EVIDEN

Identity and Access Management

DirX Directory

LDAP Proxy

Version 9.1, Edition June 2025

Table of Contents

Preface

This document provides information on how to configure and operate DirX Directory (DirX) as an LDAP Proxy.

DirX Directory Documentation Set

DirX Directory provides a powerful set of documentation that helps you configure your directory server and its applications.

The DirX Directory document set consists of the following manuals:

- *DirX Directory Introduction*. Use this book to obtain a description of the concepts of DirX Directory.
- *DirX Directory Administration Guide*. Use this book to understand the basic DirX Directory administration tasks and how to perform them with the DirX Directory administration tools.
- *DirX Directory Administration Reference*. Use this book to obtain reference information about DirX Directory administration tools and their command syntax, configuration files, environment variables and file locations of the DirX Directory installation.
- *DirX Directory Syntaxes and Attributes*. Use this book to obtain reference information about DirX Directory syntaxes and attributes.
- *DirX Directory LDAP Extended Operations*. Use this book to obtain reference information about DirX Directory LDAP Extended Operations.
- *DirX Directory External Authentication*. Use this book to obtain reference information about external authentication.
- *DirX Directory Supervisor*. Use this book to obtain reference information about the DirX Directory supervisor.
- *DirX Directory Plugins for Nagios*. Use this book to obtain reference information about DirX Directory plugins for Nagios.
- *DirX Directory Disc Dimensioning Guide*. Use this book to understand how to calculate and organize necessary disc space for initial database configuration and enhancing existing configurations.
- *DirX Directory Guide for CSP Administrators*. Use this book to obtain information about installing, configuring and managing DirX Directory in the context of a Certificate Provisioning Service operating in accordance with regulations like the German “Signaturgesetz”.
- *DirX Directory Release Notes*. Use this book to install DirX Directory and to understand the features and limitations of the current release.

Notation Conventions

Boldface type
In command syntax, bold words and characters represent commands or keywords that must be entered exactly as shown.

In examples, bold words and characters represent user input.

Italic type
In command syntax, italic words and characters represent placeholders for information that you must supply.

[ ]
In command syntax, square braces enclose optional items.

{ }
In command syntax, braces enclose a list from which you must choose one item.

In Tcl syntax, you must actually type in the braces, which will appear in boldface type.

|
In command syntax, the vertical bar separates items in a list of choices.

...
In command syntax, ellipses indicate that the previous item can be repeated.

install_path
The exact name of the root of the directory where DirX Identity programs and files are installed. The default installation directory is *userID_home_directory**/DirX Identity* on UNIX systems and **C:\Program Files\DirX\Identity** on Windows systems. During installation the installation directory can be specified. In this manual, the installation-specific portion of pathnames is represented by the notation *install_path*.

1. Introduction

The DirX Directory LDAP server can run in one of two modes:

- As a plain LDAP frontend to a DSA. In this mode, the LDAP server forwards incoming requests to the DSA via DAP.
- As a pure LDAPv3 proxy frontend to another LDAP server. In this mode, the LDAP server forwards incoming requests to another LDAP server, which can be a DirX Directory LDAP server or another type of LDAPv3 server (for example, Active Directory).

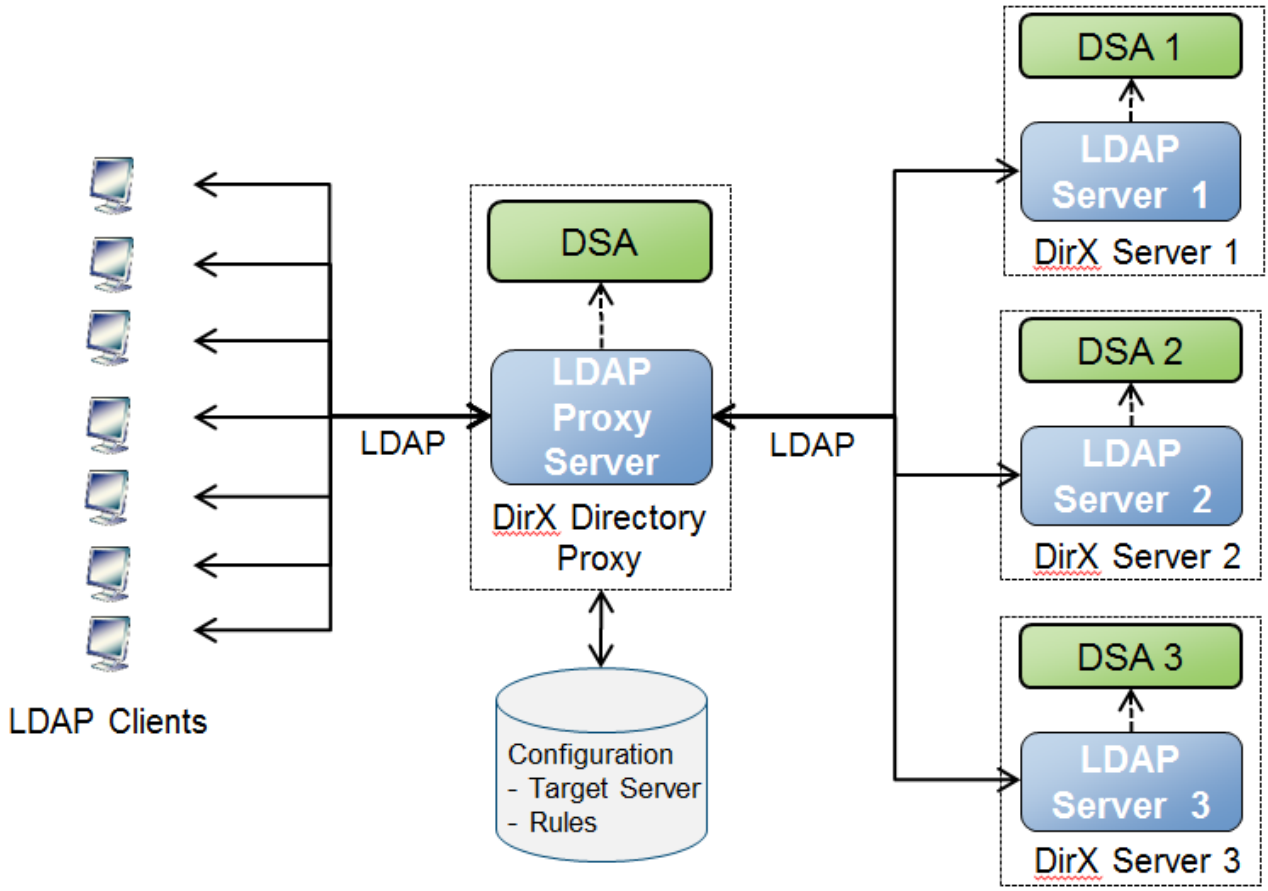
When configured to run in proxy mode, a DirX Directory LDAP server can be configured to:

- Redirect incoming LDAPv3 operations to other LDAP servers based on user name or operation type. This feature allows the routing of specific users, groups of users or special operations to specific servers only; for example, all searches for base object Y can be routed to LDAP server Z.
- Modify the content of incoming LDAPv3 client requests and outgoing LDAPv3 results before sending them on for further processing. This feature allows for adding, replacing or removing unwanted or incompatible data from LDAP requests and results – for example, to hide certain data from LDAP clients – or to provide a transparent naming schema between legacy clients and the DIT.

Running the DirX Directory LDAP server in proxy mode has the following advantages:

- It creates a central access point for LDAP clients
- It hides the actual processing server knowledge
- It hides server outages (the feature implies automatic request forwarding to other available servers in the event of network problems or server outages)
- It transparently provides load balancing to the caller
- It provides an additional access control layer to the directory service

The following figure illustrates a typical DirX Directory LDAP proxy (DLP) server configuration within a homogeneous DirX Directory environment:



As shown in the figure:

- Configuration information for the DLP server backend consists of the available target servers and rules for backend server operation. **Routing rules** specify how client requests are selected and forwarded to the configured target servers while **rewriting rules** specify how client requests or results are selected for modification and what the modifications to be made to that data shall be.
- The DLP server's contact DSA is the local DSA. Note that although the DLP server's contact DSA could be DSA1, DSA2 or DSA3, we do not recommend setting it up this way.

1.1. Configuration Elements

Configuration elements for the DLP server include:

- The LDAP configuration subentry
- The proxy mode attribute of the LDAP configuration subentry
- The DLP server configuration file

1.1.1. LDAP Configuration Subentry

Both the LDAP server and the DLP server require the presence of an LDAP configuration subentry in the DSA to configure the settings related to frontend handling, like ports, allowed and denied IP addresses and the like.This information is shared between both modes of operation.See the "DirX Directory Attributes" chapter in the *DirX Directory*

Syntaxes and Attributes reference document for a complete description of attributes defined for the LDAP configuration subentry.

1.1.2. Proxy Mode Attribute

Operation as a plain LDAP server or a DLP server is controlled by the ldapProxyMode attribute in the LDAP configuration subentry. The DirX Directory LDAP server process (**dirxldapv3**) automatically reads the value of the **ldapProxyMode** attribute at startup to determine whether to run in server mode or proxy mode. See the “Configuration” chapter in this guide for more information about the **ldapProxyMode** attribute.

1.1.3. DLP Server Configuration File

When the LDAP server is running in proxy mode - in other words, as a DLP server - the backend to which the client requests are forwarded is another LDAP server instead of a DSA. Therefore, settings in the LDAP configuration subentry that affect the backend for plain LDAP server operation - for example, DAP pool size, unbind delay time, and so on - can still be configured but will not be relevant for the DLP server.

The backend for DLP server operation is configured in the DLP server configuration file, which is a separate JavaScript Object Notation (JSON)-formatted file that defines:

- The LDAP server to be enabled as a DLP server
- The LDAP servers to which the DLP server can forward incoming requests
- The rules for routing these requests to these target servers
- The LDAP servers to be used as fallback target servers when there is no routing rule match on a request; the DLP server selects from this default set of target servers, called LB-servers.
- The rules for rewriting incoming requests before they are forwarded to a target server and for rewriting outgoing results before they are returned to the client

See the “Configuration” chapter for more information about the DLP server configuration file and how to specify the DLP server, target systems, LB-servers and rules.

1.2. Features and Limitations

The DLP server has the following features:

- Once the DLP server selects a target server, it forwards all subsequent operations on the same LDAP client connection to this same target server.All results from the target server will be received by the DLP server and returned to the LDAP client unchanged unless result rewriting rules are in effect and have been applied.
- The DLP server supports anonymous and simple authenticated LDAP users via plain and SSL/TLS.
- The DLP server can act as a TLS gateway for non-TLS clients; that is, it can receive requests from plain clients (without SSL/TLS) and then forward these requests over a secure connection to the target servers.

- The DLP server supports operation forwarding via plain and SSL connections to LDAP target servers.
- The DLP server forwards all types of LDAP operations except for LDAP extended operations, which are always executed locally either against the DLP server or the local DSA.Only the RFC3062 extended operation will be forwarded.
- If LDAP extended operations are to be executed against a target server, the operation must be sent directly to it.
- The DLP server requires a local DSA to host the DLP server's LDAP configuration subentry and the schema information used in client requests.
- The DLP server supports logging and auditing through the DirX Directory LDAP audit interfaces and tools (**dirxauddecode**). See the logging and auditing chapters in this guide for examples of output returned by these interfaces for DLP server mode.
- The DLP server supports a DirX Directory Manager interface and DirX Directory LDAP extended operations to query server status and initiate dynamic updates to its configuration without interrupting service. It supports DirX Directory LDAP extended operations to disable and enable the target LDAP servers configured to be available to the DLP server. See the Extended Operations chapter in this guide for reference descriptions of these operations.

The DLP server has the following limitations:

- The DLP server only accepts LDAPv3 requests. LDAPv2 is not supported.
- The DLP server can only forward LDAPv3 traffic. It cannot forward any other type of traffic.
- The DLP server does not support client-based authentication (SASL EXTERNAL)
- Because the DLP server establishes the same number of backend connections as frontend connections, be careful not to exceed the descriptor limits (usually a maximum of 4,000 LDAP frontend connections).
- Unlike the DSA backend, which has backend sharing capabilities, each LDAP client will always occupy two descriptors in the DLP server backend.
- Although the DLP server can establish up to 4,000 client connections on Linux platforms, we recommend limiting the number of possible client connections (since having huge arrays of sockets to multiplex can slow down server responsivity) and configuring multiple DLP servers instead.

The DLP server's routing mechanism has the following considerations:

- The possible target servers for a user should share the same knowledge in their DIT; that is, in terms of replication, they should be "full shadows" of each other. Mixing target servers with different knowledge should be avoided.
- Since DLP server forwarding can be based on the user's DN, the DLP server requires knowledge of the naming attributes of the DN. As a result, all attributes that may appear in a user's DN - including those of any non-DirX Directory target servers present in the configuration - should be defined in the DLP server's local schema.

2. Configuration

Setting up a DirX Directory LDAP server for operation as a DLP server requires two configuration tasks:

- Configuring the ldapProxyMode attribute in the LDAP server configuration subentry
- Setting up the DLP server configuration file

2.1. LDAP Proxy Mode Attribute

The ldapProxyMode attribute is an optional single-valued attribute that controls whether the LDAP server runs as:

- A normal LDAP server
- A DLP server without support for SSL/TLS backends
- A DLP server with support for SSL/TLS backends

If the ldapProxyMode attribute is not present in the LDAP configuration subentry, the LDAP server runs as a plain LDAP server.

Specify one of the following integer values:

- **0** – Run as a normal LDAP server.
- **1** – Run as a DLP server that does not support SSL/TLS backends. In this mode, the SSL/TLS functionality for the DLP server backend is not initialized which saves time and memory and does not require the configuration of SSL/TLS key material.
- **2** – Run as a DLP server that supports SSL/TLS backends. This mode requires the presence of valid SSL/TLS key material; in particular, the necessary trusted CA certificates must be provided in order to set up SSL/TLS connection to LDAPS backend servers. If your configuration currently uses plain connections but you expect to upgrade to using SSL/TLS in the near future, we recommend setting this mode and providing the required SSL/TLS key material.

The default value is **0**.

You cannot change from mode **1** to mode **2** during DLP server runtime, but you are allowed to change from mode **2** to mode **1** (downgrade) during DLP server runtime. To change the mode from **1** to **2**, you must restart the LDAP server process. A value of **0** always switches to plain LDAP server mode.

Note that proxy modes **1** and **2** have no effect on the LDAP frontend's SSL/TLS connectivity; that is, even if mode **1** is set, LDAP clients can still connect to the DLP server via SSL/TLS. The setting only prohibits the use of SSL/TLS when forwarding the request to the target servers.

2.1.1. Abbreviation

LPROM

2.1.2. LDAP Name(s)

ldapProxyMode

2.1.3. Syntax (DAP)

Integer

2.1.4. Syntax (LDAP)

Integer (LDAP Style)

2.1.5. Example (DAP)

LPROM=1

2.1.6. Example (LDAP)

ldapProxyMode=2

2.2. DLP Server Configuration File

To run the LDAP server as a DLP server, its backend configuration must be defined in a JavaScript Object Notation (JSON)-formatted configuration file named **ldap_proxy.json** which is located by default at *install_path*/**ldap/conf**.This file defines:

- The LDAP server process to be enabled as a DLP server
- The target systems to be used by the DLP server
- The defaults to be applied to operations handled by the DLP server
- The operation-forwarding and rewriting rules to be applied when running as a DLP server

The next sections describe configuration file syntax and how to debug problems with JSON syntax.

2.2.1. DLP Server Configuration File Syntax

The DLP server configuration file is a text file that you can edit with any common text editor. It consists of object definitions arranged in a single array. The array starts with an opening square bracket ([) and ends with a closing square bracket (]). Array elements (called **objects**) are separated by commas (,). Each object starts with an open curly brace (\{) and ends with a closing curly brace (}).

Thus, the fundamental layout of the configuration file looks like this:

[{_O1_},{_O2_},...{*On*}]

Where *O1...On* are JSON objects.The array can have any number of objects in an arbitrary order.Each object can have its own different syntax.

Each single object consists of a list of key-value pairs separated by colons (:).

All keys are strings, while the values can be integers, simple strings or arrays of strings like:

```
{
    “key1”    :   “value1”,             // string
    “key2”    :   [ “elem1”, “elem2” ], // array of 2 strings
    “key3”    :   4711,                 // integer
    “key4”    :   “value4”              // another string
}
```

Objects can also have values, such as:

```
{
    “key5”    :   “value5”,
    “key6”    :   123,
    “key7”    :   {                     // sub-object as value
                    “option1” : 0,
                    “option2” : “foobar”
                  }
}
```

Within an object, the order of key-value pairs is arbitrary. Each key-value pair is separated by a comma (,).

You can use C/C++-style comments in the file; that is, if the two characters “//” are found, the JSON parser will ignore the rest of the current line. The same is valid if the two characters “/**” are found; in this case, the parser will ignore everything until it finds the two chars “**/” (such comments can span multiple lines.)

2.2.2. Locating JSON Syntax Errors

If the DLP server configuration file does not contain valid JSON syntax, the LDAP server will not start. When JSON syntax errors are detected, the JSON parser will report them with messages like:

FATAL! JSON parsing error.
Illegal Proxy-Config file!! (must start as an array of objects)

FATAL: Cannot read PROXY configuration (ldap_proxy.json)!

LDAP Server exit (ExitCode: 19 Reason: FATAL: Cannot process PROXY configuration! Server stopped.)

Unfortunately, the parser does not provide more details about the location of the error. Thus, it requires some manual effort to locate the problem.

The first step should be to look at the JSON file and make sure that all required commas (,) are present. Most of the errors are due to missing commas. Remember that all objects must be separated by commas, all components within an object must be separated by commas and all server arrays must be separated by commas.

Another cause of errors is missing trailing quotation marks for strings; for example, “LDAP2.

Take care, too, that you do not specify integers (for example, **4711**) as strings (for example, **“4711”**).

You can also set the environment variable

```
DIRX_JSON_DEBUG=1
```

and then restart the DLP server.This action writes the JSON parsing process to **stderr** so that you can view the parser’s progress and identify the possible error location where parsing aborts.

If these steps don’t work, you can use an online JSON checker like

`http://json.parser.online.fr/`

The checker tries to analyze the syntax and show where the problem may exist.Be sure to remove any comments before using the checker, as JSON usually does not allow C/C++ comments.

