SeeBeyond ICAN Suite

e*Way Intelligent Adapter for LDAP User's Guide

Release 5.0.5 for Schema Run-time Environment (SRE)

Monk Version

Version 20050406011333.

Contents

Chapter 1

Introduction

This document describes how to install and configure the e*Way Intelligent Adapter for the Lightweight Directory Access Protocol (LDAP), version 3.0.

1.1 LDAP - Lightweight Directory Access Protocol

LDAP, Lightweight Directory Access Protocol, is a directory service protocol that runs over TCP/IP. LDAP provides a method for accessing different directory services based on entries. An entry is made up of a collection of attributes that have a unique identifier called a distinguished name (DN). The DN is used to refer to the entry specifically. Each of the entry's attributes has a type and one or more values. The types are most often mnemonic strings, such as "cn" for common name, or "mail" for email address. These values depend on the attribute's type. For example, a mail attribute might contain the value "jdoe@xyz.com."

1.2 Overview

The LDAP e*Way enables the e*Gate system to exchange data with online directory services. By contacting the root directory, identified by distinguished names, the LDAP e*Way is able to perform searches, enter, modify and delete information from the structure. To perform the requests, the contact distinguished name must reside in a position above the subordinate node being accessed. Access to the various distinguished names is defined within the LDAP Server.

1.2.1 Intended Reader

The reader of this guide is presumed to be a developer or system administrator with responsibility for maintaining the e*Gate system; to have expert-level knowledge of Windows and/or UNIX operations and administration; to be thoroughly familiar with *Transmission Control Protocol/Internet Protocol* (TCP/IP); and to be thoroughly familiar with Windows-style GUI operations.

1.2.2 Components

The LDAP e*Way is comprised of the following:

- **stcewgenericmonk.exe**, the executable component
- Configuration files, which the e*Way Editor uses to define configuration parameters

A complete list of installed files appears in **Table 1 on page 10**.

1.3 Supported Operating Systems

Although the LDAP e*Way components will run on the supported platforms listed below.

The LDAP e*Way is available on the following operating systems:

- Windows 2000, Windows XP, and Windows Server 2003
- HP-UX 11.0 and 11i (PA-RISC)
- Sun Solaris 8 and 9

1.4 System Requirements

To use the LDAP e*Way, you need the following:

- An e*Gate Participating Host.
- A TCP/IP network connection.

1.5 External System Requirements

To enable the e*Way to communicate properly with the LDAP system, the following are required:

- Host on which the LDAP server is running
- Port location on which the LDAP server is listening

1.6 Supported Data Types

The native APIs developed for the LDAP e*Way query the result of a search performed, based on specified criteria. Each return message may consist of a number of entries or references. The current release of the LDAP e*Way will indicate the number of

references returned, but does not support the retrieval of references or searches through the use of references. Likewise, attribute values of binary data types are not supported in the current release.

Chapter 2

Installation

This chapter describes how to install the LDAP e*Way.

2.1 Installation on Windows Systems

2.1.1 Pre-installation

1. Exit all Windows programs before running the setup program, including any anti-virus applications.
2. You must have Administrator privileges to install this e*Way.

2.1.2 Installation Procedure

To install the LDAP e*Way on a Windows

1. Log in as an Administrator on the workstation on which you want to install the e*Way.
2. Insert the e*Way installation CD-ROM into the CD-ROM drive.
3. If the CD-ROM drive's "Autorun" feature is enabled, the setup application should launch automatically; skip ahead to step 4. Otherwise, use the Windows Explorer or the Control Panel's **Add/Remove Applications** feature to launch the file **setup.exe** on the CD-ROM drive.
4. The InstallShield setup application will launch. Follow the on-screen instructions to install the e*Way.

Note: *Be sure to install the e*Way files in the suggested "client" installation directory. The installation utility detects and suggests the appropriate installation directory.* ***Unless you are directed to do so by SeeBeyond support personnel, do not change the suggested "installation directory" setting.***

2.2 Installation on UNIX Systems

2.2.1 Pre-installation

1 You do not require root privileges to install this e*Way. Log in under the user name that you wish to own the e*Way files. Be sure that this user has sufficient privilege to create files in the e*Gate directory tree.

2.2.2 Installation Procedure

To install the LDAP e*Way on UNIX

1 Log in as root on the workstation containing the CD-ROM drive, and insert the CD-ROM into the drive.

2 If necessary, mount the CD-ROM drive.

3 At the shell prompt, type

cd /cdrom

4 Start the installation script by typing:

setup.sh

5 A menu of options will appear. Select the "install e*Way" option. Then, follow any additional on-screen directions.

Note: *Be sure to install the e*Way files in the suggested "client" installation directory. The installation utility detects and suggests the appropriate installation directory.* ***Unless you are directed to do so by SeeBeyond support personnel, do not change the suggested "installation directory" setting.***

2.3 Files/Directories Created by the Installation

The LDAP e*Way installation process will install the following files within the e*Gate directory tree. Files will be installed within the "egate\client" tree on the Participating Host and committed to the "default" schema on the Registry Host.

Table 1 Files created by the installation

e*Gate Directory	File(s)
bin\	stcewgenericmonk.exe

Table 1 Files created by the installation

e*Gate Directory	File(s)
bin\ (Windows) (Solaris) (Solaris) (HP) (HP)	stc_monkldap.dll nsldap32v32.dll libldap.so liblber.so liblber.sl libldap.sl
configs\stcewgenericmonk\	stcewldap.def
monk_library\ewldap\	ldapsearchentries.ssc ldapentries.ssc ldapwriteentries.ssc display_ldapentries.tsc ldap-verify.monk ldap-startup.monk ldap-shutdown.monk ldap-outgoing.monk ldap-outgoing-write.monk ldap-notify.monk ldap-nack.monk ldap-init.monk ldap-extconnect.monk ldap-exchange.monk ldap-ack.monk

Chapter 3

Configuration

This chapter describes how to configure the LDAP e*Way.

3.1 e*Way Configuration Parameters

e*Way configuration parameters are set using the e*Way Editor.

To change e*Way configuration parameters:

1. In the Schema Designer's Component editor, select the e*Way you want to configure and display its properties.
2. Under **Configuration File**, click **New** to create a new file, **Find** to select an existing configuration file, or **Edit** to edit the currently selected file.
3. In the **Additional Command Line Arguments** box, type any additional command line arguments that the e*Way may require, taking care to insert them *at the end* of the existing command-line string. Be careful not to change any of the default arguments unless you have a specific need to do so.

For more information about how to use the e*Way Editor, see the e*Way Editor's online Help or the *e*Gate Integrator User's Guide*.

The e*Way's configuration parameters are organized into the following sections:

- General settings
- Communication Setup
- Monk Configuration
- LDAP Configuration

3.1.1 General Settings

The General Settings control basic operational parameters.

Journal File Name

Description

Specifies the name of the journal file.

Required Values

A valid filename, optionally including an absolute path (for example, **c:\temp\filename.txt**). If an absolute path is not specified, the file will be stored in the e*Gate "SystemData" directory. See the *e*Gate Integrator System Administration and Operations Guide* for more information about file locations.

Additional Information

An Event will be journaled for the following conditions:

- When the number of resends is exceeded (see Max Resends Per Message below)
- When its receipt is due to an external error, but Forward External Errors is set to **No**. (See **"Forward External Errors" on page 13** for more information.)

Max Resends Per Message

Description

Specifies the number of times the e*Way will attempt to resend a message (Event) to the external system after receiving an error. When this maximum is reached, the message is considered "Failed" and is written to the journal file.

Required Values

An integer between 1 and 1,024. The default is 5.

Max Failed Messages

Description

Specifies the maximum number of failed messages (Events) that the e*Way will allow. When the specified number of failed messages is reached, the e*Way will shut down and exit.

Required Values

An integer between 1 and 1,024. The default is 3.

Forward External Errors

Description

Selects whether error messages that begin with the string "DATAERR" that are received from the external system will be queued to the e*Way's configured queue. See **"Exchange Data with External Function" on page 27** for more information.

Required Values

Yes or **No**. The default value, **No**, specifies that error messages will not be forwarded.

See **"Schedule-driven data exchange functions" on page 22** for information about how the e*Way uses this function.

3.1.2 Communication Setup

The Communication Setup parameters control the schedule by which the e*Way obtains data from the external system.

Note: *The schedule you set using the e*Way's properties in the Schema Designer controls when the e*Way executable will run. The schedule you set within the parameters discussed in this section (using the e*Way Editor) determines when data will be exchanged. Be sure you set the "exchange data" schedule to fall within the "run the executable" schedule.*

Start Exchange Data Schedule

Description

Establishes the schedule to invoke the e*Way's **Exchange Data with External** function.

Required Values

One of the following:

- One or more specific dates/times
- A single repeating interval (such as yearly, weekly, monthly, daily, or every *n* seconds).

Also required: If you set a schedule using this parameter, you must also define all three of the following:

- Exchange Data With External Function
- Positive Acknowledgment Function
- Negative Acknowledgment Function

If you do not do so, the e*Way will terminate execution when the schedule attempts to start.

Additional Information

When the schedule starts, the e*Way determines whether it is waiting to send an ACK or NAK to the external system (using the Positive and Negative Acknowledgment functions) and whether the connection to the external system is active. If no ACK/NAK is pending and the connection is active, the e*Way immediately executes the **Exchange Data with External** function. Thereafter, the **Exchange Data with External** function will be called according to the **Exchange Data Interval** parameter until the **Stop Exchange Data Schedule** time is reached.

See **"Exchange Data with External Function" on page 27**, **"Exchange Data Interval" on page 15**, and **"Stop Exchange Data Schedule" on page 14** for more information.

Stop Exchange Data Schedule

Description

Establishes the schedule to stop data exchange.

Required Values

One of the following:

- One or more specific dates/times
- A single repeating interval (such as yearly, weekly, monthly, daily, or every *n* seconds).

Exchange Data Interval

Description

Specifies the number of seconds the e*Way waits between calls to the **Exchange Data with External** function during scheduled data exchanges.

Required Values

An integer between 0 and 86,400. The default is 120.

Additional Information

If **Zero Wait Between Successful Exchanges** is set to **Yes** and the **Exchange Data with External Function** returns data, The **Exchange Data Interval** setting will be ignored and the e*Way will invoke the **Exchange Data with External Function** immediately.

If this parameter is set to zero, there will be no exchange data schedule set and the **Exchange Data with External Function** will never be called.

See **"Down Timeout" on page 15** and **"Stop Exchange Data Schedule" on page 14** for more information about the data-exchange schedule.

Down Timeout

Description

Specifies the number of seconds that the e*Way will wait between calls to the **External Connection Establishment** function. See **"External Connection Establishment Function" on page 28** for more information.

Required Values

An integer between 1 and 86,400. The default is 15.

