Novell Developer Kit

www.novell.com

LDAP LIBRARIES FOR C#

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 products, and to get updates, see www.novell.com/documentation.

Novell Trademarks

Contents

About This Guide

LDAP libraries for C# enable you to write applications for accessing, managing, and updating information stored in Novell® eDirectory™ or other LDAP compliant directories.

This guide consists of the following sections:

- Getting Started
- Concepts
- Tasks
- Referral Handling in LDAPv3
- Controls and Extensions
- LDAP Event Services
- Revision History

Audience

This guide is intended for C# developers who desire to write applications to access, manage, update, and search for information stored in Novell eDirectory and other LDAP-aware directories.

Feedback

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

Additional Information

For additional information, see the following resources:

- **Novell eDirectory and LDAP Information:**
    - *NDK: Novell eDirectory Technical Overview*
    - *NDK: LDAP and eDirectory Integration*

Documentation Updates

For the most recent version of this guide, see the LDAP Libraries for C# NDK page (http://developer.novell.com/ndk/ldapcsharp.htm).

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.

When a single pathname can be written with a backslash for some platforms or a forward slash for other platforms, the pathname is presented with a backslash. Users of platforms that require a forward slash, such as Linux* and UNIX*, should use forward slashes as required by your software.

Getting Started 1

This section contains information about how to set up your development environment for using the Novell.Directory.Ldap namespace. For conceptual information, refer to Chapter 2, "Concepts," on page 17.

The following sections cover a few basic requirements for getting set up and started with the LDAP Libraries for C#:

- Section 1.1, "Dependencies," on page 11
- Section 1.2, "Supported Platforms," on page 11
- Section 1.3, "Using Novell.Directory.Ldap on Windows," on page 12
- Section 1.4, "Using Novell.Directory.Ldap on Linux," on page 13
- Section 1.5, "Integrating SSL with LDAP Libraries for C#," on page 13
- Section 1.6, "Sample Code," on page 15
- Section 1.7, "LDAP Test Server," on page 15

1.1 Dependencies

You need the following to take full advantage of the functionality offered in the classes:

❑ Novell® eDirectory™

- Novell eDirectory 8.5 or higher to develop or run applications using the extensions for naming context and replica management.
- Novell eDirectory 8.7 or higher if you wish to develop or run applications that start/stop Transport Layer Security (TLS).

❑ For Windows*:

To use Novell.Directroy.Ldap on the Windows OS, you need to setup your Microsoft* .NET project using Novell.Directory.Ldap.

You need mono.security.dll, if the setup is Microsoft* .NET on the Windows OS. You can download Mono™ from http://www.mono-project.com/downloads (http://www.mono-project.com/downloads) and copy mono.security.dll to an appropriate location in your project.

❑ For Linux*:

To use Novell.Directroy.Ldap on Linux, you need to have Mono® (compiler, runtime and class libraries) installed on your Linux box. You can download Mono from http://www.mono-project.com/downloads/ (http://www.mono-project.com/downloads).

1.2 Supported Platforms

The LDAP Libraries for C# SDK enables application developers to write applications to access, manage, update and search for information stored in eDirectory and other LDAP-aware directories.

Client applications remotely access directory information stored on an LDAP server. The libraries currently support development of such applications on the following platforms:

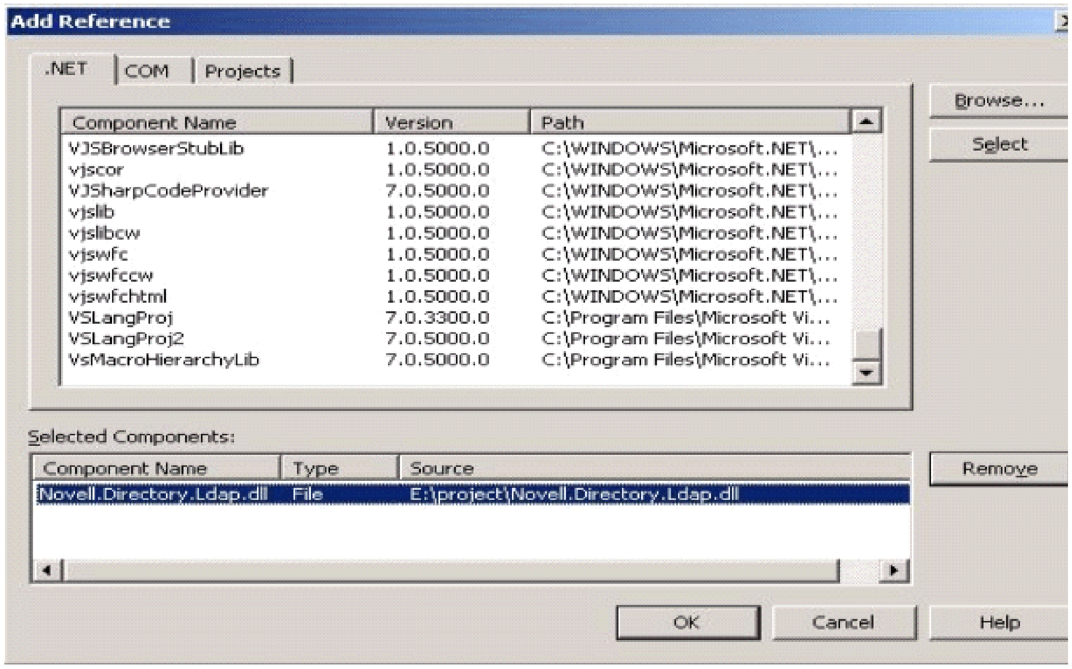
- Windows NT* Server 4.0 with SP 4 (with .NET framework installed)
- Linux (tested on Red Hat* 9.0)

1.3 Using Novell.Directory.Ldap on Windows

To use Novell.Directory.Ldap on Windows:

1. Download the `Novell.Directory.Ldap.dll` file from http://forge.novell.com/modules/xfmod/project/?ldapcsharp (http://forge.novell.com/modules/xfmod/project/?ldapcsharp) if you don't have it.
2. Copy `Novell.Directory.Ldap.dll` and `mono.security.dll` to an appropriate location in your project.
3. Start Visual Studio .NET.
4. Select *File* > *New* > *Project*.
5. In the *Project Type* column, select the project type to create C#.
6. In the *Template* column, select a project template (like Console Application, Windows Application).
7. Enter a name for your project.
8. Click *OK* to create your new project.
9. Select *Project* > *Add reference* > *Browse*.
10. In Browse, select `Novell.Directory.Ldap.dll` and `mono.security.dll` from the location you copied in Step 2.

Figure 1-1 *Windows Setup: Adding a Reference*



Component Name	Version	Path
VJSBrowserStubLib	1.0.5000.0	C:\WINDOWS\Microsoft.NET\...
vjscor	1.0.5000.0	C:\WINDOWS\Microsoft.NET\...
VJSharpCodeProvider	7.0.5000.0	C:\WINDOWS\Microsoft.NET\...
vjslib	1.0.5000.0	C:\WINDOWS\Microsoft.NET\...
vjslibcw	1.0.5000.0	C:\WINDOWS\Microsoft.NET\...
vjswfc	1.0.5000.0	C:\WINDOWS\Microsoft.NET\...
vjswfccw	1.0.5000.0	C:\WINDOWS\Microsoft.NET\...
vjswfchtml	1.0.5000.0	C:\WINDOWS\Microsoft.NET\...
VSLangProj	7.0.3300.0	C:\Program Files\Microsoft Vi...
VSLangProj2	7.0.5000.0	C:\Program Files\Microsoft Vi...
VsMacroHierarchyLib	7.0.5000.0	C:\Program Files\Microsoft Vi...

Component Name	Type	Source
Novell.Directory.Ldap.dll	File	E:\project\Novell.Directory.Ldap.dll

11 Click *OK* to *Add Reference*.

12 Add the following line to your code:

```
using Novell.Directory.Ldap;
```

1.4 Using Novell.Directory.Ldap on Linux

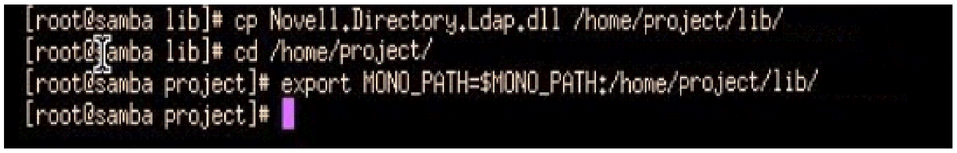
To use Novell.Directory.Ldap:

1 Download the `Novell.Directory.Ldap.dll` file from http://forge.novell.com/modules/xfmod/project/?ldapcsharp (http://forge.novell.com/modules/xfmod/project/?ldapcsharp) if you don't have it.

2 Copy `Novell.Directory.Ldap.dll` to an appropriate location in your project.

3 Set the directory containing `Novell.Directory.Ldap.dll` in your *MONO_PATH* variable as follows:

Figure 1-2 *Linux Setup*



```
[root@samba lib]# cp Novell.Directory.Ldap.dll /home/project/lib/
[root@samba lib]# cd /home/project/
[root@samba project]# export MONO_PATH=$MONO_PATH:/home/project/lib/
[root@samba project]#
```

4 In your project makefile set reference to `Novell.Directory.Ldap.dll` using

```
/r:/home/project/lib/Novell.Directory.Ldap.dll
```

5 Add the following line to your code:

```
Using Novell.Directory.Ldap;
```

NOTE: Adding the namespace using statement to your code is unnecessary, but can simplify object names. If you do not add this statement, then you must declare an object as Novell.Directory.Ldap.LdapConnection, instead of LdapConnection.

1.5 Integrating SSL with LDAP Libraries for C#

The LDAP libraries for C# perform their own authentication. To authenticate using SSL, the LDAP server must have a certificate to use with SSL, the .NET client must have a place to store the certificates, and the LDAP library must be set up to use SSL.

Thus to integrate SSL with the LDAP libraries for C#, you need to do the following:

- Section 1.5.1, "Setting Up the LDAP Server," on page 14
- Section 1.5.2, "Setting Up the .NET Client Application," on page 14
- Section 1.5.3, "Integrating the LDAP Libraries for C#," on page 15

1.5.1 Setting Up the LDAP Server

To set up the LDAP server:

- **Set up a digital certificate from a certificate authority.**
  See the documentation on Novell Certificate Server Version 1 (http://developer.novell.com/ndk/doc/ncslib/npkitenu/data/h1axgumd.html), for information on setting up a certificate on the Netware server.
- **Configure the LDAP server to use the certificate in ConsoleOne.**
  For instructions on this configuration, see Configuring LDAP Services for eDirectory (http://support.novell.com/techcenter/articles/dnd19981101.html) in the November 1998 issue of Novell Developer Notes.

1.5.2 Setting Up the .NET Client Application

You need to set up the .NET client application to store the certificates in a Mono Trust Store. Before setting up the trust store, ensure that you have:

- Mono Security Library, that is, Mono.Security.dll
- KeyStore for storing root certificates

On Linux, Mono.Security.dll and certmgr.exe utility are installed by default with the Mono packages.

On Windows, you need to install Mono.Security.dll and certmgr.exe (http://www.mono-project.com/about) . You also need to set the location in your .NET client application path.

`Mono.Security.dll` and the certmgr utility are used to create a Mono Trust Store that contains the server certificate.

Creating the Mono Trust Store

To create a trust store using Mono certmgr utility:

1. From ConsoleOne, create a trusted root certificate (a `.der` file).
2. Export the trusted root certificate to your local disk.
3. Rename the file

   ```
   [trusted root certificate].der
   ```

   to

   ```
   [trusted root certificate].cer
   ```

   This is because Mono currently does not recognize the .der extension.
4. Use the `certmgr.exe` utility to create a trust store file. If `/home/exports/TrustedRootCert.cer` is the certificate filename, the command would be as follows:

   ```
   certmgr -add -c Trust /home/exports/TrustedRootCert.cer
   ```
5. The certificate will be added to the Mono Trust Store location which you can find at:

   ```
   ~/.mono/certs/Trust directory
   ```

NOTE: The format and location of the trust store depends upon Mono releases. You should use certmgr tool to interact safely with the certificate stores. To get more information about certmgr, refer to the certmgr manpage.

1.5.3 Integrating the LDAP Libraries for C#

To integrate the Mono Security Library with the LDAP libraries for C#, set the SecureSocketLayer Property to true, after creating LdapConnection instance, as follows:

```
LdapConnection conn= new LdapConnection();conn.SecureSocketLayer=true;
```

For an example of setting up a .NET client to use SSL, see SecureBind.cs in the LDAP libraries for C# samples.

1.6 Sample Code

The LDAP Libraries for C# contain a number of samples demonstrating common operations. These samples are available on the Web (http://forge.novell.com/modules/xfmod/project/?ldapcsharp), or on the local drive once they have been installed.

1.7 LDAP Test Server

Novell has set up an LDAP server that you can access over the Internet to test your LDAP application. The server's name is www.nldap.com, and it listens on the default LDAP port (389). To use authenticated access, you must set up your own account in your own eDirectory container. Your account is limited to 1MB of disk storage.

To access this site, go to the NDS/LDAP Services Access Test Site (http://www.nldap.com/NLDAP).

2 Concepts

LDAP libraries for C# enable you to write applications for accessing, managing, and updating information stored in Novell® eDirectory™ or other LDAP compliant directories.

This chapter consists of the following sections:

2.1 Knowing the LDAP Model

Lightweight Directory Access Protocol (LDAP) is described in RFC 2251-2256 and RFC 2829-2830. It defines a lightweight access mechanism in which clients send requests to and receive responses from LDAP servers.

