_ONLY 0x0004	Invoke handler if event is successful.
HF_FAIL_ONLY 0x0008	Invoke handler if event fails.

HF_SUCCESS_ONLY and HF_FAIL_ONLY are mutually exclusive. If they are used together, no events will be passed to this handler.

The value returned by the callback must be 0 for success and any other value for failure. If the callback returns a nonzero value during a EP_INLINE priority event, the event will be aborted. The callback's return values for the EP_JOURNAL and EP_WORK priority events are ignored.

WARNING: Your application must not modify the data at the location pointed to by data. Multiple callbacks can be registered for each event, and all of the callbacks receive that same data. When a callback returns, the information pointed to by data is passed into the next callback, if one is registered. Changing the information pointed to by data can produce unpredictable behavior in other callbacks. If you are going to modify the information pointed to by data, make a local copy of the information.

The callbacks are run on threads that do not have CLIB context. If you are using functions that need CLIB context, you must establish the context by calling SetThreadGroupID (http://developer.novell.com/ndk/doc/clib/thmp_enu/data/sdk347.html#sdk347) (NLM Threads Management (http://developer.novell.com/ndk/doc/clib/thmp_enu/data/h7g6q8vc.html#bktitle)).

See Also

NWDSEUnRegisterForEvent (page 43), NWDSERegisterForEvent (page 38)

NWDSEUnRegisterForEvent

Unregisters a callback that has been registered to be called when a specified eDirectory event occurs.

Local Servers: blocking

Remote Servers: N/A

Classification: 4.x, 5.x, 6.x

Platform: NLM

Service: eDirectory Event

Syntax

```
#include <nwdsdsa.h>
#include <nwdsevnt.h>

N_EXTERN_LIBRARY (NWDSCCODE)  NWDSEUnRegisterForEvent  (
   nint       priority,
   nuint32    type,
   nint      (*handler) (
       nuint32   type,
       nuint     size,
       nptr     *data));
```

Parameters

priority

(IN) Specifies the state for which the eDirectory event reporting was registered (see Section 5.1, “Event Priorities,” on page 79).

type

(IN) Specifies the type of the event for which the callback was registered (see Section 5.2, “Event Types,” on page 80).

handler

(IN) Points to the function that was registered to be used as a callback when the event occurred.

Return Values

0x0000	Successful
Negative Value	Negative values indicate errors. For error values, see “NDS Return Values” (*NDK: Novell eDirectory Core Services*).

Remarks

For more details about the parameters for this function, see NWDSERegisterForEvent (page 38) and NWDSERegisterForEventWithResult (page 40).

See Also

NWDSERegisterForEvent (page 38), NWDSERegisterForEventWithResult (page 40)

Structures

4

This section describes the structures used by Novell® eDirectory™ Event Services.

DSEACL

Contains information about an ACL on an object.

Service: eDirectory Event

Defined In: nwdsevnt.h

Structure

```
typedef struct
{
   uint32   entryID ;
   uint32   attrID ;
   uint32   privileges ;
} DSEACL;
```

Fields

entryID

Specifies the local ID for the object that received the rights.

attrID

Specifies the ID of the attribute for which the rights apply.

privileges

Specifies the effective privilege set for subject/object or subject/attribute pair.

Remarks

This structure is used to fill a parameter in the DSEVALData (page 75) structure.

The special attributes, [All Attribute Rights] and [Entry Rights] have local IDs.

Privileges are defined as follows.

Table 4-1 *All Attribute Rights*

C Value	Pascal Value	Value Name
0x00000001L	$00000001	DS_ATTR_COMPARE
0x00000002L	$00000002	DS_ATTR_READ
0x00000004L	$00000004	DS_ATTR_WRITE
0x00000008L	$00000008	DS_ATTR_SELF
0x00000020L	$00000020	DS_ATTR_SUPERVISOR
0x00000040L	$00000040	DS_ATTR_INHERIT_CTL

Table 4-2 *Entry Rights*

C Value	Pascal Value	Value Name
0x00000001L	$00000001	DS_ENTRY_BROWSE
0x00000002L	$00000002	DS_ENTRY_ADD
0x00000004L	$00000004	DS_ENTRY_DELETE
0x00000008L	$00000008	DS_ENTRY_RENAME
0x00000010L	$00000010	DS_ENTRY_SUPERVISOR
0x00000040L	$00000040	DS_ENTRY_INHERIT_CTL

DSEBackLink

Contains information about a server holding a back link.

Service: eDirectory Event

Defined In: nwdsevnt.h

Structure

```
typedef struct
{
   uint32   serverID ;
   unit32   remoteID ;
} DSEBackLink;
```

Fields

serverID

Specifies the local ID for the server that knows about the object.

remoteID

Specifies the object's local ID on the remote server specified by serverID.

Remarks

The Back Link syntax is used to identify a server that knows about the object that owns the Back Link information. This structure is used to fill a parameter in the DSEVALData (page 75) structure.

DSEBinderyObjectInfo

Contains information about a bindery object associated with an event.

Service: eDirectory Event

Defined In: nwdsevnt.h

Structure

```
typedef struct
{
   uint32   entryID ;
   uint32   parentID ;
   uint32   type ;
   uint32   emuObjFlags ;
   uint32   security ;
   char     name [48];
} DBEBinderyObjectInfo;
```

Fields

entryID

Specifies the local ID for the Directory object that is being created to represent the bindery object.

parentID

Specifies the local ID for the parent of the object specified by entryID.

type

Specifies the bindery object type.

emuObjFlags

Specifies the bindery object flags.

security

Specifies the bindery object security.

name

Specifies the name of the bindery object.

Associated Events

DSE_CREATE_BINDERY_OBJECT
DSE_DELETE_BINDERY_OBJECT

DSEBitString

Service: eDirectory Event

Defined In: nwdsevnt.h

Structure

```
typedef struct
{
   uint32   numberOfBits ;
   uint32   numberOfBytes ;
   char     data ;
} DSEBitString;
```

Fields

numberOfBits

Specifies the number of bits in the bit string.

numberOfBytes

Specifies the number of bytes in the bit string.

data

Specifies the data for the string.

Remarks

Bit strings are padded to 4-byte boundaries.

This structure is used to fill a parameter in the DSEFaxNumber (page 62) structure.

DSEChangeConnState

Contains information about a connection whose state is being changed

Service: eDirectory Event

Defined In: nwdsevnt.h

Structure

```
typedef struct
{
   uint32   connID ;
   uint32   entryID ;
   uint32   oldFlags ;
   uint32   newFlags ;
} DSEChangeConnState;
```

Fields

connID

Specifies the local ID for the connection whose state is being changed.

entryID

Specifies the entryID of the object that owns the authenticated connection.

oldFlags

Specifies the flag associated with the previous connection state.

oldFlags can have one of the following values:

C Value	Value Name
0x0001	DSE_CONN_VALID
0x0002	DSE_CONN_AUTHENTIC
0x0004	DSE_CONN_SUPERVISOR
0x0008	DSE_CONN_OPERATOR
0x0010	DSE_CONN_LICENSED
0x0020	DSE_CONN_SEV_IS_STALE
0x0040	DSE_CONN_IS_NCP
0x0080	DSE_CONN_CHECKSUMMING
0x00FF	DSE_CONN_OPERATIONAL_FLAGS
0x0100	DSE_CONN_SIGNATURES
0x0200	DSE_CONN_CSIGNATURES
0x0400	DSE_CONN_ENCRYPTION

C Value	Value Name
0x0700	DSE_CONN_SECURITY_FLAGS

newFlags

Specifies the flag that indicates the new connection state. Uses the same flags as oldFlags.