Up Timeout

Description

Specifies the number of seconds the e*Way will wait between calls to the **External Connection Verification** function. See **"External Connection Verification Function" on page 29** for more information.

Required Values

An integer between 1 and 86,400. The default is 15.

Resend Timeout

Description

Specifies the number of seconds the e*Way will wait between attempts to resend a message (Event) to the external system, after receiving an error message from the external system.

Required Values

An integer between 1 and 86,400. The default is 10.

Zero Wait Between Successful Exchanges

Description

Selects whether to initiate data exchange after the **Exchange Data Interval** or immediately after a successful previous exchange.

Required Values

Yes or **No**. If this parameter is set to **Yes**, the e*Way will immediately invoke the **Exchange Data with External** function if the previous exchange function returned data. If this parameter is set to **No**, the e*Way will always wait the number of seconds specified by **Exchange Data Interval** between invocations of the **Exchange Data with External** function. The default is **No**.

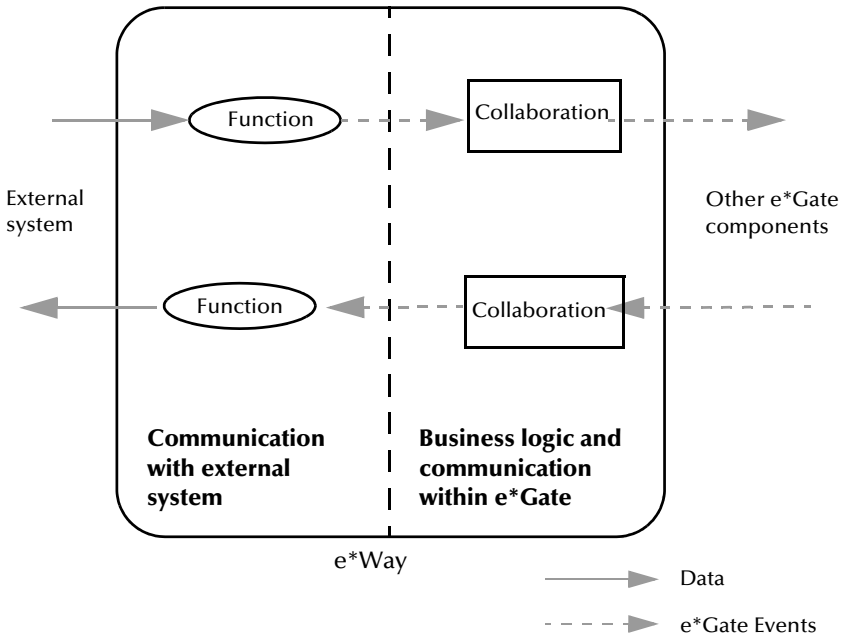
See **"Exchange Data with External Function" on page 27** for more information.

3.1.3 Monk Configuration

The parameters in this section help you set up the information required by the e*Way to utilize Monk for communication with the external system.

Conceptually, an e*Way is divided into two halves. One half of the e*Way (shown on the left in Figure 1 below) handles communication with the external system; the other half manages the Collaborations that process data and subscribe or publish to other e*Gate components.

Figure 1 e*Way internal architecture



The "communications half" of the e*Way uses Monk functions to start and stop scheduled operations, exchange data with the external system, package data as e*Gate "Events" and send those Events to Collaborations, and manage the connection between the e*Way and the external system. The **Monk Configuration** options discussed in this section control the Monk environment and define the Monk functions used to perform these basic e*Way operations. You can create and modify these functions using the SeeBeyond Collaboration Rules Editor or a text editor (such as **Notepad**, or UNIX **vi**).

The "communications half" of the e*Way is single-threaded. Functions run serially, and only one function can be executed at a time. The "business logic" side of the e*Way is multi-threaded, with one executable thread for each Collaboration. Each thread maintains its own Monk environment; therefore, information such as variables, functions, path information, and so on cannot be shared between threads.

Operational Details

The Monk functions in the "communications half" of the e*Way fall into the following groups:

Type of Operation	Name
Initialization	**Startup Function** on page 26 (also see **Monk Environment Initialization File** on page 25)

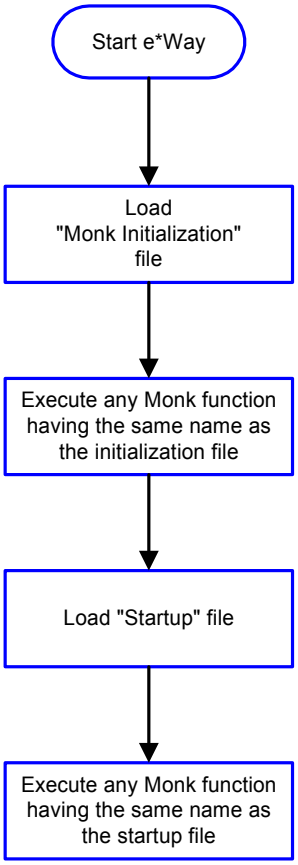
Type of Operation	Name
Connection	**External Connection Establishment Function** on page 28 **External Connection Verification Function** on page 29 **External Connection Shutdown Function** on page 29
Schedule-driven data exchange	**Exchange Data with External Function** on page 27 **Positive Acknowledgment Function** on page 29 **Negative Acknowledgment Function** on page 30
Shutdown	**Shutdown Command Notification Function** on page 31
Event-driven data exchange	**Process Outgoing Message Function** on page 27

A series of figures on the next several pages illustrates the interaction and operation of these functions.

Initialization Functions

Figure 2 illustrates how the e*Way executes its initialization functions.

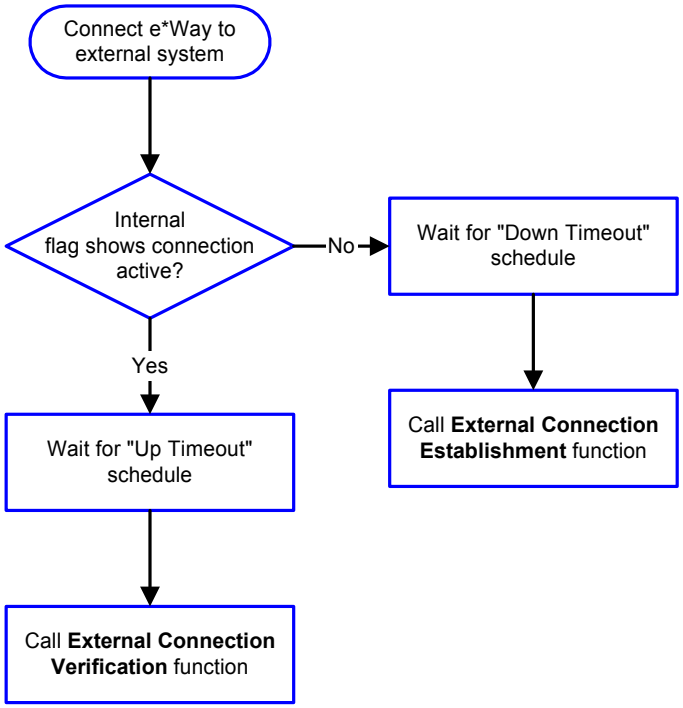
Figure 2 Initialization Functions



Connection Functions

Figure 3 illustrates how the e*Way executes the connection establishment and verification functions.

Figure 3 Connection establishment and verification functions

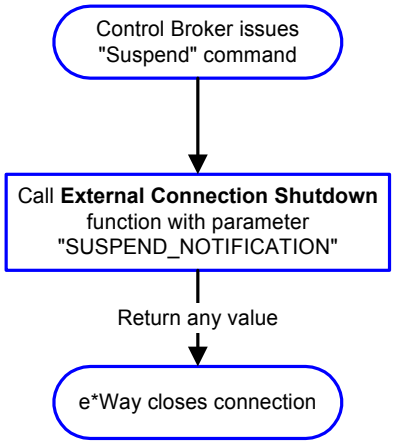


Note: *The e*Way selects the connection function based on an internal "up/down" flag rather than a poll to the external system. See* **Figure 5 on page 22** *and* **Figure 7 on page 24** *for examples of how different functions use this flag.*

User functions can manually set this flag using Monk functions. See **send-external-up** *on page 46 and* **send-external-down** *on page 47 for more information.*

Figure 4 illustrates how the e*Way executes its "connection shutdown" function.

Figure 4 Connection shutdown function



Schedule-driven Data Exchange Functions

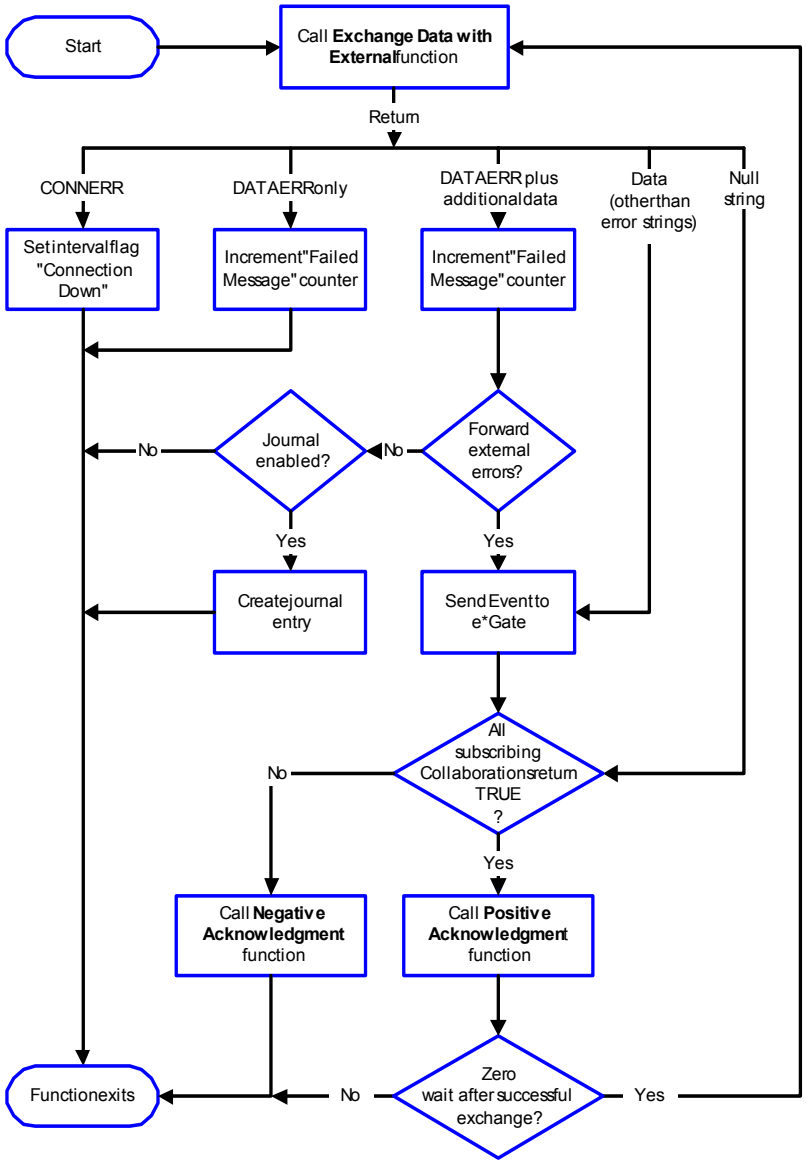
Figure 5 (on the next page) illustrates how the e*Way performs schedule-driven data exchange using the **Exchange Data with External Function**. The **Positive Acknowledgement Function** and **Negative Acknowledgement Function** are also called during this process.