The LDAP information model comes from X.500 and is based on the entry, which contains information about some object (for example, a person). Entries are composed of attributes, which have a type and one or more values. Each attribute has a syntax that determines what kinds of values are allowed in the attribute (for example, ASCII characters, a JPEG photograph, etc.) and how those values behave during directory operations (for example, case significant during comparisons).

Entries can be organized in a tree structure, usually based on political, geographical, and organizational boundaries. Other structures are possible, including a flat namespace. Each entry is uniquely named relative to its sibling entries by its relative distinguished name (RDN) consisting of one or more distinguished attribute values from the entry. At the most, one value from each attribute may be used in the RDN. For example, the entry for the person "James Smith" might be named with the "Jonathan Smith" value from the CN (commonName) attribute.

A globally unique name for an entry, called a distinguished name or DN, is constructed by concatenating the sequence of RDNs from the entry up to the root of the tree. For example, if James worked for the Novell Inc., the DN of his Novell entry might be "cn= Jonathan smith,o=Novell,c=US". The DN format used by LDAP is defined in RFC2253.

Operations are provided to authenticate, search and retrieve information, modify, add and delete entries from the tree.

An LDAP server may return referrals if it cannot completely service a request (for example if the request specifies a directory base outside of the tree managed by the server).

The LDAP libraries for C# offers a programmer the following options:

- Catch the referrals as exceptions and explicitly issue new requests to the referred-to servers
- Provide an object to establish a new connection to a referred-to server
- Let the library automatically follow the referrals

2.2 How to Use LDAP Libraries for C#

An application generally uses the following procedure to utilize LDAP libraries for C#:

1. **SSL Integration:** Integrate the Mono Security Library with the LDAP libraries for C#. For integrating instructions, refer to Section 1.5.3, "Integrating the LDAP Libraries for C#," on page 15.
2. **Establish an LDAP Connection:** Initialize an LDAP session with a directory server. The LdapConnection.Connect() call establishes a handle to the session, allowing multiple sessions to be open at the same time, on different instances of LdapConnection.
3. **Authenticate to the LDAP Server:** Authenticate to the LDAP server with LdapConnection.Bind().
4. **Perform LDAP Operations and Obtain Results:** The synchronous version of LdapConnection.Search() returns an LdapSearchResults, which can be enumerated to access all entries found. The asynchronous version of LdapConnection.Search() returns an LdapSearchQueue, which is used to read the results of the search. LdapConnection.Read() returns a single entry.
5. **Close the Connection:** Close the connection. The LdapConnection.Disconnect() call closes the connection.

2.3 LDAP Libraries for C# Namespaces

The LDAP libraries for C# developer kit is released under the Novell.Directory.Ldap namespace and all the class files under this namespace are further divided into the six different namespaces:

- Section 2.3.1, "Novell.Directory.Ldap," on page 18
- Section 2.3.2, "Novell.Directory.Ldap.Asn1," on page 18
- Section 2.3.3, "Novell.Directory.Ldap.Controls," on page 19
- Section 2.3.4, "Novell.Directory.Ldap.Extensions," on page 19
- Section 2.3.5, "Novell.Directory.Ldap.Rfc2251," on page 19
- Section 2.3.6, "Novell.Directory.Ldap.Utilclass," on page 19

2.3.1 Novell.Directory.Ldap

Allows you to manage entries and schema definitions on LDAPv3 compliant servers. It provides classes for the core C# LDAP library, which is most frequently used by applications. These classes are based on Internet drafts maintained by IETF.

NOTE: The schema functionality is yet to be implemented in the .NET C# Library.

2.3.2 Novell.Directory.Ldap.Asn1

Allows you to encode and decode Abstract Syntax Notation One (ASN.1) object types using Basic Encoding Rules (BER).

ASN.1 is the language used by the OSI protocols for describing abstract syntax. ASN.1 is defined in ISO documents 8824.2 and 8825.2.

BER is historically the original encoding rules for ASN.1.

The LDAP protocol uses the BER encoding format and Novell.Directory.Ldap.Asn1 includes classes that allow ASN.1 to be encoded and decoded into the BER format. However, the classes have been built to be flexible enough to allow an application to provide its own ASN.1 encoder class. This class could encode data into any encoding format. For example, a particular application might want to use Packed Encoding Rules (PER) to encode the supported ASN.1 objects. This application would have to supply its own PER encoder and PER decoder classes. These application-provided classes will need to implement the ASN1Encoder and ASN1Decoder interfaces defined in this package.

NOTE: LDAP uses BER encoding and the Novell provided namespace already includes a BEREncoder and BERDEcoder class. These classes can be used by third party developers who wish to develop new LDAP controls or extensions. These classes could also be used by an arbitrary .NET C# application that wishes to encode and decode data as defined in the ASN.1 format.

2.3.3 Novell.Directory.Ldap.Controls

Provides classes for using LDAP controls. This namespace uses LDAP controls that are supported in LDAPv3. The use of these controls requires an LDAP server that supports them.

2.3.4 Novell.Directory.Ldap.Extensions

Provides classes for using the Novell LDAP extensions that manage replicas, naming contexts, and the synchronization of replicas and the schema. This namespace uses LDAP extensions that are supported in LDAPv3. These extensions require the LDAP server to run on eDirectory.

2.3.5 Novell.Directory.Ldap.Rfc2251

Provides classes that represent protocol elements as defined by the IETF LDAP RFC 2251. This namespace is designed to work on LDAPv3 servers. It does not support the T.61 character set used by the LDAPv2 protocol.

2.3.6 Novell.Directory.Ldap.Utilclass

Provides utility classes for use by LDAP applications. This namespace includes the DN class and RDN class supporting DN and RDN encapsulation respectively. It also provides classes perform client functions related to the LDAP protocol. This package is designed to work on LDAPv3 servers. It does not support the T.61 character set used by the LDAPv2 protocol.

The central LDAP class is LdapConnection. It provides methods to establish an authenticated or anonymous connection to an LDAP server, as well as methods to search for, modify, compare, and delete entries in the directory.

The LdapConnection class also provides fields for storing settings that are specific to the LDAP session (such as, limits on the number of results returned or timeout limits). An LdapConnection object can be cloned, allowing objects to share a single network connection but use different settings (using LdapConstraints or LdapSearchConstraints).

A synchronous search conducted by an LdapConnection object returns results in an LdapSearchResults object, which can be enumerated to access the entries found. Each entry (represented by an LdapEntry object) provides access to the attributes (represented by LdapAttribute objects) returned for that entry. Each attribute can produce the values found as byte arrays or as Strings.

2.4 LDAP Directory Access Methods

The LDAP protocol provides synchronous as well as asynchronous directory access methods.

Synchronous methods do not return until the operation has completed. To facilitate user feedback during synchronous searches, intermediate search results can be obtained before the entire search operation is completed by specifying the number of entries to return at a time.

Asynchronous methods take a MessageQueue parameter (either LdapResponseQueue or LDAPSearchQueue) and return a MessageQueue object which is used to enumerate the responses from the server. MessageQueue is associated with the request, and it is the responsibility of the client to read the messages from the queue and process them. A loop is typically used to read from the MessageQueue object, which blocks until there is a response available, until the operation has completed.

Messages Retrieved From	**Result Objects Derived From**
LdapResponseQueue	LdapResponse
LdapSearchQueue	LdapResponse, search results, or search result references.

An LdapResponseQueue can be shared between operations, for multiplexing the results. In this case, the object returned on one operation is passed in to one or more other operations, rather than passing in null.

For the asynchronous methods, exceptions are raised only for connection errors. LDAP result messages are converted into LdapResponse objects, which are to be checked by the client for errors and referrals, whereas, the synchronous methods throw an LdapException on result codes other than 0.

An asynchronous search conducted by an LdapConnection object returns results through the getResponse method of the LdapSearchQueue returned by the search operation. The getResponse method typically returns an LdapSearchResult object which has an Entry Property that returns the LdapEntry that represents the search entry.

None of the ancillary asynchronous classes are intended to be instantiated by a client, so they lack public constructors.

2.4.1 Error Handling

Errors result in the throwing of an LdapException, with a specific error code and context-specific textual information available.

If null is passed as the value of an LdapConstraints or LdapSearchConstraints parameter to an operation, the default constraints are used for that operation.

If null is passed as the value of a DN to an operation it is treated as if it was the empty string.

The API doesn’t distinguish between LDAP search continuation references and LDAP referrals, presenting a unified interface to the client for handling the two.

Implementations of the API must ensure that the LdapConnection class is thread-safe. Other classes and methods can be thread-safe and the implementor must indicate which classes and methods are thread-safe.

2.5 Using the LDAP Classes

This section contains general information that is helpful to understand before you begin developing with the LDAP Classes for C#. This section contains information on LDAP connections, asynchronous and synchronous methods, constraints, LDAP messages, and LDAP URLs. The namespace used in the C# LDAP SDK is Novell.Directory.Ldap.

2.5.1 LDAP Connections

The central LDAP class is an LdapConnection. This class provides methods to establish an authenticated or anonymous connection to an LDAP server, as well as methods to search for, modify, compare, and delete entries in the directory.

The following code demonstrates the use of an LdapConnection object to connect to an LDAP server:

```
String ldapHost = "localhost";
int ldapPort = 389;
LdapConnection ldapConn = new LdapConnection();
ldapConn.Connect( ldapHost, ldapPort );
```

These four lines use the LdapConnection object to create an anonymous connection to the LDAP server specified by *ldapHost*, and the port specified by *ldapPort*. At this point, you may authenticate to the server using the bind method, or perform another operation.

The LdapConnection class also provides methods for managing settings that are specific to the LDAP session (such as limits on the number of results returned or time-out limits). An LdapConnection object can be cloned, allowing objects to share a single network connection in a thread-safe manner.

2.5.2 Using Synchronous or Asynchronous Functions

Blocking versus Non-Blocking: The LDAP protocol provides both synchronous and asynchronous functions. For the synchronous search methods you can set the batch size parameter for functionality similar to the asynchronous search methods.

Asynchronous Functions

Asynchronous functions do not block, they return immediately after initiating the operation. One of the Listener class functions is used to retrieve the results.

Synchronous Functions

Synchronous functions with batch size of zero block until all the results have been received from the server. Synchronous search functions with batch size = n: (non-zero) Block until "n" messages have been received from the server, then let enumeration proceed while queuing additional messages.

The default value of the batch size parameter is 1. Thus by default, an enumeration of search results from a synchronous search operation will return messages as they are received from the server. The enumeration will block if no messages are waiting.

Other differences between asynchronous and synchronous operations are detailed in the operation-specific sections, such as exception handling and referral handling.

2.5.3 Clear Text vs. Encrypted Passwords

Before you can make a non-encrypted connection, the LDAP server must be configured to allow the clear-text passwords.

2.5.4 Using Constraints to Control Operations

LDAP constraints are used to control LDAP operations, allowing you to control the way in which operations are performed. Using constraints you can, for example, enable referral handling, set referral hop limits, and set controls to be sent to the server.

2.5.5 LDAP URLs

LDAP URLs provide a uniform method to access information on an LDAP server. Defined in RFC 2255, LDAP URLs begin with the prefix `LDAP://` or `LDAPS://`. The following provides the syntax and descriptions of an LDAP URL.

```
ldap[s]://<hostname>:<port>/
<base_dn>?<attributes>?<scope>?<filter>?<extension>
```

Note that ldaps is a common enhancement used to denote SSL, and is not defined in an RFC.

Table 2-1 *Field Descriptions of an LDAP URL*

URL Element	Default Value	Description
hostname	None	DNS name or IP address of the LDAP server
port	389	Port of the LDAP server.
base_dn	root	Base DN for the LDAP operation.
attributes	all attributes	A comma delimited list of attributes to return.
scope	base	Search scope.
filter	(objectClass=*)	Search filter.
extension	none	LDAP extended operations.

NOTE: An attribute list is required if you want to provide a scope (even if the attribute list is blank). To return all attributes within a specific scope you must include <base_dn>??<scope>.

The SDK provides an LdapUrl class to handle LDAP URLs. This class has methods to store, parse, and manage LDAP URLs.

2.5.6 Using LDAP URLs When Handling Referrals

If you receive an LdapReferralExeption, you can retrieve a list of referral URLs using the LdapReferralException.getReferrals method. This method returns an array of LDAP URL Strings, which can be converted to LDAPUrls and passed directly to LDAP searches, or can be examined to determine whether or not you wish to follow the referrals.

2.5.7 LDAP Messages

The LdapMessage class represents the base class for LDAP response messages for asynchronous commands.

For all asynchronous operations you are returned a listener object. Methods of this listener object return an LdapResponse (a subclass of LdapMessage), which contains the result of the operation.

When performing an asynchronous search, a number of LdapMessage objects are returned. These messages can be one of three sub-types:

- LdapSearchResult represents an entry returned from your search.
- LdapSearchResultReference contains a search result reference (referral information) to continue your search
- An LdapResponse, signals the end of the results.

In your code, you need to determine the message type and handle it appropriately.

For example, you could perform an asynchronous search and receive nine LDAP messages. Seven of these could be LdapSearchResults, one could be an LdapSearchResultReference, and the last one is an LdapResponse. In your code, you set up conditional statements to determine the message type and handle it appropriately.

2.6 Exception Handling

There are two types of exceptions in the C# LDAP SDK:

- LdapException
- LdapReferralException

Most errors that occur throw an LdapException. An LdapException is a general exception including an error message and an LDAP result code. An LdapException can result from physical problems (such as network errors) as well as problems with LDAP operations (for example, if the LDAP add operation fails because of a duplicate entry).