2.3. DLP Server Configuration Objects

The DLP server configuration file supports the following object types for defining the information necessary for DLP server backend operation:

- LdapProxy – defines which LDAP server process runs as a DLP server
- LdapServer – defines a target server
- AttributeList – defines a list of attributes
- Defaults – defines explicit defaults to be applied to all operations handled by the DLP server
- ProxyRule – defines operation forwarding and LDAP request/result rewriting rules

LdapProxy and LdapServer are mandatory objects.The ProxyRule objects are optional.When specified, they allow the DLP server to redirect LDAP client requests from particular users or with particular operation types to particular LDAP servers and to rewrite incoming client requests and outgoing server results.The Defaults object is also optional; if it is not used, the DLP server uses its internal defaults.Note that no matter what the DLP

server configuration file contains, it must be specified as an array of 1 to *n* objects as required by JSON syntax.

Each object must define two keys:

- The **object** key – the value of this key is a string and defines the object's type and its components.
- The **name** key – the value of this key is a unique string that gives the object a symbolic name.

Additional key-value pairs may be mandatory depending on the object.

2.3.1. The LdapProxy Object

The LdapProxy object is a mandatory object in the DLP server configuration file and specifies the LDAP server to be enabled as a DLP server at process startup. Here is an example:

```
{
   "object"      : "LdapProxy",
   "name"        : "ldapConfiguration",
   "LBservers"   : [ "LDAP1", "LDAP2", "LDAP3" ]
}
```

In the object definition:

- The **object** key is mandatory and must be the string **LdapProxy**.
- The **name** key is mandatory and must be the LDAP configuration subentry name of the LDAP server to be configured as a DLP server. By default, this name is **ldapConfiguration**.
- The **LBservers** key is mandatory and defines the target servers (1-*n*) to which the DLP server can forward requests from users that do not match any of the user-routing or operation-routing rules defined in the DLP server configuration file. Its value is an array of symbolic LDAP server names that must match the names of LdapServer objects defined in the same configuration file. Note that the value must be specified as an array even if only one LDAP server is listed or an error occurs and the LDAP server will not start. If a user is controlled by a user- or operation-routing rule, DLP server ignores the **LBservers** value(s). The method the DLP server uses to select from a list of LB-servers for forwarding depends on the settings of **loadbalance** and **failover** keys in the Defaults object definition. See the description of the Defaults object definition in this guide for details.

2.3.2. The LdapServer Object

The LdapServer object defines a target LDAP server to which the DLP server can forward incoming client requests. Here is an example:

```
{
   "object"   : "LdapServer",    // this is a target server
   "name"     : "LDAP3”,         // symbolic name
   "protocol" : "ldap",          // use plain connection
   "host"     : "192.168.10.7",  // target host or IP
   "port"     :  389             // target port
}
```

In the object definition:

- The **object** key is mandatory and must be the string **LdapServer**.
- The **name** key is mandatory and can have any string value as long as the value is unique. The name is used in other objects in the configuration file (for example, in LdapProxy objects) to refer to a specific target server.
- The **protocol** key is mandatory and defines how the request is forwarded to the target server. The value **ldap** results in unencrypted plain socket mode. The value **ldaps** results in an SSL connection to the target server. When **ldaps** is used, the DLP server must run in **ProxyMode=2** and suitable TLS key material must be provided.
- The **host** key is mandatory and defines the IP or DNS name of the target LDAP server.
- The **port** key is mandatory and defines the port of the target LDAP server.

If the target server is to be addressed via SSL/TLS, the **ssl** key must also be defined. Here is an example:

```
{       // Defines an SSL LDAP server
   "object"   : "LdapServer",
   "name"     : "LDAP2",
   "protocol" : "ldaps",        // use SSL/TLS
   "host"     : "192.168.10.8",
   "port"     :  636,           // secure port
   "ssl"      : {
       "trusted_ca_file" : "C:/Program
Files/DirX/Directory/conf/testCA.pem",
       "ssl_protocol"    : "TLSv12",
       "ssl_cipher"      : "HIGH"
   }
}
```

The value of the **ssl** key is a sub-object (enclosed in curly braces (\{ }) with the following mandatory components:

- **ssl_protocol** – defines which SSL/TLS protocol to use. Possible values are **TLSv10**, **TLSv11**, **TLSv12** or **TLSv13**. The value is case sensitive. Make sure that the target server supports and allows the selected protocol.
- **ssl_cipher** – a string that defines the cipher list. The DLP server uses OpenSSL to contact target servers via SSL/TLS. The recommended value for TLS versions ⇐ TLSv12 is **HIGH**, which typically excludes unsafe ciphers from being selected. If TLSv13 is chosen special ciphers must be selected. See the OpenSSL documentation for information on possible cipher values.
- **trusted_ca_files** – defines the full qualified PEM file name that contains the chains of trusted public CA certificates used to verify the received server certificates from the target servers during the SSL handshake.

The DLP server currently supports only server-based SSL/TLS authentication against the backend, which means that the client verifies the server's certificate but not vice versa. Client-based SSL/TLS authentication against the backend is not supported. Thus, target servers that require client-based authentication should not be configured as target servers.

2.3.3. The AttributeList Object

You can use the AttributeList object to define a list of attributes for the **showonly** action of the **search.res.attributes** token of a rewriting rule in operations handled by the DLP server. Here is an example:

```
{       // Defines AttributeLists
   {
      "object"     : "AttributeList",
      "name"       : "ATTRLIST1",
      "attributes" : [ "sn", "surName", "cn", "commonName", "tn",
                       "telephoneNumber", "title", "dxidummy" ]
   },
   {
      "object"     : "AttributeList",
      "name"       : "ATTRLIST2",
      "attributes" : [
                       "tn",
                       "telephoneNumber",
                       "title",
                       "dxidummy",
                       "description
                     ]
   }
}
```

In the object definition:

- The **object** key is mandatory and must be the string **AttributeList**.
- The **name** key is mandatory and must be a unique string within multiple AttributeList definitions.
- The **attributes** key is a list of LDAP attribute names separated by a comma (**,**) and enclosed in square brackets [ ]. If an attribute has multiple LDAP names, for example, **telephoneNumber** and **tn**, all LDAP names must be specified to ensure the visibility in the result. OIDs cannot be specified instead of the LDAP name.

Attribute lists are only supported for the **showonly** action.

Keep in mind that the order of multiple actions is important that is the output of an action is the input of the subsequent action.

2.3.4. The Defaults Object

You can use the Defaults object to define explicit defaults to be applied to all operations handled by the DLP server. If used, the Defaults object must be the first object defined in the DLP server configuration file.

All of the values in the Defaults object are integers. Here is an example:

```
{
   "object"     : "Defaults",
   "LdapProxy"  : {
       "lb_failover"  : 1,
       "lb_balance"   : 1
   }
   "TLSLogging"       : 0,
   "Tracing"          : 0,
   "ConnectTimeout"   : 5,
   "OfflineRetryTime" : 30
   "JSONCodeSet"      : 1
   "NotifyRewrite"    : 1
   "NullParamStr"     : "NULL"
   "DiscloseTarget"   : 1
}
```

In the Defaults object definition, the **object** key value is mandatory and must be **Defaults**. Although the other keys are optional, we recommend specifying all of them in the object definition. These keys include:

- **LdapProxy.lb_failover**, which determines whether or not the DLP server contacts the next server from the **LBServer** list defined in the LdapProxy object using a round-robin scheme. By default (**LdapProxy.lb_failover** is set to **1**), if more than one LB-server is configured and an I/O error occurs to a selected target server, the DLP server fails over

to the next server. If this one also fails, it fails over to the next, and so on until all servers have been tried. To change this behavior, set the property **LdapProxy.lb_failover** to **0**. Now the operation returns an error if the selected target server raises an I/O error.

- The **LdapProxy.lb_balance**, which determines whether or not the DLP server applies a round-robin scheme to select the primary server from the **LBserver** list specified in the LdapProxy object definition. By default (**LdapProxy.lb_balance** set to of **1**), if more than one LB-server is configured, each user that has no other governing rule is forwarded to one of the LB-servers using round-robin selection. For example, suppose C1, C2, C3, C4 are LDAP client users that are not controlled by other rules (the numbers indicate the time sequence in which the users perform their binds) and **LBservers** lists LDAP1, LDAP2, and LDAP 3 servers. DLP server forwarding will be C1 to LDAP1, C2 to LDAP2, C3 to LDAP3, C4 to LDAP1. To change this default behavior, set the **LdapProxy.lb_balance** property to **0**. Now the DLP server forwards every user request that has no governing rule to the same primary server (the first server in the **LBserver** list, which is **LDAP1** in this case).
- **TLSLogging**, which controls whether (**1**) or not (**0**) the DLP server generates a readable log file *install_path*/ldap/log/proxyssl**pid*.txt* where *pid* is the PID of the DLP server process. (The default value is **0**.) Note that enabling **TLSLogging** will slow down performance significantly and is only intended for error analysis.
- **Tracing**, which controls whether (**1**, **2** or **3**) or not (**0**) tracing information is sent to **stderr** and the verbosity of tracing information sent (**1**, **2** or **3**). The higher the value, the more verbose the tracing information generated. (The default value is **0**.) Enabling **Tracing** will slow down performance noticeably.
- **ConnectTimeout**, which defines the number of seconds the DLP server waits for a TCP connect to succeed before it tries another server (if available and failover is allowed) or the operation fails. The default value is **5** seconds. Be careful not to supply a value that's too low, as it may lead to frequent timeouts when network performance is poor or the contacted target server is under high load. For a description of how this setting affects DLP server operation, see the section "Connect Timeout".
- **OfflineRetryTimeout**, which defines the number of seconds for which the DLP server disables a failed target server for further selection. The default value is **60** seconds. For a description of how this setting affects DLP server operation, see the section "Offline Handling and Server Retry".
- **JSONCodeSet**, which specifies the code set in which the DLP server configuration file is encoded (the configuration file is a JSON file, as described in the "Configuration" chapter). A value of **0** indicates to the DLP server that the configuration file is encoded in Latin-1 and needs to be converted to UTF-8 format for LDAP, which only supports the UTF-8 code set. A value of **1** indicates that the configuration file is encoded in UTF-8 and needs no conversion. The default value is **1** (UTF-8). For a description of how this setting affects DLP server operation, see the "Operations" chapter.
- **NotifyRewrite**, which controls whether (**1**) or not (**0**) notification of request/result rewriting is included in the LDAP response error message. When set to **1**, the client receives a string like **Proxy-Modified=yes**, which indicates that the DLP server has modified the request or the result. Set this key to **0** to hide information from clients about DLP server changes to requests and results.
- **NullParamStr**, which defines a placeholder string for an empty rewrite action

parameter in a DLP server rewriting rule. The default placeholder string is NULL. For a description of how this setting applies to DLP server rewriting rules, see the section “Protocol-Specific Actions” in the chapter “Proxy Rules”.

- **DiscloseTarget**, which controls whether (**1**) or not (**0**) the DLP server identifies the target server that processed the LDAP request in the LDAP response message. For example, when set to 1, the following output is returned for the following **dirxcp** command:

```
dirxcp> bind -prot ldapv3 -user cn=richter,ou=sales,o=pqr -auth
simple -address localhost:8080 -pass abc123
{{LDAP-Result: Bind succeeded. (Proxy-Target:LDAP2)}}
```

Note that **TLSLogging** and **Tracing** produce a lot of output and slow down performance drastically. They are intended to be used only for error analysis and not for normal operation.

2.3.5. The ProxyRule Object

The ProxyRule object is an optional object in the DLP server configuration file that specifies a rule to be applied to an LDAP operation. A ProxyRule object can specify:

- Rules for operation forwarding based on the user’s DN or on the operation type
- Rules for rewriting an LDAP client request or the result returned by the target LDAP server.

Here are examples of ProxyRule object definitions for each of these rules:

```
{
   // user-routing rule: redirect user ‘richter’ to LDAP1
   "object"      : "ProxyRule",
   "ruleType"    : "UserRouting",
   "name"        : "USERROUTING1",
   "condition"   : "(user=cn=richter,ou=sales,o=pqr)",
   "actions"     : [ "forwardto(LDAP1)" ],
   "loadbalance" : 1,
   "failover"    : 1
},

{
   // operation-routing rule: redirect ADDs to LDAP2
   "object"    : "ProxyRule",
   "ruleType"  : "OprRouting",
   "name"      : "OPRROUTING2",
   "condition" : "opr.req.type=add",
```

```
   "actions"   : [ "singleforwardto(LDAP2)" ],
   "failover"  : 0,
   "keepconn"  : 1
},

{
    // request rewriting rule: change o=my-company to o=pqr for
search bases
   "object"    :  "ProxyRule",
   "ruleType"  :  "ReqRewrite",
   "name"      :  "ReqRewrite1",
   "condition" :  "opr.req.type=search",
   "actions"   :  [ "search.req.baseObject.replace(o=my-
company,o=pqr,NULL)" ]
}

{
   // result rewriting rule: remove sn=Digger from any search result
   // add description=blabla to all result entries
   "object"    :  "ProxyRule",
   "ruleType"  :  "ResRewrite",
   "name"      :  "Test 04",
   "condition" :  "opr.req.type=search",
   "actions"   :  [ "search.res.attributes.del(sn,Digger,NULL)",
                  "search.res.attributes.add(description,blabla,NULL)"
]
}
```

In the object definition:

- The **object** key is mandatory and must be the string **ProxyRule**.
- The **ruleType** key is mandatory and must be one of the following strings:
- **UserRouting** - defines a rule for operation forwarding based on the user's DN
- **OprRouting** - defines a rule for operation forwarding based on the operation type
- **ReqRewrite** – defines a rule for modifying the data in an incoming client request
- **ResRewrite** – defines a rule for modifying the data in an outgoing result of a client request
- The **name** key is mandatory and can have any string value as long as the value is unique.
- The **condition** key is mandatory and specifies the selection criteria that an LDAP operation must satisfy in order for the actions specified in the **actions** key to be

executed on the operation. Condition values are specified in LDAP filter string format; the value depends on the value of the **ruleType** key:

- **UserRouting**, **OprRouting** - the condition value selects for forwarding the LDAP operation to a target server
- **ReqRewrite**, **ResRewrite** - the condition value selects for modifying a component of the LDAP operation

Rule conditions follow the syntax of LDAP strings and are evaluated just like LDAP filters (including nested filters with **and**/**or**/**not**). The only difference is that the assignments in the filter are well-defined keywords (for example, **wcuser** or **opr.req.type**) instead of attribute types. See the chapter “Proxy Rules” for details on how to specify rule conditions for each rule type.

- The **actions** key is mandatory and defines the operations to be performed on the LDAP operation if it matches the rule condition selection criteria. An action is defined as a JSON array; the value depends on the **ruleType** value:
- **UserRouting**, **OprRouting** – the action is a forwarding operation to a selected set of LDAP servers or a general action, such as denying the request. See the descriptions of the user- and operation-routing rules in the chapter “Proxy Rules” for details on how to specify a forwarding action.
- **ReqRewrite**, **ResRewrite** – the action is a rewriting operation for a specific component of an LDAP operation – for example, to change the base object of an incoming search request - or a general action, such as denying the request. Multiple actions can be defined for an individual rule and are executed in the order in which they are defined. See the description of the rewriting rules in the chapter “Proxy Rules” for details on how to specify rewriting actions.

A ProxyRule object definition can contain additional keys depending on the value of the **ruleType** key. See the descriptions of the different rule types in the chapter “Proxy Rules” for details.

The configuration file can contain any number of ProxyRule object definitions. More than one rule may be applied to an operation if it matches multiple rule conditions. Rules are evaluated and executed in the order in which they appear in the configuration file. See the chapter “Proxy Rules” for examples of this process.

The DLP server allows Latin-1 characters like the German umlaut to be used in proxy rule conditions and actions and converts them to UTF-8 format before passing them to LDAP, which only supports the UTF-8 encoding. The value of the **JSONCodeSet** key in the Defaults object definition indicates to the DLP server whether or not it should make this conversion. Consequently, the ProxyRule object definitions in a DLP server configuration file must all be in the same character set.

Rule conditions and actions may need to use characters that LDAP, JSON or the proxy rule action syntax define as special characters. These characters require special handling in the proxy rule definition. See the section “Handling Special Characters in Rule Conditions and Actions” in the chapter “Proxy Rules” for a description of these special characters and how to specify them in proxy rule definitions.

3. Proxy Rules

A DLP server proxy rule consists primarily of a condition and a set of actions to be executed by the DLP server if the data in the running operation match the condition. Proxy rule types include:

- User-routing rules, where the target server selection is made according to an explicit user DN, a regular expression for a user's DN and/or a node DN below which a user resides. User-routing rules can be defined, for example, to redirect groups of users to a specific target server or servers. Note that if the DN represents a group, the DNs inside the group are *not* dereferenced; that is, users inside a group are not implicitly used if the DN describes a group.
- Operation-routing rules, where the target server selection is made according to the type of operation to be forwarded. Operation-routing rules can be defined, for example, to direct all modification operations to a master server while searches are directed to a shadow server, or to direct all searches with a particular base object to a dedicated server while all other operations are forwarded to another target server.
- Request-and result-rewriting rules, where modifications to an operation's data are made according to the actions defined in the rule if the running operation matches the condition defined in the rule. Rewriting rules can be defined, for example, to update attributes like organization names associated with communication between legacy clients and a directory service or to remove sensitive information returned in search results before returning them to clients.

The **ProxyRule** DLP server configuration object defines a proxy rule. Multiple proxy rules can be defined; the DLP server evaluates and executes proxy rules in the order in which they appear in the DLP server configuration file.

This chapter provides detailed syntax and operational information for each proxy rule type and also provides information about special character handling that applies to all proxy rule types. The section "The ProxyRule Object" in the "Configuration" chapter provides syntax and operational information that is common to all proxy rule types.

3.1. User-routing Rules

The **UserRouting** proxy rule type defines an operation-forwarding rule for one explicitly-specified user, for users whose DN match a regular expression (wildcard) or for users below a given node in the DIT.If the DN of the bound user matches the configured condition in this rule, the rule is applied.The next sections describe how to specify user-routing rules and how the DLP server processes them.