"Start" can occur in any of the following ways:

- The "Start Data Exchange" time occurs
- Periodically during data-exchange schedule (after "Start Data Exchange" time, but before "Stop Data Exchange" time), as set by the Exchange Data Interval
- The **start-schedule** Monk function is called

After the function exits, the e*Way waits for the next "start schedule" time or command.

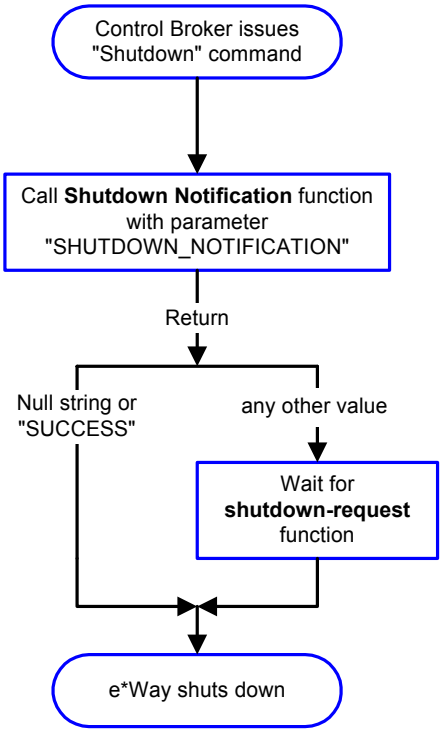
Figure 5 Schedule-driven data exchange functions



Shutdown Functions

Figure 6 illustrates how the e*Way implements the shutdown request function.

Figure 6 Shutdown functions



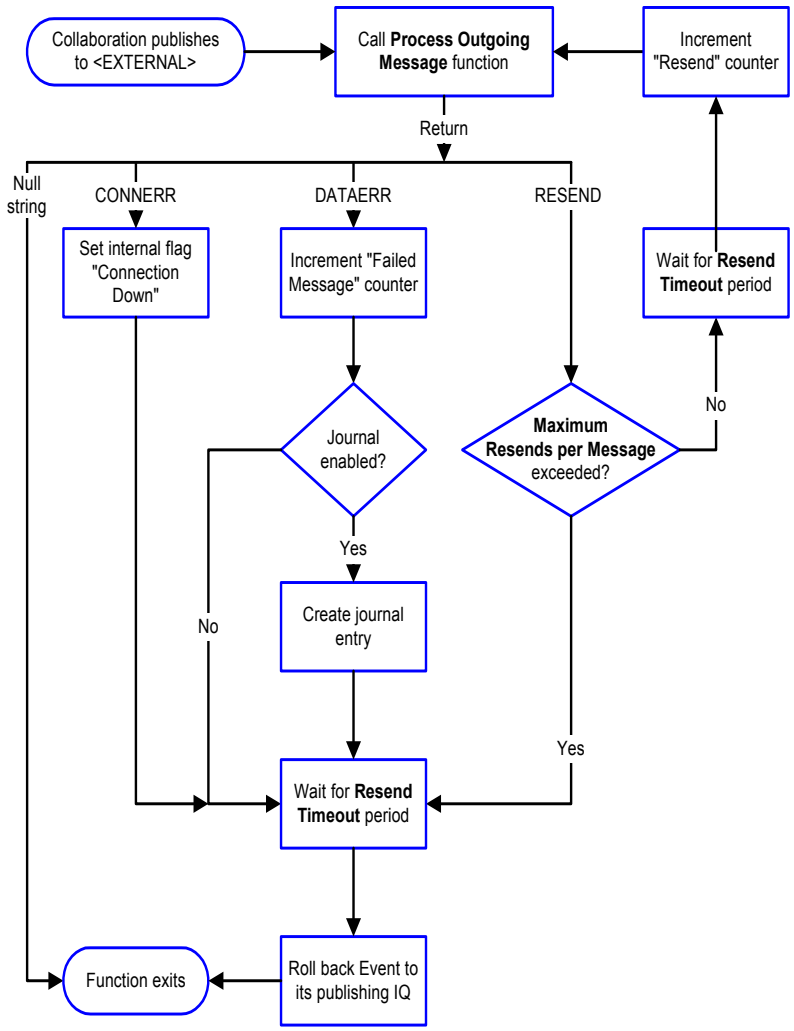
Event-driven Data Exchange Functions

Figure 7 on the next page illustrates event-driven data-exchange using the **Process Outgoing Message Function**.

Every two minutes, the e*Way checks the "Failed Message" counter against the value specified by the **Max Failed Messages** parameter. When the "Failed Message" counter exceeds the specified maximum value, the e*Way logs an error and shuts down.

After the function exits, the e*Way waits for the next outgoing Event.

Figure 7 Event-driven data-exchange functions



How to Specify Function Names or File Names

Parameters that require the name of a Monk function will accept either a function name or a file name. If you specify a file name, be sure that the file has one of the following extensions:

- .monk
- .tsc
- .dsc

Additional Path

Description

Specifies a path to be appended to the "load path," the path Monk uses to locate files and data (set internally within Monk). The directory specified in Additional Path will be searched after the default load paths.

Required Values

A pathname, or a series of paths separated by semicolons. This parameter is optional and may be left blank.

Additional information

The default load paths are determined by the "bin" and "Shared Data" settings in the .egate.store file. See the *e*Gate Integrator System Administration and Operations Guide* for more information about this file.

To specify multiple directories, manually enter the directory names rather than selecting them with the "file selection" button. Directory names must be separated with semicolons, and you can mix absolute paths with relative e*Gate paths. For example:

```
monk_scripts\my_dir;c:\my_directory
```

The internal e*Way function that loads this path information is called only once, when the e*Way first starts up.

Auxiliary Library Directories

Description

Specifies a path to auxiliary library directories. Any **.monk** files found within those directories will automatically be loaded into the e*Way's Monk environment. This parameter is optional and may be left blank.

Required Values

A pathname, or a series of paths separated by semicolons. The default is **monk_library/ewldap**.

Additional information

To specify multiple directories, manually enter the directory names rather than selecting them with the "file selection" button. Directory names must be separated with semicolons, and you can mix absolute paths with relative e*Gate paths. For example:

```
monk_scripts\my_dir;c:\my_directory
```

The internal e*Way function that loads this path information is called only once, when the e*Way first starts up.

This parameter is optional and may be left blank.

Monk Environment Initialization File

Specifies a file that contains environment initialization functions, which will be loaded after the auxiliary library directories are loaded. Use this feature to initialize the

e*Way's Monk environment (for example, to define Monk variables that are used by the e*Way's function scripts).

Required Values

A filename within the "load path", or filename plus path information (relative or absolute). If path information is specified, that path will be appended to the "load path." See **"Additional Path" on page 25** for more information about the "load path."The default is **monk_library/ewldap/ldap-init.monk**.

Additional information

Any environment-initialization functions called by this file accept no input, and must return a string. The e*Way will load this file and try to invoke a function of the same base name as the file name (for example, for a file named **my-init.monk**, the e*Way would attempt to execute the function **my-init**).

Typically, it is a good practice to initialize any global Monk variables that may be used by any other Monk Extension scripts.

The internal function that loads this file is called once when the e*Way first starts up (see **Figure 2 on page 19**).

Startup Function

Description

Specifies a Monk function that the e*Way will load and invoke upon startup or whenever the e*Way's configuration is reloaded. This function should be used to initialize the external system before data exchange starts.

Required Values

The name of a Monk function, or the name of a file (optionally including path information) containing a Monk function. This parameter is optional and may be left blank. The default is **ldap-startup** on page 63.

Additional information

The function accepts no input, and must return a string.

The string "FAILURE" indicates that the function failed; any other string (including a null string) indicates success.

This function will be called after the e*Way loads the specified "Monk Environment Initialization file" and any files within the specified **Auxiliary Directories**.

The e*Way will load this file and try to invoke a function of the same base name as the file name (see **Figure 2 on page 19**). For example, for a file named **my-startup.monk**, the e*Way would attempt to execute the function **my-startup**.

Process Outgoing Message Function

Description

Specifies the Monk function responsible for sending outgoing messages (Events) from the e*Way to the external system. This function is event-driven (unlike the Exchange Data with External function, which is schedule-driven).

Required Values

The name of a Monk function, or the name of a file (optionally including path information) containing a Monk function. ***You may not leave this field blank.*** The default is **ldap-outgoing** on page 54.

Additional Information

The function requires a non-null string as input (the outgoing Event to be sent) and must return a string.

The e*Way invokes this function when one of its Collaborations publishes an Event to an <EXTERNAL> destination (as specified within the Schema Designer). The function returns one of the following (see **Figure 7 on page 24** for more details):

- Null string: Indicates that the Event was published successfully to the external system.
- "RESEND": Indicates that the Event should be resent.
- "CONNERR": Indicates that there is a problem communicating with the external system.
- "DATAERR": Indicates that there is a problem with the message (Event) data itself.

If a string other than the above is returned, the e*Way will create an entry in the log file indicating that an attempt has been made to access an unsupported function.

Note: *If you wish to use* ***event-send-to-egate*** *to enqueue failed Events in a separate IQ, the e*Way must have an inbound Collaboration (with appropriate IQs) configured to process those Events. See* **event-send-to-egate** *on page 49 for more information.*

Exchange Data with External Function

Description

Specifies a Monk function that initiates the transmission of data from the external system to the e*Gate system and forwards that data as an inbound Event to one or more e*Gate Collaborations. This function is called according to a schedule (unlike the **Process Outgoing Message Function**, which is event-driven).

Required Values

The name of a Monk function, or the name of a file (optionally including path information) containing a Monk function. This parameter is optional and may be left blank. The default is **ldap-exchange** on page 53.

Additional Information

The function accepts no input and must return a string (see **Figure 5 on page 22** for more details):

- Null string: Indicates that the data exchange was completed successfully. No information will be sent into the e*Gate system.
- "CONNERR": Indicates that a problem with the connection to the external system has occurred.
- "DATAERR": Indicates that a problem with the data itself has occurred. The e*Way handles the string "DATAERR" and "DATAERR" plus additional data differently; see **Figure 5 on page 22** for more details.
- Any other string: The contents of the string are packaged as an inbound Event. The e*Way must have at least one Collaboration configured suitably to process the inbound Event, as well as any required IQs.