An LdapException can also contain a nested exception or a Throwable class with more information about the cause of the error. To retrieve the nested exception or Throwable class, use LdapException.getCause, which returns the lower-level exception which caused the failure. For example, an IOException with additional information may be returned on a CONNECT_ERROR failure. The various result codes returned from an LDAP request are defined as constants in the LdapException class.

Depending whether you are using asynchronous or synchronous functions, exceptions are handled differently. These differences are outlined in the following sections.

2.6.1 Synchronous Methods

Synchronous methods throw an LdapException on result codes other than SUCCESS. This eliminates the need to check for errors, therefore a standard try/catch statement can be used to handle exceptions.

The following is an example using the LdapException.toString method to print error information:

```
try {
      [Attempt an LDAP operation]
}
catch( LdapException e ) {
   Console.WriteLine( "Error: " + e.ToString() );
}
```

To facilitate user feedback during synchronous searches, intermediate search results can be obtained before the entire search operation is completed by specifying the number of entries to return at a time. By calling the LdapSearchConstraints.BatchSize property, you can set the number of results to obtain before returning information back to the application. The default setting is 1, allowing results to be obtained as they are received by the server. By setting the batch size to 0, you ensure all the results have been received from the server and stored in local memory before returning to the application. This will allow you to enumerate through the results without ever blocking

2.6.2 Asynchronous Methods

For the asynchronous methods, exceptions are thrown only for local or non-server errors, such as connection errors or parameter errors. For server errors, LDAP result messages are returned as LdapResponse objects which must be checked by the client for errors.

The following is an example of error handling with a failed asynchronous search.

NOTE: Before you check the response for an error message you need to determine which type of LDAP message has been returned.

```
//previously determined that the message is an LdapResponse
if (message is LdapResponse)
{
  LdapResponse response = (LdapResponse) message;
  int status = response.getResultCode();
  if (status != LdapException.SUCCESS )
  {
    Console.WriteLine("Asynchronous search failed.");
    throw new LdapException( response.getErrorMessage(),
                             status,
                             response.getMatchedDN());
  }
}
```

2.6.3 Referral Exceptions

LdapReferralExceptions are encountered when performing synchronous LDAP operations. They are derived from LdapException and contain a list of URL strings corresponding to referrals received on an LDAP operation.

LdapReferralExceptions are thrown when referral handling is turned off in your application, or if the API attempted to follow a referral and the referral could not be followed. For example, if you have enabled automatic referral handling and the API throws an LdapReferralException, it means the referral could not be followed and you most likely have incomplete results. The LdapReferral exception will contain a list of LDAP URL strings the API attempted to follow.

Tasks

3

This section contains information about basic LDAP operations.

3.1 Binding an Entry to an LDAP Server

The bind operation allows the entry to authenticate to the server. An entry in a directory is uniquely identified using its distinguished name (DN). A client application can choose to bind to the directory using an identity (DN and password) or anonymously. The C# code snippet below shows how to bind a user entry to an LDAP server:

Anonymous Binding

```
// C# Library namespace
using Novell.Directory.Ldap;

// Creating an LdapConnection instance
LdapConnection ldapConn= new LdapConnection();

//Connect function will create a socket connection to the server
ldapConn.Connect (ldapHost,ldapPort);

//Bind function with null user dn and password value will perform
anonymous bind to LDAP server
ldapConn.Bind (null, null);
```

Binding Using an Identity

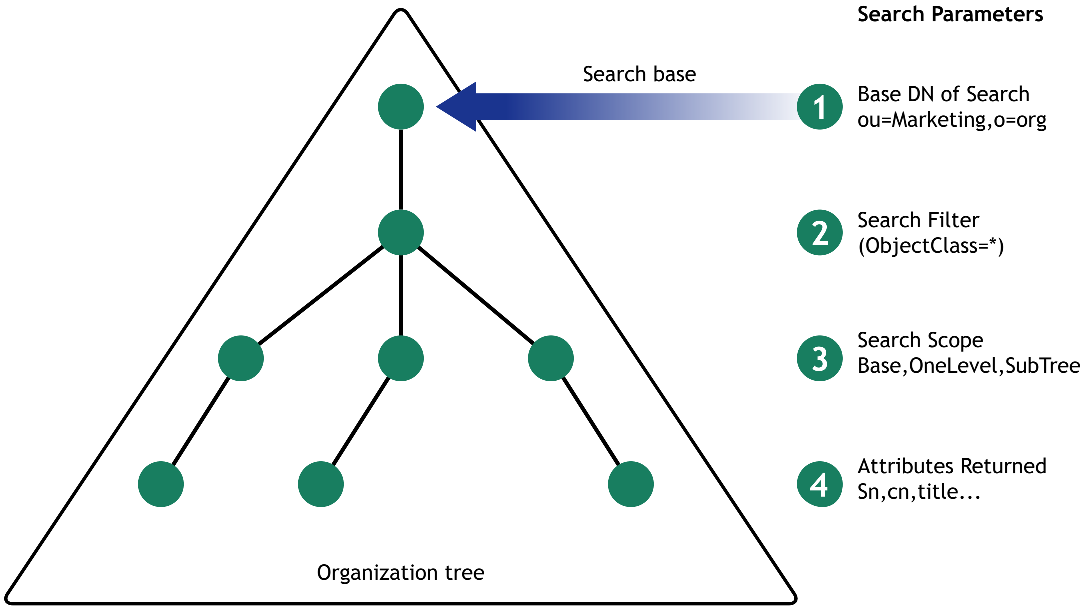
```
// C# Library namespace
using Novell.Directory.Ldap;

// Creating an LdapConnection instance
LdapConnection ldapConn= new LdapConnection();

//Connect function will create a socket connection to the server
ldapConn.Connect(ldapHost,ldapPort);

//Bind function will Bind the user object Credentials to the Server
ldapConn.Bind(userDN,userPasswd);
```

3.2 Searching the Directory

To perform a search, your application must first bind to the LDAP server and then select the root point in the directory (base object DN). For optimal performance, select a point that provides the smallest result set.

The following diagram illustrates a search that selects the marketing container as the root point for search. While setting up this search, decide on the type of object to search for and set up a search filter for that type of object.

Figure 3-1 *Search*



3.2.1 Specifying Search Parameters

Searching is performed using the LdapConnection.Search function. When you perform an LDAPsearch, you need to specify the following basic parameters:

- "Search Base" on page 28
- "Search Scope" on page 29
- "Search Filter" on page 29
- "Attribute List" on page 31
- "Types Only" on page 31

Search Base

The search base parameter specifies the DN of the entry where you want to begin the search, such as ou=development, o=acme.

If you want the search to begin at the tree root pass an empty string.

NOTE: Beginning a search at the tree root is handled differently by the various LDAP server implementations. If your search does not return results, read the root DSE to retrieve valid naming contexts for a valid starting point.

Search Scope

The search scope parameter specifies the depth of the search and can be one of three values:

- **SCOPE_BASE:** Only the entry specified as the search base is included in the search. This is used when you already know the DN of the object and you want to read its attributes. The read method may also be used to read the values of a single entry.
- **SCOPE_ONE:** Objects one level below the base (but not including the base) are included in the search. If we specified o=acme as our search base, then entries in the o=acme container would be included, but not the object o=acme.
- **SCOPE_SUB:** All objects below the base, including the base itself, are included in the search.

Search Filter

The search filter defines the entries that will be returned by the search. The LDAP search filter grammar is specified in RFC 2254 and 2251. The grammar uses ABNF notation. If you are looking for all employees with a title of engineer, the search filter would be (title=engineer).

Search Filters

The LDAP search filter grammar is specified in RFC 2254 and 2251. The grammar uses ABNF notation.

```
filter = " ( " filtercomp " ) "
filtercomp = and / or / not / item

and = "&" filterlist
   filterlist = 1*filter

or = "|" filterlist
   filterlist = 1*filter

not = "!" filterlist
   filterlist = 1*filter

item = simple / present / substring / extensible

simple = attr filtertype value
   attr =  name | name;binary
   filtertype = equal / approx / greater / less
   value = data valid for the attribute's syntax

equal = "="
approx = "~="
greater = ">="
less = "<="

present = attr "=*"
```

```
   attr =  name | name;binary

substing = attr "=" [initial] any [final]
   attr =  name | name;binary
   initial = value
   any = "*" *(value "*")
   final = value

extensible = attr [":dn"] [":" matchingrule] ":="value
            / [":dn] ":" matchingrule ":=" value
            / matchingrule = name | OID
```

For additional options for the attr option, see Section 4.1.5 of RFC 2251.

For additional information on the value option, see Section 4.1.6 of RFC 2251.

IMPORTANT: Novell® eDirectory™ does not support LDAP approximate (~=) matching or extensible matching rules.

Operators

Table 3-1 LDAP Filter Operators

Operator	Description
=	Used for presence and equality matching. To test if an attribute exists in the directory, use (attributename=*). All entries that have the specified attribute will be returned. To test for equality, use attributename=value. All entries that have attributename=value are returned. For example, (cn=Kim Smith) would return entries with Kim Smith as the common name attribute. (cn=*) would return all entries that contained a cn attribute. The = operator can also be used with wildcards to find a substring, (cn=*ary*) would return mary, hillary, and gary.
>=	Used to return attributes that are greater than or equal to the specified value. For this to work, the matching rule defined by the attribute syntax must have defined a mechanism to make this comparison. For example, (cn>=Kim Smith) would return all entries from Kim Smith to Z.
<=	Used to return attributes that are less than or equal to the specified value. For this to work, the matching rule defined by the attribute syntax must have defined a mechanism to make this comparison. For example, (cn<=Kim Smith) would return all entries from A to Kim Smith.
~=	Used for approximate matching. The algorithm used for approximate matching varies with different LDAP implementations.

The following Boolean operators can be combined with the standard operators to form more complex filters. Note that the Boolean operator syntax used is different in search filters than in the C and Java programming languages, but they are conceptually similar.

Table 3-2 *LDAP Filter Boolean Operators*

Boolean Operators	Description
&	And. For example, (&(Kim Smith) (telephonenumber=555-5555)) would return entries with common name of Kim Smith and a telephone number of 555-5555.
|	Or. For example, (|(cn=Kim Smith)(cn=Kimberly Smith)) would return entries with common name Kim Smith or Kimberly Smith.
!	Not. For example, (!(cn=Kim Smith)) would return entries with any cn other than Kim Smith. Note that the ! operator is unary, i.e. operates only on a single filter expression.

Examples

Filter and Description
(cn = Kim Smith)
Returns entries with a common name of Kim Smith.
(&(cn=Kim Smith)(telephonenumber=555*)(emailaddress=*acme.com))
Returns entries with a common name of Kim Smith, a telephone number that starts with 555, and an e-mail address that ends in acme.com
(!(cn = Chris Jones))
Returns entries that do not have a common name of Chris Jones.
(&(objectClass=inetOrgPerson) (| (sn=Smith) (cn=Chris S*)))
Returns entries that are of type inetOrgPerson with a surname of Smith or a common name beginning with Chris S.
(&(o=acme)(objectclass=Country)(!(|(c=spain)(c=us))
Returns entries that are of type Country from the organization Acme, that are not countries spain or us.

Operational Attributes

Operational attributes are not automatically returned in search results; they must be requested by name in the search operation. For a list of supported operational attributes in Novell eDirectory 8.5, see "LDAP Operational Attributes" in the *LDAP and eDirectory Integration Guide*. The LDAP servers in releases previous to 8.5 do not support requesting operational attributes in a search operation.

Attribute List

A null-terminated array of strings indicating which attributes to return for each matching entry.

Types Only

A Boolean specifying whether you want to return only the attribute names or the attribute types and the attribute values.

3.2.2 Getting Search Results

Each result returned by the Search function is stored in a MessageQueue which can be retrieved either in a synchronous way using LdapSearchResults class or in an asynchronous way using LdapSearchQueue class.

The C# code snippet below shows how to do a synchronous and asynchronous search in a LDAP server:

```
string searchBase = "ou=development,o=acme";
int searchScope = LdapConnection.SCOPE_BASE;
string searchFilter = "(title=engineer)";
```

Searching the Directory - Asynchronous

```
// C# Library namespace
using Novell.Directory.Ldap;

// Creating an LdapConnection instance
LdapConnection ldapConn= new LdapConnection();

//Connect function will create a socket connection to the server
ldapConn.Connect(ldapHost,ldapPort);

//Bind function will Bind the user object Credentials to theServer
ldapConn.Bind(userDN,userPasswd);

// Searches in the Marketing container and return all child entries
just below this container i.e Single level search

LdapSearchQueue queue=ldapConn.Search (searchBase,
LdapConnection.SCOPE_ONE, searchFilter,null,false,(LdapSearchQueue)
null,(LdapSearchConstraints)null );


LdapMessage message;
while ((message = queue.getResponse()) !=null)
{
   if (message is LdapSearchResult)
   {
          LdapEntry entry = (LdapSearchResult) message.Entry;
          System.Console.Out.WriteLine("\n" + entry.DN);
          System.Console.Out.WriteLine("\tAttributes: ");

          // Get the attribute set of the entry
          LdapAttributeSet attributeSet = entry.getAttributeSet();
          System.Collections.IEnumerator ienum =
attributeSet.GetEnumerator();

         // Parse through the attribute set to get the attributes and
the corresponding values
          while(ienum.MoveNext())
          {
                 LdapAttribute attribute=(LdapAttribute)ienum.Current;
```

```
                string attributeName =attribute.Name;
                string attributeVal = attribute.StringValue;
                Console.WriteLine( attributeName + "value:" +
attributeVal);}}
          }

   //Procced

   //While all the required entries are parsed, disconnect
   ldapConn.Disconnect();
```

Searching the Directory - Synchronous

```
// C# Library namespace
using Novell.Directory.Ldap;

// Creating an LdapConnection instance
LdapConnection ldapConn= new LdapConnection();

//Connect function will create a socket connection to the server
ldapConn.Connect(ldapHost,ldapPort);

//Bind function will bind the user object Credentials to the Server
ldapConn.Bind(userDN,userPasswd);

// Searches in the Marketing container and return all child entries
just below this container i.e. Single level search

LdapSearchResults
lsc=ldapConn.Search("ou=Marketing,o=Sales",LdapConnection.SCOPE_ONE,"
objectClass=*",null,false);
while (lsc.hasMore())
{
   LdapEntry nextEntry = null;
   try
   {
       nextEntry = lsc.next();
   } catch(LdapException e)
     {
         Console.WriteLine("Error: " + e.LdapErrorMessage);
         //Exception is thrown, go for next entry
         continue;
     }

Console.WriteLine("\n" + nextEntry.DN);

// Get the attribute set of the entry
LdapAttributeSet attributeSet = nextEntry.getAttributeSet();
System.Collections.IEnumerator ienum = attributeSet.GetEnumerator();

// Parse through the attribute set to get the attributes and the
corresponding values
while(ienum.MoveNext())
{
   LdapAttribute attribute=(LdapAttribute)ienum.Current;
   string attributeName = attribute.Name;
   string attributeVal = attribute.StringValue;
   Console.WriteLine( attributeName + "value:" + attributeVal);}
}

//Procced

//While all the entries are parsed, disconnect
ldapConn.Disconnect();
```

3.3 Creating an Entry in the Directory

This section contains information about how to add a new entry inside a directory using classes provided by Novell.Directory.Ldap namespace. To add a new directory entry, use the Add function of LdapConnection class.