Associated Events

DSE_CHANGE_CONN_STATE

DSECIList

Service: eDirectory Event

Defined In: nwdsevnt.h

Structure

```
typedef struct
{
   uint32    numberOfStrings ;
   uint32    length1 ;
   unicode   string [1];
} DSECIList;
```

Fields

numberOfStrings

Specifies the number of strings the structure holds.

length1

Specifies the length (in bytes) of the first string.

string

Specifies the location of the first string.

Remarks

The strings in this structure are null terminated, and aligned on 4-byte boundaries. If necessary, the strings are padded to fit those boundaries. The value in length does not include the bytes used for padding.

The first uint32 (4 bytes) after the first string contains the length of the second string. The second string follows the length. Any remaining strings follow this pattern.

The Unicode strings are in Intel format, lo-hi order.

DSEDebugInfo

Contains DSTrace information corresponding to the information in DSTraceInfo.

Service: eDirectory Event

Defined In: nwdsevnt.h

Structure

```
typedef struct
{
   uint32   dstime, milliseconds ;
   uint32   curThread ;
   uint32   connID ;
   uint32   perpetratorID ;
   char    *fmtStr ;
   char    *parms ;
   char     data ;
} DSEDebugInfo;
```

Fields

dstime, milliseconds

Specifies the time the event occurred.

curThread

Specifies the thread that was running when the event occurred.

connID

Specifies the number of the connection that generated the event.

perpetratorID

Specifies the local ID for the object that requested the action. For example, Admin.Acmecorp that an entry be created.

fmtStr

Points to a sprintf type format string describing the parameters.

parms

Points to a pseudo-stack, or variable argument list, of parameters.

data

Specifies the parameter data.

Associated Events

All events having a DSE_DB_ prefix.

DSEEmailAddress

Contains e-mail addresses being reported in an event.

Service: eDirectory Event

Defined In: nwdsevnt.h

Structure

```
typedef struct
{
   uint32    type ;
   uint32    length ;
   unicode   address [1 /*or more*/];
} DSEEmailAddress;
```

Fields

type

Specifies the type of the e-mail address.

length

Specifies the length of the first e-mail address.

address

Specifies the location where the first e-mail address begins.

Remarks

type can be either zero or one.

If type is set to zero, the address is an e-mail address, in the form of non_MHS_Email_Protocol:non_MHS_Email_Address. Where non_MHS_Email_Protocol is a 1-8 character string, and non-MHS-Email_Address is a string for the actual address value, such as the following:

SMTP:JohnD@Novell.Com

If type is set to one, the address is an E-mail alias, in the form of non-MHS_Email_Protocol:non-MHS_Email_Alias. Where non_MHS_Email_Protocol is a 1-8 character string, and non-MHS_Email_Address is a string for the actual alias value, such as the following:

SMTP:JohnD@Novell.Com

DSEEntryInfo

Contains information about an entry involved in a DSEvent (returned for NetWare® 4.x events).

Service: eDirectory Event

Defined In: nwdsevnt.h

Structure

```
typedef struct
{
   uint32          perpetratorID ;
   uint32          verb ;
   uint32          entryID ;
   uint32          parentID ;
   uint32          classID ;
   uint32          flags ;
   DSETimeStamp    creationTime ;
   const unicode  *dn ;
   const unicode  *newDN ;
   char            data [1];
} DSEEntryInfo;
```

Fields

perpetratorID

Specifies the local ID for the object that requested the action. For example, Admin.Acmecorp requesting that an entry be created.

verb

Specifies the action that caused the event to occur. These verbs, such as DSV_MODIFY_ENTRY are defined in NWDSDEFS.H.

entryID

Specifies the local ID for the object that was acted upon.

parentID

Specifies the local ID for the parent of the object that was acted upon.

classID

Specifies the local ID for the object class type, such as User, of the object that was acted upon. This value is not set for DSE_CREATE_ENTRY events.

flags

Specifies the flags identifying the object type. For most object types, flags will be set to zero. For partition roots, external references, and aliases, flags will have the following values:

0x0001	DSEF_PARTITION_ROOT
0x0002	DSEF_EXTREF
0x0004	DSEF_ALIAS

creationTime

Specifies the creation time of the object that is associated with entryID and points to DSETimeStamp (page 71).

dn

Points to the distinguished name of the object that was acted upon.

newDN

Points to the new distinguished name of the object that was acted upon. This is valid only if the object's distinguished name has been changed.

data

Specifies the location where the data is stored for the dn and the newDN fields. (Do not access this data directly. Access it through the dn and newDN fields.)

Remarks

NetWare® 4.x events return this structure. NetWare 5.x events return DSEEntryInfo2. The distinguished names pointed to by dn and newDN are not in a form that is consistent with the names used by the eDirectory functions. To use these names, you must convert the names to the proper form by calling NWDSEConvertEntryName.

DSEEntryInfo2

Contains information about an entry involved in a DSEvent (returned for NetWare® 5.x events).

Service: eDirectory Event

Defined In: nwdsevnt.h

Structure

```
typedef struct
{
   uint32          perpetratorID ;
   uint32          verb ;
   uint32          entryID ;
   uint32          parentID ;
   uint32          classID ;
   uint32          flags ;
   DSETimeStamp    creationTime ;
   const unicode  *dn ;
   const unicode  *newDN ;
   uint32         *connID ;
   char            data [1];
} DSEEntryInfo;
```

Fields

perpetratorID

Specifies the local ID for the object that requested the action. For example, Admin.Acmecorp creating an entry.

verb

Specifies the action that caused the event to occur. These verbs, such as DSV_MODIFY_ENTRY, are defined in NWDSDEFS.H.

entryID

Specifies the local ID for the object that was acted upon.

parentID

Specifies the local ID for the parent of the object that was acted upon.

classID

Specifies the local ID for the object class type, such as User, of the object that was acted upon.

flags

Specifies the flags identifying the object type. For most object types, flags will be set to zero. For partition roots, external references, and aliases, flags will have the following values:

0x0001	DSEF_PARTITION_ROOT
0x0002	DSEF_EXTREF
0x0004	DSEF_ALIAS

creationTime

Specifies the creation time of the object that is associated with entryID and points to DSETimeStamp (page 71).

dn

Points to the distinguished name of the object that was acted upon.

newDN

Points to the new distinguished name of the object that was acted upon. This is valid only if the object's distinguished name has been changed.

connID

Specifies the connection that generated the event.

data

Specifies the location where the data is stored for the dn and the newDN fields. (Do not access this data directly. Access it through the dn and newDN fields.)

Remarks

NetWare 5.x events return this structure. NetWare® 4.x events return DSEEntryInfo. The distinguished names pointed to by dn and newDN are not in a form that is consistent with the names used by the eDirectory functions. To use these names, you must convert the names to the proper form by calling NWDSEConvertEntryName.

DSEEventData

Contains data from a DSEvent.

Service: eDirectory Event

Defined In: nwdsevnt.h

Structure

```
typedef struct
{
   uint32        dsTime, milliseconds ;
   uint32        curProcess ;
   uint32        connID ;
   uint32        verb ;
   uint32        perpetratorID ;
   uint32        d1 ;
   uint32        d2 ;
   uint32        d3 ;
   uint32        d4 ;
   uint32        dataType ;
   const void   *dataPtr ;
   char          data ;
} DSEventData;
```

Fields

dsTime

Specifies the time in milliseconds when the event occurred.

curProcess

Specifies the process that was running when the event occurred.

connID

Specifies the number of the connection that generated the event.

verb

Specifies the eDirectory verb that generated the event.

perpetratorID

Specifies the local ID for the object that requested the action. For example, Admin.Acmecorp requesting that an entry be created.

d1

The contents of this field depend on the type of event.

d2

The contents of this field depend on the type of event.

d3

The contents of this field depend on the type of event.

d4

The contents of this field depend on the type of event.

dataType

Specifies the type of data.

dataPtr

Points to the event data.

data

The event data.