This function is initially triggered by the **Start Data Exchange** schedule or manually by the Monk function **start-schedule**. After the function has returned true and the data received by this function has been ACKed or NAKed (by the **Positive Acknowledgment Function** or **Negative Acknowledgment Function**, respectively), the e*Way checks the **Zero Wait Between Successful Exchanges** parameter. If this parameter is set to **Yes**, the e*Way will immediately call the **Exchange Data with External** function again; otherwise, the e*Way will not call the function until the next scheduled "start exchange" time or the schedule is manually invoked using the Monk function **start-schedule.** (see **start-schedule** on page 44 for more information.)

External Connection Establishment Function

Description

Specifies a Monk function that the e*Way will call when it has determined that the connection to the external system is down.

Required Values

The name of a Monk function, or the name of a file (optionally including path information) containing a Monk function. ***This field cannot be left blank.*** The default is **ldap-extconnect** on page 58.

Additional Information

The function accepts no input and must return a string:

- "SUCCESS" or "UP": Indicates that the connection was established successfully.
- Any other string (including the null string): Indicates that the attempt to establish the connection failed.

This function is executed according to the interval specified within the **Down Timeout** parameter, and is *only* called according to this schedule.

The **External Connection Verification** function (see below) is called when the e*Way has determined that its connection to the external system is up.

External Connection Verification Function

Description

Specifies a Monk function that the e*Way will call when its internal variables show that the connection to the external system is up.

Required Values

The name of a Monk function. This function is optional; if no **External Connection Verification** function is specified, the e*Way will execute the **External Connection Establishment** function in its place. The default is **ldap-verify** on page 64.

Additional Information

The function accepts no input and must return a string:

- "SUCCESS" or "UP": Indicates that the connection was established successfully.
- Any other string (including the null string): Indicates that the attempt to establish the connection failed.

This function is executed according to the interval specified within the **Up Timeout** parameter, and is *only* called according to this schedule.

The **External Connection Establishment** function (see above) is called when the e*Way has determined that its connection to the external system is down.

External Connection Shutdown Function

Description

Specifies a Monk function that the e*Way will call to shut down the connection to the external system.

Required Values

The name of a Monk function. This parameter is optional. The default is **ldap-shutdown** on page 62.

Additional Information

This function requires a string as input, and may return a string.

This function will only be invoked when the e*Way receives a "suspend" command from a Control Broker. When the "suspend" command is received, the e*Way will invoke this function, passing the string "SUSPEND_NOTIFICATION" as an argument.

Any return value indicates that the "suspend" command can proceed and that the connection to the external system can be broken immediately.

Positive Acknowledgment Function

Description

Specifies a Monk function that the e*Way will call when ***all*** the Collaborations to which the e*Way sent data have processed and enqueued that data successfully.

Required Values

The name of a Monk function, or the name of a file (optionally including path information) containing a Monk function. This parameter is required if the **Exchange Data with External** function is defined. The default is **ldap-ack** on page 52.

Additional Information

The function requires a non-null string as input (the Event to be sent to the external system) and must return a string:

- "CONNERR": Indicates a problem with the connection to the external system. When the connection is re-established, the Positive Acknowledgment function will be called again, with the same input data.
- Null string: The function completed execution successfully.

After the **Exchange Data with External** function returns a string that is transformed into an inbound Event, the Event is handed off to one or more Collaborations for further processing. If the Event's processing is completed successfully by *all* the Collaborations to which it was sent, the e*Way executes the Positive Acknowledgment function (otherwise, the e*Way executes the Negative Acknowledgment function).

Negative Acknowledgment Function

Description

Specifies a Monk function that the e*Way will call when the e*Way fails to process and queue Events from the external system.

Required Values

The name of a Monk function, or the name of a file (optionally including path information) containing a Monk function. This parameter is required if the **Exchange Data with External** function is defined. The default is **ldap-nack** on page 60.

Additional Information

The function requires a non-null string as input (the Event to be sent to the external system) and must return a string:

- "CONNERR": Indicates a problem with the connection to the external system. When the connection is re-established, the function will be called again.
- Null string: The function completed execution successfully.

This function is only called during the processing of inbound Events. After the **Exchange Data with External** function returns a string that is transformed into an inbound Event, the Event is handed off to one or more Collaborations for further processing. If the Event's processing is not completed successfully by *all* the Collaborations to which it was sent, the e*Way executes the Negative Acknowledgment function (otherwise, the e*Way executes the Positive Acknowledgment function).

Shutdown Command Notification Function

Description

Specifies a Monk function that will be called when the e*Way receives a "shut down" command from the Control Broker. This parameter is optional.

Required Values

The name of a Monk function.

Additional Information

When the Control Broker issues a shutdown command to the e*Way, the e*Way will call this function with the string "SHUTDOWN_NOTIFICATION" passed as a parameter.

The function accepts a string as input and must return a string:

- A null string or "SUCCESS": Indicates that the shutdown can occur immediately.
- Any other string: Indicates that shutdown must be postponed. Once postponed, shutdown will not proceed until the Monk function **shutdown-request** is executed (see **shutdown-request** on page 50).

Note: *If you postpone a shutdown using this function, be sure to use the (**shutdown-request)** function to complete the process in a timely manner.*

3.1.4 LDAP Configuration

The parameters in this section help to set up the required information for the e*Way to access the TCP/IP system.

Host

Description

Specifies the host on which the server is running.

Required Values

A string. Any valid server name.

Port

Description

Specifies the server listening for connection requests.

Required Values

An integer between 1 and 864000.

Username

Description

Specifies the username for logon authentication.

Required Values

Any valid username.

Password

Description

Specifies password associated with the username indicated.

Required Values

A string.

Timeout

Description

Specifies the number of milliseconds to wait for a response when contacting the server.

Required Values

An integer between 1 and 864000.

3.2 External Configuration Requirements

There are no configuration changes required in the external system. All necessary configuration changes can be made within e*Gate.

Chapter 4

Implementation

This chapter includes information pertinent to implementing a sample of the LDAP e*Way in a production environment.

4.1 Sample Implementation

To implement the LDAP e*Way within an e*Gate system, do the following:

- Define Event Type Definitions (ETDs) to package the data being exchanged with the external system.
- In the e*Gate Schema Designer, do the following:
  - Define Collaboration Rules to process Event data.
  - Define any IQs to which Event data will be published prior to sending it to the external system.
  - Define the e*Way component.
  - Within the e*Way component, configure Collaborations to apply the required Collaboration Rules.
- Use the e*Way Editor to set the e*Way's configuration parameters.
- Be sure that any other e*Gate components are configured as necessary to complete the schema.
- Test the schema and make any necessary corrections.

Note: *For more information about creating or modifying any component within the e*Gate Schema Designer, see the Schema Designer's Help system.*

4.1.1 Sample Event Type Definitions

The following files are loaded with the LDAP e*Way installation, under ..egate\monk_scripts\common:

- ldapentries.ssc
- ldapwriteentries.ssc
- ldapwriteentries.ssc
- out.ssc

These files are provided as sample structures for use with the **ldap-outgoing** or **ldap-outgoing-write** functions. For details, see **"ldap-outgoing" on page 54** and **"ldap-outgoing-write" on page 56**.

In lieu of using the ETD Build Tool (see below), you may use the sample files as a starting point, and make modifications in the ETD Editor as needed. You may also use these files in the Sample Schema discussed later in this chapter (see **"Creating a Schema Using ldap-outgoing" on page 35**.)

4.1.2 Creating Event Type Definitions from Sample Data

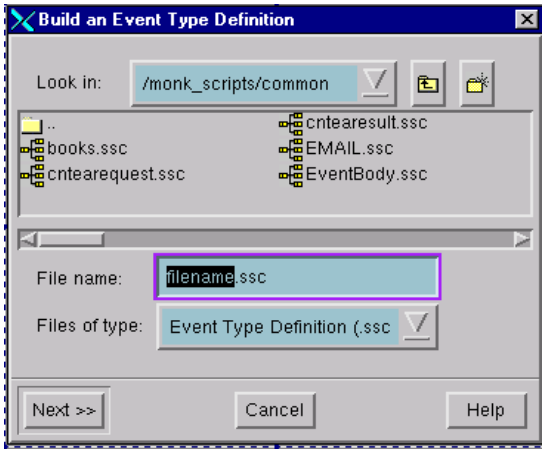
You can use the ETD Editor to create or modify any necessary Event Type Definitions. However, if you wish to base ETDs upon existing data, you can automatically create these ETDs using the ETD Build Tool.

Note: *Be sure to set the* **Default Editor** *to* **Monk**, *from the* **Options** *menu in the SeeBeyond Schema Designer.*

To create an Event Type Definition with the ETD Build Tool:

1. Launch the ETD Editor.
2. On the ETD Editor's Toolbar, click **Build**. The **Build an Event Type Definition** dialog box appears.

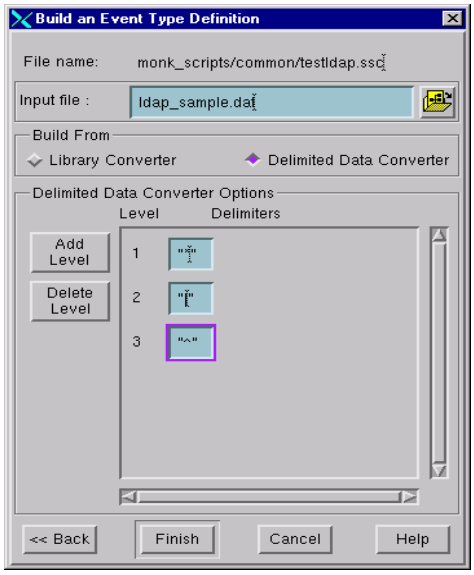
Figure 8 Build an Event Type Definition



3 In the **File name** field, type the name of the ETD file you wish to build. Do not specify any file extension—the Editor will supply the .ssc extension automatically.