Adding an entry involves four steps:

1. Create the attributes of the entry and add them to an attribute set.
2. Specify the DN of the entry to be created.
3. Create an LdapEntry object with the DN and the attribute set.
4. Call the LdapConnection.Add method to add it to the directory.

The C# code fragments below shows how to add an entry using Novell.Directory.Ldap namespace:

Example

```
// C# Library namespace
using Novell.Directory.Ldap;

// Creating an LdapConnection instance
LdapConnection ldapConn= new LdapConnection();

//Connect function will create a socket connection to the server
ldapConn.Connect(ldapHost,ldapPort);

//Bind function will Bind the user object  Credentials to the Server
ldapConn.Bind(userDN,userPasswd);

//Creates the List attributes of the entry and add them to attribute 
set
LdapAttributeSet attributeSet = new LdapAttributeSet();
attributeSet.Add( new LdapAttribute( "objectclass","inetOrgPerson"));
attributeSet.Add( new LdapAttribute("cn", new string[]{"James Smith", 
"Jimmy Smith"}));
attributeSet.Add( new LdapAttribute("givenname", "James"));
attributeSet.Add( new LdapAttribute("sn", "Smith"));
attributeSet.Add( new LdapAttribute("mail", "JSmith@Acme.com"));

// DN of the entry to be added
string dn = "cn=KSmith," + containerName;
LdapEntry newEntry = new LdapEntry( dn, attributeSet );
```

```
//Add the entry to the directory
ldapConn.Add( newEntry );
```

3.4 Modifying Entry Properties

This section contains information about how to modify the attributes of an existing entry inside a directory using classes provided by Novell.Directory.Ldap namespace. To modify the attributes of an existing entry, use the Modify function of LdapConnection class.

Modifying an entry attributes involves three steps:

1. Create an LdapModification object for each of the attributes to be modified using an attribute with a new value (in case of add/replace) and the modification type, for example, add, replace or delete.
2. Create an LdapModification array with all the LdapModification objects created above.
3. Call the LdapConnection.Modify method to modify the entry attributes

Example

```
// C# Library namespace
using Novell.Directory.Ldap;

// Creating an LdapConnection instance
LdapConnection ldapConn= new LdapConnection();

//Connect function will create a socket connection to the server
ldapConn.Connect(ldapHost,ldapPort);

//Bind function will bind the user object Credentials to the server
ldapConn.Bind(userDN,userPasswd);

ArrayList modList = new ArrayList();
String desc = "This object belongs to test user";

// Add a new value to the description attribute
LdapAttribute attribute = new LdapAttribute( "description", desc);
modList.Add( new LdapModification(LdapModification.ADD, attribute));

//Replace the existing email with the new email value
attribute = new LdapAttribute( "mail", "James_Smith@Acme.com");
modList.Add( new LdapModification(LdapModification.REPLACE,
attribute));

LdapModification[] mods = new LdapModification[modList.Count];
Type mtype=Type.GetType("Novell.Directory.LdapModification");
mods = (LdapModification[])modList.ToArray(typeof(LdapModification));

//Modify the entry in the directory
ldapConn.Modify ( dn, mods );
```

3.5 Renaming an Entry

This section contains information about how to rename (or change the RDN) an entry inside a directory using classes provided by Novell.Directory.Ldap namespace. To rename an entry use the Rename function of LdapConnection class.

Renaming an entry involves two steps:

1. Select the new RDN of the entry.
2. Specify whether you want to keep the old name as an attribute value or not.

The C# code fragments below shows how to rename an entry using Novell.Directory.Ldap namespace:

Example

```
// C# Library namespace
using Novell.Directory.Ldap;

// Creating an LdapConnection instance LdapConnection
ldapConn= new LdapConnection();

//Connect function will create a socket connection to the server
ldapConn.Connect(ldapHost,ldapPort);

//Bind function will bind the user object Credentials to the server
ldapConn.Bind(userDN,userPasswd);

//Renames the entry to newRDN. If third parameter is true it means, the
old name is not retained as an attribute value. If false, the old name
is retained as an attribute value.

ldapConn.Rename(oldDN, newRDN, true);
```

3.6 Moving an Entry

This section contains information about how to move an entry (changing the parent DN) inside a directory from one container to another using classes provided by Novell.Directory.Ldap namespace. To move an entry, use the Rename function of LdapConnection class, do the following:

1. Select the new container DN (parentDN) where the entry has to be moved.
2. Specify whether you want to keep the old name as an attribute value or not.

The C# code fragments below shows how to move an entry using Novell.Directory.Ldap namespace:

Example

```
// C# Library namespace
using Novell.Directory.Ldap;

// Creating an LdapConnection instance
LdapConnection ldapConn= new LdapConnection();
```

```
//Connect function will create a socket connection to the server
ldapConn.Connect(ldapHost,ldapPort);

//Bind function will bind the user object Credentials to the server
ldapConn.Bind(userDN,userPasswd);

//Moves the entry to new container named parentDN. If third parameter
//is true it means, the old name is not retained as an attribute value.
//If false, the old name is retained as an attribute value.
ldapConn.Rename(oldDN, oldDN,parentDN, true);
```

3.7 Deleting an Entry

This section contains information about how to delete an entry from the directory using classes provided by Novell.Directory.Ldap namespace. To delete an entry, use the Delete function of LdapConnection class.

Example

```
// C# Library namespace
using Novell.Directory.Ldap;

// Creating an LdapConnection instance
LdapConnection ldapConn= new LdapConnection();

//Connect function will create a socket connection to the server
ldapConn.Connect(ldapHost,ldapPort);

//Bind function will bind the user object Credentials to the server
ldapConn.Bind(userDN,userPasswd);

//Deletes the entry from the directory
ldapConn.Delete(entryDN);
```

Referral Handling in LDAPv3 4

Because of the distributed nature of directory services, a search sent to an LDAP server on eDirectory™ has a high probability of returning partial data, and referrals for the rest of the data.

When an LDAP server does not contain the requested data and a referral is necessary, the LDAPGroup object in eDirectory can be configured to handle them in one of four ways:

- Configure eDirectory to return complete data and never referrals to the client (always chain).
- Send referrals to the client only for eDirectory servers that do not support chaining.
- Always send referrals to the client (never chain).
- The client applications can be configured to have the API follow referrals, or the applications can perform their own handling of the referrals.

When you are using the LDAP SDK, note that referrals are handled differently for asynchronous and synchronous requests. Details are outlined in the following sections:

- Section 4.1, "Configuring eDirectory to Return Complete Data," on page 39
- Section 4.2, "Configuring eDirectory to Return Referrals," on page 39
- Section 4.3, "Enabling Referral Handling in the Application," on page 39
- Section 4.4, "Following Referrals Using Synchronous Requests," on page 40

4.1 Configuring eDirectory to Return Complete Data

In eDirectory, the LDAP server can be configured to return complete data and not return referrals. This is done through the LDAP Group Object using ConsoleOne. For possible configurations in e-Directory, see the documentation for the LDAP Group Object (http://www.novell.com/documentation/lg/ndsse/ndsseenu/data/fbabcieb.html).

4.2 Configuring eDirectory to Return Referrals

The LDAP server in eDirectory can also be configured to return referrals to your application. This is done through the LDAP Group Object using ConsoleOne®. For possible configurations in Novell e-Directory, see the documentation for the LDAP Group Object (http://www.novell.com/documentation/lg/ndsse/ndsseenu/data/fbabcieb.html)

4.3 Enabling Referral Handling in the Application

To enable referral following, use LDAPConstraints.setReferralFollowing passing TRUE to enable referrals, or FALSE (default) to disable referrals.

4.4 Following Referrals Using Synchronous Requests

When performing synchronous operations, referrals can be followed automatically with or without authentication, or they can be handled manually.

4.4.1 Following Referrals Manually

When referral handling is disabled, an LdapReferralException is thrown if the search result is a referral or a continuation reference.

You can use LdapReferralExceptions to follow referrals or continuation references by retrieving the URLs from the LdapReferralException and manually following them.

If you receive some data and an LdapReferralException is thrown, this is not an error. The server has probably returned partial data and a continuation reference for the remaining data. A separate LdapReferralException is thrown for each continuation reference received during a search.

4.4.2 Following Referrals Automatically as Anonymous

If referral following is enabled, referrals are followed by default using an anonymous bind to the next server. If your application does not require authentication, this default behavior is ideal.

If the server encounters a problem following a referral, an LdapReferralException is thrown. This exception provides information on the URLs that could not be followed, and it may contain a nested exception or throwable class with more information on what caused the exception. Be aware that if you receive some data and an LdapReferralException when using automatic referral handling, you most likely have incomplete results. This does not indicate the end of the data in your enumeration.

4.4.3 Following Referrals Automatically with Authentication

If your application requires more than anonymous authentication, you will need to implement a referral handler. The LDAP SDK provides interfaces your application can implement to provide credentials when following referrals. These interfaces are LdapAuthHandler and LdapBindHandler.

- **LdapAuthHandler:** This interface is the simplest to use. Your application creates an object that implements this interface and the SDK uses this class under-the-covers to authenticate.
- **LdapBindHandler:** Used to do explicit bind processing on a referral. This interface provides greater control over the bind process when following a referral but requires more work.

LdapAuthHandler

To use LdapAuthHandler you must create a class that extends the LdapAuthHandler interface. The following is an example of an LdapAuthHandler class:

```
class AuthImpl implements LdapAuthHandler
{
  private LdapAuthHandler auth;
  AuthImpl( String dn, byte[] pw )
  {
    auth = new LdapAuthProvider( dn, pw);
    return;
  }

  public LdapAuthProvider getAuthProvider(String host, int port)
  {
    return auth;
  }
}
```

LdapBindHandler

To use LdapBindHandler, you must create a class that extends the LdapBindHandler interface. The LdapBindHandler class provides the most flexibility, but you must perform your own bind operation. If the bind is successful, the referral will then be followed automatically.

Controls and Extensions

5

Controls and extensions were added to the LDAPv3 protocol. In version 2, there was no standard mechanism to extend the protocol, requiring developers to extend the protocol in nonstandard ways. In version 3, extensions and controls were defined to provide consistent expansion of the protocol.

5.1 Supported Controls

The following table contains a list of controls supported in the LDAP libraries for C#.

Table 5-1 *Supported Controls*

OID	Description
1.2.840.113556.1.4.473	Sever-side sort control request
1.2.840.113556.1.4.474	Server-side sort control response
2.16.840.1.113730.3.4.9	Virtual list view request
2.16.840.1.113730.3.4.10	Virtual list view response
2.16.840.1.113730.3.4.3	Persistent search
2.16.840.1.113730.3.4.7	Entry change notification
2.16.840.1.113730.3.4.2	Manage Dsa IT

- **Server Side Sort:** Returns results from a search operation in sorted order. This can be used to off-load processing from the client, or if you cannot sort the results for some reason.
- **Vertical List View:** Works in conjunction with the server side sort control to provide a dynamic view of a scrolling list. This works in conjunction with the server side sort control.
- **Persistent Search and Entry Change Notification:** Provides a control to perform a continuous search, notifying the application of changes.
- **Manage Dsa IT:** Causes directory-specific entries, regardless of type, to be treated as normal entries.

The C# code fragments below shows how to use the server side control using the Novell.Directory.Ldap namespace. The code below is hard coded to sort based on the “sn” attribute, and it searches for all objects at the specified searchBase.