3.1.1. Syntax Description

User-routing rules are specified as ProxyRule objects in the DLP server configuration file.Here is an example of a user-routing rule:

```
{
```

```
        "object"      : "ProxyRule",
        "ruleType"    : "UserRouting",
        "name"        : "USERROUTING1",
        "condition"   : "(|(user=cn=richter,ou=sales,o=my-
company)(wcuser=^cn=D.*o=my-company)(subuser=ou=sales,o=my-
company))",
        "actions"     : [ "forwardto(LDAP2,LDAP1)" ],
        "loadbalance" : 0,
        "failover"    : 1
}
```

In the object definition:

- The **object** key is mandatory and must be the string **ProxyRule**.
- The **ruleType** key is mandatory and must be **UserRouting**.
- The **name** key is mandatory and can have any string value as long as the value is unique.
- The **condition** key is a mandatory LDAP filter string in the format *token assertion value* that describes the criteria that the user DN in the incoming **bind** operation must satisfy in order for the DLP server to perform the forwarding action defined in the rule.

For user-routing rules, *token* is one of the following user classes:

- **user** – the rule is to be applied to one explicitly specified user. If the DN of the bound user matches the configured DN in this rule, the rule is applied. The value must be a syntactically legal LDAP DN. The rule will be selected according to this DN when this user invokes an LDAP bind operation. You can specify the string **anonymous** for the user DN to define a rule that applies to anonymous users that do not perform a bind operation or bind as an anonymous user. As authentication via method EXTERNAL SASL is not supported, users identified by their X.509 certificate cannot be used.
- **subuser** – the rule is to be applied to the users below a given node in the DIT. The subuser rule works like the user rule except that instead of defining a single user, all users below the specified node are affected. However, if a user in the specified subtree has an explicit **user** rule assigned, this specific rule overrides the **subuser** rule because a **user** rule takes precedence over a **subuser** rule.
- **wcuser** – defines a rule for users whose DN match a regular expression (wildcard). This rule works exactly like the **user** rule except that the user's DN is matched against a regular expression given in the **wcuser** value. The regular expression follows the Linux/Perl regular expression syntax and is performed case-insensitive.

Although *assertion* can be any valid LDAP assertion (see the section "Rewriting Rules → "The Condition Key" → "Condition Rule Syntax" for the list of supported and unsupported assertions), the power of using the wildcard user (**wcuser**) condition, where you can assign almost any kind of abbreviation for a DN, means that the **equal** assertion is the most meaningful assertion to use in user-routing rules.

In the example **wcuser=^cn=D.*o=my-company** above, users like

```
cn=Digger,ou=development,o=my-company
```

```
cn=Digger,ou=sales,o=my-company
```

```
cn=Digger,ou=support2,ou=support1,o=my-company
```

match the rule.

- The **actions** key is mandatory. For user-routing rules, the **forwardto** action is the only action supported (besides the general action **denyreq**) and defines a JSON array of LDAP target servers enclosed in square brackets [ ]. The target servers are indicated by their names and must match the name of an LDAPserver object definition in the same configuration file. At least one name must be present in the array (an empty array is not allowed). The first server given in the list is considered to be the default primary server. Be sure to avoid leading and trailing blanks in LDAP server names as they will be interpreted as part of the name.

User-routing rules can also specify the **denyreq** general action. For more information on what this action does, see “Rewriting Rules” → “Syntax Description” → “The actions Key” → “General Actions”.

- The **loadbalance** key is optional and defines how the DLP server handles the servers listed in the **forwardto** action during primary server selection (the first server to be contacted from the list). The DLP server does not perform load balancing by default: the first server from the list is always the primary server. To change the default behavior, specify the **loadbalance** key and set it to **1**. Now, for every new bind from the user, the DLP server selects the next server from the list using a round-robin scheme.
- The **failover** key is optional and defines how the DLP server handles the servers listed in the **forwardto** action when an error occurs while sending a request to the target server. By default, there is a failover to the next server if an error occurs with the selected target server. To change the default behavior, specify the **failover** key and set it to **0**. Now the operation will fail if the primary server fails.

The default behavior for **loadbalance** and **failover** differs between **user**, **subuser** and **wcuser** rule conditions. The reason for this difference is the general assumption that multi-user rules (**subuser** and **wcuser**) will generate too much traffic for a single target server and thus load balancing is the desired behavior. In contrast, the traffic from a single user is not so relevant to the total server load.

If failover is active and all configured servers fail, the entire operation fails.

The section “How User-routing rules are Processed” provides more information about how the DLP server selects user-routing rules and target servers given load-balancing and failover settings.

Each LDAP client has its own backend LDAP connection; no backend sharing takes place even if credential match or backend sharing has been configured in the LDAP

configuration subentry. Once established, the backend connection to the selected target remains open until the client performs an **unbind** operation, drops the connection to the DLP server, the target server closes the connection (for example, because of client-idle-timeout) or a network error occurs. Due to the one-to-one relationship between frontend client connections and backend server connections, the possible number of parallel client connection depends on the number of available socket descriptors.

3.1.2. How User-routing Rules are Processed

A target server from a user-routing rule is selected by one the following events:

- A client performs a **bind** operation. It doesn't matter whether the bind operation is the first operation on a LDAP connection or if it occurs at later time on the existing connection.

  Whenever the DLP server detects a bind, it searches for a user-routing rule for the user defined by the incoming bind operation. If the DLP server finds a rule, it sets the target server according to the rule. If a target server has already been selected (for example, because a **bind** occurred earlier on this connection) the DLP server drops the existing connection and uses the target server from the last bind operation for all subsequent operations; that is, a total target shift is performed. If the new target server(s) cannot be connected, the existing connection remains intact and all further operations will go to the old target (shift on success).

- The first operation on a newly created LDAP connection is not a **bind**. In this case, the DLP server assumes an anonymous user, searches for a user-routing rule for anonymous and sets the target server accordingly. If no rule is found, the DLP server uses the LB-servers as a fallback.

The DLP server evaluates user-routing rules in the order in which they are specified in the DLP server configuration file; that is, it evaluate the first user-routing rule if finds, then the second and so on.

If a condition matches the user, the DLP server stops searching and assigns the servers from the corresponding user-routing rule to the user as new target servers for LDAP requests. The servers from a matching user-routing rule are called the "relevant servers" for the current user. Note that servers that have been previously detected to fail or are marked as offline are automatically excluded from the relevant servers.

In a list of servers, the server that the DLP server selects first is defined by the **loadbalance** option in the rule.A value of **0** specifies that the first server in the list is always selected for the requests.A value of **1** specifies that a target server is to be selected using a simple round-robin selection process.

Once a target server is selected and successfully contacted, all subsequent requests are sent to the same selected server until a new **bind** operation is performed on the same LDAP connection or the connection ends.If the DLP server cannot contact a target server or an established connection to this server fails, the **failover** option defines whether or not the next server from the relevant servers list is selected and the request resubmitted to this server.If **failover** is set to **1** and the connection to a server breaks, the next server from the list is selected as the new target server and all subsequent requests are sent to this new

server (until this server also fails) This automatic reselection is transparent to the client and continues until all relevant servers have been contacted and have failed at least once.

For example: Let's assume that the list of target servers for user X is **LDAP1**, **LDAP2**, **LDAP3** and that **failover=1** and **loadbalancing=1** are set.

When user X binds for the first time **LDAP1** is selected.If **LDAP1** can be contacted successfully, LDAP1 is the target server for all subsequent operations.

After performing some operations, user X closes his connection (**unbind**).Sometime later, user X again performs a **bind** and **LDAP2** is selected (**loadbalancing=1**) but now **LDAP2** is down and cannot be reached.As **failover=1** is set, the next selected target server is **LDAP3**, which is up and the bind succeeds.Again, user X performs some operations against **LDAP3** and finally closes the connection.A few moments later, user X binds again and because **loadbalancing=1**, **LDAP3** is selected (last time, **LDAP2** was selected due to load-balancing and then failed; failover occurred but the selection for load-balancing is not influenced by failover) Thus we now have **LDAP3** again as the target server for all remaining operations.This example shows that load-balancing selection and failover selection are independent from each other although they select from the same list.

3.2. Operation-routing Rules

The **OprRouting** proxy rule type defines an operation-forwarding rule for one explicitly-specified LDAP operation.Operation-routing rules can direct single operations, based upon the request parameters, to dedicated target servers.If the operation matches the configured condition in the rule, the rule is applied.

3.2.1. Syntax Description

Operation-routing rules are specified as ProxyRule objects in the DLP server configuration file.Here is an example of an operation-routing rule:

```
{
        "object"      : "ProxyRule",
        "ruleType"    : "OprRouting",
        "name"        : "OPRROUTING2",
        "condition"   :
"(&(opr.req.type=search)(search.req.baseObject=o=pqr))",
        "actions"     : [ "singleforwardto(LDAP1,LDAP3)" ],
        "failover"    : 1,
        "keepconn"    : 1
}
```

In the object definition:

- The **object** key is mandatory and must be the string **ProxyRule**.

- The **ruleType** key is mandatory and must be **OprRouting**.
- The **name** key is mandatory and can have any string value as long as the value is unique.
- The **condition** key is a mandatory LDAP filter string in the format *token assertion value* that describes the criteria that the running operation must satisfy in order for the DLP server to perform the action on the operation defined in the rule. See the condition syntax description in the section “Rewriting Rules” → “Syntax Description” → “The Condition Key” → “Condition Rule Syntax” for details about this format as it applies to operation-routing rules and rewriting rules.

For operation-routing rules, *token* can be any of the tokens described in the table “Condition Token Names and Assignments” provided in the section “Rewriting Rules”. Note that condition keys that specify bind operations - for example, **opr.req.type=bind opr.req.type=*** - are not permitted to be used in operation-routing rules because bind operations are handled separately by the user-routing rules and specifying them here may lead to unwanted behavior.

- The **actions** key is mandatory. For operation-routing rules, the **singleforwardto** action is the only action supported (besides the general action **denyreq**) and defines a JSON array of LDAP target servers enclosed in square brackets [ ]. The target servers are indicated by their names and must match the name of a definition of an LDAPserver object in the same configuration file. At least one name must be present in the array (an empty array is not allowed). The first server given in the list is considered to be the primary server. The next servers are only contacted if the previous servers are unreachable and the **failover** key is set to **1**. Avoid leading and trailing blanks in LDAP server names because they will be interpreted as part of the server names.

Operation-routing rules can also specify the **denyreq** action. For more information on how this action works, see “Rewriting Rules” → “Syntax Description” → “The actions Key” → “General Actions”.

- The **failover** key defines how the DLP server handles the servers listed in **singleforwardto** when an error occurs while sending a request to the target server. If **failover** is set to **1** the request is sent to the next server if an error occurs with the selected target server. To change this behavior, set the **failover** key to **0**. Now the operation will fail if the primary server fails.
- The **keepconn** key defines whether (**1**) or not (**0**) the connection to the selected target server is to remain open after the operation is performed. Keeping the new connection open may be important if the operation performed was a paged search: the cookies for paged search are maintained by the servers only for the same connection, so closing a new connection that was caused by an operation-routing rule will break subsequent paged searches. Note: If **keepconn** is set to **1** for an operation, subsequent operations for which no other operation-routing rule is defined will use the connection that has been kept open.

Note, too, that when **keepconn=1** is set and the operation governed by an operation rule succeeds, the target server that was selected by the bind operation is overlaid with the target from the operation-routing rule and therefore becomes obsolete. For more information about this behavior, see the examples in the chapter "Examples and

Considerations".

Note that operation-routing rules are applied before any defined rewriting rules, so target server selection is based on the original LDAP input from the client.

3.2.2. How Operation-routing Rules are Processed

When a client connects to the DLP server for the first time (mostly via the bind operation), a user-routing procedure is performed that determines the target server for all operations of this user.

Target server selection is governed either by an existing user-routing rule or, if no suitable rule is found in the list of user-routing rules, by the default LB-server selection.

The result of this process is a set of 1-*n* target servers that are used whenever the user issues an operation that does not match an operation-routing rule; in other words, an operation-routing rule overrides the target server settings from the user-routing rule.

Thus, when an operation comes in from a user, the DLP server first determines whether there is a matching operation-routing rule and if so, uses the target servers from this rule. If there is no matching operation-routing rule, the DLP server uses the user-routing target servers as selections for forwarding the request.

When the DLP server searches for a matching operation-routing rule, it processes the operation-routing rules in the order in which they are defined in the DLP server configuration file and stops searching when it finds a match. Consequently, we recommend placing the rules with specific conditions in the configuration file before the more general ones.

For example, suppose there are two operation-routing rules ORR#1 and ORR#2:

```
ORR#1 (opr.req.type=search)

ORR#2 (&(opr.req.type=search)(search.req.baseObject=o=my-company))
```

In this example, ORR#2 should appear before ORR#1 in the DLP server configuration file, otherwise the more specific ORR#2 will never be reached, as **opr.req.type=search** is true for both rules but the search will stop after the first match.

Note that the same restrictions apply for operation-routing conditions as for **ReqRewrite** rule conditions.For example, you are not allowed to OR the **opr.req.type** token with other tokens, for example:

```
(|(opr.req.type=search)(user=cn=admin,o=my-company))
```

See “Rewriting Rules” → “Syntax Description” → “The condition Key” → “Using the opr.req.type Token” for more information.

3.3. Rewriting Rules

The **ReqRewrite** and **ResRewrite** rule types define rules for changing data in client requests and server responses.

3.3.1. Syntax Description

Request and result rewriting rules are specified as ProxyRule object definitions in the DLP server configuration file.This section describes their syntax, while the chapter “Examples and Considerations” provides several examples of their function.

3.3.1.1. The object Key

The **object** key is mandatory and must be the string **ProxyRule**.

3.3.1.2. The ruleType key

The **ruleType** key is mandatory; for rewriting rules, it determines when the DLP server is to invoke the rule. Use the string **ReqRewrite** for a rule to be invoked on LDAP requests. Use the string **ResRewrite** for a rule to be invoked on LDAP results. **ReqRewrite** rules are applied *before* the request is forwarded to an LDAP server. **ResRewrite** rules are applied *after* the result is received from the LDAP server and before it is returned to the client. Although it is syntactically allowed to have actions that operate on request parameters in **ResRewrite** rule types, they have no effect on the result since changing the request *after* the result is received does not change anything in the result for the client. Note that it is the rule condition that determines whether or not the rule is applied; the **ruleType** value simply determines when a rule is invoked.

3.3.1.3. The name Key

The **name** key is mandatory and can have any string value as long as the value is unique. The name appears in DLP server audit records to indicate which rules were applied to a specific request. We recommend using descriptive names like **SetSearchSizeLimitTo500** to make it easier for offline analysis of traffic in audit records later on.

3.3.1.4. The condition Key

The **condition** key is mandatory and defines whether or not a rule will be applied to the running operation. For rewriting rules, a condition can be seen as a further refinement that tells the DLP server how the incoming request or received result must appear in order to be modified by one or more actions.

A simple condition can look something like this:

```
"(&(opr.req.type=search)(search.req.baseObject=o=my-company))"
```

This condition defines two sub-conditions that must both match in order to execute the rule actions:

1. opr.req.type=search

2. search.req.baseObject=o=my-company

Because the two sub-conditions are combined with a logical **and** (**&**), both must evaluate to **true** to create a match.

The next sections explain the syntax for condition rules in more detail.

3.3.1.4.1. Condition Rule Syntax

For rewriting rules and operation-routing rules, a condition rule is an LDAP assertion in the format:

token assertion value

where:

token is a predefined string that represents an LDAP PDU operation. The predefined token **opr.req.type** is a mandatory element of an operation-routing or a rewriting condition rule and defines the operation type for which the rule is targeted; for example, **opr.req.type=search**. The sections on token syntax provide more detail about how to use **opr.req.type** and list the predefined tokens available for use in operation-routing and rewriting rule conditions.

assertion is one of the LDAP assertion operators **equal**, **present**, **substr** (**initial**, **final**, **any**), **lessorequal** and **greaterorequal** (**approx** and **ext_match** are not supported) in LDAP filter notation syntax:

equal	***=****value*
present	**=***
initial (begins with)	***=value*** **
final (ends with)	***=****value*
any (contains)	***=value***
lessorequal	***⇐****value*
greaterorequal	***>=****value*

Rule conditions follow LDAP filter semantics:

- You can use complex combinations of **and**, **or** and **not** sub-filters items within a condition.
- You can use substring values like **initial**, **final** and **any** as well as **present** items.
- **lessorequal** and **greaterorequal** are also supported.

Approximate and extensible filters are currently unsupported.

The DLP server recognizes all assigned values as strings and automatically converts them to integer for comparison if necessary. For example, the DLP server automatically recognizes a condition like:

`search.req.sizeLimit>=1000`

as an integer with a value of **1000** when comparing the request against the condition.

You are responsible for assigning meaningful values to the LDAP PDU components mapped by the pre-defined tokens, or undefined behavior may occur. For example, the condition definition

`search.req.baseObject=700`

is not meaningful and should be avoided.

The DLP server does not perform any syntax checking or matching rule compliance on any assigned value. The server only compares strings. Consequently, a condition like **search.req.baseObject=700** will be accepted but during comparison it will be compared to the DN **baseObject** of the request as a string of value **700** which definitely will never match.

3.3.1.4.2. Condition Token Syntax

The general token syntax for addressing a specific LDAP component in a rule condition is:

operation-type.{**req**|**res**}.*component-name*

where

operation-type specifies the LDAP operation (**bind**, **search**, **modify**, **add**, **delete**, **modDN**, **compare**)

req|**res** specifies whether it is targeted for the request (**req**) or the result (**res**)

component-name specifies the component name as provided in the LDAP standard RFC4511ff.

Here are some examples:

`search.req.baseObject addresses the requested baseObject`

`search.res.attributes addresses the resulting attributes`

3.3.1.4.3. Using the opr.req.type Token

The **opr.req.type** token is a special token that defines the targeted operation type (for example, a **search**). This token must be present in a rule condition to specify the operation type for which the rule is targeted. The absence of **opr.req.type** is considered to be an error.

The value for the **opr.req.type** token represents the necessary LDAP operations and can be one of

bind, **search**, **modify**, **add**, **delete**, **modDN**, **compare**

or "*****" to indicate that the condition can match with any of the operation types. Note that the values for **opr.req.type** are case sensitive.

Using the **OR** operator to **OR** the **opr.req.type** token to other tokens in a rule condition is not allowed due to the token's special status and function. If you want the same actions to be performed for different operation types – for example, for add or for modify – you must write separate rules for each operation type. For example, the following rule is not allowed:

Condition: **(|(opr.req.type=add)(opr.req.type=modify))**
Action: **"denyreq"**

The following rules are allowed:

Condition: **(opr.req.type=add)**
Action: **"denyreq"**

Condition: **(opr.req.type=modify)**
Action: **"denyreq"**

We recommend keeping rule conditions simple, not because of easier parsing, but to keep track of the existing rules.