4 Click **Next**. A new dialog box appears.

Figure 9 Build an Event Type Definition - Delimited Data



5 Provide the name of the Input file on which the ETD will be based.

6 Under **Build From**, select **Delimited Data Converter**.

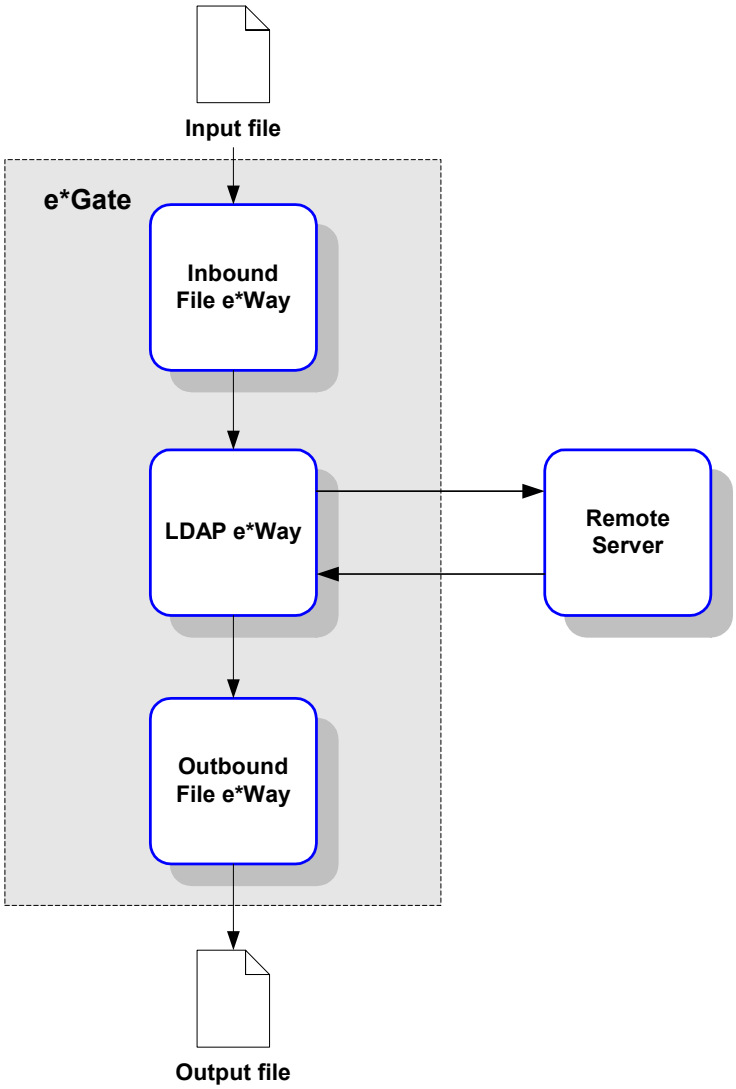
7 Add Levels and the corresponding delimiters, based on the Input file.

8 Click **Finish**. The Build tool will create the ETD.

4.1.3 Creating a Schema Using ldap-outgoing

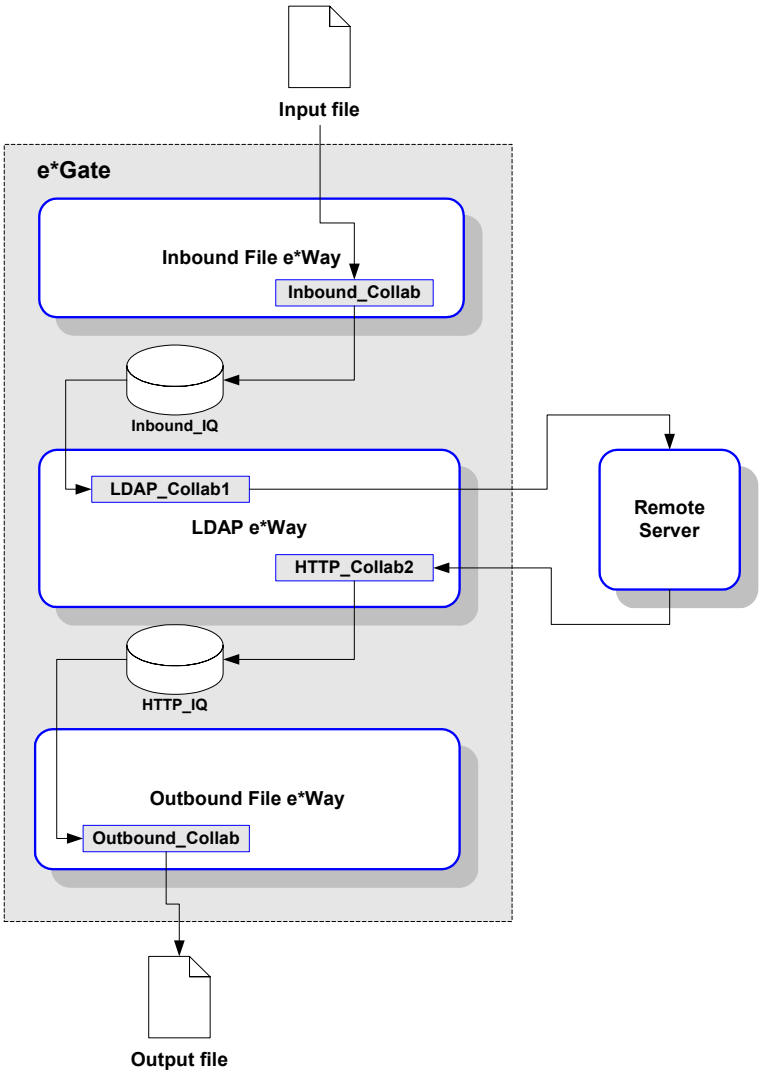
This section demonstrates how to set up a basic schema using the **ldap-outgoing** function. In this sample, data is drawn from a text file using the file e*Way and sent to an external system using the LDAP e*Way. The data returned from the external system is received by the LDAP e*Way, then forwarded to another file e*Way and stored in an output file on the local system (see **Figure 10 on page 36**).

Figure 10 Sample schema: basic architecture



This schema requires a number of components, as illustrated in **Figure 11 on page 37**.

Figure 11 Sample schema (component view)



Note: *For more information about creating or modifying any component within the e*Gate Schema Designer, see the Schema Designer's Help system.*

1 Log into the e*Gate Schema Designer and click **New** to create a new schema. Name the schema "LDAP_sample."

The Schema Designer main screen appears.

2 If the Navigator's **Components** tab is not selected already, select it now.

3 Create an Event Type named "In."

4 Display the properties of the **In** Event Type. Then, use the **Find** button, navigate to the **"common"** folder to assign the file **GenericInEvent.ssc**.

5 Create a Collaboration Rule named "Passthrough_Data."

6 Edit the Properties of this Collaboration Rule as follows:

Service	Pass Through
Subscription	**In** (the Event Type defined in Step 1 above)
Publication	**In** (Event Type defined in Step 1 above)

7 Create two IQs, named "Inbound_IQ" and "LDAP_IQ."

8 Create an e*Way named "Inbound."

9 Display the e*Way's properties. Then, use the **Find** button to assign the file **stcewfile.exe**.

The next part of the procedure requires that you launch the e*Way editor and define the file-based e*Way's properties.

1 With the e*Way's Properties page still displayed, click **New** to launch the e*Way Editor.

2 Using the e*Way Editor, make the following configuration settings:

Section	Parameter and setting
General Settings	**AllowIncoming**:Yes **AllowOutgoing**:No
Poller(inbound) Settings	**Polldirectory**: **C:\TEMP** (or other "temporary" directory) **Input File Mask**: leave unchanged

3 Save the settings, promote to run time, and exit the e*Way Editor.

4 When you return to the e*Way's Properties page, click **OK** to save all changes and return to the Schema Designer's main window.

Next, create a Collaboration for the Inbound e*Way.

1 Open the **Inbound** e*Way and create a Collaboration named "Inbound_collab".

2 Set the Collaboration's properties as follows:

Collaboration Rule	**Passthrough_Data**
Subscriptions	Event: **In** Source: **<External>**.
Publications	Event: **In** Publish to: **Inbound_IQ**.

Now that the "inbound" e*Way is completely configured, you must create an outbound LDAP e*Way.

1 Create a new e*Way component named "ldap_eway".

2 Display the e*Way's properties. Then, use the **Find** button to assign the file **stcewgenericmonk.exe**.

3 Click **New** to launch the e*Way Editor. When prompted with a list of templates, select **stcewldap**.

4 Use the e*Way Editor to define the following parameters:

Section	Parameter and Settings
General Settings	Leave all settings unchanged
Communication Setup	**Exchange Data Interval**: 0 (zero) **Zero Wait Between Successful Exchanges**: No
Monk Configuration	**Auxiliary Library Directories**: **monk_library/ewldap** **Monk Environment Initialization File**: **monk_library/ewldap/ldap-init.monk** **Startup Function**: **ldap-startup** **Process Outgoing Message Function**: **ldap-outgoing** **Exchange Data With External Function**: **ldap-exchange** **External Connection Establishment Function**: **ldap-extconnect** **External Connection Verification Function**: **ldap-verify** **External Connection Shutdown Function**: **ldap-shutdown** **Positive Acknowledgment Function**: **ldap-ack** **Negative Acknowledgment Function**: **ldap-nack** The remaining parameter may be left blank for this sample.
LDAP Configuration (Enter the appropriate information for your system. (See **"LDAP Configuration" on page 31** for more information.)	Host: Port: UserName: Encrypted Password: Timeout:

5 Save the settings, promote to runtime, and exit the e*Way Editor.

6 When you return to the e*Way's Properties page, click **OK** to save all changes and return to the Schema Designer's main window.

Next, create the Collaboration for the LDAP e*Way.

1 Select the **ldap_eway** component and create a Collaboration named "ldap_collab1".

2 Assign the following properties to the Collaboration:

Collaboration Rules	Passthrough_Data
Subscriptions	Event: In Source: Inbound_collab
Publications	Event: In Publish to: <External>

3 Create a second Collaboration for the **ldap_eway**, naming it "ldap_collab2".

4 Assign the following properties to the Collaboration:

Collaboration Rules	Passthrough_Data
Subscriptions	Event: In Source: <External>
Publications	Event: In Publish to: LDAP_IQ

Now create and configure the final e*Way component.

1. Create a new e*Way named "Outbound".
2. In its Properties Page, specify the executable file of "Outbound" as **stcewfile.exe**.
3. Display the e*Way's properties. Then, use the **Find** button to assign the file **stcewfile.exe**.
4. With the e*Way's Properties page still displayed, click **New** to launch the e*Way Editor.
5. Using the e*Way Editor, specify the following configuration settings:

Section	Parameter and setting
General Settings	**AllowIncoming**: No **AllowOutgoing**: Yes
Outbound (sender) Settings	**Output directory**: **C:\TEMP** (or other "temporary" directory) **Output File Name**: ldap_out.txt

6. Save the settings, promote to run time, and exit the e*Way Editor.
7. When you return to the e*Way's Properties page, click **OK** to save all changes and return to the Schema Designer's main window.
8. Create a Collaboration for the "Outbound" e*Way, naming it "outbound_collab".
9. Set the Collaboration's properties as follows:

Collaboration Rules:	Passthrough_Data
Subscriptions	**Event**: In **Source**: ldap_collab2
Publications	**Event**: In **Publish to**: <External>

The Schema Designer configuration is now complete. Now, you must create some test data which will be sent to external sites. The results of these requests will be saved to the output data file.

Note: *When using an input file, it is necessary to modify the fields within the configuration file to match those within the input file, or to leave the fields blank. If within the input file, the delimiters are left empty the action within the configuration file will be used.*

10 Save the file as C:\TEMP\TESTDATA.FIN (if you specified a different input directory, please make the appropriate substitution).