Using Sort Control

```
// C# Library namespace
using Novell.Directory.Ldap;

// Creating an LdapConnection instance
LdapConnection ldapConn= new LdapConnection();

//Connect function will create a socket connection to the server
```

```
ldapConn.Connect(ldapHost,ldapPort);

//Bind function will Bind the user object Credentials to the Server
ldapConn.Bind(userDN,userPasswd);

String[] attrs = new String[1];
attrs[0] = "sn";
LdapSortKey[] keys = new LdapSortKey[1];
keys[0] = new LdapSortKey( "sn" );

// Create a LDAPSortControl object - Fail if cannot sort
LdapSortControl sort = new LdapSortControl( keys, true );

// Set the Sort control to be sent as part of search request
LdapSearchConstraints cons = ldapConn.SearchConstraints;

cons.setControls( sort );
ldapConn.Constraints = cons;
LdapSearchResults lsc=ldapConn.Search(searchBase,
LdapConnection.SCOPE_SUB,searchFilter,attrs,false,
LdapSearchConstraints)null );

while (lsc.hasMore())
{
LdapEntry nextEntry = null;

 try
 {
  nextEntry = lsc.next();
 }
 catch(LdapException e)
 {
 Console.WriteLine("Error: " + e.LdapErrorMessage);
// Exception is thrown, go for next entry
 continue;
 }
Console.WriteLine("\n" + nextEntry.DN);

LdapAttributeSet attributeSet =nextEntry.getAttributeSet();
System.Collections.IEnumerator ienum = attributeSet.GetEnumerator();

while(ienum.MoveNext())
{
LdapAttribute attribute=(LdapAttribute)ienum.Current;
string attributeName = attribute.Name;
string attributeVal = attribute.StringValue;
Console.WriteLine( attributeName + "value:" + attributeVal);
}
}

ldapConn.Disconnect();
```

5.2 Supported Extensions

The following table contains a list of extensions supported in the LDAP libraries for C#.

Table 5-2 *Supported Extensions*

OID	Name
2.16.840.1.113719.1.27.100.1	ndsToLdapResponse
2.16.840.1.113719.1.27.100.2	ndsToLdapRequest
2.16.840.1.113719.1.27.100.3	Split Partition Request
2.16.840.1.113719.1.27.100.4	Split Partition Response
2.16.840.1.113719.1.27.100.5	MergePartitionRequest
2.16.840.1.113719.1.27.100.6	MergePartitionResponse
2.16.840.1.113719.1.27.100.7	addReplicaRequest
2.16.840.1.113719.1.27.100.8	addReplicaResponse
2.16.840.1.113719.1.27.100.9	refreshLDAPServerRequest
2.16.840.1.113719.1.27.100.10	refreshLDAPServerResponse
2.16.840.1.113719.1.27.100.11	removeReplicaRequest
2.16.840.1.113719.1.27.100.12	removeReplicaResponse
2.16.840.1.113719.1.27.100.13	PartitionEntryCountRequest
2.16.840.1.113719.1.27.100.14	PartitionEntryCountResponse
2.16.840.1.113719.1.27.100.15	changeReplicaTypeRequest
2.16.840.1.113719.1.27.100.16	changeReplicaTypeResponse
2.16.840.1.113719.1.27.100.17	getReplicaInfoRequest
2.16.840.1.113719.1.27.100.18	getReplicaInfoResponse
2.16.840.1.113719.1.27.100.19	listReplicaRequest
2.16.840.1.113719.1.27.100.20	listReplicaResponse
2.16.840.1.113719.1.27.100.21	receiveAllUpdatesRequest
2.16.840.1.113719.1.27.100.22	receiveAllUpdatesResponse
2.16.840.1.113719.1.27.100.23	sendAllUpdatesRequest
2.16.840.1.113719.1.27.100.24	sendAllUpdatesResponse
2.16.840.1.113719.1.27.100.25	RequestPartitionSyncRequest
2.16.840.1.113719.1.27.100.26	RequestPartitionSyncResponse
2.16.840.1.113719.1.27.100.27	requestSchemaSyncRequest
2.16.840.1.113719.1.27.100.28	requestSchemaSyncResponse

OID	Name
2.16.840.1.113719.1.27.100.29	AbortPartitionOperationRequest
2.16.840.1.113719.1.27.100.30	AbortPartitionOperationResponse
2.16.840.1.113719.1.27.100.31	GetBindDNRequest
2.16.840.1.113719.1.27.100.32	Response Get Bind DN
2.16.840.1.113719.1.27.100.33	getEffectivePrivilegesRequest
2.16.840.1.113719.1.27.100.34	getEffectivePrivilegesResponse
2.16.840.1.113719.1.27.100.35	setReplicationFilterRequest
2.16.840.1.113719.1.27.100.36	setReplicationFilterResponse
2.16.840.1.113719.1.27.100.37	getReplicationFilterRequest
2.16.840.1.113719.1.27.100.38	getReplicationFilterResponse
2.16.840.1.113719.1.27.100.39	CreateOrphanPartitionRequest
2.16.840.1.113719.1.27.100.40	CreateOrphanPartitionResponse
2.16.840.1.113719.1.27.100.41	RemoveOrphanPartitionRequest
2.16.840.1.113719.1.27.100.42	RemoveOrphanPartitionResponse

The C# code fragment below shows how to use the extensions using the Novell.Directory.Ldap namespace. The code below shows a sample extension demonstrating how to reload the LDAP server module using LDAP extensions.

Using Extensions

```
// C# Library namespace
using Novell.Directory.Ldap;


// Creating an LdapConnection instance
LdapConnection ldapConn= new LdapConnection();


//Connect function will create a socket connection to the server
ldapConn.Connect(ldapHost,ldapPort);


//Bind function will Bind the user object  Credentials to the Server
ldapConn.Bind(userDN,userPasswd);


// Creating Refresh LDAP server extension.
LdapExtendedOperation request = new RefreshLdapServerRequest();


//Sending the extended request and getting back the result
LdapExtendedResponse response = ldapConn.ExtendedOperation(request);
if ( response.ResultCode == LdapException.SUCCESS )
{
      Console.WriteLine("Refresh Ldap Server Request succeeded\n");
}
ldapConn.Disconnect();
```

LDAP Event Services 6

LDAP Event Services provide a way for applications to monitor the activity of eDirectory™ on an individual server using LDAP. LDAP Event Services are available on eDirectory 8.7 and higher.

6.1 Concepts

LDAP Event Services utilizes the standard LDAP extension mechanism to expose the eDirectory event system. The LDAP Libraries for C# (http://developer.novell.com/ndk/ldapcsharp.htm) are enhanced to provide support functions to simplify the use of the event system extension.

The event system extension allows the client to specify the events for which it wants to receive notification. This information is sent in the extension request. If the extension request specifies valid events, the LDAP server keeps the connection open and uses the intermediate extended response to notify the client when events occur. Any data associated with an event is also sent in the response. If an error occurs when processing the extended request or during the subsequent processing of events, the server sends an extended response to the client containing error information and then terminates the processing of the request.

6.1.1 Configuring the eDirectory Event System

The eDirectory Event System extension is configured on a per LDAP server basis using the iManager utility (for information, see the iManager Documentation (http://www.novell.com/documentation/lg/imanager20/index.html)). There are two parameters that need to be set. The "allow event monitoring" parameter turns event monitoring on or off on that particular server. If event monitoring is turned off, the monitor events request fails. The second parameter is the maximum event monitoring load for the server. A zero value indicates no load limit. Each event type is assigned an integer valued load factor. The load factor is a representation of the loading effect monitoring this event has on the server relative to all other event types. The load is calculated based on each monitored event's load factor and the number of clients registered for that event.

Client Access Rights to Event Data

Any LDAP client can register to monitor any event. Access retrictions are enforced at the time of notification. If the authenticated client does not have access rights to view all of the information in the event, the event will not be sent. The one exception to this rule is the perpetrator DN. If the client does not have rights to the perpertrator object it will be sent as a zero length string. The event notification will still be sent.

6.1.2 Monitoring the eDirectory Events

The C# sample class EdirEventSample below shows how to monitor the eDirectory events using the Novell.Directory.Ldap namespace. The code below is hard coded to monitor the event type EVT_CREATE_ENTRY and it uses the command line parameters to get the inputs such as the host name, login DN and password.

```
using System;
```

```
//Use the C# LDAP namespaces
using Novell.Directory.Ldap;
using Novell.Directory.Ldap.Events;
using Novell.Directory.Ldap.Events.Edir;
using Novell.Directory.Ldap.Events.Edir.EventData;

public class EdirEventSample
{
 //Set the time out parameter.
 public const double TIME_OUT_IN_MINUTES = 5;
 public static DateTime timeOut;

 /**
  * Check the queue for a response. If a response has been received,
  * print the response information.
  */

 static private bool checkForAChange(LdapResponseQueue queue)
 {
   LdapMessage message;
   bool result = true;

   try
   {
     //check if a response has been received so we don't block
     //when calling getResponse()

     if (queue.isResponseReceived())
     {
        message = queue.getResponse();
        if (message != null)
        {
      // is the response a search result reference?

      if (message is MonitorEventResponse)
      {
        MonitorEventResponse eventerrorresponse =
(MonitorEventResponse) message;
        Console.WriteLine("\nError in Registration ResultCode  = " +
eventerrorresponse.ResultCode);
        EdirEventSpecifier[] specifiers =
eventerrorresponse.SpecifierList;
        for (int i = 0; i < specifiers.Length; i++)
        {
           Console.WriteLine("Specifier:" + "EventType  = " +
specifiers[i].EventType);
        }
        Environment.Exit(-1);

      }

      // is the response a event response ?

      else if ( message is EdirEventIntermediateResponse)
      {
         Console.WriteLine("Edir Event Occured");
         EdirEventIntermediateResponse eventresponse =
(EdirEventIntermediateResponse) message;

         //process the eventresponse Data, depending on the
         // type of response
         processEventData(eventresponse.EventResponseDataObject,
eventresponse.EventType);

      }

      // the message is an Unknown response
      else
      {
         Console.WriteLine("UnKnown Message =" + message);
      }
        }
     }

  }
  catch (LdapException e)
  {
     Console.WriteLine("Error: " + e.ToString());
     result = false;
  }

  return result;
 }

 public static void Main(String[] args)
 {
   if (args.Length != 3)
   {
     Console.WriteLine("Usage:   mint EdirEventSample <host name>
<login dn>"
                       + " <password> ");
     Console.WriteLine("Example: mint EdirEventSample Acme.com
\"cn=admin,o=Acme\""
                       + " secret ");
     Environment.Exit(0);
   }

   int ldapPort = LdapConnection.DEFAULT_PORT;  // Set to the default
LDAP Port
   int ldapVersion = LdapConnection.Ldap_V3;           // Set to the
LDAP version 3
   String ldapHost = args[0];
   String loginDN = args[1];
   String password = args[2];

   LdapResponseQueue queue = null;

   //Create the connection
   LdapConnection lc = new LdapConnection();

   try
   {
      // connect to the server
      lc.Connect(ldapHost, ldapPort);

      // authenticate to the server
      lc.Bind(ldapVersion, loginDN, password);

      //Create an Array of EdirEventSpecifier
      EdirEventSpecifier[] specifier = new EdirEventSpecifier[1];

      //Register for all Add Value events.

      specifier[0] = new
EdirEventSpecifier(EdirEventType.EVT_CREATE_ENTRY,
                  //Generate an Value Event of Type Add Value
                  EdirEventResultType.EVT_STATUS_ALL
                  //Generate Event for all status
                  );

      //Create an MonitorEventRequest using the specifiers.
      MonitorEventRequest requestoperation = new
MonitorEventRequest(specifier);

     //Send the request to server and get the response queue.
     queue = lc.ExtendedOperation(requestoperation, null, null);

   }

   //Catch the exception
   catch (LdapException e)
   {
      Console.WriteLine("Error: " + e.ToString());
      try
      {
      //Disconnect
      lc.Disconnect();
      }
      catch (LdapException e2)
      {
      Console.WriteLine("Error: " + e2.ToString());
      }
      Environment.Exit(1);
    }

    catch (Exception e)
    {
      Console.WriteLine("Error: " + e.ToString());
    }

    Console.WriteLine("Monitoring the events for {0} minutes..",
TIME_OUT_IN_MINUTES );
```

```
    //Set the timeout value
    timeOut= DateTime.Now.AddMinutes(TIME_OUT_IN_MINUTES);

    try
   {
      //Monitor till the timeout happens
      while (DateTime.Now.CompareTo(timeOut) < 0)
      {
         if (!checkForAChange(queue))
           break;
         System.Threading.Thread.Sleep(10);
      }
   }

    catch (System.IO.IOException e)
    {
      Console.WriteLine(e.Message);
    }

    catch (System.Threading.ThreadInterruptedException e)
    {
      Console.WriteLine(e.Message);
    }

    //disconnect from the server before exiting
    try
    {
        lc.Abandon(queue);   //abandon the queue
        lc.Disconnect();
    }

     catch (LdapException e)
     {
        Console.WriteLine("Error: " + e.ToString());
     }

      Environment.Exit(0);

 } // end main


 /**
  * Processes the Event Data depending on the Type.
  */

 static private void processEventData( BaseEdirEventData data,
EdirEventType type)
 {
    switch (type)
    {
      case EdirEventType.EVT_CREATE_ENTRY :
      // Value event.
      //Output the relevant Data.
      EntryEventData valueevent = (EntryEventData) data;
      Console.WriteLine("Entry         = " + valueevent.Entry);
      Console.WriteLine("PrepetratorDN = " +
valueevent.PerpetratorDN);
      Console.WriteLine("TimeStamp     = " + valueevent.TimeStamp);
      Console.WriteLine();
      break;

      default :
      //Unknow Event.
      break;
   }

 }
}
```

6.2 Event Types

The eDirectory event system supports over 200 events. Each event is identified by an integer eventType and most events have associated event data with additional information about the event. This information is returned in one of several structs depending upon the event type.