3.3.1.4.4. Condition Token Names and Assignments

The following table shows the token names and assignments that are supported for rewriting and operation-routing rule conditions. Assignments can be: **E** (**equal**), **SI** (**substring-initial**), **SF** (**substring-final**), **SA** (**substring-any**), **P** (**present**), **LE** (**lessorequal**), **GE** (**greaterorequal**):

Token Name	Supported Assignments	Description
opr.req.type	E, P	The operation type for which this condition is targeted. The value must be one of **bind**, **search**, **add**, **modify**, **delete**, **compare** or **modDN**. If the value is "**", any operation type will match. This token is *mandatory* for a condition. Example: *opr.req.type=search**.
user	E, SI, SA, SF, P	The user for which the condition is targeted. The user-name is determined by the last successful bind operation or is "anonymous" if no successful authenticated bind was performed. The value must be a legal LDAP distinguished name of a user; for example: **cn=richter,ou=sales,o=my-company**. If the value is "**", any user will match. If the value is "anonymous", the unauthenticated user will match. Example: *user=*cn=admin*** (any user that has **cn=admin** in its name).

Token Name	Supported Assignments	Description
ip	E	The IP address for which the condition is targeted. The IP is expected to be a valid IPv4 address; for example, **111.22.33.76**. If the value is "**...**", any IP address will match. You can also specify subnet IP ranges like **ip=192.33.**.** which evaluates the condition to match for all IP addresses from the B-subnet 192.33. Only IPv4 addresses are currently supported.
security	E	The security level of the operation to be processed. Possible values are **plain** or **tls**. Example: the condition **(&(opr.req.type=**)(security=plain))* combined with the action **denyreq** rejects incoming client operations unless they are received via the secure TLS channel.
bind.req.name	E, SI, SA, SF,P	The string value that must be present in the bind- DN. If the value is "**", any DN will match except anonymous users. If the value is "anonymous", the unauthenticated user will match. This token is very similar to the *user** token except that this one is only allowed for bind operations. Example: **bind.req.name=*o=my-company*** matches to all bind requests for users with a component **o=my-company** in their bind-DN name; that is, it matches to users like **cn=admin,o=my-company** or **cn=richter,ou=sales,o=my-company** but not to **cn=admin,o=pqr**.
search.req.baseObject	E, SI, SA, SF,P	The baseObject that must be present in an incoming search request to match the condition. The baseObject must be specified as a legal LDAP DN; for example, **search.req.baseObject=ou=sales,o=my-company**. The value can also be a substring of a DN; for example, **o=my-comp*** which matches to a baseObject **o=my-company** via an INITIAL substring match. You can also use FINAL and CONTAINS substrings.

Token Name	Supported Assignments	Description
search.req.scope	E	The scope that must be present in an incoming search request to match the condition. The value must be one of • baseObject • singleLevel • wholeSubtree This token is only applicable to search operations.
search.req.sizeLimit	E, LE, GE	The sizeLimit value that must be present in an incoming search request to match the condition. The assigned value can either be an **equal**, **lessorequal** or **greaterorequal** assignment; for example, **search.req.sizeLimit>=1000**. The assigned string (here, **1000**) is automatically converted and compared to the integer value of sizeLimit in the incoming LDAP PDU.
search.req.timeLimit	E, LE, GE	The timeLimit value that must be present in an incoming search request to match the condition. The assigned value can either be an **equal**, **lessorequal** or **greaterorequal** assignment; for example, **search.req.timeLimit⇐1**. The assigned string (here, **1**) is automatically converted and compared to the integer value of **timeLimit** in the incoming LDAP PDU.

Token Name	Supported Assignments	Description
search.req.attributes	E	The name of a requested attribute that must be present in an incoming search request to match the condition. The assigned value must be a legal LDAP attribute name. Multiple attribute names must be added to the condition by adding another AND item, for example, **(&(search.req.attributes=street)(search.req.attributes=sn))**, which evaluates to true if the incoming search contains both **street** *and* **sn** in the list of requested attributes.  if the incoming request contains shortcuts like “**” (for all user attributes) or “+” (for all operational attributes), no match against explicit values will be performed that might be contained within these shortcuts; that is, no match is detected if the incoming request contains “**” and the condition contains an explicit name from the user-attribute list. Example: if the condition is **search.req.attributes=title** but the incoming request contains “**”, no match will occur even though *title** is implicitly contained in the all-user attributes list defined by “*”.
search.req.control	E	The LDAPV3 control that must be present in the incoming search request. Possible values are **simplePagedResult** or **serverSideSorting**. These values can be abbreviated to **PR** and **SSS**.

Token Name	Supported Assignments	Description
modify.req.object	E, SI, SA, SF	The name of the target entry that must be present in the incoming request. The assigned value must be a legal LDAP DN; for example: **modify.req.object=cn=richter,ou=sales,o=pqr**. The assigned value can also consist of a combination of one or more substrings; for example, **modify.req.object=*richter*** or **modify.req.object=cn*abele*o=pqr** which evaluate to a filter expression of three substrings: initial : cn any : abele final : o=pqr This condition means that the incoming target entry name of the modification must begin with **cn**, must contain the substring **abele** and must end with **o=pqr**. The condition will evaluate to true only if all three conditions match.

Token Name	Supported Assignments	Description
add.req.entry	E, SI, SA, SF	The name of the target entry that must be present in the incoming request. The assigned value must be a legal LDAP DN; for example: **add.req.entry=cn=richter,ou=sales,o=pqr** The assigned value can also consist of a combination of one or more substrings; for example: **add.req.entry=*richter*** or **add.req.entry=cn*abele*o=pqr** which evaluates to a filter expression of three substrings: **initial** : **cn** **any** : **abele** **final** : **o=pqr** This condition means that the incoming target entry name of the add operation must begin with **cn**, must contain the substring **abele** and must end with **o=pqr**. The condition evaluates to true only if all three substring conditions match.

Token Name	Supported Assignments	Description
delete.req.entry	E, SI, SA, SF	The name of the target entry that must be present in the incoming request. The assigned value must be a legal LDAP DN; for example: **delete.req.entry=cn=richter,ou=sales,o=pqr**. The assigned value can also consist of a combination of one or more substrings; for example: **delete.req.entry=*richter*** or **delete.req.entry=cn*abele*o=pqr*** **which evaluate to a filter expression of three substrings:** ***initial** : **cn** **any** : **abele** **final** : **o=pqr** This condition means that the incoming target entry name of the delete operation must begin with **cn**, must contain the sub-string **abele** and must end with **o=pqr**. The condition evaluates to true only if all three substrings match.

Token Name	Supported Assignments	Description
modDN.req.entry	E, SI, SA, SF	The name of the target entry that must be present in the incoming request. The assigned value must be a legal LDAP DN; for example: **delete.req.entry=cn=richter,ou=sales,o=pqr** The assigned value can also consist of a combination of one or more substrings; for example: **delete.req.entry=*richter*** or **delete.req.entry=cn*abele*o=pqr** which evaluates to a filter expression of three substrings: **initial** : **cn** **any** : **abele** **final** : **o=pqr** This condition means that the incoming target entry name of the **modDN** operation must begin with **cn**, must contain the sub-string **abele** and must end with **o=pqr**; the condition evaluates to true only if all three substring conditions match.

Token Name	Supported Assignments	Description
compare.req.entry	E, SI, SA, SF	The name of the target entry that must be present in the incoming request. The assigned value must be a legal LDAP DN; for example: **delete.req.entry=cn=richter,ou=sales,o=pqr** The assigned value may also consist of a combination of one or more substrings, e.g. **delete.req.entry=*richter*** or **delete.req.entry=cn*abele*o=pqr*** **which evaluates to a filter expression of three substrings:** ***initial** : **cn** **any** : **abele** **final** : **o=pqr** This condition means that the incoming target entry name of the compare operation must begin with **cn**, must contain the substring **abele** and must end with **o=pqr**. The condition evaluates to true only if all three substring conditions match.
compare.req.attr	E	The name of the attribute in the incoming request that is to be compared.

LDAP search filters cannot be configured as conditions for proxy rules.

3.3.1.5. The actions Key

Actions operate on the request or the result. Actions can be general (for example, to reject the entire request) or they can be specific to the LDAP PDU components (for example, to change the **baseObject** of an incoming search request).

A general action is described by a simple token like **denyreq** while a protocol-specific action has a more complex syntax described in this section.

Every proxy rule can have 1-*n* actions. If multiple actions are defined for a rule, they are executed in the order in which they are defined within the rule (top-down). The input for a subsequent action is the result of the all previous actions performed and not the original request, that is

OriginalReq → Action1 → ChangedReq1 → Action2 → ChangedReq2 → .....

Consequently, when you are defining multiple actions in a rule, make sure that each action

definition considers the actions that precede it.

3.3.1.5.1. General Actions

The only general action currently supported for all rule types is **denyreq**, which rejects the incoming request with an UNWILLING_TO_PERFORM error. The operation is not forwarded to any LDAP server. The DLP server stops its processing operation when it detects a **denyreq** action within a sequence of actions. Be careful when using the **denyreq** action with general conditions. For example, consider the following condition/action pair:

```
condition : (&(opr.req.type=)(user=))
action : denyreq
```

This combination effectively locks out all users and all operations from the server.

3.3.1.5.2. Protocol-Specific Actions

Rewriting rules can specify protocol-specific actions to execute rewriting operations on specific LDAP components within a request or result. The first three elements of a protocol-specific action definition specify the component to be modified and mirror the condition rule syntax. The fourth element is the action to be performed (for example, replace) and the last three elements are the necessary parameters for the action.

Protocol-Specific Action Syntax

The general token syntax for addressing a specific LDAP component in a rule condition is:

operation-type.\{**req**|**res**}.*component-name*.*action*(*parameter1*, *parameter2*, *parameter3*)

where

operation-type specifies the LDAP operation (**bind**, **search**, **modify**, **add**, **delete**, **modDN**, **compare**)

req|**res** specifies whether it is targeted for the request (**req**) or the result (**res**)

component-name specifies the component name as provided in the LDAP standard RFC4511ff.

action is a string that specifies the operation to be performed on the component and can be one of:

- **replace** – exchange an existing value with a new one (note that wildcard (*****) replacements are not supported)
- **add** – add new value to the existing values
- **delete** – delete an existing value
- **clear** – remove either all values of an attribute or all values
- **set** – set a single value – existing values are lost
- **hide** – hide a resulting entry from a search result

- **showonly** – show only a list of attributes

Not all actions are possible for all LDAP components; for example, an **add** operation on a **baseObject** is not possible. Also note that only the following LDAP controls can be **add**ed:

- LDAP_CTRL_SESSION_ID 1.3.6.1.4.1.21008.108.63.1
- LDAP_CTRL_RELAXED_UPD 1.3.12.2.1107.1.3.2.12.5

The rewriting of LDAP controls (for example, changing attribute names in server-side-sorting) is currently not supported.

parameter1, *parameter2* and *parameter3* are action parameters that may be necessary to fulfill a specific rewriting operation and can be one of the following values:

- The string **NULL**, which indicates that the parameter is not used.
- An assignment string in the format *type*=***value*; for example, *cn=smith**
- A plain string; for example, **objectClass**
- An integer; for example, **700**

The required syntax for these parameters depends on the specific action and on the intended change to be applied. See the tables “Tokens for Actions on Requests” and “Tokens for Actions on Results” for complete details about which parameter must be set for a specific action. If a parameter is not of use for an action, the **NULL** string must be specified. (You can use the **NullParamStr** key in the **Defaults** object to change the string to be used for unused parameters to something other than **NULL**. See the section "The Defaults Object" for details.)

Do not enter unnecessary blanks before or after the enclosing parentheses and before or after the comma. If *parameter1*, *parameter2* or *parameter3* is an assignment (for example, **cn=abele**), do not enter blanks on either side of the equal sign because they will be interpreted as part of the value and may therefore generate undesired results.

Here is an example of an action definition:

```
search.req.baseObject.replace(o=pqr,o=my-company,NULL)
```

In this example, *parameter1* **o=pqr** indicates to the action (**replace**) that an incoming **baseObject o=pqr** is to be replaced by a new value of **o=my-company** in a search request. You are responsible for defining meaningful actions; for example, defining **baseObject** rewriting actions for operations other than **search** should be avoided.

The DLP server ignores anything beyond the first closing parenthesis “**)**” in an action string; for example, in **search.req.baseObject.replace(cn=a,,NULL))**, the extra close parenthesis “**)**” is ignored.

Tokens for Actions on Requests

The following table shows the supported tokens and actions for request rewriting:

Token Name	Action	P1	P2	P3	Description
bind.req.name	*replace*	Old LDAP DN pattern	New LDAP DN pattern	NULL	Replaces the pattern specified by P! in the binder's distinguished name (DN).
search.req.baseObject	*replace*	Old LDAP DN pattern	New LDAP DN pattern	NULL	Replaces the pattern specified by P1 with pattern specified by P2 in the baseObject of the search request. Example: with P1: **ou=sales** and P2: **ou=NewSales**, an incoming baseObject **cn=richter,ou=sales,o=pqr** is modified to **cn=richter,ou=NewSales,o=pqr.** If P2 is an empty string, the pattern P1 is removed from the original baseObject DN.
	set	New LDAP DN	NULL	NULL	Replaces the existing baseObject DN with the one given by P1.
search.req.attributes	*del*	Attribute name. Example: **cn**.	NULL	NULL	Removes the attribute (for example, **cn**) specified by P1 from the list of requested attributes.
	replace	Old attribute name. Example: **strasse**.	New attribute name. Example:* street*.	NULL	Renames an existing attribute whose name is specified by P1 with the name specified by P2 in the requested attributes list.
	add	Attribute name. Ex:*cn*.	NULL	NULL	Adds a new attribute (specified by P1) to the list of requested attributes.
	set	NULL	NULL	NULL	Empties the list of requested attributes; that is, no attributes will be returned.
	replace	Old attribute name. Example: **road**.	New attribute name. Example: **street**.	NULL	Replaces an existing attribute type name (P1) with a new attribute type name (P2) in the filter.

Token Name	Action	P1	P2	P3	Description
search.req.filter	*replace*	Attribute name. Example: **member**.	Old string pattern. Example: **o=pqr**.	New string pattern Example:*o=my-company*.	Replaces a substring in the filter value of an attribute specified by P1. The old substring (P2) is replaced with the new substring (P3). If P1 is "**", the replacement is made in any existing attribute-value independent of the attribute type. Both patterns (old and new) must not contain the character "**".
search.req.sizeLimit	*set*	New limit. Example: **100**.	NULL	NULL	Sets sizeLimit of the search request to the value specified by P1.
search.req.timeLimit	*set*	New limit. Example: **300**.	NULL	NULL	Sets timeLimit of the search request to the value specified by P1.
***.req.controls**	*add*	Symbolic name or OID of control to be added.	NULL	NULL	Adds a well-defined LDAPv3 control specified by P1 to the current operation. Controls that already exist in the incoming request are not touched. The control has the criticality set to TRUE.
modify.req.object	*replace*	Old LDAP DN pattern	New LDAP DN pattern	NULL	Replaces the pattern specified by P1 with the pattern given by P2 in the target entry DN of the. Example: with P1: **ou=sales** and P2: **ou=NewSales**, an incoming target entry **cn=richter,ou=sales,o=pqr** is modified to **cn=richter,ou=NewSales,o=pqr**.
modify.req.object	*add*	Attribute name. Example: **cn**.	Assertion value. Example: **Digger**.	NULL	Adds the attribute specified by P1 to the incoming list of attribute values to be added in the target entry. If no incoming list exists, a new list is created containing the attribute from P1.

Token Name	Action	P1	P2	P3	Description
modify.req.changes_add	*del*	Attribute name. Example: **cn**.	Assertion value. Example: **Digger**.	NULL	Removes the given attribute value (P2) of the attribute specified by P1 from the incoming list of attribute values to be added in the target entry. If the value does not exist in the list, no action is performed. If the value to be removed is the only value of the attribute in the list, the entire list is removed.
	del	Attribute name. Example:* cn*.	NULL	NULL	Removes the attribute specified by P1 (including all of its values) from the incoming list of attribute values to be added in the target entry. If the attribute does not exist in the list, no action is performed.
	replace	Old attribute name. Example:* cn*.	New attribute name. Example: **mycn**.	NULL	Renames the attribute type specified by P1 with the new name specified in P2 in the incoming list of attributes to be added to the target entry. If the old name does not exist, no action is performed. The values of the attribute in the list are not affected.

Token Name	Action	P1	P2	P3	Description
	replace	Attribute name. Example: **member**.	Old string pattern Example: **sales**.	New string pattern Example: **support**.	Replaces the pattern from P2 in the existing value of the attribute specified by P1 with the pattern from P3. Example: Old attribute-value: **member=cn=smith,ou=sales,o=pqr**. By defining P1 as **member** and P2 as **ou=sales** and P3 as **ou=support**, the new attribute value will be: **member=cn=smith,ou=support,o=pqr**. Multiple replaces within the same value are supported. The replace is made to all values of the same attribute specified by P1. If P1 is "*****", the replacement is made to all values for all existing attributes.
	clear	NULL	NULL	NULL	Removes all attributes and all values from the incoming list of attributes to be added in the target entry. The incoming lists for attributes to be deleted or replaced are not touched.
	add	Attribute name. Example: **cn**.	Attribute value. Example: **Digger**.	NULL	Adds the given attribute value (P2) to the attribute specified by P1 to the incoming list of attribute values to be deleted in the target entry. If no incoming list exists, a new list will be created containing the attribute from P1

Token Name	Action	P1	P2	P3	Description
modify.req.changes_delete	*del*	Attribute name. Example: **cn**.	Attribute value. Example: **Digger**.	NULL	Removes the given attribute value specified by P2 for attribute P1 from the incoming list of attribute values to be deleted in the target entry. If the value does not exist in the list, no action is performed. If the value to be removed is the only value of the attribute in the list, the entire list will be removed.
	del	Attribute name. Example: **cn**.	NULL	NULL	Removes the attribute specified by P1 (including all its values) from the incoming list of attribute values to be deleted in the target entry. If the attribute does not exist in the list, no action is performed.
	replace	Old attribute name. Example: **cn**.	New attribute name. Example: **mycn**.	NULL	Renames the attribute type specified by P1 with the new name specified by P2 in the incoming list of attributes to be deleted to the target entry. If the old name does not exist, no action is performed. The values of the attribute in the list are not affected.