Associated Events

DSE_PARTITION_OPERATION_EVENT
Events 53 through 203, except as noted in "Values" on page 79 .

DSEFaxNumber

Contains a fax number associated with a DSEvent.

Service: eDirectory Event

Defined In: nwdsevnt.h

Structure

```
typedef struct
{
   uint32          length ;
   unicode         telephoneNumber [1 /*or more*/];
   DSEBitString    parameters ;
} DSEFaxNumber;
```

Fields

length

Specifies the number of characters used in the telephone number.

telephoneNumber

Specifies the telephone number.

parameters

Specifies a bit string containing additional information and points to DSEBitString (page 50).

DSEHold

Contains information about server holds associated with a DSEvent.

Service: eDirectory Event

Defined In: nwdsevnt.h

Structure

```
typedef struct
{
   uint32   entryID ;
   uint32   amount ;
} DSEHold;
```

Fields

entryID

Specifies the ID of the object that owns the accounting information.

amount

Specifies the number of charges that are on hold.

Remarks

See the “Server Holds” attribute in the *eDirectory Schema Reference* to see how this information is used.

DSEModuleState

Contains information about a module state that is being changed.

Service: eDirectory Event

Defined In: nwdsevnt.h

Structure

```
typedef struct
{
   uint32    connID ;
   uint32    flags ;
   long      handle ;
   unicode   name [DSE_MAX_MODULE_NAME_CHARS];
} DSEModuleState;
```

Fields

connID

Specifies the connection used by the module.

flags

Specifies a flag identifying the module's state. Flags can have the following values:

0x01	DSE_MOD_CHANGING
0x02	DSE_MOD_LOADED
0x04	DSE_MOD_AUTOLOAD
0x08	DSE_MOD_HIDDEN
0x10	DSE_MOD_ENGINE
0x20	DSE_MOD_AUTOMATIC
0x40	DSE_MOD_DISABLED
0x80	DSE_MOD_MANUAL
0x100	DSE_MOD_SYSTEM
0x200	DSE_MOD_WAITING

When the Changing flag is combined with the Loaded flag, they indicate that the module is starting to load. When the Changing flag is not combined with the Loaded flag, it indicates that the module is starting to unload.

The Waiting flag indicates that a state change for the module has been queued.

handle

Specifies the module handle.

name

Specifies the name used by the module. The maximum number of characters in the module name is 32.

Associated Events

DSE_CHANGE_MODULE_STATE

DSENetAddress

Contains a network address associated with a DSEvent.

Service: eDirectory Event

Defined In: nwdsevnt.h

Structure

```
typedef struct
{
   uint32   type ;
   uint32   length ;
   uint8    data [1];
} DSENetAddress;
```

Fields

type

Specifies the type of the address.

length

Specifies the number of bytes in which the address is stored.

data

Specifies the place where the network address is stored.

Remarks

type can have the following values:

NT_IPX
NT_IP
NT_SDLC
NT_TOKENRING_ETHERNET
NT_OSI
NT_APPLETALK
NT_COUNT

The address is stored as a binary string. This string is the literal value of the address. To display it as a hexadecimal value, you must convert each 4-bit nibble to the correct character (0,1,2,3,...F).

For two net addresses to match, the type, length, and value of the addresses must match.

DSEOctetList

Contains a list of strings associated with a DSEvent.

Service: eDirectory Event

Defined In: nwdsevnt.h

Structure

```
typedef struct
{
   uint32   numOfStrings ;
   uint8    string1 ;
} DSEOctetList;
```

Fields

numOfStrings

Specifies the number of strings contained in the structure.

string1

Specifies the location of the data for the strings.

Remarks

The strings are length preceded.

DSEPath

Contains a path associated with a DSEvent.

Service: eDirectory Event

Defined In: nwdsevnt.h

Structure

```
typedef struct
{
   uint32   nameSpaceType ;
   uint32   volumeEntryID ;
   uint32   length ;
   uint8    data [1];
} DSEPath;
```

Fields

nameSpaceType

Specifies the type of the name space.

volumeEntryID

Specifies the local ID for the volume where the path is located.

length

Specifies the length of the path.

data

Specifies the location where the path is stored.

Remarks

The following name-space types have been defined:

DS_DOS
DS_MACINTOSH
DS_UNIX
FTAM
OS/2

DSEReplicaPointer

Contains a replica pointer associated with a DSEvent.

Service: eDirectory Event

Defined In: nwdsevnt.h

Structure

```
typedef struct
{
   uint32   serverID,
   uint32   type ;
   uint32   number ;
   uint32   replicaRootID ;
   char     referral [1];
} DSEReplicaPointer;
```

Fields

serverID

Specifies the local ID for the name server that holds the replica.

type

Specifies the replica type.

number

Specifies the replica number.

replicaRootID

Specifies the remote ID for the object that is the partition root. This is the replica root's local ID on the remote server.

referral

Specifies an array of network addresses for the server specified in serverID.

Remarks

These types of partitions are defined as follows:

RT_MASTER
RT_SECONDARY
RT_READONLY
RT_SUBREF

A server can have more than one address, such as IPX and IP.

DSESEVInfo

Contains a Security Equivalence Vector associated with a DSEvent.

Service: eDirectory Event

Defined In: nwdsevnt.h

Structure

```
typedef struct
{
   uint32    entryID ;
   uint32    retryCount ;
   uint32    valueID ;
   unicode   valueDN [MAX_DN_CHARS + 1];
   char      referral ;
} DBESEVInfo;
```

Fields

entryID

Specifies the local ID for the Directory object whose Security Equivalence Vector (SEV) is being checked.

retryCount

Reserved.

valueID

Reserved.

valueDN

Specifies the distinguished name of an object or group being checked.

referral

Specifies the server holding the object designated in the valueDN parameter.

Associated Events

DSE_CHECK_SEV

DSETimeStamp

Contains a time stamp associated with a DSEvent.

Service: eDirectory Event

Defined In: nwdsevnt.h

Structure

```
typedef struct
{
   uint32   seconds ;
   uint16   replicaNumber ;
   uint16   event ;
} DSETimeStamp;
```

Fields

seconds

Specifies in seconds when the event occurred. Zero equals 12:00 midnight, January 1, 1970, UTC.

replicaNumber

Specifies the number of the replica on which the change or event occurred.

event

Specifies an integer that further orders events occurring within the same whole-second interval.

Remarks

Two time stamp values are compared by comparing the seconds fields first and the event fields second. If the seconds fields are unequal, order is determined by the seconds field alone. If the seconds fields are equal, and the eventID fields are unequal, order is determined by the eventID fields. If the seconds and the event fields are equal, the time stamps are equal.

DSETraceInfo

Contains information about a DSTrace event. DSTrace events are now handled by the DSE_DB_ events and use the DSEDebugInfo structure.

Service: eDirectory Event

Defined In: nwdsevnt.h

Structure

```
typedef struct
{
   unsigned long   traceVector1 ;
   unsigned long   traceVector2 ;
   unsigned long   dstime ;
   unsigned long   milliseconds ;
   char            string [1024];
} DSETraceInfo;
```

Fields

traceVector1

Specifies a bit flag identifying the type of trace event.

traceVector2

Reserved.

dstime

Specifies in seconds when the event occurred. Zero equals 12:00 midnight, January 1, 1970, UTC.

milliseconds

Specifies a further ordering (in milliseconds) of the time specified by dstime.

string

Specifies a string containing a message about the event.