The eDirectory event system events are grouped according to the structure of the associated event data. This event grouping is outlined in the following table:

Table 6-1 *Event Grouping*

Event Type	**Description**
Entry Events	These events indicate the occurrence of individual entry operations such as creating or deleting an entry. The event data is contained in an EntryEventData Class.
Value Events	These events indicate the occurrence of attribute value operations such as deleting or adding a value. The event data is contained in a ValueEventData Class.
General DS Events	These are general events used to indicate a wide variety of DS operations. A generic structure, GeneralDSEventData Class, is used to hold the event data which needs to be interpreted based on the event type.
Bindery Events	These events occurrence of bindery emulation operations. The event data is contained in a BinderyObjectEventData Class.
Security Equivalence Event	This event indicates an entry’s security equivalence vector is being checked. The event data is contained in a SecurityEquivalenceEventData Class.
Network Address Events	The data for these events is contained in a NetworkAddressEventData Class.
Events without Data	This classification includes all events that do not have associated data.

6.2.1 Entry Events

The following table lists the event types that are associated with changes to individual attributes.

Table 6-2 *Event Types*

Val	Event Type	Description
1	EVT_CREATE_ENTRY	A new eDirectory object has been created. This event does not include className and is set to null.
2	EVT_DELETE_ENTRY	An existing eDirectory object has been deleted.
3	EVT_RENAME_ENTRY	An existing eDirectory object has been renamed.
4	EVT_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 EVT_MOVE_DEST_ENTRY).
14	EVT_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 EVT_MOVE_SOURCE_ENTRY.) This generates EVT_ADD_VALUE events for all of the values associated with the object.
15	EVT_DELETE_UNUSED_EXTREF	An unused external reference has been deleted.
228	EVT_PRE_DELETE_ENTRY	A pre-delete event posted when an entry is about to be deleted.

6.2.2 Value Events

The following table lists the events that are associated with changes to individual attributes:

	Event Type	Structure Returned
5	EVT_ADD_VALUE	A value has been added to an object attribute.
6	EVT_DELETE_VALUE	A value has been deleted from an object attribute.
7	EVT_CLOSE_STREAM	A Stream attribute has been closed.
8	EVT_DELETE_ATTRIBUTE	An attribute has been deleted from an object. This generates EVT_DELETE_VALUE events for values associated with the attribute. The EVT_DELETE_VALUE events occur after the EVT_DELETE_ATTRIBUTE event.

6.2.3 Debug Events

The following table lists the events that are associated with debug events:

	Event Type	Description	Data Returned
26	EVT_DB_AUTHEN	An authentication debug message has been sent.	DebugEventData Class
27	EVT_DB_BACKLINK	A backlink debug message has been sent.	DebugEventData Class
28	EVT_DB_BUFFERS	A request buffer debug message has been sent.	DebugEventData Class
29	EVT_DB_COLL	A collision debug message has been sent.	DebugEventData Class
30	EVT_DB_DSAGENT	A low-level DSAgent debug message has been sent.	DebugEventData Class
31	EVT_DB_EMU	A Bindery emulation debug message has been sent.	DebugEventData Class
32	EVT_DB_FRAGGER	A Fragger debug message has been sent.	DebugEventData Class
33	EVT_DB_INIT	An initialization debug message has been sent.	DebugEventData Class
34	EVT_DB_INSPECTOR	An inspector debug message has been sent.	DebugEventData Class
35	EVT_DB_JANITOR	A Janitor process debug message has been sent.	DebugEventData Class
36	EVT_DB_LIMBER	A Limber process debug message has been sent.	DebugEventData Class
37	EVT_DB_LOCKING	A locking debug message has been sent.	DebugEventData Class
38	EVT_DB_MOVE	A move debug message has been sent.	DebugEventData Class
39	EVT_DB_MIN	A default DSTrace (equivalent to ON) debug message has been sent.	DebugEventData Class
40	EVT_DB_MISC	A miscellaneous debug message has been sent	DebugEventData Class
41	EVT_DB_PART	A partition operations debug message has been sent.	DebugEventData Class
42	EVT_DB_RECMAN	A Record Manager debug message has been sent.	DebugEventData Class
44	EVT_DB_RESNAME	A Resolve Name debug message has been sent.	DebugEventData Class
45	EVT_DB_SAP	A SAP* debug message has been sent.	DebugEventData Class
46	EVT_DB_SCHEMA	A schema debug message has been sent.	DebugEventData Class

	Event Type	Description	Data Returned
47	EVT_DB_SKULKER	A synchronization debug message has been sent.	DebugEventData Class
48	EVT_DB_STREAMS	A streams debug message has been sent.	DebugEventData Class
49	EVT_DB_SYNC_IN	An incoming synchronization debug message has been sent.	DebugEventData Class
50	EVT_DB_THREADS	An eDirectory thread scheduling debug message has been sent.	DebugEventData Class
52	EVT_DB_TIMEVECTOR	A time vectors debug message has been sent.	DebugEventData Class
53	EVT_DB_VCLIENT	A virtual client debug message has been sent.	DebugEventData Class
166	EVT_DB_NCPENG	An NCPENG debug message has been sent.	DebugEventData Class
175	EVT_DB_AUDIT	An auditing debug message has been sent.	DebugEventData Class
176	EVT_DB_AUDIT_NCP	An auditing NCP™ debug message has been sent.	DebugEventData Class
177	EVT_DB_AUDIT_SKULK	An auditing debug message concerning synchronization has been sent.	DebugEventData Class
184	EVT_DB-CHANGE_CACHE	A change cache debug message has been sent.	DebugEventData Class
186	EVT_DB_PURGE	A purge debug message has been sent.	DebugEventData Class
189	EVT_DB_CLIENT_BUFFERS	A client buffers debug message has been sent.	DebugEventData Class
190	EVT_DB_WANMAN	A WAN Traffic Manager debug message has been sent	DebugEventData Class
198	EVT_DB_DRL	A Distribute Reference Link (DRL) has been created.	DebugEventData Class
202	EVT_DB_ALLOC	A memory allocation debug message has been generated.	DebugEventData Class
204	EVT_DB_SERVER_PACKET	Not implemented.	DebugEventData Class
207	EVT_DB_OBIT	An obituary debug message has been generated.	DebugEventData Class
209	EVT_DB_SYNC_DETAIL	A synchronization detail debug message has been generated.	DebugEventData Class
210	EVT_DB_CONN_TRACE	A connection trace debug message has been generated.	DebugEventData Class

	Event Type	Description	Data Returned
214	EVT_DB_DIRXML	A DirXML® debug message has been sent.	DebugEventData Class
217	EVT_DB_DIRXML_DRIVERS	A DirXML Drivers debug message has been sent.	DebugEventData Class
218	EVT_DB_NDSMON	A NDSMON debug message has been sent.	DebugEventData Class
220	EVT_DB_DNS	A DNS debug message has been sent.	DebugEventData Class
221	EVT_DB_REPAIR	A DS Repair debug message has been sent.	DebugEventData Class
222	EVT_DB_REPAIR_DEBUG	A Repair Debug debug message has been sent.	DebugEventData Class
225	EVT_DB_SCHEMA_DETAIL	A Schema Detail debug message has been sent.	DebugEventData Class
227	EVT_DB_IN_SYNC_DETAIL	A Sync Detail debug message has been sent.	DebugEventData Class
229	EVT_DB_SSL	An SSL debug message has been sent.	DebugEventData Class
230	EVT_DB_PKI	A PKI debug message has been sent.	DebugEventData Class
231	EVT_DB_HTTPSTK	A HTTPSTK debug message has been sent.	DebugEventData Class
232	EVT_DB_LDAPSTK	An LDAPSTK debug message has been sent.	DebugEventData Class
233	EVT_DB_NICIEXT	A NICI Ext debug message has been sent.	DebugEventData Class
234	EVT_DB_SECRET_STORE	A SecretStore debug message has been sent.	DebugEventData Class
235	EVT_DB_NMAS	A NMAS™ debug message has been sent.	DebugEventData Class
236	EVT_DB_BACKLINK_DETAIL	A Backlink Detail debug message has been sent.	DebugEventData Class
237	EVT_DB_DRL_DETAIL	A DRL debug message has been sent.	DebugEventData Class
238	EVT_DB_OBJECT_PRODUCER	An Object Producer debug message has been sent.	DebugEventData Class
239	EVT_DB_SEARCH	A Search debug message has been sent.	DebugEventData Class
240	EVT_DB_SEARCH_DETAIL	A Search Detail debug message has been sent.	DebugEventData Class

	Event Type	Description	Data Returned
242	EVT_DB_NPKI_API	An NPKI debug message has been sent.	DebugEventData Class

6.2.4 General DS Events

A large number of events are classified as general events. The meaning of the data returned is dependent on the type of the event.

For example, when a EVT_LOGIN (event type 100) occurs, the GeneralDSEventData Class contains several general data fields about the event (dstime, milliseconds, curProcess, and verb). The final two output parameters (intValues[], and strValues[]) contain information specific to the EVT_LOGIN event. The description of this information for each event is contained in the Data Returned column in the following table.

In the following example, each integer and string value from the Data Returned column is paired with its corresponding value in the GeneralDSEventData Class:

```
Integer Values
[0] (corresponds to) intValues[0]
[1] (corresponds to) intValues[1]
...

String Values:
[0] (corresponds to) strValues[0]
[1] (corresponds to) strValues[1]
...
```

The Data Returned column for the EVT_LOGIN event contains the following:

```
Integer Values
[0] 0 if a non-null password was used, 1 if a null password was used
[1] 0 if a bindery login was used, -1 if an NDS login was performed

String Values:
[0] DN of the parent of the entry that performed the login
[1] DN of the entry that performed the login
```

The data described by Integer Values [0] is contained in the first location in the intValues array, intValues[0]. If you wanted to determine if a non-null password was used during the login, intValues[0] would be checked to determine if it is 1 or 0 (with a value of 0 indicating that a non-null password was used).

Table 6-3 *Event Type*

	Event Type	Description	Data Returned
53	EVT_AGENT_OPEN_LOCAL	The local Directory agent has been opened.	Integer Values: [0] end state of the open operation String Values: [0] end [1] start [2] audit
54	EVT_AGENT_CLOSE_LOCAL	The local Directory agent has been closed.	Integer Values: [0] the state of the operation String Values: [0] end [1] start
55	EVT_DS_ERR_VIA_BINDERY	An error was returned from the bindery.	Integer Values: [0] error code returned from the bindery
56	EVT_DSA_BAD_VERB	An incorrect verb number was given in a DSAgent request.	Integer Values: [0] bad verb number given to the DSA Request (NCP 104, 2)
57	EVT_DSA_REQUEST_START	A DSAgent request has been started.	Integer Values: [0] verb number (NCP 104, 2)
58	EVT_DSA_REQUEST_END	A DSAgent request has completed.	Integer Values: [0] verb number [1] primary ID [2] request size [3] reply size
59	EVT_MOVE_SUBTREE	A container and its subordinate objects have been moved.	String Values: [0] DN of source container [1] DN of destination container
60	EVT_NO_REPLICA_PTR	A replica exists that has no replica pointer associated with it.	String Values: [0] DN of associated partition object
61	EVT_SYNC_IN_END	Inbound synchronization has finished.	Integer Values: [0] number of entries sent String Values: [0] DN of the server entry associted with the server sending the changes [1] DN of root entry of the partition

	Event Type	Description	Data Returned
62	EVT_BKLINK_SEV	A backlink operation has updated an object’s Security Equivalence Vector.	String Values: [0] DN of the updated object
63	EVT_BKLINK_OPERATOR	A backlink operation has changed an object’s console operator privileges.	String Values: [0] DN of updated entry [1] DN of server entry for which the privleges were changed
64	EVT_DELETE_SUBTREE	A container and its subordinate objects have been deleted.	Integer Values: [0] number of entries deleted String Values: [0] DN of the root object of the deleted subtree
67	EVT_REFERRAL	A referral has been created.	Integer Values: [0] Referral type String Values: [0] DN of the partition [1] DN of the entry
68	EVT_UPDATE_CLASS_DEF	A schema class definition has been updated.	String Values: [0] Name of updated class
69	EVT_UPDATE_ATTR_DEF	A schema attribute definition has been updated.	String Values: [0] Name of updated or added attribute
70	EVT_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.	Integer Values: [0] Seconds field of entry's timestamp [1] replicaNumber field of entry's timestamp [2] Event field of the entry's timestamp String Values: [0] DN of the entry's parent
71	EVT_PURGE_ENTRY_FAIL	A purge operation on an entry has failed.	String Values: [0] DN of the entry for which the purge operation failed
72	EVT_PURGE_START	A purge operation has started.	Integer Values: [0] Replica type String Values: [0] DN of the partition being purged

	Event Type	Description	Data Returned
73	EVT_PURGE_END	A purge operation has ended.	Integer Values: [0] Number of entries purged [1] Number of values purged String Values: [0] DN of the purged partition
76	EVT_LIMBER_DONE	A Limber operation has completed.	Integer Values: [0] 1 indicates all initialized, 0 indicates not all initialized [1] 1 indicates that a new RDN was found, 0 indicates that a new RDN was not found
77	EVT_SPLIT_DONE	A Split Partition operation has completed.	String Values: [0] DN of the parent partition's root [1] DN of the child partition's root
78	EVT_SYNC_SVR_OUT_START	Outbound synchronization has begun from a particular server.	Integer Values: [0] Replica number [1] Replica state, type, and flags String Values: [0] DN of the associated server entry [1] DN of the partition root
79	EVT_SYNC_SVR_OUT_END	Outbound synchronization from a particular server has finished.	Integer Values: [0] Number of objects sent [1] Number of values sent String Values: [0] DN of the associated server entry [1] DN of the partition root
80	EVT_SYNC_PART_START	Synchronization of a partition has begun.	Integer Values: [0] Partition state [1] Replication type String Values: [0] DN of associated partition entry

	Event Type	Description	Data Returned
81	EVT_SYNC_PART_END	Synchronization of a partition has finished.	Integer Values: [0] 1 indicates all processed, 0 indicates not all processed String Values: [0] DN of the associated partition entry
82	EVT_MOVE_TREE_START	A Move Subtree operation has started.	String Values: [0] DN of the root of the subtree to be moved [1] DN of the destination parent entry [2] DN of the server the operation is starting with
83	EVT_MOVE_TREE_END	A Move Subtree operation has finished.	String Values: [0] DN of the root of the moved subtree [1] DN o fthe server the operation started from
86	EVT_JOIN_DONE	A Join Partitions operation has completed.	String Values: [0] DN of the parent partition root entry [1] DN of the child partition root entry
87	EVT_PARTITION_LOCKED	A partition has been locked.	String Values: [0] DN of the locked partition
88	EVT_PARTITION_UNLOCKED	A partition has been unlocked.	String Values: [0] DN of the unlocked partition
89	EVT_SCHEMA_SYNC	The schema has been synchronized.	Integer Values: [0] 1 indicates all changes processed, 0 indicates not all changes processed
90	EVT_NAME_COLLISION	A name collision (two entries with the same name) has occurred.	String Values: [0] DN of the original entry [1] DN of the duplicate entry
91	EVT_NLM_LOADED	An NLM™ has been loaded.	Integer Values: [0] Module handle of the loaded NLM.