Token Name	Action	P1	P2	P3	Description
	replace	Attribute name. Example: **member**.	Old string pattern. Example: **sales**.	New string pattern. Example: **support**.	Replaces the pattern from P2 in the existing value of the attribute specified by P1 with the pattern from P3. Example: Old attribute-value: member=cn=smith,ou=sales,o=pqr. By defining P1 as **member** and P2 as **ou=sales** and P3 as **ou=support**, the new attribute value will be: **member=cn=smith,ou=support,o=pqr**. Multiple replaces within same value are supported. The replacement is made to all values of the same attribute specified by P1. If P1 is “*”, the replacement is made to all values for all existing attributes.
	clear	NULL	NULL	NULL	Removes all attributes and all values from the incoming list of attributes to be deleted in the target entry. The incoming lists for attributes to be added or replaced are not touched. Please note that in LDAPv3, a modification must have at least one change present (that is, an empty list is not allowed).
	add	Attribute name. Example: **cn**.	Attribute value. Example: **Digger**.	NULL	Adds the given attribute value (P2) to the incoming list of attribute values to be replaced in the target entry for attribute P1. If no incoming list exists, a new list is created containing the attribute from P1/P2.

Token Name	Action	P1	P2	P3	Description
modify.req.changes_replace	*del*	Attribute name. Example: **cn**.	Attribute value. Example: **Digger**.	NULL	Removes the given attribute value given by P2 from the incoming list of attribute values to be replaced in the target entry for attribute P1. If the value does not exist in the list, no action is performed. If the value to be replaced is the only value of the attribute in the list, the entire list is removed.
	del	Attribute name. Example: **cn**.	NULL	NULL	Removes the attribute given by P1 (including all its values) from the incoming list of attribute values to be replaced in the target entry. If the attribute does not exist in the list, no action is performed.
	replace	Old attribute name. Example: **cn**.	New attribute name. Example: **mycn**.	NULL	Renames the attribute type given by P1 with the new name given in P2 in the incoming list of attributes to be replaced to the target entry. If the old name does not exist, no action is performed. The values of the attribute in the list are not affected.

Token Name	Action	P1	P2	P3	Description
	replace	Attribute name. Example: **member**.	Old string pattern. Example: **sales**.	New string pattern. Example: **support**.	Replaces the pattern from P2 in the existing value of the attribute specified by P1 with the pattern from P3. Example: Old attribute-value: **member=cn=smith,ou=sales,o=pqr**. By defining P1 as **member** and P2 as **ou=sales** and P3 as **ou=support**, the new attribute value will be: **member=cn=smith,ou=support,o=pqr** Multiple replaces within same value are supported. The replacement is made to all values of the same attribute specified by P1. If P1 is "*", the replacement is made to all values for all existing attributes.
	clear	NULL	NULL	NULL	Removes all attributes and all values from the incoming list of attributes to be replaced in the target entry. The incoming lists for attributes to be added or deleted are not touched.
add.req.entry	*replace*	Old LDAP DN pattern	New LDAP DN pattern	NULL	Replaces the pattern specified by P1 with the pattern specified by P2 in the DN to be added. Example: with P1: **ou=sales** and P2: **ou=NewSales**, an incoming DN **cn=richter,ou=sales,o=pqr** will be modified to **cn=richter,ou=NewSales,o=pqr**.

Token Name	Action	P1	P2	P3	Description
	del	Attribute name. Example: **cn**.	NULL	NULL	Removes all values for the attribute (example: **cn**) specified by P1 from the list of incoming attributes to be added in the new entry. Note: This action may cause the add operation to fail if mandatory attributes are removed.
add.req.attributes	*del*	Attribute name. Example: **cn**.	Attribute value. Example: **Digger**.	NULL	Removes the attribute value specified by P2 from the incoming list of attribute values for the attribute given by P1 in the target entry. If the value does not exist in the list, no action is performed. Note: This action may cause the add operation to fail if mandatory attributes are removed.
	replace	Old attribute name. Example: **road**.	New attribute name. Example: **street**.		Renames an existing attribute whose name is given by P1 with the name given by P2 in the list of attributes to be added.

Token Name	Action	P1	P2	P3	Description
	replace	Attribute name. Example: **member**.	Old string pattern. Example: **sales**.	New string pattern. Example: **support**.	Replaces the pattern from P2 in the existing value of the attribute specified by P1 with the pattern from P3. Example: Old attribute-value: **member=cn=smith,ou=sales,o=pqr**. By defining P1 as **member** and P2 as **ou=sales** and P3 as **ou=support**, the new attribute value will be: **member=cn=smith,ou=support,o=pqr**. Multiple replaces within the same value are supported. The replacement is made to all values of the same attribute specified by P1. If P1 is "*", the replacement is made to all values for all existing attributes.
	add	Attribute name. Example: **cn**.	Attribute value. Example: **Digger**.	NULL	Adds a new value (specified by P2) to the attribute specified by P1 in the list of attributes to be added.
	clear	NULL	NULL	NULL	Removes all attributes and all values from the incoming list of attributes to be added in the new target entry.
delete.req.entry	*replace*	Old LDAP DN pattern	New LDAP DN pattern	NULL	Replaces the pattern specified by P1 with the pattern specified by P2 in the entry-DN to be renamed/moved. Example: with P1: **ou=sales** and P2: **ou=NewSales**, an incoming DN **cn=richter,ou=sales,o=pqr** will be modified to **cn=richter,ou=NewSales,o=pqr**.

Token Name	Action	P1	P2	P3	Description
modDN.req.entry	*replace*	Old LDAP DN pattern	New LDAP DN pattern	NULL	Replaces the pattern specified by P1 with the pattern specified by P2 in the DN of an incoming ModifyDN request.
modDN.req.newrdn	*replace*	Old LDAP DN pattern	New LDAP DN pattern	NULL	Replaces the pattern specified by P1 with the pattern specified by P2 in the **newrdn** component of an incoming ModifyDN request.
modDN.req.newsup	*replace*	Old LDAP DN pattern	New LDAP DN pattern	NULL	Replaces the pattern specified by P1 with the pattern specified by P2 in the **newsuperior** component of an incoming ModifyDN request.
compare.req.entry	*replace*	Old attribute name. Example: **strasse**	New attribute name. Example: **street**.	NULL	Replaces the attribute type name for the attribute to be compared specified by P1 with the one specified by P2.

Token Name	Action	P1	P2	P3	Description
compare.req.attr	*replace*	Attribute name. Example: **member**.	Old string pattern. Example:* sales*.	New string pattern. Example: **suppose**.	Replaces the pattern from P2 in the existing value of the attribute specified by P1 with the pattern from P3. Example: Old attribute-value: **member=cn=smith,ou=sales,o=pqr**. By defining P1 as **member** and P2 as **ou=sales** and P3 as **ou=support**, the new attribute value will be: **member=cn=smith,ou=support,o=pqr** Multiple replaces within same value are supported. The replacement is made to all values of the same attribute specified by P1. If P1 is "*, the replacement is made to all values for all existing attributes.

Tokens for Actions on Results

The following table shows all allowed/supported tokens and actions for result rewriting. Currently only result rewriting for **search** operations is supported.

Token Name	Action	P1	P2	P	Description
search.res.objectName	*replace*	Old LDAP DN pattern	New LDAP DN pattern	NULL	Replaces the pattern specified by P1 with the pattern specified by P2 in the resulting entry DN. Example: P1 is **ou=sales** and P2 is **ou=NewSales**; a resulting entry **cn=richter,ou=sales,o=pqr** appears as **cn=richter,ou=NewSales,o=pqr** in the client result.
search.res.entry	*hide*	LDAP DN pattern	NULL	NULL	Hides all resulting entries if their DN matches the pattern specified in P1.

Token Name	Action	P1	P2	P	Description
search.res.attributes	*del*	Attribute name. Example: **cn**.	NULL	NULL	Removes the attribute (example: **cn**) specified by P1 from the result (including all values).
	del	Attribute name. Example: **cn**.	Attribute value. Example: **Digger**.	NULL	Removes a single value (example: **cn=Digger**) of the attribute specified by P1 from the result. If the removed value is the only value of the attribute, the entire attribute is removed.
	replace	Old attribute name. Example: **strasse**.	New attribute name. Example: **street**.	NULL	Renames an existing attribute whose name is specified by P1 with the name specified by P2 in the LDAP result.
	replace	Attribute name. Example: **member**.	Old string pattern. Example: **sales**.	New string pattern. Example: **support**.	Replaces the pattern from P2 in the existing value of the attribute specified by P1 with the pattern from P3. Example: Old attribute-value: **member=cn=smith,ou=sales,o=pqr** By defining P1 as **member**, P2 as **ou=sales** and P3 as **ou=support**, the new attribute value will be: **member=cn=smith,ou=support,o=pqr**. Multiple replaces within same value are supported. The replacement is made to all values of the same attribute specified by P1. If P1 is "*****", the replacement is made to all values for all existing attributes.

Token Name	Action	P1	P2	P	Description
	add	Attribute name. Example: **cn**.	Attribute value. Example: **Digger**.	NULL	Adds a new value specified by P2 to the attribute specified by P1 to the result.
	clear	Attribute name. Example: **cn**.	NULL	NULL	Clears all existing values of the attribute specified by P1. This action is equal to **search.res.attributes.del(attr,NULL)**.
	clear	NULL	NULL	NULL	Clears all existing values of all attributes. The effect is that no attributes are returned.

Token Name	Action	P1	P2	P	Description
	showonly	List of attribute names separated by a plus-sign **+** Example: **tn+cn+description+sn** Or an attribute-list name that refers to a previously defined list of attribute names. (See “The AttributeList Objet” for details.) The attribute list name is prefixed by a **@** Example **@ATTRLIST1** Must be not an empty string or NULL.	NULL	NULL	Removes all attributes that are not contained in P1.

Please be careful not to define too many search result rules as they will be applied to every resulting entry, which might be a performance issue if huge results are retrieved.

3.3.2. How the Rule Processing Sequence Affects Result Rewriting Rules

As mentioned in the ProxyRule object definition description in the “Configuration” chapter, multiple proxy rules are executed in the order in which they are defined in the DLP server configuration file, and multiple actions are executed in the order in which they’re defined in the rule.

For example, let’s assume that three rules R1, R2 and R3 have been defined in the configuration file: R1 is the first rule to be defined, R2 is the second, and R3 is the third. The DLP server checks the conditions for each rule in the same order as the rule definitions:

R1 → R2 → R3

Now let’s assume that R1 defines two actions: A11 and A12 R2 defines only one action: A21 and R3 defines three actions: A31, A32, and A33. If all three rule conditions – R1, R2 and R3 - match the running operation O, the DLP server modifies the incoming operation O(in) in the following sequence to the outgoing operation O(out):

O(in) →A11 →A12 →A21 →A31 →A32 →A33 → O(out)

As shown in this example, the DLP server executes the actions on the incoming operation in the same sequence as they are presented in the configuration file.

Now let’s consider what this execution sequence means for constructing result rewriting rules, since the same execution sequence applies: the request/result input for an action is always the request/result output of its preceding action. Suppose we have the following two rules:

R1:

C1: **(&(opr.req.type=search)(search.req.baseObject=o=pqr))**

A1: **change base object to O=my-company in the running search**

R2:

C2: **(&(opr.req.type=search)(search.req.baseObject=o=pqr))**

A2: **add an attribute cn to the list of requested attributes in the running search**

It’s clear that both conditions are identical and will match to an incoming search request with **baseObject o=pqr**. However, because R1 is executed before R2, the action A1 will change the **baseObject** of the incoming request from **o=pqr** to **o=my-company**. Therefore, when R2 is processed, the rule condition is no longer true and the action A2 will never be executed.

3.3.2.1. Handling Attribute Name Aliases in Rewriting Rules

LDAP attribute names can have aliases. For example, the attribute **telephonenumber** has the alias **tn**. If you intend to apply rewrite rules to attributes that have aliases, you may need to specify multiple actions in order to cover all possible versions of an attribute name; for example:

```
{
    "object" :        "ProxyRule",
    "ruleType" :      "ReqRewrite",
    "name" :          "Test 03",
    "condition" :     "opr.req.type=modify",
    "actions" :     [
"modify.req.changes_add.del(telephoneNumber,123,NULL)",
                      "modify.req.changes_add.del(tn,123,NULL)" ]
}
```

3.3.3. Using Virtual Names in Rewriting Actions on Search Results

When defining rewriting rules for search results, a special problem can occur. For example, suppose there is a single rule that rewrites the search result with an action like

`search.res.attributes.replace(street,road)`

This action changes the name of the attribute **street** in all resulting entries into a virtual new name **road**. Therefore, the client receives the name **road** instead of **street** which is (usually) not an attribute defined in the schema. Most clients will ignore this mismatch against the schema (or simply do not even read the schema) and just display the attribute with the new virtual name **road** like they would display the **street** attribute. As long as it is only about viewing, it is (mostly) not of too much relevance as long as the client is familiar with the new name. However, problems can occur if the received virtual attribute name is used by the client for a subsequent operation (for example, modify). If **road** will be used in a subsequent operation and if no further rule exists, the attribute **road** will be transmitted to the LDAP server, which will possibly return an error due to this unknown attribute (remember: the mapping is only known to the DLP server – not to the backend servers).

Consequently, if you intend for the rewritten attribute to be re-usable by the client in subsequent operations, you need to define additional rules to perform the corresponding conversion from client to server again (for example, **road** to **street**) in order for the operation to succeed in the backend servers, which are unaware that a rewrite has occurred in the DLP server.

To support the **road** to **street** re-conversion from our example in a subsequent **modify** operation, it may be necessary to add other rules with actions like

`modify.req.changes_add.replace(road,street)`
`modify.req.changes_delete.replace(road,street)`
`modify.req.changes_replace.replace(road,street)`

This will do the job for modify operations if the attribute is not a naming attribute (that is, an attribute that does not appear in DNs).

To exchange the virtual name **road** in all possible modification operations, similar rules for other operations like **add**, **compare** may be necessary in order to translate **road** into **street**

again.

These actions must be defined in the first rules if other actions are to be present, as actions are applied sequentially in the order in which they're defined in a rule.

Things get more complicated if the re-written attribute may also appear in DNs (as a type of an RDN that builds up the DN).

The chapter "Result-Rewriting Considerations" provides some additional guidance for administrators on how to maintain a consistent approach to the database in subsequent calls after result rewriting takes place.

3.4. Character Set Requirements in Rule Conditions and Actions

UTF-8 is the only character encoding allowed by LDAP.However, special Latin-1 characters like the German umlaut (**ä**, **ö**, **ü**) may need to be used in condition and action assignments, especially in DN strings.The DLP server can convert these Latin-1 characters to their UTF-8 representation for processing by LDAP.The **JSONCodeSet** key in the Defaults object definition indicates to the DLP server in which character set the proxy rules in the DLP server configuration file are encoded and thus whether or not they need to be converted.Make sure that all conditions and all actions in all ProxyRule objects you define in a DLP server configuration file use the same character encoding: Latin-1 or UTF-8. Do not mix character sets within a single DLP server configuration file.See the "Configuration" chapter for more information on the Defaults and ProxyRule objects.

3.5. Handling Special Characters in Rule Conditions and Actions

Both LDAP and JSON define characters that require special handling in certain cases.The proxy rule parameter syntax also defines special characters.When a condition assignment or an action parameter needs to contain one of these special characters, the character must be converted to its hex string representation and prefixed with the **#** character in the proxy rule definition.

LDAP RFC4514 requires escaping the following characters when they appear in DNs:

- A space (U+0020) or number sign (**#** U+0023) that occurs at the beginning of the string
- A space (U+0020) character that occurs at the end of the string
- One of the characters **"**, **+**, **,**, **;**, **<**, **>**, or **** (U+0022, U+002B, U+002C, U+003B, U+003C, U+003E, or U+005C, respectively)
- The null (U+0000) character

JSON RFC4627 requires the following characters to be escaped when they appear in property names or property values:

Special Character	Escaped Output
Quotation mark (**"**)	\"
Backslash (****)	\\
Slash (**/**)	\/
Backspace	\b
Form feed	\f
New line	\n
Carriage return	\r
Horizontal tab	\t

The proxy rule condition rule and parameter syntax requires the following characters to be escaped:

- Open parenthesis **(**
- Close parenthesis **)**
- Comma **,**

When you find that a condition assignment or an action parameter needs to use one of these special LDAP, JSON or proxy rule syntax characters, you must use the character's hex string representation and prefix it with the **#** character.

For example, let's assume the value **James, Bond (007)**, which contains the special characters "**,**", "**(**" and "**)**" is to be set as a parameter. The following hex string can be used as the parameter:

```
#4A616D65732C20426F6E64202830303729
J a m e s , B o n d ( 0 0 7 )
```

The hex string consists of the hex bytes of each character in text format (J=4A, a=61, m=6D, and so on). This is the same method that LDAP allows for representing DNs that contain special or binary characters.

If a value to be added as a parameter begins with a **#** but does not contain any special characters, the value itself must start with another **#**, for example, the value **#Hello There#** must be represented as **##Hello There#**. This extra **#** prefix is only necessary if the *first* character is a **#**. If the **#** appears at any other position, the string remains unchanged.

The parameter represented by the hex string must still obey the code-set (Latin-1 or UTF-8) from the **Default** object's **JSONCodeSet** setting. See the Default object description in the "Configuration" chapter and the chapter "Operations" for more details.

4. Operation

This chapter provides information about DLP server operation in the areas of server process startup, connect timeout, off-line handling and server retry, round-robin selection and failover and character set handling and provides a general workflow example that shows how server selection works and how failover is handled for a user-routing rule.

4.1. LDAP Server Process Startup for DLP

When the LDAP server process (**dirxldapv3**) starts, it reads the ldapProxyMode attribute from its corresponding LDAP configuration subentry.The subentry name is specified on the command line with the **-n** option; if it is not specified, the server process uses the subentry name **ldapConfiguration**.

When the ldapProxyMode attribute is set to **1** or **2**, the server process runs as a DLP server.All well-known settings from the LDAP configuration subentry continue to apply as long as they refer to the client side (for example, LDAP port, number of pool threads, maximum number of client connections).Settings that affect the DAP backend (for example, unbind delay time, DAP share count) are ignored.