Remarks

Your application should not depend upon the text strings in the DSETraceInfo structure. eDirectory Trace Information is for internal development purposes. The text strings returned in string may change with any version of the OS.

The bits of the traceVector1 field are defined as follows:

Bit	Meaning
TV_ON	If set, tracing is enabled. This bit will always be set when trace events are received.

Bit	Meaning
TV_AUDIT	Auditing
TV_INIT	Initialization
TV_FRAGGER	Fragger
TV_MISC	Miscellaneous
TV_RESNAME	Resolve name
TV_STREAMS	Streams
TV_LIMBER	Limber
TV_JANITOR	Janitor
TV_BACKLINK	Backlink
TV_MERGE	Merge
TV_SKULKER	Skulker
TV_LOCKING	Locking
TV_SAP	SAP
TV_SCHEMA	Schema
TV_COLL	Collisions
TV_INSPECTOR	Inspector
TV_ERRORS	Errors
TV_PART	Partition operations
TV_EMU	Bindery Emulator
TV_VCLIENT	Virtual Client
TV_AUTHEN	Authentication
TV_RECMAN	Record Manager
TV_TIMEVECTOR	Time vectors
TV_REPAIR	DS_Repair
TV_DSAgent	Low-level DSA tracing
TV_ERRET	ERRET and ERRTRACE
TV_SYNC_IN	Incoming sync traffic
TV_THREADS	DS thread scheduling
TV_MIN	Default DSTRACE messages
TV_CHECK_BIT	All TV_ values must have this bit

Associated Events

DSE_OBSOLETE_TRACE

DSETypedName

Contains a Typed Name structure associated with a DSEvent.

Service: eDirectory Event

Defined In: nwdsevnt.h

Structure

```
typedef struct
{
   uint32   entryID ;
   uint32   level ;
   uint32   interval ;
} DSETypedName;
```

Fields

entryID

Specifies the local ID for the object.

level

Specifies the priority.

interval

Specifies the frequency of reference.

Remarks

The meaning of the information for this structure is determined by the attribute to which the information belongs.

DSEVALData

Contains information about an attribute value.

Service: eDirectory Event

Defined In: nwdsevnt.h

Structure

```
typedef union
{
   unicode              string [1/*or more*/];
   uint32               num ;
   uint32               entryID ;
   uint32               classID ;
   uint8                boolean ;
   DSENetAddress        netAddress ;
   DSEPath              path ;
   DSEReplicaPointer    replica ;
   DSEACL               acl ;
   DSETimeStamp         timeStamp ;
   DSEBackLink          backLink ;
   DSETypedName         typedName ;
   DSEHold              hold ;
   DSEEmailAddress      emailAddress ;
   DSEFaxNumber         faxNumber ;
   DSECIList            ciList ;
   uint8                octedString [1];
   DSEOctetList         octetList ;
} DSEValData;
```

Fields

string

Specifies the following syntaxes:

Case Exact String
Case Ignore String
Numeric String
Printable String
Telephone Number

num

Specifies the following syntaxes:

Counter
Integer
Interval
Time

entryID

Specifies the distinguished name.

classID

Specifies the class name.

boolean

Specifies a Boolean string value.

netAddress

Specifies DSENetAddress (page 66).

path

Specifies DSEPath (page 68).

replica

Specifies DSEReplicaPointer (page 69).

acl

Specifies DSEACL (page 46).

timeStamp

Specifies DSETimeStamp (page 71).

backLink

Specifies DSEBackLink (page 48).

typedName

Specifies DSETypedName (page 74)

hold

Specifies DSEHold (page 63).

emailAddress

Specifies DSEEmailAddress (page 55).

faxNumber

Points to DSEFaxNumber (page 62).

ciList

Points to DSECIList (page 53).

octetString

Indicates the octet string stream.

octetList

Points to DSEOctetList (page 67).

DSEValueInfo

Contains information about an attribute value.

Service: eDirectory Event

Defined In: nwdsevnt.h

Structure

```
typedef struct
{
   uint32          perpetratorID ;
   uint32          verb ;
   uint32          entryID ;
   uint32          attrID ;
   uint32          syntaxID ;
   uint32          classID ;
   DSETimeStamp    timeStamp ;
   unsigned        size ;
   char            data [1];
} DSEValueInfo;
```

Fields

perpetratorID

Specifies the local ID for the object that requested the action. (For example, Admin.Acmecorp requesting that a phone number be added.)

verb

Specifies the action that caused the event to occur. These verbs, such as DSV_MODIFY_ENTRY, are defined in NWDSDEFS.H.

entryID

Specifies the local ID for the object that was acted upon.

attrID

Specifies the local ID that identifies the type of schema attribute that was changed.

syntaxID

Specifies the syntax that the data is stored by.

classID

Specifies the local ID that identifies the class of the object identified by entryID.

timeStamp

Specifies the time when the event occurred and points to DSETimeStamp (page 71).

size

Specifies the size (in bytes) of the information stored in the location identified by data.

data

Specifies the information that further identifies the changes that were made.

Remarks

The information stored in the data field of this structure is stored in a union called DSEVALData (page 75).

Associated Events

DSE_ADD_VALUE
DSE_CLOSE_STREAM
DSE_DELETE_ATTRIBUTE
DSE_DELETE_VALUE

Values

5

This section describes the values used by Novell® eDirectory™ Event Services functions and structures.

- Section 5.1, "Event Priorities," on page 79
- Section 5.2, "Event Types," on page 80

5.1 Event Priorities

The functions for registering and unregistering eDirectory events are NWDSERegisterForEvent (page 38) and NWDSEUnRegisterForEvent (page 43), both of which require values for the priority, type, and handler parameters. The priority parameter can have one of the following values:

EP_INLINE	Provides synchronous pre-event reporting because the callback can determine whether or not the event is allowable. If the callback returns a nonzero value, the transaction is aborted and the callback's return value is returned to the module that created the event. Since the module that created the event waits for a response while the callback processes, callbacks need to return as quickly as possible. The callback can sleep, but usually only sleeps to allocate memory. A zero (0) return value indicates success. A nonzero value indicates failure. This priority is the most difficult one to use for chained event handlers. You cannot assume that an eDirectory event will complete if your callback returns zero. The next callback in the chain could abort the transaction. To verify changes occurred, register a callback for the DSEP_JOURNAL or DSEP_WORK priorities. Warning: While inside this callback, use discretion in calling functions that create more eDirectory events. This is a closed loop where the growth of the journal queue could be uncontrollable.
EP_JOURNAL	Provides synchronous post-event reporting because event information is stored in a journal queue that records the events in the order they occurred. A single thread services all of the callbacks for the events, so the callback's execution time should be minimized. One method is to have the callback determine if the data should be used. If it should be used, the callback stores the data in a list that another thread processes. If multiple callbacks are registered for the same event, the current callback must be processed before the next callback is called. The callback can sleep.

EP_WORK	Provides asynchronous post-event reporting because events are reported after they occur, but not necessarily in the order they occurred. They are reported only after all of the event's callbacks registered for the EP_JOURNAL priority have completed. Each callback is run on a separate thread. This frees the event handler from the time constraints of the other two priorities. The callback can sleep. Time is not a critical issue with this priority.

5.2 Event Types

The type parameter describes the type of event being registered. There are more than 200 event types, and they are described in the following tables. For each type of event, a structure is returned that contains information specific to that type of event. Each type in the following table returns a DSEEntryInfo (page 56) structure, except where noted.