Launch the sample schema. If the schema was configured properly and your connection to the remote location site is good, you should find response data from your requests in the file **C:\TEMP\ldap_out.txt** (if you specified a different output directory, please make the appropriate substitution).

4.1.4 Using Query Strings to Extract Data from LDAP

To search a directory, a search filter is used to define the search. Below is the basic syntax for a search filter:

(*attribute operator value*)

For example:

```
(cn=George Jetsen)
```

Where **cn** is the attribute, **=** is the operator, and **George Jetsen** is the value. The filter finds entries with the common name "George Jetsen."

Operators

The following table gives details on the use of each operator.:

Table 2 Operator Usage

Operator	Description	Example Filter	Search Result
=	Returns entries whose attribute is equal to the value.	`(cn=George Jetsen)`	Finds the entry "cn=George Jetsen."
>=	Returns entries whose attribute is greater than or equal to the value.	`(sn >= jensen)`	Finds all entries from "sn=jetsen" to "sn=z..."
<=	Returns entries whose attribute is less than or equal to the value.	`(sn <= jetsen)`	Finds all entries from "sn=a..." to "sn=jetsen."
=*	Returns entries that have a value set for that attribute.	`(sn =*)`	Finds all entries that have the **sn** attribute.
~=	Returns entries whose attribute value approximately matches the specified value. Typically, this is an algorithm that matches words that sound alike.	`(sn ~= jetsen)`	Finds the entry "sn = jedsen"

Note that when comparing values containing letters, the letter **a** is less than the value **z**. For example, the following filter finds all entries with last names beginning with **a** through **jetsen**:

```
(sn<=jetsen)
```

Boolean Operators

The following table gives details on the use of each boolean operator:

Boolean Operator	Description	Example
&	Returns entries matching all specified filter criteria.	`(& (sn>=b) (sn<=u))` Returns entries with last name begining with b through u.
|	Returns entries matching one or more of the filter criteria.	`(|(sn=Jetsen)(sn=Johnson))` Searches for all entries with the last name Jetsen or the last name Johnson.

!	Returns entries for which the filter is not true. You can only apply this operator to a single filter.	`(!(filter))`
		
		but not:
		
		`(!(filter1)(filter2))`

Wildcards

You can also include wildcards to search for entries that start with, contain, or end with a given value. For example, you can use this filter to search for all entries whose names begin with the letter F:

(givenName=F*)

Information on Search Scope

There are three possible search scopes: **BASE**, **ONELEVEL**, **SUBTREE**.

When sending a search request, you need to specify the base DN and scope of the search to identify the entries that you want searched. The base DN (the base argument) is the DN of the entry that serves as the starting point of the search.

To specify the scope of the search, pass one of the following values as the scope parameter:

- SUBTREE searches the base entry and all entries at all levels below the base entry
- ONELEVEL searches all entries at one level below the base entry. The base entry is not included in the search. Use this setting if you just want a list of the entries under a given entry.

Chapter 5

LDAP e*Way Functions

The LDAP e*Way's functions fall into the following categories:

- **Basic Functions** on page 43 control the e*Way's basic operations.
- **LDAP Standard Functions** on page 51 control the e*Way's communication center, and are defined from the configuration file.
- **LDAP Native Functions** on page 65 control the e*Way's business logic and are defined from within a Collaboration Rules script.

5.1 Basic Functions

The functions in this category control the e*Way's most basic operations.

Note: *The functions described in this section can only be used by the functions defined within the e*Way's configuration file. None of the functions are available to Collaboration Rules scripts executed by the e*Way.*

The basic functions are

start-schedule

Syntax

```
(start-schedule)
```

Description

start-schedule requests that the e*Way execute the "Exchange Data with External" function specified within the e*Way's configuration file. Does not affect any defined schedules.

Parameters

None.

Return Values

None.

Throws

None.

stop-schedule

Syntax

```
(stop-schedule)
```

Description

stop-schedule requests that the e*Way halt execution of the "Exchange Data with External" function specified within the e*Way's configuration file. Execution will be stopped when the e*Way concludes any open transaction. Does not affect any defined schedules, and does not halt the e*Way process itself.

Parameters

None.

Return Values

None.

Throws

None.

send-external-up

Syntax

```
(send-external-up)
```

Description

send-external-up instructs the e*Way that the connection to the external system is up.

Parameters

None.

Return Values

None.

Throws

None.

send-external-down

Syntax

```
(send-external-down)
```

Description

send-external down instructs the e*Way that the connection to the external system is down.

Parameters

None.

Return Values

None.

Throws

None.

get-logical-name

Syntax

```
(get-logical-name)
```

Description

get-logical-name returns the logical name of the e*Way.

Parameters

None.

Return Values

string
Returns the name of the e*Way (as defined by the Schema Designer).

Throws

None.

event-send-to-egate

Syntax

```
(event-send-to-egate string)
```

Description

event-send-to-egate sends data that the e*Way has already received from the external system into the e*Gate system as an Event.

Parameters

Name	Type	Description
string	string	The data to be sent to the e*Gate system.

Return Values

Boolean

Returns **#t** (true) if the data is sent successfully; otherwise, returns **#f** (false).

Throws

None.

Additional information

This function can be called by any e*Way function when it is necessary to send data to the e*Gate system in a blocking fashion.

shutdown-request

Syntax

```
(shutdown-request)
```

Description

shutdown-request completes the e*Gate shutdown procedure that was initiated by the Control Broker but was interrupted by returning a non-null value within the Shutdown Command Notification Function (see **"Shutdown Command Notification Function" on page 31**). Once this function is called, shutdown proceeds immediately.

Once interrupted, the e*Way's shutdown cannot proceed until this Monk function is called. If you do interrupt an e*Way shutdown, we recommend that you complete the process in a timely fashion.

Parameters

None.

Return Values

None.

Throws

None.

5.2 LDAP Standard Functions

The functions in this section control the e*Way's communications center and are defined within the conifiguration file.

Note: *The functions described in this section can only be used by the functions defined within the e*Way's configuration file. None of the functions are available to Collaboration Rules scripts executed by the e*Way.*

The current suite of standard functions are:

ldap-ack

Syntax

```
(ldap-ack message-string)
```

Description

ldap-ack is used to send a positive acknowledgment to the external system, and for post processing after successfully sending data to e*Gate.

Parameters

Name	Type	Description
message-string	string	The Event for which an acknowledgment is sent.

Return Values

string

An empty string indicates a successful operation. The e*Way will then be able proceed with the next request.

- The string "CONNERR" indicates a loss of connection with the external system. The client system moves to a down state and attempts to connect. When the connection is re-established, **ldap-ack** will be called again, with the same input data.

Throws

None.

ldap-exchange

Syntax

```
(ldap-exchange)
```

Description

ldap-exchange sends a received event from the external system to e*Gate. The function accepts no input.

Parameters

None.

Return Values

string

Returns "**UP**" when the connection is established. Anything else indicates no connection was established.

Throws

None.

ldap-outgoing

Syntax

```
(ldap-outgoing message-string)
```

Description

ldap-outgoing sends a received message (Event) from e*Gate to the external system.

Parameters

Name	Type	Description
message-string	string	The Event to be processed.

Return Values

string

Returns one of the following:

- An empty string indicates a successful operation.
- "RESEND" causes the message to be immediately resent. The e*Way will compare the number of attempts it has made to send the Event to the number specified in the Max Resends per Messages parameter, and does one of the following:
  - If the number of attempts does not exceed the maximum, the e*Way will pause the number of seconds specified by the **Resend Timeout** parameter, increment the "resend attempts" counter for that message, then repeat the attempt to send the message.
  - If the number of attempts exceeds the maximum, the function returns false and rolls back the message to the e*Gate IQ from which it was obtained.
- "CONNERR" indicates that there is a problem communicating with the external system. First, the e*Way will pause the number of seconds specified by the **Resend Timeout** parameter. Then, the e*Way will call the **External Connection Establishment function according to the Down Timeout schedule, and** will roll back the message (Event) to the IQ from which it was obtained.
- "DATAERR" indicates that there is a problem with the message (Event) data itself. First, the e*Way will pause the number of seconds specified by the **Resend Timeout** parameter. Then, the e*Way increments its "failed message (Event)" counter, and rolls back the message (Event) to the IQ from which it was obtained. If the e*Way's journal is enabled (see **"Journal File Name" on page 12**) the message (Event) will be journaled.
- If a string other than the above is returned, the e*Way will create an entry in the log file indicating that an attempt has been made to access an unsupported function.

Throws

None.

Examples

```
(define ldap-outgoing
 (lambda ( message-string )
  (let ((result ""))
;;The body of the function goes here. For example:
;;(display "[++]Executing e*Way external process outgoing message function.")
;; (display message-string)
;;
;;
;;The function must return a string.
result
  )
))
```

Notes

See **Figure 7 on page 24** for more information regarding Process Outgoing Message functions.

This function is used to query the LDAP directory database.

ldap-outgoing-write

Syntax

```
(ldap-outgoing-write event-string)
```

Description

ldap-outgoing-write sends a received message (Event) from e*Gate to the external system.

Parameters

Name	Type	Description
event-string	string	The Event to be processed.

Return Values

string

Returns one of the following:

- An empty string indicates a successful operation.
- **RESEND** causes the message to be immediately resent. The e*Way will compare the number of attempts it has made to send the Event to the number specified in the Max Resends per Messages parameter, and does one of the following:
  - If the number of attempts does not exceed the maximum, the e*Way will pause the number of seconds specified by the **Resend Timeout** parameter, increment the "resend attempts" counter for that message, then repeat the attempt to send the message.
  - If the number of attempts exceeds the maximum, the function returns false and rolls back the message to the e*Gate IQ from which it was obtained.
- **CONNERR** indicates that there is a problem communicating with the external system. First, the e*Way will pause the number of seconds specified by the **Resend Timeout** parameter. Then, the e*Way will call the **External Connection Establishment function according to the Down Timeout schedule, and** will roll back the message (Event) to the IQ from which it was obtained.
- **DATAERR** indicates that there is a problem with the message (Event) data itself. First, the e*Way will pause the number of seconds specified by the **Resend Timeout** parameter. Then, the e*Way increments its "failed message (Event)" counter, and rolls back the message (Event) to the IQ from which it was obtained. If the e*Way's journal is enabled (see **"Journal File Name" on page 12**) the message (Event) will be journaled.
- If a string other than the above is returned, the e*Way will create an entry in the log file indicating that an attempt has been made to access an unsupported function.

Throws

None.

Additional Information

This function is used to contact and modify/enter data into the LDAP directory database.

ldap-extconnect

Syntax

```
(ldap-extconnect)
```

Description

ldap-extconnect is used to establish the external system connection.

Parameters

None.

Return Values

string

Returns "**UP**," which indicates the connection is established. Any other return value indicates that no connection was established.