Note the following about LDAP server process startup as a DLP:

- If the ldapProxyMode attribute is not present in the LDAP configuration subentry, the LDAP server continues to run as a plain LDAP server.
- If the server process starts with a proxy mode >=1 but it cannot locate the DLP server configuration file, it starts in plain LDAP server mode.
- If the server process starts with a proxy mode >=1 but the DLP server configuration file contains syntax errors, the LDAP server will not start until the configuration file is correctly specified.
- If the server process starts with a proxy mode >=1 but the LDAP configuration subentry name specified in the **-n** option (or the default name **ldapConfiguration** if the **-n** option is not specified) does not match any of the **LdapProxy** object names defined in the DLP server configuration file, it starts in plain LDAP server mode. Therefore, it is important to note that the LDAP server process must find 1) a proxy mode >= 1 in the LDAP configuration subentry and 2) a matching **LdapProxy** object in the DLP server configuration file in order to establish itself as a working DLP server.

4.1.1. Connect Timeout

Before the DLP server can forward a client request to a selected target server, it must perform a TCP-connect operation (possibly followed by an SSL-connect()).The connection is usually established rather quickly, but timeout effects may occur if the peer is unavailable or unreachable.A normal successful TCP connect consists of a three-way handshake between the initiator and the responder (we assume here that all IP addresses are properly configured and routable):

State	Initiator (DLP Server)	Direction	Responder (LDAP Server)
1	send SYN	→	recv SYN
2	recv SYN-ACK	←	send SYN-ACK
3	send ACK	→	recv ACK

The connect timeout typically appears at state 2 when the initiator waits for the SYN-ACK from the peer. The amount of time spent in state 2 depends heavily on the state of the responder:

a. Responder-host is running, application listens on port
b. Responder-host is running, no application listens on port
c. Responder-host is running, application listens but firewall blocks initiator IP
d. Responder host is down
e. Network to responder is broken

For cases a. and b., an immediate response is sent from responder to initiator.

For case c., it depends on the firewall whether or not it will create a RST (reset) to reject the connection.If it doesn't, the SYN is un-answered and times out on the initiator side.

For cases d. and e., there is no peer TCP that could respond with error and so the initial SYN packet is lost and is never answered.After some time, the initiator TCP starts a re-transmission of the SYN packet, likely with the same unanswered result.Usually the re-transmission occurs *N* times (a TCP parameter) and the time the sender is willing to wait is doubled.On the Windows platform, *N* is **3** and the initial wait is 3 seconds, which results in 3 + 6 + 12 = 21 second timeout for cases c., d. and e.

The DLP server contacts the target servers sequentially one after the other.Therefore, if a rule contains *M* servers, each try might take up to 21 seconds to detect that the target is down before it tries the next server from the list.You can use the **ConnectTimeout** value in the Defaults object definition to help reduce the time it takes to detect a server outage during the TCP-connect().By default, the DLP server uses 3 seconds - which is the first retransmission timeout on Windows - to speed up detection of cases c., d. and e.We recommend leaving the default in place until TCP analysis has shown that some other value helps to overcome connection problems.Never set **ConnectTimeout** to **0** as it may render the DLP server completely unable to connect.

Note that there is no timeout for normal I/O read/write operations because introducing one imposes the requirement of determining the worst-case runtime of any of the connected target servers for all possible legal searches.For example, when a search request is sent out, what is the right amount of time to wait before returning with a timeout?Choosing the right timeout requires the ability to predict the maximum search runtime that can occur with any possible search.This is almost impossible to calculate, as some simple searches are fast but more complicated searches can last a long time.

4.2. Offline Handling and Server Retry

When the DLP server forwards a request to a target LDAP server, it may detect a network failure.There are basically two incidents for such a failure:

- Failure detected while processing the TCP connect() while establishing a connection to the target server
- Failure detected during read/write I/O while the connection was already established

Both detections lead the DLP server to mark the target server as OFFLINE.

If more than one target server is configured and the rule allows for failover, the DLP server continues to try to process the request to the next target server.This action continues until the request can be processed or all until servers fail.Switching to another target server is transparent to the client: the DLP server automatically re-establishes the LDAP connection with the credentials that existed at failure time if the error occurs while a connection was established at error time.

If a target server is marked OFFLINE, the DLP server will not select it for any further operations for a certain amount of time, no matter for which rule the server was configured. Even if a server outage was detected by user X, the target server will not be selectable for any other user Y after the detection.

The selection of possible target servers for a user occurs at the time of the first operation on the corresponding client LDAP connection; usually this operation is a bind, but LDAPv3 allows starting with any operation, in which case the anonymous user is then assumed.As a result, if a target server is marked as OFFLINE due to failure detection, it does not affect the target servers that are already selected for other user connections that existed at failure time.Users that establish a new LDAP connection will not have servers marked as OFFLINE as their target server choice.

+ You can use the **OfflineRetryTimeout** key in the Defaults object definition to control the duration for which an OFFLINE server is not selectable; after this amount of time has passed, the server is selectable again for a retry for new users.The default is 60 seconds

Please note that there is a difference between “selectable” and a real retry.

When a target server is selectable, it means that the DLP server adds it to the list of possible servers configured by the corresponding rule.If the target server is not the rule’s primary server (the first one in the list) it’s possible that it may never be contacted.Thus, being selectable does not necessarily imply that an actual retry occurs.Therefore, the server remains selectable until a real retry returns another error, in which case it is marked as OFFLINE again and will not be selectable again for the configured retry time.If a target server is retried successfully after an offline-retry timeout, it remains selectable.

Be careful not to set **OfflineRetryTime** too high.As we have seen, if a server cannot be reached and is marked as OFFLINE, a server is not retried until the **OfflineRetryTime** expires.Thus, if all relevant servers cannot be reached, they will all stay in the OFFLINE state for at least **OfflineRetryTime** and will therefore not be selected even if the server is physically up before the

> **OfflineRetryTime** expires.If you set this time to a high value – for example, 5 minutes – for users that only have these servers configured for their use, further operation is not be possible until the **OfflineRetryTime** expires.On the other hand, if you set **OfflineRetryTime** to a very low value – for example, 1 second – a lot of operations will retry these servers very frequently and may experience a significant TCP timeout for their operation duration if the server is down for a long time.

We recommend setting **OfflineRetryTime** in the range of 30-60 seconds to establish a good balance between not frequently calling servers that are down and having a reasonable early detection timeframe once they are up again.

4.3. Round-Robin Selection and Failover

If round-robin (RR) selection is enabled for a rule and failover (FO) is also enabled, they are both applied separately, which may lead to unexpected target selection for subsequent binds.

Let's illustrate this concept with the following example:

- User A has a rule where both RR and FO are enabled.
- User A uses the target servers L1, L2 and L3.
- The target server L3 is down.
- The time between each bind is longer than the **OfflineRetryTime** setting.

On user A's first bind, the selected target servers are L1, L2, L3. The first bind goes to L1 and succeeds.When user A makes a second bind later on, L2 is selected (due to RR), resulting in a successful bind.After some time, user A issues a third bind; L3 is selected (RR) and fails.Because FO is active, the next server L1 is selected and succeeds.

What happens when user A performs a fourth bind?Which server is contacted?It is server L1, because RR and FO are treated separately; that is, a failover selection (next server) will not change the next selection for the RR algorithm.Therefore, as RR selected L3 for bind #3 (which failed and was shifted to L1 by FO), the RR algorithm will select the next after L3, which is L1.

This behavior may look strange, as L1 has received the last two binds from user A although RR is active, but it is due to the fact that FO simply selects the next server from the current failing server, and RR simply selects the next server in line after the last RR selection.

4.4. Character Set Handling

Although proxy rule tokens, keys and components are plain ASCII strings, there may be some assignment values within a proxy rule condition or action that contain special Latin-1 characters like **ä**, **ö**, **ü**, and so on (for example, for DNs).In order to match a condition or action containing these Latin-1 characters properly to the UTF-8 character values contained in LDAP, it is necessary to identify to the DLP server which format the DLP server configuration file uses.There are currently two choices defined in the Defaults object

definition of the DLP server configuration file:

"JSONCodeSet" : 0 // declares the configuration file to be a Latin-1 content file
"JSONCodeSet" : 1 // declares the configuration file to be a UTF-8 content file

If a value of **0** is specified, the DLP server interprets all condition and action strings contained in all ProxyRule objects as Latin-1 characters and performs an implicit conversion to UTF-8 before storing them to its internal configuration.This option might be useful if your JSON text editor does not support storing Latin-1 characters like **ö** as their multi-byte UTF-8 **char** representations.

A value of **1** indicates that all condition and action strings in the DLP server configuration file are encoded in UTF-8 format and do not require conversion.By default, UTF-8 is assumed (**JSONCodeSet = 1**).

4.5. General Operation Forwarding Example

This example describes internal operation when a user X performs a sequence of LDAP operations **bind** → **search** → **unbind**→ **bind**→**search**→**unbind** on a plain (non SSL) connection against the DLP server.We assume that the primary server LDAP3 is down and unavailable.Servers LDAP1 and LDAP2 are up and available.We further assume that the DLP server has not yet contacted LDAP3 and so is not aware that it is down.We also assume that user X has an explicit rule of the form:

```
{
   "object"      : "ProxyRule",
   "ruleType"    : "UserRouting",
   "name"        : "USERROUTING2",
   "condition"   : "(user=cn=admin,o=pqr)",
   "actions"     : [ "forwardto(LDAP3,LDAP1)" ],
   "loadbalance" : 0,
   "failover"    : 1
}
```

before he can send out the bind PDU, he must establish a TCP connection. During TCP connection establishment, the DLP server recognizes a new LDAP connection and creates an internal object called LdapConnection. As no authentication happened so far (remember that the bind PDU has not yet been sent), the DLP server assigns the ‘anonymous’ user to this new connection and waits for further data to arrive.

Once the TCP connection is established, the client can now send the LDAP bind PDU containing the credentials (user+pwd) for this connection. The DLP server receives this PDU, detects that it is a bind PDU and extracts the user name (DN) from it.

Next, the DLP server reads the configured rule sets and checks to see if there is a rule for user X. It finds the user rule and then builds a list of possible target servers by reading the **forwardto** servers from the rule and checking whether or not a round-robin selection

should be used on the list. Because the rule does not supply the **loadbalance** key and it is a specific-user rule, round-robin selection is not performed and the first server in the list (LDAP3) becomes the primary server. As LDAP3 has never been contacted, the DLP server assumes that it is available and selects it as the target server for user X.

Now the DLP server opens a TCP connection to LDAP3 and detects the outage (this may take a few seconds) and marks LDAP3 as OFFLINE. Next, it checks whether the user rule allows failover and whether there are other servers configured for failover. Both conditions are true. Thus, LDAP2 becomes the new primary target server. The DLP server performs a TCP connect against LDAP2 which now succeeds.

Next, the DLP server forwards the bind to this target server and receives the bind result PDU indicating a successful bind. The received PDU is now sent back to the client.

Once the client receives the bind success, it issues the search by sending the search PDU out to the DLP server. The DLP server receives the PDU and recognizes a search. For the DLP server, this means that no target server selection is necessary as there is already a working connection to LDAP2. Thus the DLP server simply sends out the search PDU to LDAP2 and receives the search result. After receiving the result, the DLP server returns the result to the client.

After the client has received the search result, it invokes the unbind operation. The unbind PDU is sent to the DLP server and is received. Again the DLP server knows that the target server is LDAP2 and sends out the unbind operation. As unbind is not a confirmed operation, the DLP server does not need to wait for a response and closes the TCP connection to LDAP2. It also closes the frontend connection to the client and destroys the internal LdapConnection object. As a result, the DLP server no longer has any frontend or backend connection and is back to the state it was in before the first bind.

Now the client sends the second bind. The DLP server establishes a TCP connection, creates a new LdapConnection object and assigns the anonymous user. The client sends the second bind PDU which is received by the DLP server. The DLP server extracts user X from the bind and performs a lookup against the rules. It again finds a user rule without load balancing where the primary server is LDAP3. However, as it has detected that LDAP3 is OFFLINE, the DLP server ignores it, selects LDAP2 as the primary server, establishes a new TCP connection and sends the bind to LDAP2 again. The successful bind response is received and returned to the client. The remaining operations search and unbind operations work exactly as for the first sequence.

5. Monitoring and Analysis

As request/result re-writing is an on-the-fly process within the LDAP proxy server and thus only either the client will see the rewritten results or the target server will see the rewritten request it is hard to actually see whether the rules are applied correctly as intended.

In order to provide better insight some extensions to the existing monitoring capabilities have been made.

5.1. Analyzing Errors in Rewriting Rule Definitions

As everything regarding rewriting is controlled by the rules contained within the DLP server configuration file, it is pretty easy to make incorrect definitions either by making syntax errors or by making logical mistakes between relations.

Let's assume we made a rewriting rule definition like this one:

```
{
        "object"    : "ProxyRule",
        "ruleType"  : "ReqRewrite",
        "name"      : "REQREWRITE1",
        "condition" :
"(&(opr.req.typ=search)(search.req.baseObject=o=my-company))",
        "actions"   : [ "search.req.baseObject.replace(NULL,o=pqr)" ]
}
```

The definition looks fine at first glance, but still the DLP server refuses to start.

5.1.1. Finding Syntax Errors

One of the most common errors is one or more syntax errors in a condition or an action definition.

With the definition from the previous section, the server will not start. Why not?

```
ProxyConf-Error: Illegal condition-string
'(&(opr.req.typ=search)(search.req.baseObject=o=my-company))' found
in ProxyRule object 'REQREWRITE1'!
FATAL: Cannot read PROXY configuration (ldap_proxy.json)!
LDAP Server exit (ExitCode: 19 Reason: FATAL: Cannot process PROXY
configuration! Server stopped.)
```

Chances are that you can't see the problem immediately, so what can we do to find it?

The best way to localize the problem is to activate DLP server tracing by setting:

"Tracing" : 3

in the Defaults object definition in the DLP server configuration file and then restarting the DLP server.

Restarting the server causes a trace file to be generated at *install_path*/ldap/log/proxytrace**PID*.txt*, where *PID* is the process ID of the newly started server.

When we look at the end of the file, we find the following data:

```
pid:8788,tid:10704, 10:42:46.401
(..\..\dirxldapv3\mainthread\mn_proxy.cpp:3786)
ConditionListToLdapFilter((opr.req.typ=search)(search.req.baseObject=
o=my-company))

pid:8788,tid:10704, 10:42:46.401
(..\..\dirxldapv3\mainthread\mn_proxy.cpp:3932)
ConditionToLdapFilter((opr.req.typ=search))

pid:8788,tid:10704, 10:42:46.401
(..\..\dirxldapv3\mainthread\mn_proxy.cpp:3639)
SimpleConditionToLdapFilter(opr.req.typ=search)

pid:8788,tid:10704, 10:42:46.401
(..\..\dirxldapv3\mainthread\mn_proxy.cpp:3533)
IsLegalConditionToken(opr.req.typ=search)

pid:8788,tid:10704, 10:42:46.401
(..\..\dirxldapv3\mainthread\mn_proxy.cpp:3622)
ILLEGAL TOKEN or VALUE found in 'opr.req.typ=search'!

pid:8788,tid:10704, 10:42:46.401
(..\..\dirxldapv3\mainthread\mn_proxy.cpp:3709)
[JSON] Illegal Token found in condition string: 'opr.req.typ'

pid:8788,tid:10704, 10:42:46.401
(..\..\dirxldapv3\mainthread\mn_proxy.cpp:3763)
[JSON] Syntax error in Condition-Filter
```

From this output, we can see that the error is caused by the incorrect token name **opr.req.typ** in the condition definition. The correct token name is **opr.req.type**. We can

easily correct this error in the corresponding proxyRule object in the configuration file.

```
ProxyConf-Error: Illegal CMD-String in Action
'search.req.baseObject.replace(NULL,o=pqr)'!
FATAL: Cannot read PROXY configuration (ldap_proxy.json)!
LDAP Server exit (ExitCode: 19 Reason: FATAL: Cannot process PROXY
configuration! Server stopped.)
```

As we can see, the error has changed. Now it reports an error in the **action** string.

Looking again at the trace file, we find the following data:

```
pid:8400,tid:10292, 13:10:59.032
(..\..\dirxldapv3\mainthread\mn_proxy.cpp:3018)
PrepareCmdParamsStr(search.req.baseObject.replace(NULL,o=pqr))

pid:8400,tid:10292, 13:10:59.032
(..\..\dirxldapv3\mainthread\mn_proxy.cpp:3055)
Bad parameters found in 'search.req.baseObject.replace(NULL,o=pqr)'
-> P1 & P2 must contain old-value and new-value!

pid:8400,tid:10292, 13:10:59.032
(..\..\dirxldapv3\mainthread\mn_proxy.cpp:4214)
PrepareAction():Illegal action specification:
'search.req.baseObject.replace(NULL,o=pqr)'
```

This gives the error reason. The problem is that the parameter list (**NULL,o=pqr**) is not a legal parameter list for the **replace** action on the component **search.req.baseObject**.

The first parameter must be the old string to be replaced by the string in the second parameter.

The definition contains **NULL** as the first parameter, which is not allowed for a **replace** operation. Let's correct this mistake in the proxyRule object in the configuration file:

```
{
    "object"    : "ProxyRule",
    "ruleType"  : "ReqRewrite",
    "name"      : "REQREWRITE1",
    "condition" :
"(&(opr.req.type=search)(search.req.baseObject=o=my-company))",
    "actions"   : [ "search.req.baseObject.replace(o=my-
```

```
company,o=pqr)" ]
}
```

After doing so, our server starts up.

Don’t forget to disable tracing after fixing, otherwise performance will decrease dramatically.

5.1.2. Detecting Logical Errors

These types of errors are much harder to detect as the server will not necessarily report an error at all. Suppose we have the following rule definition:

```
{
    "object"    : "ProxyRule",
    "ruleType"  : "ReqRewrite",
    "name"      : "REQREWRITE1",
    "condition" :
"(&(opr.req.type=modify)(search.req.baseObject=o=my-company))",
    "actions"   : [ "search.req.baseObject.replace(o=my-
company,o=pqr)" ]
}
```

This definition has no syntax errors so the server can start.

The problem here is that the condition will never match any request because there is no modify operation for a search **baseObject**.We wanted to address a search, but we mistyped a modify instead.The server will not detect such incorrect use/mixing of incompatible tokens.It only checks whether the syntax is OK and whether all mandatory components are described.The DLP server will not detect logical errors.

In our example, the rule will never become active but the DLP server administrator may never discover the problem.Therefore, we strongly recommend running a quick check on any new rule you create by performing an operation that should match the rule and then checking whether or not everything is applied as expected.

5.2. DLP Server Logging

The LDAP server binary is the same whether it is running as a DLP server or a plain LDAP server.Consequently, the same logging environment applies to both modes.