Note that for events using the DSEEntryInfo (page 56) structure, NetWare 4.x returns DSEEntryInfo (page 56) ; NetWare 5.x returns a DSEEntryInfo2 (page 58) structure

	Event Type	Structure Returned
1	DSE_CREATE_ENTRY: A new eDirectory object has been created.	DSEEntryInfo
2	DSE_DELETE_ENTRY: An existing eDirectory object has been deleted.	DSEEntryInfo
3	DSE_RENAME_ENTRY: An existing eDirectory object has been renamed.	DSEEntryInfo
4	DSE_MOVE_SOURCE_ENTRY: This is the second of two events reported for a move operation. This event specifies the deletion of a eDirectory object from its original location in the Directory tree. (See DSE_MOVE_DEST_ENTRY.	DSEEntryInfo
5	DSE_ADD_VALUE: A value has been added to an object attribute.	DSEValueInfo (page 77)
6	DSE_DELETE_VALUE: A value has been deleted from an object attribute.	DSEValueInfo (page 77)
7	DSE_CLOSE_STREAM: A Stream attribute has been closed.	DSEValueInfo (page 77)
8	DSE_DELETE_ATTRIBUTE: An attribute has been deleted from an object. This generates DSE_DELETE_VALUE events for values associated with the attribute. The DSE_DELETE_VALUE events occur after the DSE_DELETE_ATTRIBUTE event.	DSEValueInfo (page 77)
9	DSE_SET_BINDERY_CONTEXT: The bindery context has been set on the server.	No data is associated with this event.
10	DSE_CREATE_BINDERY_OBJECT: A bindery object has been created.	DSEBinderyObjectInfo (page 49)
11	DSE_DELETE_BINDERY_OBJECT: A bindery object has been deleted.	DSEBinderyObjectInfo (page 49)
12	DSE_CHECK_SEV: The Security Equivalence Vector has been checked.	DSESEVInfo

	Event Type	Structure Returned
13	DSE_UPDATE_SEV: The Security Equivalence Vector has been updated.	No data is associated with this event.
14	DSE_MOVE_DEST_ENTRY: This is the first of two events reported for a move operation. This event specifies the placement of the eDirectory object into its new location in the Directory tree. (See DSE_MOVE_SOURCE_ENTRY.) This generates DSE_ADD_VALUE events for all of the values associated with the object.	DSEEntryInfo
15	DSE_DELETE_UNUSED_EXTREF: An unused external reference has been deleted.	DSEEntryInfo
16	DSE_OBSOLETE_TRACE: A DSTrace event has occurred. The specific DSTrace event is designated by a TV_ flag returned in the DSETraceInfo structrue	DSETraceInfo (page 72)
17	DSE_REMOTE_SERVER_DOWN: A remote server has gone down.	DSENetAddress
18	DSE_NCP_RETRY_EXPENDED: The number of retries for an NCP™ request has been expended.	DSENetAddress
19	DSE_REMOTE_CONN_CLEARED: A remote connection has been cleared.	DSENetAddress
20	DSE_PARTITION_OPERATION_EVENT: A partition operation has occurred.	DSEEventData
21	DSE_CHANGE_MODULE_STATE: The eDirectory module's state has changed.	DSEModuleState (page 64)
22	DSE_RESERVED_2: Not used.	
23	DSE_RESERVED_3: Not used	
24	DSE_RESERVED_4: Not used.	
25	DSE_RESERVED_5: Not used.	

The events described in the following table are debug trace events, which correspond to the older DSTrace type TV_. All use the DSEDebugInfo (page 54) structure, and the result field is always 0 (zero).

	Event Type	Structure Returned
26	DSE_DB_AUTHEN: An authentication debug message has been sent.	DSEDebugInfo
27	DSE_DB_BACKLINK: A backlink debug message has been sent.	DSEDebugInfo
28	DSE_DB_BUFFERS: A request buffer debug message has been sent.	DSEDebugInfo
29	DSE_DB_COLL: A collision debug message has been sent.	DSEDebugInfo
30	DSE_DB_DSAGENT: A low-level DSAgent debug message has been sent.	DSEDebugInfo
31	DSE_DB_EMU: A Bindery emulation debug message has been sent.	DSEDebugInfo
32	DSE_DB_FRAGGER: A Fragger debug message has been sent.	DSEDebugInfo
33	DSE_DB_INIT: An initialization debug message has been sent.	DSEDebugInfo

34	DSE_DB_INSPECTOR: An inspector debug message has been sent.	DSEDebugInfo
35	DSE_DB_JANITOR: A Janitor process debug message has been sent.	DSEDebugInfo
36	DSE_DB_LIMBER: A Limber process debug message has been sent.	DSEDebugInfo
37	DSE_DB_LOCKING: A locking debug message has been sent.	DSEDebugInfo
38	DSE_DB_MOVE: A move debug message has been sent.	DSEDebugInfo
39	DSE_DB_MIN: A default DSTrace (equivalent to ON) debug message has been sent.	DSEDebugInfo
40	DSE_DB_MISC: A miscellaneous debug message has been sent	DSEDebugInfo
41`	DSE_DB_PART: A partition operations debug message has been sent.	DSEDebugInfo
42	DSE_DB_RECMAN: A Record Manager debug message has been sent.	DSEDebugInfo
43	DSE_DB_OBSOLETEREPAIR: Not used.	DSEDebugInfo
44	DSE_DB_RESNAME: A Resolve Name debug message has been sent.	DSEDebugInfo
45	DSE_DB_SAP: A SAP debug message has been sent.	DSEDebugInfo
46	DSE_DB_SCHEMA: A schema debug message has been sent.	DSEDebugInfo
47	DSE_DB_SKULKER: A synchronization debug message has been sent.	DSEDebugInfo
48	DSE_DB_STREAMS: A streams debug message has been sent.	DSEDebugInfo
49	DSE_DB_SYNC_IN: An incoming synchronization debug message has been sent.	DSEDebugInfo
50	DSE_DB_THREADS: An eDirectory thread scheduling debug message has been sent.	DSEDebugInfo
51	DSE_DB_TIMEVECTOR: A time vectors debug message has been sent.	DSEDebugInfo
52	DSE_DB_VCLIENT: A virtual client debug message has been sent.	DSEDebugInfo

Most of the events described in the following table use theDSEEventData (page 60) structure. The Data Returned column indicates the structure returned for those events that do not use the DSEventData structure.

For the events using the DSEEventData structure, not all the fields are filled in for each event, so the Data Returned column lists the fields and their content for each event. Unused fields are set to -1 for IDs or values, or are set to 0 (zero) for pointers.

	Event Type	Data Returned
53	DSE_AGENT_OPEN_LOCAL: The local Directory agent has been opened.	d1: state (1: start, 0: end). The result is valid for the end state only. Outside the locks.