	Event Type	Description	Data Returned
96	EVT_SERVER_RENAME	A server has been renamed.	String Values: [0] New server name
97	EVT_SYNTHETIC_TIME	To bring eDirectory servers into synchronization, synthetic time has been invoked.	Integer Values: [0] Number of time stamps requested String Values: [0] DN of the root entry of the parition issuing the time stamp [1] DN of the partiton
99	EVT_DSA_READ	A Read operation has been performed on an entry.	String Values: [0] DN of read entry
100	EVT_LOGIN	A user has logged in.	Integer Values: [0] 0 if a non-null password was used, 1 if a null password was used [1] 0 if a bindery login was used, -1 if an eDirectory login was performed String Values: [0] DN of the parent [1] DN of the entry
101	EVT_CHGPASS	A user's password has changed.	String Values: [0] DN of the parent entry of the changed entry [1] DN of the entry whose password was changed
102	EVT_LOGOUT	A user has logged out.	String Values: [0] DN of the parent entry of entry that logged out [1] DN of the entry that logged out
103	EVT_ADD_REPLICA	A replica of a partition has been added to a server.	Integer Values: [0] Replica type String Values: [0] DN of the root entry of the partition [1] DN of the server entry [2] Name of the server

	Event Type	Description	Data Returned
104	EVT_REMOVE_REPLICA	A replica of a partition has been removed from a server.	String Values: [0] DN of the root entry of the partition [1] DN of the server entry [2] Name of the server
105	EVT_SPLIT_PARTITION	A partition has been split.	String Values: [0] DN of the root entry of the parent partition [1] DN of the root entry of the new partition [2] Name of the new partition entry
106	EVT_JOIN_PARTITIONS	A parent partition has been joined with a child partition.	String Values: [0] DN of the root entry of the parent partition [1] DN of the root entry of the child partition
107	EVT_CHANGE_REPLICA_TYPE	A partition replica's type has been changed.	Integer Values: [0] Old replica type [1] New replica type String Values: [0] DN of the partition's root entry [1] DN of the target server's entry
108	EVT_REMOVE_ENTRY	An entry has been removed beneath a container.	String Values: [0] DN of the container entry [1] DN of the deleted entry
109	EVT_ABORT_PARTITION_OP	A partition operation has been aborted.	String Values: [0] DN of the partition's parent entry [1] DN of the partition entry
110	EVT_RECV_REPLICA_UPDATES	A replica has received an update during synchronization.	String Values: [0] DN of the replica's root entry
111	EVT_REPAIR_TIMESTAMPS	A replica's time stamps have been repaired.	String Values: [0] DN of the replica's root entry
112	EVT_SEND_REPLICA_UPDATES	A replica has sent an update during synchronization.	String Values: [0] DN of the replica's root entry

	Event Type	Description	Data Returned
113	EVT_VERIFY_PASS	A password has been verified.	String Values: [0] DN of the entry's parent [0] DN of the entry
114	EVT_BACKUP_ENTRY	An entry has been backed up.	String Values: [0] Backed-up entry's DN
115	EVT_RESTORE_ENTRY	An entry has been restored.	String Values: [0] DN of the entry's parent [1] RDN of the entry
116	EVT_DEFINE_ATTR_DEF	An attribute definition has been added to the schema.	String Values: [0] Attribute's name
117	EVT_REMOVE_ATTR_DEF	An attribute definition has been removed from the schema.	String Values: [0] Attribute name
118	EVT_REMOVE_CLASS_DEF	A class definition has been removed from the schema.	String Values: [0] Class name
119	EVT_DEFINE_CLASS_DEF	A class definition has been added to the schema.	String Values: [0] Class name
120	EVT_MODIFY_CLASS_DEF	A class definition has been modified.	String Values: [0] Class name
121	EVT_RESET_DS_COUNTERS	The internal eDirectory counters have been reset.	String Values: [0] DN of the server entry
122	EVT_REMOVE_ENTRY_DIR	A file directory associated with an entry has been removed.	String Values: [0] DN of the entry's parent [1] DN of the entry
123	EVT_COMPARE_ATTR_VALUE	A Compare operation has been performed on an attribute.	String Values: [0] DN of the entry's parent [1] DN of the entry [2] Attribute name
124	EVT_STREAM	A stream attribute has been opened or closed.	Integer Values: [0] 0 if the stream was opened, 1 if the stream was closed [1] requested rights (only present if the stream was opened) String Values: [0] DN of the entry [1] Attribute name

	Event Type	Description	Data Returned
125	EVT_LIST_SUBORDINATES	A List Subordinate Entries operation has been performed on a container object.	String Values: [0] DN of the entry's parent [1] DN of the entry
126	EVT_LIST_CONT_CLASSES	A List Containable Classes operation has been performed on an entry.	String Values: [0] DN of the entry's parent [1] DN of the entry
127	EVT_INSPECT_ENTRY	An Inspect Entry operation has been performed on an entry.	String Values: [0] DN of the entry's parent [1] DN of the entry
128	EVT_RESEND_ENTRY	A Resend Entry operation has been performed on an entry.	String Values: [0] DN of the entry's parent [1] DN of the entry
129	EVT_MUTATE_ENTRY	A Mutate Entry operation has been performed on an entry.	String Values: [0] DN of the entry [1] OID of the new class [2] Name of the new class
130	EVT_MERGE_ENTRIES	Two entries have been merged.	String Values: [0] DN of the parent of the winner entry [1] DN of the winner entry [2] DN of the loser entry
131	EVT_MERGE_TREE	Two eDirectory trees have been merged.	String Values: [0] DN of the root entry
132	EVT_CREATE_SUBREF	A subordinate reference has been created.	String Values: [0] subordinate reference ID
133	EVT_LIST_PARTITIONS	A List Partitions operation has been performed.	String Values: [0] DN of the partitions root entry
134	EVT_READ_ATTR	An entry’s attributes have been read.	String Values: [0] DN of the entry [1] Attribute's name
135	EVT_READ_REFERENCES	The references on a given object have been read.	String Values: [0] DN of the entry
136	EVT_UPDATE_REPLICA	An Update Replica operation has been performed on a partition replica.	String Values: [0] DN of the partition's root entry [1] DN of the partition entry

	Event Type	Description	Data Returned
137	EVT_START_UPDATE_REPLICA	A Start Update Replica operation has been performed on a partition replica.	String Values: [0] DN of the partition's root entry
138	EVT_END_UPDATE_REPLICA	An End Update Replica operation has been performed on a partition replica.	String Values: [0] DN of the partition's root entry
139	EVT_SYNC_PARTITION	A Synchronize Partition operation has been performed on a partition replica.	String Values: [0] DN of the partition's root entry
141	EVT_CREATE_BACKLINK	A backlink has been created.	Integer Values: [0] Remote entry ID String Values: [0] DN of the server entry making the request [1] DN of the local entry
142	EVT_CHECK_CONSOLE_OPERATOR	An object has been checked for Console Operator rights.	Integer Values: [0] 0 if the entry does not have console rights, 1 if it does String Values: [0] DN of server entry [1] DN of entry being checked
143	EVT_CHANGE_TREE_NAME	The tree name has been changed.	String Values: [0] New tree name
144	EVT_START_JOIN	A Start Join operation has been performed.	String Values: [0] DN of the parent partition's root entry [1] DN of the child partition's root entry
145	EVT_ABORT_JOIN	A Join operation has been aborted.	String Values: [0] DN of the parent partition root [1] DN of the child partition root
146	EVT_UPDATE_SCHEMA	An Update Schema operation has been performed.	String Values: [0] DN of the server entry
147	EVT_START_UPDATE_SCHEMA	A Start Update Schema operation has been performed.	String Values: [0] Name of the Tree root [1] DN of the server entry

	Event Type	Description	Data Returned
148	EVT_END_UPDATE_SCHEMA	An End Update Schema operation has been performed.	String Values: New: [0] Name of the Tree root [1] DN of the server entry
149	EVT_MOVE_TREE	A Move Tree operation has been performed.	Integer Values: [0] Type of string. String Values: [0] DN of the source parent [1] DN of the destination parent [2] If integer[0] = 0, Source DN. If integer [0] = 1, new name.
151	EVT_RELOAD_DS	DS has been reloaded.	String Values: [0] DN of tree root.
151	EVT_ADD_PROPERTY	An attribute (property) has been added to an object.	Integer Values: [0] Security [1] Flags String Values: [0] DN of the entry
152	EVT_DELETE_PROPERTY	An attribute (property) has been removed from an object.	String Values: [0] DN of the entry
153	EVT_ADD_MEMBER	A member has been added to a Group object.	String Values: [0] DN of the group entry [1] DN of the new member [2] Attribute name
154	EVT_DELETE_MEMBER	A member has been deleted from a Group object.	String Values: [0] DN of the group entry [1] DN of the deleted entry [2] Attribute name
155	EVT_CHANGE_PROP_SECURITY	Security for a bindery object's property has been changed.	Integer Values: [0] New security String Values: [0] DN of the object [1] Property's name

	Event Type	Description	Data Returned
156	EVT_CHANGE_OBJ SECURITY	A bindery object’s security has been changed.	Integer Values: [0] New security String Values: [0] DN of the object's parent [1] DN of the object
159	EVT_SEARCH	A Search operation has been performed.	Integer Values: [0] Scope [1] Indicates the nodes to search [2] Info type String Values: [0] DN of the base entry
160	EVT_PARTITION_STATE_CHG	A partition’s state has changed.	Integer Values: [0] Function [1] Type [2] State String Values: [0] DN of the partition's root entry [1] DN of the partner partition entry
161	EVT_REMOVE_BACKLINK	A backlink has been removed.	String Values: [0] DN of the affected object [1] DN of the affected server entry [2] DN of the remote server entry
162	EVT_LOW_LEVEL_JOIN	A low-level join has been performed.	String Values: [0] DN of the parent's root entry [1] DN of the child partition's root entry
164	EVT_CHANGE_SECURITY_EQUALS	An object’s Security Equals attribute has been changed.	Integer Values: [0] 0=delete equivalence, 1=add equivalence String Values: [0] DN of the obect whos security has changed. [1] DN of the equivalent object.

	Event Type	Description	Data Returned
167	EVT_CRC_FAILURE	A CRC failure has occurred when fragmented NCP requests were reconstructed.	Integer Values: [0] Failure type, 0=server, 1=client [1] Server/Client CRC error count
168	EVT_ADD_ENTRY	A new object has been added under a container object.	String Values: [0] Container entry's DN [1] Entry's DN
169	EVT_MODIFY_ENTRY	An attribute has been modified on an object.	String Values: [0] Dn of the entry's parent entry [1] Entry's DN
178	EVT_MODIFY_RDN	A Modify RDN operation has been performed.	String Values: [0] Dn of the entry's parent entry [1] Entry's DN [2] Entry's former DN
181	EVT_ENTRYID_SWAP	A Swap Entry ID operation has been performed.	String Values: [0] DN of the source entry [1] DN of the destination entry
185	EVT_LOW_LEVEL_SPLIT	A low-level partition split has been performed.	String Values: [0] DN of the parent partition root entry [1] DN of the child partition root entry
188	EVT_ALLOW_LOGIN	A user has been allowed to log in.	Integer Values: [0] Flags String Values: [0] Entry's DN

6.2.5 Events Without Data

The following events do not have any associated data. When these events occur, the eventData field of the EventMonitorResponse is not present.

	Event	Description
9	EVT_SET_BINDERY_CONTEXT	
13	EVT_UPDATE_SEV	
94	EVT_LUMBER_DONE	Signals that a lumber operation has finished.
95	EVT_BACKLINK_PROC_DONE	Signals that a backlink process has finished.
98	EVT_SERVER_ADDRESS_CHANGE	Signals that a server address has changed.