To view details about DLP server processing, add the following components to the **dirxlog.cfg** file:

ldap_op.1.4.5

Because logging can have a significant performance impact, it is recommended to enable this only for diagnostic purposes, not for normal operation.

5.2.1. Logging Example

This example shows sample output of the logging generated when a client performs an LDAP bind operation against the DLP server:

```
------------------------------------------------------------------------
-----------
LDAP Listener        0x000024fc LDAP 9.2.104     dir      Di
28.03.2017 16:14:21
------------------------------------------------------------------------
-----------
          -- 0x45046755 DEBUG1   ldap_op  mn_ldap_listener 429
14:21:997
 1 New LDAP connection (Con1) accepted from client IP:
"127.0.0.1":63210
```

The DLP server recognizes traffic on the LDAP port by the listener thread – listener creates a new connection object (Con0) and hands processing over to the pool thread.

```
------------------------------------------------------------------------
-----------
PoolThread#1         0x00002b60 LDAP 9.2.104     dir      Di
28.03.2017 16:14:21
------------------------------------------------------------------------
-----------
          -- 0x4504674e DEBUG1   ldap_op  op_ldap_operatio 535
14:21:997
 0 New operation: "LDAP_Con1_Op0" "BindReq" V3
```

The pool thread creates a new operation object on Con0 and receives data from client: it detects a bind request.

```
          -- 0x45046766 DEBUG5   ldap_op  mn_proxy.cpp     2842
14:21:997
 1 lookup_user_rule(user:"cn=richter,ou=sales,o=my-company")="FOUND"
```

The DLP server extracts the user DN from the **bind** operation and performs a lookup against the operation-forwarding rules for this user.

```
          -- 0x4504676a DEBUG5   ldap_op  mn_proxy.cpp     3014
14:21:997
 1 "get_relevant_servers(cn=richter,ou=sales,o=my-company)=USER_RULE
   URule: USERRULE3 (type:2) (match:cn=richter,ou=sales,o=my-company)
   ORule: none"
```

The DLP server's lookup has found a forwarding rule (USERRULE3) for the user and prepares the possible target servers.

```
          -- 0x4504676a DEBUG5   ldap_op  mn_proxy.cpp     3017
14:21:998
 1 "Found 2 Relevant Servers from Rule"
          -- 0x4504676a DEBUG5   ldap_op  mn_proxy.cpp     3044
14:21:998
 1 "ServerName: LDAP1
   Address:127.0.0.1:1636
   Protocol:LDAPS - TLSv1.2
   Trusted-CA-File:C:/Program Files/DirX/Directory/conf/testCA.pem
   Cipher:HIGH"
          -- 0x4504676a DEBUG5   ldap_op  mn_proxy.cpp     3044
14:21:998
 1 "ServerName: LDAP3
   Address:127.0.0.1:3333
   Protocol:LDAP"
```

The identified user rule defines two target servers (LDAP1, LDAP3) with corresponding server parameters. LDAP1 is the primary server.

```
          -- 0x4504675c DEBUG1   ldap_op  mn_proxy.cpp     742
14:22:005
 2 ldap_connect(sd:836)=PROXY_OK(0)
   OpName:"LDAP_Con1_Op0",
   Server:"127.0.0.1":1636 ("LDAP1") sec:"ssl"
```

The DLP server prepares to establish a LDAP connection via SSL/TLS to LDAP1.

```
          -- 0x4504676d DEBUG4   ldap_op  mn_proxy.cpp     3403
14:22:005
```

```
 1 GetTargetServerConnection(): Using Connection to Server '"LDAP1"'
```

The DLP establishes a connection to the server.

```
          -- 0x45046762 DEBUG4   ldap_op  mn_proxy.cpp     1611
14:22:005
 2 SendLdapDataToServer("LDAP1",sd:836)=PROXY_OK(0)
   Data: (len:277)
        0000  30 82 01 11 02 01 01 60 26 02 01 03 04 19 63 6E
0,.....`&.....cn
        0010  3D 72 69 63 68 74 65 72 2C 6F 75 3D 73 61 6C 65
=richter,ou=sale
        0020  73 2C 6F 3D 70 71 72 80 06 61 62 63 31 32 33 A0
s,o=my-company..abc123
        0030  81 E3 30 75 04 1A 31 2E 33 2E 36 2E 31 2E 34 2E
.ã0u..1.3.6.1.4.
        0040  31 2E 32 31 30 30 38 2E 31 30 38 2E 36 33 2E 31
1.21008.108.63.1
        0050  04 57 30 55 04 0C 31 30 2E 39 33 2E 32 35 2E 31
.W0U..10.93.25.1
        0060  36 32 04 0D 64 69 72 78 63 70 20 5B 37 32 36 30
62..dirxcp [7260
        0070  5D 04 1C 31 2E 33 2E 36 2E 31 2E 34 2E 31 2E 32
]..1.3.6.1.4.1.2
        0080  31 30 30 38 2E 31 30 38 2E 36 33 2E 31 2E 33 04
1008.108.63.1.3.
        0090  18 63 6E 3D 61 62 65 6C 65 39 2C 6F 75 3D 73 61
.cn=abele9,ou=sa
        00A0  6C 65 73 2C 6F 3D 70 71 72 30 6A 04 18 31 2E 33
les,o=my-company0j..1.3
        00B0  2E 31 32 2E 32 2E 31 31 30 37 2E 31 2E 33 2E 32
.12.2.1107.1.3.2
        00C0  2E 31 32 2E 38 04 4E 30 4C 04 24 36 36 65 63 38
.12.8.N0L.$66ec8
        00D0  62 35 31 2D 32 35 31 31 2D 34 38 32 32 2D 38 32  b51-
2511-4822-82
        00E0  62 33 2D 32 33 32 63 30 31 66 61 66 35 34 35 04  b3-
232c01faf545.
        00F0  24 64 66 37 35 64 32 31 35 2D 36 30 36 64 2D 34
$df75d215-606d-4
```

```
        0100  33 62 30 2D 39 37 61 64 2D 39 35 61 37 31 31 61  3b0-
97ad-95a711a
        0110  62 63 63 37 30
bcc70.....
```

The DLP server sends the client **bind** request to the target server (implicitly the OpUUID was appended to the request – this is visible at the end of the hex-dump).

```
          -- 0x45046763 DEBUG4   ldap_op  mn_proxy.cpp     1801
14:22:021
 3 RecvLdapDataFromServer("LDAP1",sd:836)=PROXY_OK(0)
   Data: (len:52)
        0000  30 32 02 01 01 61 2D 0A 01 00 04 00 04 26 42 69
02...a-......&Bi
        0010  6E 64 20 73 75 63 63 65 65 64 65 64 2E 20 28 43  nd
succeeded. (C
        0020  6F 6E 74 61 63 74 2D 44 53 41 3A 2F 43 4E 3D 44
ontact-DSA:/CN=D
        0030  53 41 33 29
SA3).....
```

The DLP server receives the bind result PDU from the LDAP server.

```
          -- 0x45046765 DEBUG5   ldap_op  mn_proxy.cpp     1438
14:22:021
 4 DecodeLdapPdu(pduType:1, data:0x0000000004C24F50,
data_len:52)=PROXY_OK(0)
   Decoded:
        STRUCT LDAP_Message_V3 {
          bit_mask = 0 ;
          messageID = 0x1 ;
          protoOp = UNION ProtoOp {
            choice = 0x2 ;
            u.bindResp_V3 = STRUCT LDAP_BindResp_V3 {
              bit_mask = 0 ;
              resultCode = LDAP_SUCCESS(0) ;
              matchedDN = <ABSENT> ;
              errorMessage = 38 char value(s)
                0000  42 69 6E 64 20 73 75 63 63 65 65 64  Bind
succeed
```

```
               000C  65 64 2E 20 28 43 6F 6E 74 61 63 74  ed.
(Contact
               0018  2D 44 53 41 3A 2F 43 4E 3D 44 53 41
-DSA:/CN=DSA
               0024  33 29
3).........
             referral_V3 = <ABSENT> ;
             serverSaslCreds_V3 = <ABSENT> ;
           }
         }
         controls_V3 = <ABSENT> ;
       }
```

The DLP server decodes the received bind result PDU for two reasons: a) to get the LDAP result code and b) to check whether the received data is really a correct bind response PDU. In this example, PDU and bind were OK.

```
         -- 0x45046764 DEBUG4   ldap_op  mn_proxy.cpp     1491
14:22:021
 3 SendLdapDataToClient(sd:1216)=PROXY_OK(0)
   Data: (len:52)
        0000  30 32 02 01 01 61 2D 0A 01 00 04 00 04 26 42 69
02...a-......&Bi
        0010  6E 64 20 73 75 63 63 65 65 64 65 64 2E 20 28 43  nd
succeeded. (C
        0020  6F 6E 74 61 63 74 2D 44 53 41 3A 2F 43 4E 3D 44
ontact-DSA:/CN=D
        0030  53 41 33 29
```

The DLP server sends the received result back to the client.

```
                                   SA3)............
         -- 0x4504675b DEBUG1   ldap_op  mn_proxy.cpp     1259
14:22:021
 1 ldap_bind(user:"cn=richter,ou=sales,o=my-company")=PROXY_OK(0)
   OpName:"LDAP_Con1_Op0",
   Server:"127.0.0.1":1636 ("LDAP1") sec:"ssl" sd:836
```

The final log indicates that the forwarded bind was OK.

5.3. DLP Server Audit

The DLP server uses the same audit mechanism as the plain LDAP server.The only major difference is the record layout.The same common tool **dirxauddecode** can be used to evaluate DSA audit, LDAP server audit and DLP server audit.Note that the **dirxaudstatistics** tool does not recognize proxy-mode records and thus cannot be used to evaluate DLP server audit files

5.3.1. DLP Server Audit Record Layout

The layout of DLP server audit records is quite different from the LDAP server records due to the different tasks of customizing or forwarding a request rather than processing it.Nevertheless, the evaluation of a DLP server audit file is still the same.For example, consider the following command:

```
dirxauddecode -i audit.log -a audit.log.txt -v -v
```

The command produces an ASCII file with an initial header that looks very similar to the LDAP audit header.The only difference is that the Content Type field is PROXY instead of LDAP:

```
#################  DIR.X AUDIT TRAIL (c) Eviden
################################
Cmd-Line: -i audit.log -a audit.log.txt -v -v
================================================================================
============
Audit  File #             :1
Input  File               :audit.log
Output File               :audit.log.txt
Audit Version             :9.2
Server UUID               :0235b963-f69d-403f-8918-2425fef3ae34
Audit Start Time (local)  :Thu Apr 20 10:13:39 2017
Audit Start Time (GMT)    :Thu Apr 20 08:13:39 2017
Content Type              :PROXY
OpSelection               :all
OpErrors                  :yes
Audit Level               :max
Audit Encryption          :none
Overflow Action:          :move
Max Records per File      :5000
Value Limit               :128
Server-PID                :7680
DB Master-Entries         :118503
DB Copy-Entries           :0
```

```
Filter Records with IP    :---
Ignore Records with IP    :---
Included LDAP Ops         :All
Included DSA  Ops         :All
OS Name                   :Microsoft Windows 7  64-bit- Service Pack
1 (build 7601)
Total Phys Memory         :16266 MB
Avail Phys Memory         :11851 MB
Allocated CTX Size        :56 MB
HWM CTX Size              :88 MB
CTX ULimit                :6000 MB
MemPagesize               :4096
CPUs                      :4
Max Open Files soft       :unlimited
Max Open Files hard       :unlimited
Audit Disk Space Total    :807641084 kB
Audit Disk Space Free     :465029860 kB
PID                       :7680
Host Name                 :XXXXXXXX
Host IP                   :1.1.1.1
Server Version            :DirX Directory V8.6 9.2.104 2017:03:25
20:10 64-Bit
Server Type               :Frontend Proxy Server
Server Mode               :Read/Write
Contact-DSA               :Name=/CN=DSA1, enabled=yes, fails=0,
PSAP=TS=DSA1,NA='TCP/IP_IDM!internet=1.2.3.4+port=4711',DNS='(HOST=XX
XXXXX,SSLPORT=21201,PLAINPORT=21200,MODE=ssl)'
SSL Encryption            :SSLv3.0 TLSv1.0 TLSv1.1 TLSv1.2
Server Start Time         :Thu Apr 20 10:13:38 2017
Configuration Name        :ldapConfiguration
ClCfg File                :C:\Program
Files\DirX\Directory\ldap\conf\dirxldap.cfg
Ldap Port                 :389
SSL Port                  :636
RPC Port                  :6999
Max Conn                  :777
Client Idle Time          :300
TCP/IP Response Mode      :24
Socket Mode               :async
Thread Pool Size          :32
DN Escape Mode            :backslash
```

```
Allowed IPs                :all
Denied IPs                 :12.23.34.45
Denied IPs                 :11.22.33.44
Denied IPs                 :100.101.102.103

DB-Index-Info:
Attr:                                   uid : initial final
present
Attr:                                userid : initial final
present
Attr:                             objectClass : initial
Attr:                                   ocl : initial
Attr:                                    cn : initial final
Attr:                              commonName : initial final
Attr:                                    sn : initial final
Attr:                                 surname : initial final
Attr:                                       c : initial final
Attr:                             countryName : initial final
Attr:                                       o : initial final
Attr:                         organizationName : initial final
Attr:                collectiveOrganizationName : initial final
Attr:                                      ou : initial final
Attr:                    organizationalUnitName : initial final
Attr:         collectiveOrganizationalUnitName : initial final
Attr:                             description : initial final
Attr:                                   dsc : initial final
Attr:                                remarks : initial final
contains
========================================================================
============
DB-Index-Usage:
Attribute access counter high score at Thu Apr 20 10:12:45 2017 :
Attribute name           : Index access counter
                         :      INITIAL        FINAL     CONTAINS
PRESENT
uid                      :      2719993            0           -
0
cn                       :      1175911            0           -
-
objectClass              :           95            -           -
-
```

```
description              :           4          14           -
-
o                        :           4           0           -
-
sn                       :           0           3           -
-
ou                       :           2           0           -
-
================================================================================
============
```

that any information about the DSA or the database is derived from the **local DSA** and not from the target servers. For example, the information:

```
DB Master-Entries :118503
```

describes the local DSA DB and provides no information about the target server's database.

The single operation records follow the header. These records have a different layout from plain LDAP records since the server is running as a DLP server, not as a plain LDAP server.

5.3.2. Bind, Search, Add Example

To illustrate the meaning of an audit record, let's assume a user has performed three operations: a bind, a search and an add request with the following DLP server proxy rules defined for these three operations:

```
{
   // redirect user 'richter' to LDAP1
   "object"       : "ProxyRule",
   "ruleType"     : "UserRouting",
   "name"         : "USERROUTING1",
   "condition"    : "(user=cn=richter,ou=sales,o=pqr)",
   "actions"      : [ "forwardto(LDAP1)" ],
   "loadbalance"  : 1,
   "failover"     : 1
},

{
   // redirect ADDs to LDAP2
   "object"     : "ProxyRule",
   "ruleType"   : "OprRouting",
   "name"       : "OPRROUTING2",
```

```
   "condition"  : "opr.req.type=add",
   "actions"    : [ "singleforwardto(LDAP2)" ],
   "failover"   : 0,
   "keepconn"   : 1
},

{
    // change o=my-company to o=pqr for search bases
   "object"    : "ProxyRule",
   "ruleType"  : "ReqRewrite",
   "name"      : "ReqRewrite1",
   "condition" : "opr.req.type=search",
   "actions"   : [ "search.req.baseObject.replace(o=my-
company,o=pqr,NULL)" ]
}

{
   // remove sn=Digger from any search result
   // add description=blabla to all result entries
   "object"    : "ProxyRule",
   "ruleType"  : "ResRewrite",
   "name"      : "Test 04",
   "condition" : "opr.req.type=search",
   "actions" : [ "search.res.attributes.del(sn,Digger,NULL)",
                 “search.res.attributes.add(description,blabla,NULL)"
]
}
```

Now let’s look at the audit output. Note that in the example shown here, fields that have the same meaning as for LDAP audits are not described.

5.3.2.1. The DLP Server Bind Record

Let’s start with the DLP server **bind** record:

```
----------------- OPERATION 000001 ----------------
  Create Time    :Wed Jul 25 14:46:33.036496 2018
  Start Time     :Wed Jul 25 14:46:33.036571 2018
  SrvSend EndTime:Wed Jul 25 14:46:33.062344 2018
  SrvRecv EndTime:Wed Jul 25 14:46:33.072868 2018
  ClSend EndTime :Wed Jul 25 14:46:33.074132 2018
  End Time       :Wed Jul 25 14:46:33.075126 2018
```

```
  OpUUID          :35d6c5ad-89fa-44e3-84d3-845f70b97f51
  Concurrency     :1
  OpStackSize     :1
  OpFlow In/Out   :0/0
  Contact-Server :127.0.0.1:1636 (LDAP1)
  SrvRelRule      :USERROUTING1
  SrvSecurity     :ssl
  SrvSslProtocol :TLSv1.2
  SrvSslCipher    :ECDHE-RSA-AES256-GCM-SHA384
  #ContactedSrv   :1
  SrvSocket       :1232
  SvrErrno        :0
  SvrConnectDur   :0.014270 sec
  SvrSendDur      :0.000299 sec (1 calls)
  SvrRecvDur      :0.010132 sec (2 calls)
  SvrBytesSent    :279
  SvrBytesRecv    :29
  Duration        :0.038555 sec
  User            :cn=richter,ou=sales,o=pqr
  ClIP+Port+Sd    :[127.0.0.1]+55631+668
  Op-Name         :LDAP_Con0_Op0
  Operation       :BIND
  Version         :3
  MessageID       :1
  ClSecurity      :plain
  ClRecvDur       :0.005230 sec (3 calls, 0 WouldBlocks,
WouldblockTime:0.000000 sec)
  ClSendDur       :0.000021 sec (1 calls, 0 WouldBlocks,
WouldblockTime:0.000000 sec)
  ReqRewriteDur   :0.000584 sec
  ResRewriteDur   :0.000000 sec
  ClBytes Rcvd    :169
  ClBytes Sent    :29
  LdapResultCode :0 (success)
  ProxyResultCode:0
```

As shown in the lines above, all records start with some absolute timestamps that give the time at which the operation was processed.

The line:

```
Create Time    :Wed Jul 25 14:46:33.036496 2018
```

shows the time at which the request was recognized by the DLP server via TCP to come in from the LDAP client.