	Event Type	Data Returned
54	DSE_AGENT_CLOSE_LOCAL: The local Directory agent has been closed.	d1: state (1: start, 0: end). Outside the locks.
55	DSE_DS_ERR_VIA_BINDERY: An error was returned via the Bindery.	d1: error code is returned via the bindery. Outside the locks.
56	DSE_DSA_BAD_VERB: An incorrect verb number was given in a DSAgent request.	d1: bad verb number given to DSA request (NCP 104, 2). Outside the locks.
57	DSE_DSA_REQUEST_START: A DSAgent request has been started.	d1: verb number (NCP 104, 2). Outside locks.
58	DSE_DSA_REQUEST_END: A DSAgent request has completed.	d1: verb number. d2: primary ID. d3: request size. d4: reply size. Not in locks.
59	DSE_MOVE_SUBTREE: A container and its subordinate objects have been moved.	d1: source ID. d2: destination ID. Not in locks.
60	DSE_NO_REPLICA_PTR: A replica exists that has no replica pointer associated with it.	d1: partition ID. Inside locks.
61	DSE_SYNC_IN_END: Inbound synchronization has finished.	d1: ID of server sending changes. d2: partition root ID. d3: Number of entries sent. Outside the locks.
62	DSE_BKLINK_SEV: A backlink operation has updated an object's Security Equivalence Vector.	d1: ID of object being updated. Outside the locks

	Event Type	Data Returned
63	DSE_BKLINK_OPERATOR: A backlink operation has changed an object's console operator privileges.	d1: ID of object whose console operator privileges were changed. d2: ID of server to which privileges were changed. Outside the locks.
64	DSE_DELETE_SUBTREE: A container and its subordinate objects have been deleted.	d1: ID of subtree root. d2: count of objects deleted. Inside locks and transaction.
65	DSE_SET_NEW_MASTER: A new master replica has been designated.	d1: ID of partition being changed. Outside locks.
66	DSE_PART_STATE_CHG_REQ: A partition state change has been requested.	d1: ID of partition. d2: partnerPartID. d3: (function<<16) type. d4: state.
67	DSE_REFERRAL: A referral has been created.	d1: ID of local entry. d2: ID of partition. d3: referral type.
68	DSE_UPDATE_CLASS_DEF: A schema class definition has been updated.	uname: name of schema class updated. Inside locks and transaction.
69	DSE_UPDATE_ATTR_DEF: A schema attribute definition has been updated.	uname: name of schema attribute updated. Inside locks and transaction

	Event Type	Data Returned
70	DSE_LOST_ENTRY: eDirectory has encountered a lost entry. A lost entry is an entry for which updates are being received, but no entry exists on the local server.	d1: parent ID. d2: time stamp of event. uname: Unicode name of entry. Inside locks and transaction.
71	DSE_PURGE_ENTRY_FAIL: A purge operation on an entry has failed.	d1: ID of entry that failed. Inside lock and transaction.
72	DSE_PURGE_START: A purge operation has started.	d1: ID of partition being purged. d2: replica type. Inside lock.
73	DSE_PURGE_END: A purge operation has ended.	d1: ID of partition purged. d2: number of entries. d3: number of values purged. Outside of the locks.
74	DSE_FLAT_CLEANER_END: A Flatcleaner operation has completed.	d1: number entries purged. d2: number of values purged. Outside of locks.
75	DSE_ONE_REPLICA: A partition has been encountered that has only one replica. Novell® recommends that each partition have at least three replicas for greater fault-tolerance.	d1: ID of partition with only one replica. Inside the locks.
76	DSE_LIMBER_DONE: A Limber operation has completed.	d1: all initialized (Boolean value). d2: found new RDN (Boolean value). Outside locks.
77	DSE_SPLIT_DONE: A Split Partition operation has completed.	d1: ID of parent partition root. d2: ID of child partition root. Outside locks.

	Event Type	Data Returned
78	DSE_SYNC_SVR_OUT_START: Outbound synchronization has begun from a particular server.	d1: ID of server. d2: partition root ID. d3: replica state, type, and flags. Outside locks.
79	DSE_SYNC_SVR_OUT_END: Outbound synchronization from a particular server has finished.	d1: ID of server. d2: partition root ID. d3: objects sent. d4: values sent. Outside locks
80	DSE_SYNC_PART_START: Synchronization of a partition has begun.	d1: partition ID. d2: partition state. d3: replica type. Inside locks.
81	DSE_SYNC_PART_END: Synchronization of a partition has finished.	d1: partition ID. d2: All Processed (Boolean value). Outside locks.
82	DSE_MOVE_TREE_START: A Move Subtree operation has started.	d1: ID of subtree root being moved. d2: destination (parent) ID. d3: server ID starting from. Outside locks.
83	DSE_MOVE_TREE_END: A Move Subtree operation has finished.	d1: ID of subtree root being moved. d2: server ID starting from. Outside locks.
84	DSE_RECERT_PUB_KEY: An entry’s public key has been certified.	d1: ID of entry whose keys are being certified. Inside locks and transaction.

	Event Type	Data Returned
85	DSE_GEN_CA_KEYS: Certificate of Authority keys have been generated.	d1: ID of entry having CA Keys generated. Inside locks and transaction.
86	DSE_JOIN_DONE: A Join Partitions operation has completed.	d1: ID of parent partition root. d2: ID of child partition root. Inside locks.
87	DSE_PARTITION_LOCKED: A partition has been locked.	d1: ID of partition being locked. Outside locks.
88	DSE_PARTITION_UNLOCKED: A partition has been unlocked.	d1: ID of partition being unlocked.
89	DSE_SCHEMA_SYNC: The schema has been synchronized.	d1: allProcessed (Boolean value). Outside locks.
90	DSE_NAME_COLLISION: A name collision (two entries with the same name) has occurred.	d1: ID of original entry. d2: ID of duplicate entry. Inside locks and transaction.
91	DSE_NLM_LOADED: An NLM™ has been loaded.	d1: module handle of NLM that was loaded. Outside locks
92	Not used.	
93	Not used.	
94	DSE_LUMBER_DONE: A Lumber operation has completed.	No parameters. Outside locks.
95	DSE_BACKLINK_PROC_DONE: A backlink process has completed.	No parameters. Outside lock.
96	DSE_SERVER_RENAME: A server has been renamed.	name: ASCII new server name. Inside locks.

	Event Type	Data Returned
97	DSE_SYNTHETIC_TIME: To bring eDirectory servers into synchronization, synthetic time has been invoked.	d1: root entry of partition issuing time stamp. d2: partition ID. d3: count of time stamps requested. Inside locks.
98	DSE_SERVER_ADDRESS_CHANGE: A server's address has changed.	No parameters. Inside locks.
99	DSE_DSA_READ: A Read operation has been performed on an entry.	d1: ID of entry being read. Outside locks.

The events described in the following table are primarily used for auditing. Thus, whenever possible the event is reported within a transaction, so an error can be returned and the transaction aborted if necessary. All the events in the table use the DSEEventData (page 60) structure, but the fields are used differently by each event. The Data Returned column describes the fields used by each event.

	Event Type	Data Returned
100	DSE_LOGIN: A user has logged in.	d1: parent ID. d2: entry ID. d3: usedNullPassword (Boolean value). d4: bindery login (0) or NDS login (-1).
101	DSE_CHGPASS: A user's password has changed.	d1: parent ID. d2: entry ID.
102	DSE_LOGOUT: A user has logged out.	d1: parent ID. d2: entry ID.
103	DSE_ADD_REPLICA: A replica of a partition has been added to a server.	d1: partition root ID. d2: server ID. uname: server name.
104	DSE_REMOVE_REPLICA: A replica of a partition has been removed from a server.	d1: partition root ID. d2: server ID. uname: server name.