Throws

None.

ldap-init

Syntax

```
(ldap-init)
```

Description

ldap-init begins the initialization process for the e*Way. This function loads the stc_monkldap.dll file.

Parameters

None.

Return Values

string

If a "FAILURE" string is returned, the e*Way will shutdown. Any other returned string indicates success.

Throws

None.

ldap-nack

Syntax

```
(ldap-nack message-string)
```

Description

ldap-nack is used to send a negative acknowledgment to the external system, and for post processing after failing to send data to e*Gate.

Parameters

Name	Type	Description
message-string	string	The Event for which a negative acknowledgment is sent.

Return Values

string

An empty string indicates a successful operation. The e*Way will then be able to proceed with the next request.

- "CONNERR" indicates a loss of connection with the external system. The client system moves to a down state and attempts to connect. When the connection is re-established, **ldap-nack** will be called again, with the same input data.

Throws

None.

ldap-notify

Syntax

```
(ldap-notify)
```

Description

ldap-notify notifies the external system that the e*Way is shutting down.

Parameters

None.

Return Values

string
If a "FAILURE" string is returned, the e*Way will shutdown. Any other returned string indicates success.

Throws

None.

ldap-shutdown

Syntax

```
(ldap-shutdown shutdown)
```

Description

ldap-shutdown requests that the external shutdown.

Parameters

Name	Type	Description
shutdown	string	The string "**SUSPEND_NOTIFICATION**" is passed as the parameter, when the e*Way calls this function.

Return Values

string

"SUCCESS' allows an immediate shutdown to occur, anything else delays shutdown until the shutdown-request is executed successfully.

Throws

None.

Additional Information

A return value of SUCCESS indicates that the shutdown can occur immediately. Any other return value indicates that the shutdown Event must be delayed. If the shutdown is delayed the user is required to execute a **shutdown-request** call from within a Monk function to allow the requested shutdown process to continue. See **shutdown-request** on page 50.

ldap-startup

Syntax

```
(ldap-startup)
```

Description

ldap-startup loads instance-specific functions and invokes setup.

Parameters

None.

Return Values

string
"FAILURE" causes shutdown of the e*Way. Any other returned string indicates success.

Throws

None.

ldap-verify

Syntax

```
(ldap-verify hCon)
```

Description

ldap-verify is used to verify whether the external system connection is established.

Parameters

Name	Type	Description
hCon	opaque handle	The handle returned by **ldap-init**

Return Values

string

Returns "UP" or "SUCCESS" if the connection is established. Any other return value indicates the connection is not established.

Throws

None.

5.3 LDAP Native Functions

The functions described in this section can only be called from within a Collaboration Rules script.

The native functions that facilitate connection to the external TCP/IP system are:

ldap-open

Syntax

```
(ldap-open pszHostName dwPort [pszUserDistinguishedName pszPassWord])
```

Description

ldap-open locates and establishes the external system connection.

Parameters

Name	Type	Description
pszHostName	string	A string specifying the hostname for connection.
dwPort	integer	The port number for the connection
pszUserDistinguishedName	string	An optional parameter, a valid user name or an empty string.
pszPassWord	string	An optional parameter, implemented with the user name, the valid password associated with specified user name or an empty string.

Return Values

handle

The handle associated with the connection.

Throws

None.

Additional Information

For anonymous login, specify empty string (“”) for both pszUserDistinguishedName and pszPassWord, else the function will error out.

ldap-is-open

Syntax

```
(ldap-is-open hCon)
```

Description

ldap-is-open verifies that the connection handle is valid.

Parameters

Name	Type	Description
hCon	opaque handle	The handle returned from **ldap-open**.

Return Values

Boolean

Returns **#t** (true) if the handle is valid; otherwise, returns **#f** (false).

Throws

None.

ldap-close

Syntax

```
(ldap-close hCon)
```

Description

ldap-close deallocates the session handle obtained by ldap-open.

Parameters

Name	Type	Description
hCon	opaque handle	The handle returned from **ldap-open**.

Return Values

Boolean

Returns **#t** (true) if successful; otherwise, returns **#f** (false).

Throws

None.

ldap-begin-search

Syntax

```
(ldap-begin-search hCon pszBase pszScope pszFilter bAttrsonly
cmsTimeout cbSizeLimit)
```

Description

ldap-begin-search establishes the specifics of the search query.

Parameters

Name	Type	Description
hCon	opaque handle	The handle returned from **ldap-open**.
pszBase	string (0 delimited)	The starting distinguishing name for the search.
pszScope	string (0 delimited)	The scope of the search. (Valid selections are: BASE, ONELEVEL, and SUBTREE)
pszFilter	string (0 delimited)	The search filter to apply to the search.
bAttrsonly	Boolean	The Boolean flag to indicate whether only attribute types are wanted.
cmsTimeout	An Integer	The timeout expressed in milliseconds.
cbSizeLimit	An Integer	The maximum number of results to retrieve.

Return Values

Boolean

Returns **#t** (true) if the search obtains results; otherwise, returns **#f** (false).

Throws

None.

ldap-end-search

Syntax

```
(ldap-end-search hCon)
```

Description

ldap-end-search indicates whether the search ended successfully.

Parameters

Name	Type	Description
hCon	opaque handle	The handle returned from **ldap-open**.

Return Values

Boolean

Returns **#t** (true) if the search ended successfully; otherwise, returns **#f** (false).

Throws

None.

Additional Information

This API ***must*** be called before proceeding with another **ldap-begin-search** on the same connection handle.

ldap-howmany-messages

Syntax

```
(ldap-howmany-messages hCon)
```

Description

ldap-howmany-messages returns the number of messages returned from the search. A message (Event) is either an entry or a reference.

Parameters

Name	Type	Description
hCon	opaque handle	The handle returned from **ldap-open**.

Return Values

integer
Returns the number of messages returned from the search.

Throws

None.

Additional Information

For the current release, only entries are supported.

ldap-howmany-references

Syntax

```
(ldap-howmany-references hCon)
```

Description

ldap-howmany-references returns the number of references returned from the search.

Parameters

Name	Type	Description
hCon	opaque handle	The handle returned by **ldap-open**.

Return Values

integer
Returns the number of references returned from the search.

Throws

None.

ldap-howmany-entries

Syntax

```
(ldap-howmany-entries hCon)
```

Description

ldap-howmany-entries returns the number of entries returned from the search.

Parameters

Name	Type	Description
hCon	opaque handle	The handle returned by **ldap-open**.

Return Values

integer
Returns the number of entries returned from the search.

Throws

None.

ldap-get-next-entry

Syntax

```
(ldap-get-next-entry hCon)
```

Description

ldap-get-next-entry queries whether this call has resulted in an entry.

Parameters

Name	Type	Description
hCon	opaque handle	The handle returned by **ldap-open**.

Return Values

Boolean

Returns **#t** (true) if this call results in a successful entry; otherwise returns **#f** (false).

Throws

None.

ldap-read-distinguished-name

Syntax

```
(ldap-read-distinguished-name hCon)
```

Description

ldap-read-distinguished-name returns the DN of the entry previously returned by a call to **ldap-get-next-entry**.

Parameters

Name	Type	Description
hCon	opaque handle	The handle returned by **ldap-open**.

Return Values

string
Returns a zero-delimited string containing the DN of the entry.

Throws

None.

ldap-howmany-attributes

Syntax

```
(ldap-howmany-attributes hCon)
```

Description

ldap-howmany-attributes returns the number of attributes for the entry previously returned by a call to **ldap-get-next-entry**.

Parameters

Name	Type	Description
hCon	opaque handle	The handle returned from **ldap-open**.

Return Values

integer
Returns the number of attributes for the entry.

Throws

None.

ldap-get-next-attribute

Syntax

```
(ldap-get-next-attribute hCon)
```

Description

ldap-get-next-attribute queries whether the call resulted in any attributes for the entry.

Parameters

Name	Type	Description
hCon	opaque handle	The handle returned by **ldap-open**.

Return Values

Boolean

Returns **#t** (true) if the query results in the successful return of attributes; otherwise, returns **#f** (false).

Throws

None.

ldap-read-attribute-name

Syntax

```
(ldap-read-attribute-name hcon)
```

Description

ldap-read-attribute-name returns the name of the attribute returned as a result of the call to **ldap-get-next-attribute**.

Parameters

Name	Type	Description
hCon	opaque handle	The handle returned by **ldap-open**.

Return Values

string
Returns a zero-delimited string containing the name of an attribute.

Throws

None.

ldap-howmany-values

Syntax

```
(ldap-howmany-values hCon)
```

Description

ldap-howmany-values returns the number of values for the attribute previously returned by **ldap-get-next-attribute**.

Parameters

Name	Type	Description
hCon	opaque handle	The handle returned by **ldap-open**.

Return Values

integer
Returns the number of values for the attribute returned by **ldap-get-next-attribute**.

Throws

None.

ldap-get-next-value

Syntax

```
(ldap-get-next-value hCon)
```

Description

ldap-get-next-value queries whether the call returned any values for the attribute.

Parameters

Name	Type	Description
hCon	opaque handle	The handle returned by **ldap-open**.

Return Values

Boolean

Returns **#t** (true) if the call returned any values; otherwise, returns **#f** (false).

Throws

None.

ldap-read-value

Syntax

```
(ldap-read-value hCon)
```

Description

ldap-read-value returns the string containing the value previously returned by call to **ldap-get-next-value**.

Parameters

Name	Type	Description
hCon	opaque handle	The handle returned by **ldap-open**.

Return Values

string
Returns a zero-delimited string containing the value returned by a call to **ldap-get-next-value**.

Throws

None.

ldap-delete-entry

Syntax

```
(ldap-delete-entry hCon pszDNDelete)
```

Description

ldap-delete-entry deletes the specified distinguished name.

Parameters

Name	Type	Description
hCon	opaque handle	The handle returned by **ldap-open**.
pszDNDelete	string	The distinguished name of the entry to delete.

Return Values

Boolean

Returns **#t** (true) if the disinguished name is deleted successfully; otherwise, returns **#f** (false).

Throws

None.

ldap-rename-entry

Syntax

```
(ldap-rename-entry hCon pszDN pszRDN)
```

Description

ldap-rename-entry renames the specified entry.

Parameters

Name	Type	Description
hCon	opaque handle	The handle returned by **ldap-open**.
pszDN	string	The current distinguised name of the entry to rename.
pszRDN	string	The new relative distinguished name of the entry.

Return Values

Boolean

Returns **#t** (true) if the entry is succesfully renamed; otherwise, returns **#f** (false).

Throws

None.

ldap-create-ldapmod

Syntax

```
(ldap-create-ldapmod pszDN)
```

Description

ldap-create-ldapmod returns the handle to the newly created LDAPMods, used for setting attributes and attribute values.

Parameters

Name	Type	Description
pszDN	string	The distinguished name of an entry to add or modify.