	Event	Description
140	EVT_SYNC_SCHEMA	Signals that the schema has been synchronized.
150	EVT_RELOAD_DS	Signals that eDirectory has been reloaded.
163	EVT_CREATE_NAMEBASE	Signals that a directory namebase has been created.
171	EVT_OPEN_BINDERY	Signals that the bindery has been opened.
172	EVT_CLOSE_BINDERY	Signals that the bindery has been closed.
174	EVT_NEW_SCHEMA_EPOCH	Signals that a new schema epoch has been declared.
182	EVT_INSIDE_NCP_REQUEST	
187	EVT_END_NAMEBASE_TRANSACTION	
213	EVT_BEGIN_NAMEBASE_TRANSACTION	

6.2.6 Bindery Events

The following table lists the events that are associated with bindery events:

	Event	Description
10	EVT_CREATE_BINDERY_OBJECT	Signals that a bindery object has been created.
11	EVT_DELETE_BINDERY_OBJECT	Signals that a bindery object has been deleted.

6.2.7 Security Equivalence Event

The security equivalence event is indicated by the following eventType value:

	Event	Description
12	EVT_CHECK_SEV	

6.2.8 Module State Events

The following table lists the events that are associated with module state events:

	Event	Description
21	EVT_CHANGE_MODULE_STATE	

6.2.9 Network Address Events

The following table lists the events that are associated with network address events:

	Event	Description
17	EVT_REMOTE_SERVER_DOWN	
18	EVT_NCP_RETRY_EXPENDED	
158	EVT_CONNECT_TO_ADDRESS	A connection has been established with a particular address. Data returned: Integer Values: [0] taskID [1] Address type [2] Address_size String Values: [0] Address

6.2.10 Connection Change Events

The following table lists the events that are associated with connection change events:

	Event	Description
173	EVT_CHANGE_CONN_STATE	Signals that the connection state has changed
212	EVT_COMPUTE_CONN_SEV_INLINE	

6.2.11 Change Server Address

The following table lists the events that are associated with change server address events:

	Event	Description
219	EVT_CHANGE_SERVER_ADDRS	

6.3 Classes

This section explains the Event Data classes used by LDAP Event Services.

6.3.1 Entry Events

This section explains the Entry Events.

EntryEventData Class

The response data for entry events is returned as an EntryEventData class. This class consists of the read-only properties as explained in the table below.

Property	Type	Description
PerpetratorDN	LDAPDN	Specifies the DN of the entry that caused the event.
Entry	LDAPDN	Specifies the DN of the entry that was acted upon.
Class	OCTET STRING	Specifies the class OID of the object that was acted upon.
TimeStamp	DSETimestamp	Specifies the Time of Creation of this entry.
Verb	INTEGER	Specifies the action that caused the event to occur.
Flags	INTEGER	
NewDN	OCTET STRING	Specifies the new DN of the entry that was acted upon.

DSETimestamp Class

The class represents the time stamp data structure for eDirectory Events Notification. It contains the time (in seconds), replica number and event type.

Property	Type	Description
Seconds	INTEGER	Specifies in seconds when the event occurred. Zero equals 12:00 midnight, January 1, 1970,
ReplicaNumber	INTEGER	Specifies the number of the replica on which the change or event occurred.
Event	INTEGER	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 Event fields are unequal, order is determined by the Event fields. If the seconds and the event fields are equal, the time stamps are equal.

6.3.2 Value Events

This section explains the Value Events.

ValueEventData Class

The response data for value events is returned as an ValueEventData class. This class consists of the read-only properties as explained in the table below.

Property	Type	Description
PerpetratorDN	LDAPDN	Specifies the DN of the entry that caused the event.
Entry	LDAPDN	Specifies the DN of the entry that was acted upon.

Property	Type	Description
Attribute	OCTET STRING	Specifies the attribute OID of the attribute that was acted upon.
Syntax	OCTET STRING	Specifies the Syntax OID of the entry that was acted upon.
ClassId	OCTET STRING	Specifies the class OID of the object that was acted upon.
TimeStamp	DSETimestamp	Specifies a TimeStamp.
Data	OCTET STRING	Specifies the information that further identifies the changes that were made.
Verb	INTEGER	Specifies the action that caused the event to occur.

6.3.3 Debug Events

This section explains the different types of Debug Events.

DebugEventData Class

The response data for debug event is returned as a DebugEventData class. This class consists of the read-only properties as explained in the table below.

Property	Type	Description
DSTime	INTEGER	Specifies the time the event occurred as the number of seconds elapsed since midnight (00:00:00), January 1, 1970, coordinated universal time, according to the system clock.
MilliSeconds	INTEGER	The millisecond portion of the time the event occurred.
PerpetratorDN	LDAPDN	The DN of the object that caused this event.
FormatString	String	The format string used to create the string printed in the DS Trace utility. The format string describes the string that is displayed by the DS Trace utility. It contains literal characters as well as format characters that serve as place holder for parameter values. See the remarks for a list of valid format characters.
Verb	INTEGER	The ID of the ds verb that was executing when the event occurred.
ParameterCount	INTEGER	Number of elements in the parameters array.
Parameters	Array of DebugParameter	A list of DebugParameter class. The parameters are in the same order as the parameter characters in the format string.

DebugParameter Class

This class contains the debug parameters associated with debug events.

Property	Type	Design
DebugType	INTEGER	An integer that indicates the type of the parameter. See the Table Below.
Data	Object	Specifies different type of Object depending on Type.

Values of Type

Value of Type	Data Class contained as Data property	Description
DebugParamType.ENTRYID	INTEGER	Contains an Integer Object.
DebugParamType.STRING	String	Contains an UTF-8 encoded string value of the parameter
DebugParamType.BINARY	byte[]	Contains an byte array as data.
DebugParamType.INTEGER	INTEGER	Contains an Integer Object.
DebugParamType.ADDRESS	ReferralAddress	Contains the Network Address contained in a ReferralAddress class.
DebugParamType.TIMESTAMP	DSETimeStamp	Specifies an TimeStamp as a DSETimeStamp class.
DebugParamType.TIMEVECTOR	List of DSETimeStamp	Specifies a List of DSETimeStamp.

Remarks

The FormatString parameter is formatted according to the following:

```
%[flags][width][.precision][L,l,h,!]type
```

Element	Description
flags	-, +, #, 0
width	An optional integer indicating the width of the displayed value
precision	An optional integer indicating the precision of the displayed value
L, l, h, !	A character indication the size of the parameter, one of the following values: • L: DOUBLE_FLAG • l: LONG_FLAG • h: SHORT_FLAG • !: I64_FLAG

Element	Description
type	A character indicating the data type of the parameter, one of the following values: • C: color (no associated parameter) • t: current time (no associated parameter) • s: string, EVT_TAG_DB_STRING • a: network address • U: string, EVT_TAG_DB_STRING • T: time stamp • V: time stamp vector • S: string, EVT_TAG_DB_STRING • D: binary data • x: hex integer, EVT_TAG_DB_INTEGER • v: verb number, EVT_TAG_DB_INTEGER • o: octal integer, EVT_TAG_DB_INTEGER • e: error code value, EVT_TAG_DB_INTEGER • d: normal decimal integer, EVT_TAG_DB_INTEGER • c: single character, EVT_TAG_DB_INTEGER • p: raw memory pointer, EVT_TAG_DB_INTEGER • X: HEX integer, EVT_TAG_DB_INTEGER • E: error code value, EVT_TAG_DB_INTEGER

ReferralAddress Class

This class is used to store the network address.

Property	Type	Description
AddressType	INTEGER	An integer value indicating the address type.
Address	String	The actual address value.

6.3.4 General DS Events

This section explains different types of the General DS Events.

GeneralDSEventData Class

The response data for general DS events is returned as a GeneralDSEventData class. This class consists of the read-only properties as explained in the table below.

Property	Type	Description
DSTime	INTEGER	Specifies the time in milliseconds when the event occurred.
MilliSeconds	INTEGER	

Property	Type	Description
Verb	INTEGER	Specifies the action that caused the event to occur.
CurrentProcess	INTEGER	Specifies the process that was running when the event occurred.
PerpetratorDN	LDAPDN	Specifies the DN of the entry that caused the event.
IntegerValues	Array of integer	Contains event data determined by the event type
StringValues	Array of string	Contains event data determined by the event type.

Events Without Data

In the case of events without data , the response data is returned as an null object.

6.3.5 Bindery Events

This section explains the Bindery Data.

BinderyObjectEventData Class

The response data for bindery events is returned as a BinderyObjectEventData class. This class consists of the read-only properties as explained in the table below.

Property	Type	Description
EntryDN	LDAPDN	Specifies the DN of the Directory entry that is being created to represent the bindery object.
Type	INTEGER	Specifies the bindery object type.
EmuObjFlags	INTEGER	Specifies the bindery object flags.
Security	INTEGER	Specifies the bindery object security.
Name	LDAPString	Specifies the name of the bindery object.

6.3.6 SecurityEquivalence Event

This section explains the SecurityEquivalence Event.

SecurityEquivalenceEventData Class

The response data for SecurityEquivalence events is returned as a SecurityEquivalenceEventData class. This class consists of the read-only properties as explained in the table below.

Property	Type	Description
EntryDN	LDAPDN	Specifies the DN of the Directory object whose Security Equivalence Vector (SEV) is being checked.
RetryCount	INTEGER	Specifies the number of retries.
ValueDN	LDAPDN	Specifies the DN of an object or group being checked.

Property	Type	Description
ReferralList	List of ReferralAddress	Specifies the List of referrals.
ReferralCount	INTEGER	Specifies the number of referrals in the referrals parameter.

6.3.7 Module State Events

This section explains the Module State Events.

ModuleStateEventData Class

The response data for module state events is returned as an ModuleStateEventData class. This class consists of the read-only properties explained in the table below.

Property	Type	Design
ConnectionDN	LDAPDN	Specifies the DN of the entry associated with the connection.
Flags	INTEGER	The least significant byte of the flags field contains module attribute flags. The next byte contains event subtype flags. They indicate the type of module event in progress. See the Table below for details.
Name	LDAPSTRING	Specifies the affected module name.
Description	LDAPSTRING	Specifies the name and description of the target module.
Source	LDAPSTRING	Specifies the affecting module.

The values for flags field are contained in the following table:

0x0001	DSE_MOD_HIDDEN
0x0002	DSE_MOD_SYSTEM
0x0004	DSE_MOD_ENGINE
0x0008	DSE_MOD_AUTOMATIC
0x00FF	DSE_MOD_FILE_MASK
0x0100	DSE_MOD_POSTEVENT
0x0200	DSE_MOD_AVAILABLE
0x0400	DSE_MOD_LOADING
0x0800	DSE_MOD_MODIFY
0x8000	DSE_MOD_NEGATE_BIT
0xFF00	DSE_MOD_EVENT_MASK

6.3.8 Network Address Event

This section explains the Network Address Event.

NetworkAddressEventData Class

The response data for network address events is returned as a NetworkAddressEventData class. This class consists of the read-only properties as explained in the table below.

Property	Type	Description
ValueType	INTEGER	Specifies the type of the address. Can be one of the following values: • NT_IPX • NT_IP • NT_SDLC • NT_TOKENRING_ETHERNET • NT_OSI • NT_APPLETALK • NT_COUNT
Data	OCTET STRING	An character array containing address.

Remarks

The address is stored as a binary string. This string is the literal value of the address. To display as a hexadecimal value, you must convert each 4-bit nibble to the correct character (0,1,2,3,...) For two net addresses to match, the type, length, and value of the addresses must match.

6.3.9 Connection Change Event

This section explains the Connection Change Event.

ConnectionStateEventData Class

The response data for connection change events is returned as a ConnectionStateEventData class. This class consists of the read-only properties as explained in the table below.

Property	Type	Design
ConnectionDN	LDAPDN	Specifies the DN of the entry associated with the connection.
OldFlags	INTEGER	Specifies the flag associated with the previous connection state, and is one of the flag Specified in Table below.
NewFlags	INTEGER	Specifies the flag that indicates the new connection state. Uses the same flags as oldFlags.
SourceModule	LDAPSTRING	Specifies the module that caused the connection state to change.

Flags

0x00000001	DSE_CONN_VALID

0x00000002	DSE_CONN_AUTHENTIC
0x00000004	DSE_CONN_SUPERVISOR
0x00000008	DSE_CONN_OPERATOR
0x00000010	DSE_CONN_LICENSED
0x00000020	DSE_CONN_SEV_IS_STALE
0x000000FF	DSE_CONN_OPERATIONAL_FLAGS
0x00010000	DSE_CONN_CLEAR_ON_UNLOCK
0x00020000	DSE_CONN_LOCKED
0x00040000	DSE_CONN_CLEAR_ON_EVENT
0x000F0000	DSE_CONN_SECURITY_FLAGS

Revision History

A

The following table lists all changes made to the NDK: LDAP Libraries for C# documentation:

March 2006	Added the following: • Section 2.5, "Using the LDAP Classes," on page 21 • Section 2.6, "Exception Handling," on page 23 • Section 6.1.2, "Monitoring the eDirectory Events," on page 49 Modified the examples in the following sections: • Section 3.2, "Searching the Directory," on page 28 • Section 3.3, "Creating an Entry in the Directory," on page 34 • Section 3.4, "Modifying Entry Properties," on page 35 • Section 3.5, "Renaming an Entry," on page 36 • Section 3.6, "Moving an Entry," on page 36 • Section 3.7, "Deleting an Entry," on page 37 • Section 5.1, "Supported Controls," on page 43 Fixed formatting issues.
October 2005	Added information on "SSL Integration:" on page 18.
September 2004	Added information on integrating the Mono Security Library with the LDAP libraries for C#. See Section 2.2, "How to Use LDAP Libraries for C#," on page 18.
October 2004	Added the following: • Chapter 6, "LDAP Event Services," on page 49. • Location to obtain the Novell.Directory.Ldap.dll file.
June 2004	Added section on Section 1.5, "Integrating SSL with LDAP Libraries for C#," on page 13.
February 2004	Initial version.