	Event Type	Data Returned
105	DSE_SPLIT_PARTITION: A partition has been split.	d1: parent partition root ID.
		d2: new partition root ID.
		uname: new partition entry name.
106	DSE_JOIN_PARTITIONS: A parent partition has been joined with a child partition.	d1: parent partition root ID.
		d2: child partition root ID.
107	DSE_CHANGE_REPLICA_TYPE: A partition replica's type has been changed.	d1: partition root ID.
		d2: target server ID.
		d3: old type.
		d4: new type.
108	DSE_ADD_ENTRY: An entry has been added beneath a container.	d1: parent ID.
		d2: entry ID.
		uname: entry name (DSA op).
		name: entry name (Bindery op).
109	DSE_ABORT_PARTITION_OP: A partition operation has been aborted.	d1: parent ID.
		d2: entry ID.
110	DSE_RECV_REPLICA_UPDATES: A replica has received an update during synchronization.	d1: replica root ID
111	DSE_REPAIR_TIMESTAMPS: A replica's time stamps have been repaired.	d1: replica root ID
112	DSE_SEND_REPLICA_UPDATES: A replica has sent an update during synchronization.	d1: replica root ID
113	DSE_VERIFY_PASS: A password has been verified.	d1: parent ID.
		d2: entry ID.
114	DSE_BACKUP_ENTRY: An entry has been backed up.	d1: entry ID
115	DSE_RESTORE_ENTRY: An entry has been restored.	d1: parent ID.
		name: entry RDN.
116	DSE_DEFINE_ATTR_DEF: An attribute definition has been added to the schema.	uname: attribute name.
117	DSE_REMOVE_ATTR_DEF: An attribute definition has been removed from the schema.	d1: attr ID.
		d2: schema root ID.
		uname: attribute name.

	Event Type	Data Returned
118	DSE_REMOVE_CLASS_DEF: A class definition has been removed from the schema.	d1: class ID. d2: schema root ID. uname: class name.
119	DSE_DEFINE_CLASS_DEF: A class definition has been added to the schema.	uname: class name.
120	DSE_MODIFY_CLASS_DEF: A class definition has been modified.	d1: class ID. d2: schema root ID. uname: class name.
121	DSE_RESET_DS_COUNTERS: The internal NDS counters have been reset.	d2: server ID.
122	DSE_REMOVE_ENTRY_DIR: A file directory associated with an entry has been removed.	d1: parent ID. d2: entry ID. uname: entry name.
123	DSE_COMPARE_ATTR_VALUE: A Compare operation has been performed on an attribute.	d1: parent ID. d2: entry ID. uname: attribute name.
124	DSE_STREAM: A stream attribute has been opened or closed	Opening a stream: d1:DSE_ST_OPEN. d2: Entry ID. d3: Attribute ID. d4: Requested rights. Closing a stream: d1:DSE_ST_CLOSE. d2: Entry ID. d3: Attribute ID.
125	DSE_LIST_SUBORDINATES: A List Subordinate Entries operation has been performed on a container object.	d1: parent ID. d2: entry ID. uname: entry name.
126	DSE_LIST_CONT_CLASSES: A List Containable Classes operation has been performed on an entry.	d1: parent ID. d2 entry ID. uname: entry name.
127	DSE_INSPECT_ENTRY: An Inspect Entry operation has been performed on an entry.	d1: parent ID. d2: entry ID.

	Event Type	Data Returned
128	DSE_RESEND_ENTRY: A Resend Entry operation has been performed on an entry.	d1: parent ID.
		d2: entry ID.
129	DSE_MUTATE_ENTRY: A Mutate Entry operation has been performed on an entry.	d1: entry ID.
		d2: new class ID.
		uname: new class name.
130	DSE_MERGE_ENTRIES: Two entries have been merged.	d1: winner parent ID.
		d2: winner entry ID.
		uname: loser entry name.
131	DSE_MERGE_TREE: Two eDirectory trees have been merged.	d1: root entry ID.
132	DSE_CREATE_SUBREF: A subordinate reference has been created.	d1: subref ID.
133	DSE_LIST_PARTITIONS: A List Partitions operation has been performed.	d1: partition root entry ID.
134	DSE_READ_ATTR: An entry's attributes have been read.	d1: entry ID.
		d2: attribute ID.
135	DSE_READ_REFERENCES: The references on a given object have been read.	d1: entry ID
136	DSE_UPDATE_REPLICA: An Update Replica operation has been performed on a partition replica.	d1: partition root ID.
		d2: entry ID.
		uname: entry name.
137	DSE_START_UPDATE_REPLICA: A Start Update Replica operation has been performed on a partition replica.	d1: partition root ID.
138	DSE_END_UPDATE_REPLICA: An End Update Replica operation has been performed on a partition replica.	d1: partition root ID.
139	DSE_SYNC_PARTITION: A Synchronize Partition operation has been performed on a partition replica.	d1: partition root ID.
140	DSE_SYNC_SCHEMA: The schema has been synchronized.	d1: tree root ID.
141	DSE_CREATE_BACKLINK: A backlink has been created.	d1: tree root ID.
		d2: ID of server making request.
		d3: local entry ID.
		d4: remote entry ID.

	Event Type	Data Returned
142	DSE_CHECK_CONSOLE_OPERATOR: An object has been checked for Console Operator rights.	d1: tree root ID. d2: server ID. d3: isOperator (Boolean). d4: ID of object being checked.
143	DSE_CHANGE_TREE_NAME: The tree name has been changed.	d1: tree root ID. uname: new tree name.
144	DSE_START_JOIN: A Start Join operation has been performed.	d1: parent partition root ID. d2: child partition root ID.
145	DSE_ABORT_JOIN: A Join operation has been aborted.	d1: parent partition root ID. d2: child partition root ID.
146	DSE_UPDATE_SCHEMA: An Update Schema operation has been performed.	d1: tree root ID. d2: server ID.
147	DSE_START_UPDATE_SCHEMA: A Start Update Schema operation has been performed.	d1: tree root ID. d2: server ID.
148	DSE_END_UPDATE_SCHEMA: An End Update Schema operation has been performed.	d1: tree root ID. d2: server ID.
149	DSE_MOVE_TREE: A Move Tree operation has been performed.	d1: parent ID.
150	DSE_RELOAD_DS: eDirectory has been reloaded.	d1: tree root ID.
151	DSE_ADD_PROPERTY: An attribute (property) has been added to an object.	d1: object ID. d3: security. d4: flags. name: object name.
152	DSE_DELETE_PROPERTY: An attribute (property) has been removed from an object.	d1: object ID. name: object name.
153	DSE_ADD_MEMBER: A member has been added to a Group object.	d1: object ID. d3: member ID. name: property name.
154	DSE_DELETE_MEMBER: A member has been deleted from a Group object.	d1: object ID. d3: member ID. name: property name.

	Event Type	Data Returned
155	DSE_CHANGE_PROP_SECURITY: Security for a bindery object's property has been changed.	d1: object ID. d3: new security. name: property name.
156	DSE_CHANGE_OBJ SECURITY: A bindery object's security has been changed.	d1: object parent ID d2: object ID d3: new security
157	DSE_READ_OBJ_INFO: A Read Object Info operation has been performed on an object.	d1: parent ID. d2: entry ID.
158	DSE_CONNECT_TO_ADDRESS: A connection has been established with a particular address.	d1: task ID. d3: address type. d4: address size. name: address data.
159	DSE_SEARCH: A Search operation has been performed.	d1: base object ID d2: scope d3: nodes to search (not used) d4: information type
160	DSE_PARTITION_STATE_CHG: A partition's state has changed.	d1: parition root ID. d2: partnerPartID. d3: (function<<16)|type. d4: state.
161	DSE_REMOVE_BACKLINK: A backlink has been removed.	d1: object ID affected. d2: server ID of removed backlink. d3: remote ID of removed backlink.
162	DSE_LOW_LEVEL_JOIN: A low-level join has been performed.	d1: parent partition root ID. d2: child partition root ID.
163	DSE_CREATE_NAMEBASE: The Directory namebase has been created.	No data returned. Outside lock.