Return Values

handle
Returns the handle to the LDAPMods.

Throws

None.

ldap-set-ldapmod-attribute

Syntax

```
(ldap-set-ldapmod-attribute hLDAPMod pszAttributeName
pszValuesDelimList pszValueDelim)
```

Description

ldap-set-ldapmod-attribute sets the attribute/values for the specified LDAPMod.

Parameters

Name	Type	Description
hLDAPMod	opaque handle	The handle returned by **ldap-create-ldapmod**.
pszAttributeName	string	The attribute to add or modify.
pszValuesDelimList	string	The attribute's values delimited by a character specified by pszValueDelim.
pszValueDelim	string	The character (i.e., "|" that is used to delimit each value in pszValuesDelimList.

Return Values

Boolean

Returns **#t** (true) if the attribute/values are set successfully; otherwise, returns **#f** (false).

Throws

None.

Additional Information

Because an entry may have more than one attribute, it is necessary to repeatedly call this API to "set" the attribute and its values for all the required attributes before calling **ldap-add-entry**, **ldap-add-attributes**, **ldap-remove-attributes**, **ldap-add-values**, **ldap-remove-values**, and **ldap-replace-values**. Ensure that pszValueDelim is the character that delimits the values in pszValuesDelimList.

ldap-clear-ldapmod-attributes

Syntax

```
(ldap-clear-ldapmod-attributes hLDAPMod)
```

Description

ldap-clear-ldapmod-attributes clears the attribute and the associated values for the specified attribute.

Parameters

Name	Type	Description
hLDAPMod	opaque handle	The handle returned by **ldap-create-ldapmod**.

Return Values

Boolean

Returns **#t** (true) if the attribute and the associated values are deleted successfully; otherwise, returns **#f** (false).

Throws

None.

Additional Information

Use this API to clear any attribute(s)/value(s) set by prior calls to **ldap-set-ldapmod-attribute**.

ldap-destroy-ldapmod

Syntax

```
(ldap-destroy-ldapmod hLDAPMod)
```

Description

ldap-destroy-ldapmod deallocates the memory associated with the specified handle.

Parameters

Name	Type	Description
hLDAPMod	opaque handle	The handle returned by **ldap-create-ldapmod**.

Return Values

Boolean

Returns **#t** (true) if the handle deletes successfully; otherwise, returns **#f** (false).

Throws

None.

Additional Information

This API must be called to deallocate memory when LDAPMod handle is no longer in use.

ldap-add-entry

Syntax

```
(ldap-add-entry hCon hLDAPMod)
```

Description

ldap-add-entry adds an entry.

Parameters

Name	Type	Description
hCon	opaque handle	The handle returned by **ldap-open**.
hLDAPMod	opaque handle	The handle returned by **ldap-create-ldapmod**.

Return Values

Boolean

Returns **#t** (true) if the entry is added successfully; otherwise, returns **#f** (false).

Throws

None.

Additional Information

ldap-create-ldapmod creates LDAPMod handle for the entry to add. For each attribute of an entry, use **ldap-set-ldapmod-attribute** to set the attribute name and attribute values before calling this API to add the entry.

ldap-add-attributes

Syntax

```
(ldap-add-attributes hCon hLDAPMod)
```

Description

ldap-add-attributes adds an attribute to the entry.

Parameters

Name	Type	Description
hCon	opaque handle	The handle returned by **ldap-open**.
hLDAPMod	opaque handle	The handle returned by **ldap-create-ldapmod**.

Return Values

Boolean

Returns **#t** (true) if the attribute is added successfully; otherwise, returns **#f** (false).

Throws

None.

Additional Information

ldap-create-ldapmod creates LDAPMod hand for the entry to modify. For each attribute to add to the entry, use **ldap-set-ldapmod-attribute** to set the attribute name and attribute values before calling this API to add the attribute(s) to the entry.

ldap-remove-attributes

Syntax

```
(ldap-remove-attributes hCon hLDAPMod)
```

Description

ldap-remove-attribute removes the specified attribute from the entry.

Parameters

Name	Type	Description
hCon	opaque handle	The handle returned by **ldap-open**.
hLDAPMod	opaque handle	The handle returned by **ldap-create-ldapmod**.

Return Values

Boolean

Returns **#t** (true) if the attribute is removed successfully; otherwise, returns **#f** (false).

Throws

None.

Additional Information

ldap-create-ldapmod creates the LDAPMod handle for the entry to modify. For each attribute to remove from the entry, use **ldap-set-ldapmod-attribute** to set the attribute name and attribute values before calling this API to remove the attribute(s) from the entry.

ldap-add-values

Syntax

```
(ldap-add-values hCon hLDAPMod)
```

Description

ldap-add-value adds values to the attribute.

Parameters

Name	Type	Description
hCon	opaque handle	The handle returned by **ldap-open**.
hLDAPMod	opaque handle	The handle returned by **ldap-create-ldapmod**.

Return Values

Boolean

Returns **#t** (true) if the value is added successfully; otherwise, returns **#f** (false).

Throws

None.

Additional Information

ldap-create-ldapmod creates the LDAPMod handle for the entry to modify. Use ldap-set-ldapmod-attribute with the desired values in pszValuesDelimList to be added to the attribute specified by pszAttributeName before calling this API to add the values to the attribute of the entry.

ldap-remove-values

Syntax

```
(ldap-remove-values hCon hLDAPMod)
```

Description

ldap-remove-values removes the specified value(s) from the attribute.

Parameters

Name	Type	Description
hCon	opaque handle	The handle returned by **ldap-open**.
hLDAPMod	opaque handle	The handle returned by **ldap-create-ldapmod**.

Return Values

Boolean

Returns **#t** (true) if the value is removed successfully; otherwise, returns **#f** (false).

Throws

None.

Additional Information

ldap-create-ldapmod creates the LDAPMod handle for the entry to modify. Use **ldap-set-ldapmod-attribute** with the desired values in pszValuesDelimList to be removed from the attribute specified by pszAttributeName before calling this API to remove the values from the attribute of the entry.

ldap-replace-values

Syntax

```
(ldap-replace-values hCon hLDAPMod)
```

Description

ldap-replace-values replaces the value(s) for an attribute.

Parameters

Name	Type	Description
hCon	opaque handle	The handle returned by **ldap-open**.
hLDAPMod	opaque handle	The handle returned by **ldap-create-ldapmod**.

Return Values

Boolean

Returns **#t** (true) if the value is replaced successfully; otherwise, returns **#f** (false).

Throws

None.

Additional Information

ldap-create-ldapmod creates the LDAPMod handle for the entry to modify. Use **ldap-set-ldapmod-attribute** with the desired values in pszValuesDelimList to replace the attribute apecified by pszAttributeName before calling this API to replace the values in the attribute of the entry.

ldap-compare-value

Syntax

```
(ldap-compare-value hCon hLDAPMod)
```

Description

ldap-compare-value verifies if the value is found in desired attribute of the entry.

Parameters

Name	Type	Description
hCon	opaque handle	The handle returned by **ldap-open**.
hLDAPMod	opaque handle	The handle returned by **ldap-create-ldapmod**.

Return Values

Boolean

Returns **#t** (true) if the pszValue is found in pszAttribute of the entry pszEntryDN; otherwise, returns **#f** (false).

Throws

None.

ldap-set-search-attributes

Syntax

```
(ldap-set-search-attributes hCon pszDelimitedAttribs pszDelimList)
```

Description

ldap-set-search-attributes sets the attribute restriction for the specified search attribute.

Parameters

Name	Type	Description
hCon	opaque handle	The handle returned by **ldap-open**.
pszDelimitedAttribs	string	The delimited attribute list.
pszDelimList	string	The delimiter(s) used to delimit the attribute(s) in pszDelimitedAttribs.

Return Values

Boolean

Returns **#t** (true) if the attribute restriction is successfully set; otherwise, returns **#f** (false).

Throws

None.

Additional Information

Call this function ***before*** calling **ldap-begin-search** to set attribute restriction.

If pszDelimitedAttribs and pszDelimList are empty (“”), then no attribute restriction will take effect on searches.

If pszDelimitedAttribs is a single attribute (i.e., “sn”) and pszDelimList is empty (“”), then the searched entry results will only have the attribute specified by pszDelimitedAttribs.

If pszDelimitedAttribs is a multi-attributed delimited list (i.e., “sn|email|phone”) and pszDelimiList in non-empty (i.e., “|”), then searched entry results will have all the attributes specified in pszDelimitedAttribs.

ldap-clear-search-attributes

Syntax

```
(ldap-clear-search-attributes hCon)
```

Description

ldap-clear-search clears the attribute restriction set by **ldap-set-search-attributes**..

Parameters

Name	Type	Description
hCon	opaque handle	The handle returned by **ldap-open**.

Return Values

Boolean

Returns **#t** (true) if the attribute restriction cleared successfully; otherwise, returns **#f** (false).

Throws

None.

ldap-sort-entries-on-dn

Syntax

```
(ldap-sort-entries-on-dn hCon)
```

Description

ldap-sort-entries-on-dn sort the entries returned by **ldap-begin-search**.

Parameters

Name	Type	Description
hCon	opaque handle	The handle returned by **ldap-open**.

Return Values

Boolean

Returns **#t** (true) indicates the sort was successful; otherwise, returns **#f** (false).

Throws

None.

Additional Information

Call this function ***after*** calling **ldap-begin-search** to sort entries by their distinguised name and ***prior*** to callinging **ldap-get-next-entry**.

ldap-sort-entries-on-attributes

Syntax

```
(ldap-sort-entris-on-attributes hCon pszDelimitedAttribs
pszDelimList)
```

Description

ldap-sort-entries-on-attributes sorts the attributes in the order specified.

Parameters

Name	Type	Description
hCon	opaque handle	The handle returned by **ldap-open**.
pszDelimitedAttribs	string	The delimited attribute list.
pszDelimList	string	The delimiter(s) used to delimit the attribute(s) in pszDelimitedAttribs.

Return Values

Boolean

Returns **#t** (true) if the attribute restriction was successfully set; otherwise, **#f** (false).

Throws

None.

Additional Information

Call this function after calling **ldap-begin-search** to sort entries by their attribute(s) and prior to calling **ldap-get-next-entry**.

If pszDelimitedAttribs and pszDelimList are empty (“”), sorting is defaulted to sorted on distinguished name (same as calling **ldap-sort-entries-on-dn**).

If pszDelimitedAttribs is a single attribute (i.e., “sn”) and pszDelimList is empty (“”), then sorting is done on attribute pszDelimitedAttribs.

If pszDelimitedAttribs is a multi-attributed delimited list (i.e., “sn|email|phone”) and pszDelimList is non-empty (i.e., “|”) , then sorting will be done on attributes in the order listed from left to right in pszDelimitedAttribs (i.e., sort on “sn”, then on “email”, and then on “phone”).

Index

P

Q

R

S

U

Z