	Event Type	Data Returned
164	DSE_CHANGE_SECURITY_EQUALS: An object's Security Equals attribute has been changed.	d1: object ID. d2: equivalent ID. d3: 0=delete, 1=add equivalence. Inside locks and transaction.
165	DSE_DB_OBSOLETEGUID: A globally unique ID debug message has been sent.	GUID library and service.
166	DSE_DB_NCPENG: An NCPENG debug message has been sent.	DSEDebugInfo (page 54)
167	DSE_CRC_FAILURE: A CRC failure has occurred when fragmented NCP requests were reconstructed.	d1: CRC failure type (0=server, 1=client). d2: server/client CRC error count.
168	DSE_ADD_ENTRY: A new object has been added under a container object.	d1: Parent ID. d2: object ID. Success returns DSE_DATATYPE_ STRUCT1
169	DSE_MODIFY_ENTRY: An attribute has been modified on an object.	d1: Parent ID. d2: object ID. Success returns DSE_DATATYPE_ STRUCT1
170	DSE_DB_OBSOLETE-FORWARDLINK	
171	DSE_OPEN_BINDERY: The Bindery has been opened.	d1: Tree root ID.
172	DSE_CLOSE_BINDERY: The Bindery has been closed.	d1: Tree root ID.
173	DSE_CHANGE_CONN_STATE: The connection state has changed.	DSEChangeConnState (page 51)
174	DSE_NEW_SCHEMA_EPOCH: A new schema epoch has been declared.	d1: Tree root ID.
175	DSE_DB_AUDIT: An auditing debug message has been sent.	DSEDebugInfo (page 54)
176	DSE_DB_AUDIT_NCP: An auditing NCP debug message has been sent.	DSEDebugInfo (page 54)
177	DSE_DB_AUDIT_SKULK: An auditing debug message concerning synchronization has been sent.	DSEDebugInfo (page 54)
178	DSE_MODIFY_RDN: A Modify RDN operation has been performed.	d1: Parent ID. d2: Entry ID. uname: Old RDN.

	Event Type	Data Returned
179	DSE_DB_LDAP: An LDAP debug message has been sent.	DSEDebugInfo (page 54)
180	DSE_ORPHAN_PARTITION: An orphan partition operation has been performed. This operation has four variations: Create, Remove, Link, and Unlink.	Create: d1: DSE_OP_CREATE d2: newPartitionID. d3: targetPartitionID. Remove: d1: DSE_OP_REMOVE d2: Partition ID. Link: d1: DSE_OP_LINK d2: Partition ID. d3: Target Partition ID. d4: Target Server ID. Unlink: d1: DSE_OP_UNLINK d2: Partition ID. d3: Target Partition ID.
181	DSE_ENTRYID_SWAP: A Swap Entry ID operation has been performed.	d1: Source ID. d1: Destination ID.
182	DSE_NCP_REQUEST: An NCP request has been made.	No data returned. Used by Lock Check.
183	DSE_DB_LOST_ENTRY: A lost entry debug message has been sent. A lost entry is an entry for which updates are being received, but no entry exists on the local server.	DSEDebugInfo (page 54)
184	DSE_DB-CHANGE_CACHE: A change cache debug message has been sent.	DSEDebugInfo (page 54)
185	DSE_LOW_LEVEL_SPLIT: A low-level partition split has been performed.	d1: Parent partition Root ID Child partition root ID. Outside lock.
186	DSE_DB_PURGE: A purge debug message has been sent.	DSEDebugInfo (page 54)
187	DSE_END_NAMEBASE_TRANSACTION: An End Namebase Transaction debug message has been sent.	DSEDebugInfo (page 54)
188	DSE_ALLOW_LOGIN: A user has been allowed to log in.	d1: Entry ID. d2: Flags-.

	Event Type	Data Returned
189	DSE_DB_CLIENT_BUFFERS: A client buffers debug message has been sent.	DSEDebugInfo (page 54)

The events described in the following table are primarily associated with WAN Traffic Manager. They must be inline events so policy results can be returned to eDirectory. Those events with no structure specified use the structure with the indicated parameters filled.

	Event Type	Structure Returned
190	DSE_DB_WANMAN: A WAN Traffic Manager debug message has been sent.	DSEDebugInfo (page 54)
191	DSE_WTM_RESERVED_1: Reserved for internal use.	
192	DSE_WTM_RESERVED_2: Reserved for internal use.	
193	DSE_WTM_RESERVED_3: Reserved for internal use.	
194	DSE_WTM_RESERVED_4: Reserved for internal use.	
195	DSE_WTM_RESERVED_5: Reserved for internal use.	
196	DSE_WTM_RESERVED_6: Reserved for internal use.	
197	DSE_LOCAL_REPLICA_CHANGE: A replica on the local server has been modified.	d1: opcode: DSE_LRC_ADD DSE_LRC_REMOVE DSE_LRC_MODIFY d2: replicaRootID
198	DSE_DB_DRL: A Distribute Reference Link (DRL) has been created.	DSEDebugInfo (page 54)
199	DSE_MOVE_ENTRY_SOURCE: An entry has been moved from a source server.	d1: Parent ID. d2: Destination Parent ID. d3: Source ID. uname: Name
200	DSE_MOVE_ENTRY_DEST: An entry has been moved to a destination server.	d1: Parent ID. d2: Destination Parent ID. d3: Source ID. uname: New name
201	DSE_NOTIFY_REF_CHANGE: A Used By obituary has been added to an object.	d1: Source ID being referenced.
202	DSE_DB_ALLOC: A memory allocation debug message has been generated.	DSEDebugInfo (page 54)
203	DSE_CONSOLE_OPERATION: A wire protocol verb has been invoked for a console command.	d1: Operation Code: DSC_

	Event Type	Structure Returned
204	DSE_DB_SERVER_PACKET: Not implemented.	DSEDebugInfo (page 54)
205	DSE_START_DIB_CHECK : A RecMan DIB validation has started.	
206	DSE_END_DIB_CHECK: A RecMan DIB validation has finished.	d1: Error status
207	DSE_DB_OBIT: An obituary debug message has been generated.	DSEDebugInfo (page 54)
208	DSE_REPLICA_IN_TRANSITION:	DSEEventData (page 60)
		d1: Partition Root ID
		d2: Last ID
209	DSE_DB_SYNC_DETAIL: A synchronization detail debug message has been generated.	DSEDebugInfo (page 54)
210	DSE_DB_CONN_TRACE: A connection trace debug message has been generated.	DSEDebugInfo (page 54)

The handler parameter is a pointer to a function that is to be called when the specified eDirectory event occurs. The function is defined as follows:

type

(IN) Identifies the type of the event that has occurred. (See the type parameter above.

size

(IN) Specifies the size of the data that is returned for the event.

data

(IN) Points to the location of the data that contains information related to the event.

The data structures in the above table are defined in NWDSEVNT.H.

The value returned by the callback must be 0 for success and any other value for failure. If the callback returns a nonzero value during a EP_INLINE priority event, the event will be aborted. The callback's return values for the EP_JOURNAL and EP_WORK priority events are ignored.

See Also

- Section 1.1, "eDirectory Event Introduction," on page 11

Revision History

A

The following table lists all changes made to the eDirectory™ Event Services documentation:

March 1, 2006	Added a statement about callback threads to Section 1.3.6, "Priority 2," on page 18.
October 5, 2005	Transitioned to revised Novell documentation standards.
February 2003	Renamed the product name from "NDS" to "Novell eDirectory" at relevant instances.
September 2002	Updated the classID parameter description DSEEntryInfo (page 56) struct to state that DSE_CREATE_ENTRY events do not return this data.
July 1998	Added NDS 8 events.