The line:

```
Start Time     :Wed Jul 25 14:46:33.036571 2018
```

shows the time at which the DLP server started processing the request by reading the LDAP PDU, decoding it, etc. Usually this time is close to the CreateTime which shows that the server was not too busy and processing started immediately. If this time is significantly later (by seconds) it indicates that no pool thread was available to process the request and the request had to wait for a pool thread to become available. This kind of message may indicate a heavy load situation, especially if the concurrency (number of parallel processed operations) is high.

The line:

```
SrvSend EndTime:Wed Jul 25 14:46:33.062344 2018
```

shows the time at which the request was completely forwarded to the target LDAP server (when the local processing (deciding which target server to use, decoding the message, performing request rewrite actions) has completed.

The line:

```
SrvRecv EndTime:Wed Jul 25 14:46:33.072868 2018
```

shows the time the answer from the server was completely received (for search results, this includes the receipt of all resulting entries).

The line:

```
ClSend EndTime :Wed Jul 25 14:46:33.074132 2018
```

shows the time at which the result (after applying all result rewriting rules) was completely sent out to the client.

The line:

```
End Time       :Wed Jul 25 14:46:33.075126 2018
```

shows the time at which the operation ended within the DLP server (after cleanup, audit write and other operations.).

This line:

```
Contact-Server :127.0.0.1:1636 (LDAP1)
```

shows the target LDAP server to which the request was finally forwarded, with IP, port and logical server name taken from the DLP server configuration file.

The line:

```
SrvRelRule     :USERROUTING1
```

shows which rule defined the target server. In our example, it was USERROUTING1 for user **richter**. Note that only user-routing rules or the LB fallback servers have any effect on this property.

These lines:

```
  SrvSecurity    :ssl
  SrvSslProtocol :TLSv1.2
  SrvSslCipher   :ECDHE-RSA-AES256-GCM-SHA384
```

show that the target server (LDAP1) was contacted via SSL/TLS.

These lines:

```
  SrvSocket      :1232
  SvrErrno       :0
  SvrConnectDur  :0.014270 sec
  SvrSendDur     :0.000299 sec (1 calls)
  SvrRecvDur     :0.010132 sec (2 calls)
  SvrBytesSent   :279
  SvrBytesRecv   :29
```

provide details about the socket number used to transfer data to the target server, how long it took to establish a TCP connection (if it was necessary to create a new connection to the target), how long it took and how many send() calls were necessary to send out the

request and for the result to be received along with the amount of data sent out and received from the target LDAP server.

This line:

```
Duration       :0.038555 sec
```

shows the time the DLP server took to process the request (including the time spent at the target server).

These lines:

```
  ClSecurity     :plain
  ClRecvDur      :0.005230 sec (3 calls, 0 WouldBlocks,
WouldblockTime:0.000000 sec)
  ClSendDur      :0.000021 sec (1 calls, 0 WouldBlocks,
WouldblockTime:0.000000 sec)
  ClBytes Rcvd   :169
  ClBytes Sent   :29
```

provide information about the client activity: how it accessed the server (via a plain connection, not SSL/TLS) how long it took to read the request from the client, how long it took to send the result back to the client and the amount of data received in the request/response.

These lines:

```
  ReqRewriteDur  :0.000584 sec
  ResRewriteDur  :0.000000 sec
```

show how long it took to rewrite the request or the result. Please note that even if NO rule matches and NO rewrite takes place, the DLP server needs time to evaluate these conditions.

These lines:

```
  LdapResultCode :0 (success)
  ProxyResultCode:0
```

show that we have two result codes: the result code as received from the LDAP server and the result code about the DLP server processing. It can be that the request was successfully processed by the LDAP server but could not be sent back to the client (for example, due to network problems).

5.3.2.2. The Search Record

The subsequent search received from the client by the DLP server looks like this:

```
----------------- OPERATION 000002 ----------------
  Create Time    :Wed Jul 25 14:46:35.501099 2018
  Start Time     :Wed Jul 25 14:46:35.501191 2018
  SrvSend EndTime:Wed Jul 25 14:46:35.510630 2018
  SrvRecv EndTime:Wed Jul 25 14:46:35.761931 2018
  ClSend EndTime :Wed Jul 25 14:46:35.764063 2018
  End Time       :Wed Jul 25 14:46:35.764587 2018
  OpUUID         :a14f41da-e11a-43c1-a8e8-6afde8d3472f
  Concurrency    :1
  OpStackSize    :1
  OpFlow In/Out  :0/0
  Contact-Server :127.0.0.1:1636 (LDAP1)
  SrvSecurity    :ssl
  SrvSslProtocol :TLSv1.2
  SrvSslCipher   :ECDHE-RSA-AES256-GCM-SHA384
  SrvSocket      :1232
  SvrErrno       :0
  SvrConnectDur  :0.000000 sec
  SvrSendDur     :0.000362 sec (1 calls)
  SvrRecvDur     :0.068365 sec (60 calls)
  SvrBytesSent   :284
  SvrBytesRecv   :1301
  Duration       :0.263396 sec
  User           :cn=richter,ou=sales,o=pqr
  ClIP+Port+Sd   :[127.0.0.1]+55631+668
  Op-Name        :LDAP_Con0_Op1
  Operation      :SEARCH
  Version        :3
  MessageID      :0
  ClSecurity     :plain
  ClRecvDur      :0.000405 sec (3 calls, 0 WouldBlocks,
WouldblockTime:0.000000 sec)
  ClSendDur      :0.000340 sec (30 calls, 0 WouldBlocks,
WouldblockTime:0.000000 sec)
  ReqRewriteDur  :0.004085 sec
  ResRewriteDur  :0.115512 sec
  NumResEntries  :29
```

```
  ClBytes Rcvd   :181
  ClBytes Sent   :2043
  LdapResultCode :0 (success)
  ProxyResultCode:0
  ProxyRuleExec  :ReqRewrite1 (CallCount: 1, ItemsModified: 1)
  ProxyRuleExec  :Test 04 (CallCount: 29, ItemsModified: 29)
```

Most fields are identical to the bind record audit output. We can see that the search went to the same target (LDAP1) on the same socket as the bind operation.

The last lines of the output show the list of rules that were applied to this operation (**ProxyRuleExec**). Each line refers to a separate rule and indicates how many times the condition was evaluated (**CallCount**) and how many changes/actions were executed (**ItemsModified**). We see that one item in the request was modified. From the corresponding rule (**ReqRewrite1**), we know that the incoming **baseObject** was modified. The returned results were modified by rule **Test 04**. We know that 29 entries were returned (**NumResEntries=29**) and all of them were modified (**ItemsModified=29**).

You may notice that the following two values differ:

```
ClBytes Rcvd   :181   // bytes received from client
SvrBytesSent   :284   // bytes forwarded to backend LDAP
```

Shouldn't they be the same? In principle yes, but the DLP server adds a control to the request that allows shifting the DLP server-generated UUID to the LDAP server, which uses this UUID in its corresponding audit log record and even passes it to the DSA, which also includes it into its audit. This operation makes it easier to locate the corresponding record in DLP server/LDAP/DSA audits. As it is appended to the incoming request, the size of the forwarded request will grow a little.

5.3.2.3. The Add Record

The subsequent **add** operation received from the client by the DLP server looks like this:

```
----------------- OPERATION 000003 ----------------
  Create Time    :Wed Jul 25 14:47:30.304363 2018
  Start Time     :Wed Jul 25 14:47:30.304413 2018
  SrvSend EndTime:Wed Jul 25 14:47:30.340626 2018
  SrvRecv EndTime:Wed Jul 25 14:47:30.343205 2018
  ClSend EndTime :Wed Jul 25 14:47:30.344562 2018
  End Time       :Wed Jul 25 14:47:30.345118 2018
  OpUUID         :36fadde3-274c-4e23-a96c-508395b515d2
  Concurrency    :1
  OpStackSize    :1
```

```
  OpFlow In/Out  :0/0
  Contact-Server :127.0.0.1:2636 (LDAP2)
  SrvSecurity    :ssl
  SrvSslProtocol :TLSv1.2
  SrvSslCipher   :ECDHE-RSA-AES256-GCM-SHA384
  #ContactedSrv  :1
  SrvSocket      :836
  SvrErrno       :0
  SvrConnectDur  :0.017388 sec
  SvrSendDur     :0.000840 sec (2 calls)
  SvrRecvDur     :0.010794 sec (4 calls)
  SvrBytesSent   :570
  SvrBytesRecv   :78
  Duration       :0.040705 sec
  User           :cn=richter,ou=sales,o=pqr
  ClIP+Port+Sd   :[127.0.0.1]+55631+668
  Op-Name        :LDAP_Con0_Op2
  Operation      :ADD
  Version        :3
  MessageID      :0
  ClSecurity     :plain
  ClRecvDur      :0.000162 sec (3 calls, 0 WouldBlocks,
WouldblockTime:0.000000 sec)
  ClSendDur      :0.000033 sec (1 calls, 0 WouldBlocks,
WouldblockTime:0.000000 sec)
  ReqRewriteDur  :0.000616 sec
  ResRewriteDur  :0.000000 sec
  ClBytes Rcvd   :181
  ClBytes Sent   :49
  LdapResultCode :32 (noSuchObject)
  ProxyResultCode:0
  ProxyRuleExec  :OPRROUTING2 (CallCount: 1, ItemsModified: 0)
```

At the end of the output, we see that OPRROUTING2 was applied. From the proxy rule definition given at the start of this example, we know that this is an operation-routing rule and should redirect the operation to LDAP2, which is confirmed by the line:

```
Contact-Server :127.0.0.1:2636 (LDAP2)
```

We can also see that the LDAP operation failed with a “noSuchObject” error, but the processing was fine from the DLP server’s point of view:

```
LdapResultCode :32 (noSuchObject)
ProxyResultCode:0
```

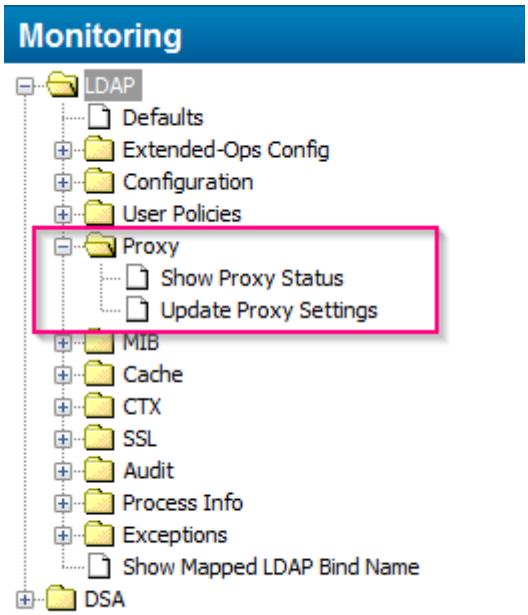
5.4. LDAP Extended Operations for DLP Servers

DirX provides the following LDAP operations to observe and control a DLP server:

- **ldap_proxy_status** - Shows the current status of the server
- **ldap_proxy_update** - Triggers a runtime update of the DLP server configuration file
- **ldap_proxy_server_disable** - Disables an LDAP target server
- **ldap_proxy_server_enable** - Re-enables a disabled LDAP target server

All of these extended operations can be invoked by the **dirxextop** tool.See the *DirX Directory Administration Reference* for a description of **dirxextop**.

The DLP server status display and update trigger can also be invoked in the DirX Directory Manager with the Monitoring view:



Disabling and enabling a target server can only be performed via **dirxextop**.

5.4.1. ldap_proxy_server_disable

5.4.1.1. Synopsis

ldap_proxy_server_disable *server_name*

5.4.1.2. Purpose

Removes an LDAP server from selection as a target server for the DLP server.

5.4.1.3. Parameters

server_name
The name of the target server to be disabled, as specified in the DLP server configuration file.

5.4.1.4. Description

The **ldap_disable_config_dsa** LDAP extended operation allows DirX Directory administrators to permanently disable a target LDAP server for a DLP server; for example, before taking it off-line for maintenance.

Use the mandatory *server_name* parameter in the **-P** option to **dirxextop** to specify the name of the LDAP server to be disabled.

The OID of this extended operation is 1.3.12.2.1107.1.3.2.11.62.

When you use this operation to disable a server, you must explicitly re-enable it with the **ldap_proxy_server_enable** LDAP extended operation before the DLP server can select it again.

5.4.1.5. Example

The following example shows how to apply the **ldap_proxy_server_disable** LDAP extended operation with the **dirxextop** command. In the example, the target LDAP server **LDAP2** is disabled:

```
dirxextop -h localhost -p 389 -D cn=admin,o=my-company -w dirx -t
ldap_proxy_server_disable -P LDAP2
```

Use the **ldap_proxy_status** extended operation to view the result:

```
------------------------------------------------------------------------
--------
---------------------------[ Servers
]-------------------------------------
------------------------------------------------------------------------
--------
NumLdapServers        : 5 (SelCount:2)
Server                : Name=LDAP5, Status:AVAILABLE
                        Addr=10.93.25.72:3333, Sec=plain,
                        ConnOK=0, ConnFail=0, OpOK=0, OpFail=0
```

```
Server                : Name=LDAP4, Status:AVAILABLE
                        Addr=127.0.0.1:3333, Sec=plain,
                        ConnOK=0, ConnFail=0, OpOK=0, OpFail=0
Server                : Name=LDAP3, Status:AVAILABLE
                        Addr=127.0.0.1:3333, Sec=plain,
                        ConnOK=0, ConnFail=0, OpOK=0, OpFail=0
Server                : Name=LDAP2, Status:DISABLED since Tue Apr 18
16:00:11.115654
                        Addr=127.0.0.1:2636, Sec=TLSv1.2,
                        ConnOK=0, ConnFail=0, OpOK=0, OpFail=0
Server                : Name=LDAP1, Status:AVAILABLE
                        Addr=127.0.0.1:1636, Sec=TLSv1.2,
                        ConnOK=2, ConnFail=0, OpOK=3, OpFail=0
```

The server section of the output shows that the server LDAP2 has the status DISABLED.

5.4.1.6. See Also

ldap_proxy_server_enabled

ldap_proxy_status

5.4.2. ldap_proxy_server_enable

5.4.2.1. Synopsis

ldap_proxy_server_enable *server_name*

5.4.2.2. Purpose

Returns an LDAP server to possible selection as a target server.

5.4.2.3. Parameters

server_name
The name of the target LDAP server to be enabled, as specified in the DLP server configuration file.

5.4.2.4. Description

The **ldap_proxy_server_enable** LDAP extended operation allows DirX Directory administrators to enable an LDAP server that was previously disabled with the **ldap_proxy_server_disable** LDAP extended operation.

Use the mandatory *server_name* parameter in the **-P** option to **dirxextop** to specify the name of the LDAP server to be enabled.

What is the OID for this operation is 1.3.12.2.1107.1.3.2.11.63.

5.4.2.5. Example

The following example shows how to apply the **ldap_enable_config_dsa** LDAP extended operation with the **dirxextop** command. In the example, the LDAP server **LDAP2** is enabled:

```
dirxextop -h localhost -p 389 -D cn=admin,o=my-company -w dirx -t
ldap_proxy_server_enable -P LDAP2
```

5.4.2.6. See Also

ldap_proxy_server_enabled

ldap_proxy_status

5.4.3. ldap_proxy_status

5.4.3.1. Synopsis

ldap_proxy_status

5.4.3.2. Purpose

Displays the current DLP server status regarding servers, rules and selections.

5.4.3.3. Description

On success, this operation returns the current state of the DLP server in readable format. The output consists of a header section, a server section and a rule section.

The header section provides general information about the DLP server, including:

- The selected LDAP configuration subentry name for the DLP server. By default, the proxy-name is **ldapConfiguration**, which is the name of the server's default LDAP configuration subentry if not set otherwise. (See the section in the *DirX Directory Administration Guide* that describes how to add multiple LDAP servers for details.)
- The proxy mode in which the DLP server is currently running. A value of **1** indicates that SSL/TLS is not in use on any target server. A value of **2** indicates that at least one target server has been contacted via SSL/TLS.
- The currently active Proxy-ID number. At DLP server startup, this field is set to **1** and is incremented with every **ldap_proxy_update** call. You can use this field to determine how many online updates of the DLP server configuration have occurred to date.
- Whether or not failover is enabled. When enabled, the DLP server uses the next configured LDAP server when a failure is detected. We recommend enabling this feature in the operation-forwarding rule object definition(s) (with the **failover** key) and/or in the Defaults object definition (with the **LdapProxy.lb_failover** key. See the

"Configuration" chapter for details.

- Whether or not the primary LDAP server function is shifted among the available load-balancing LDAP servers (LB-servers). When disabled, all traffic for users without any forwarding rule will go to the first LB-server. We recommend enabling the feature in the Defaults object definition (with the **LdapProxy.lb_balance** key). See the "Configuration" chapter for details. Note that load balancing is not performed for users that match a configured user, subuser or wildcard user rule.
- How long (in seconds) the DLP server will wait for a TCP connect to be confirmed before dropping the server and switching to the next server (if possible).
- How long (in seconds) the DLP server will wait to select a failed target server for new client connections.
- The character encoding of the DLP server configuration file.
- Whether (**1**) or not (**0**) notification of request/result rewriting is included in the LDAP response error message.
- The string used to indicate an empty parameter in protocol-specific rewriting actions.
- The level of tracing information sent to **stderr**.
- How many active subscribers (client connections) (and thus implicitly which target servers) were selected against the settings of this DLP server configuration. Every new bind operation always uses the most recent DLP server configuration and increments the subscription counter for this DLP server configuration. Every unbind operation will decrement the subscription counter (DLP server configuration objects with a subscription count of 0 are scheduled to be removed at next automatic cleanup).
- The last subscribed user to this DLP server configuration.

The "Processed LDAP Requests" section displays how many operations of which type have been processed to date and the number of result entries returned by searches.

The server section:

- Lists all available target servers by their name, IP, port and whether they will be contacted via SSL/TLS or plain connections.
- Displays the current status of each server. Possible states are:
- **AVAILABLE** (the server can be selected)
- **OFFLINE** (the server cannot be selected for some period of time)
- **DISABLED** (the server will not be selected until it has been re-enabled).
- Shows how many successful/failed connects and operations have been seen for each server. Note that for the DLP server, an operation fails if an I/O error occurs, but not because an LDAP operation returns an error code != 0. The LDAP result code never affects DLP server operation.

The user- and operation-routing rules section shows the rules of these types that have been defined in the DLP server configuration file. For each rule, the output displays:

- The rule condition that must be met

- How many times the rule has been selected to date
- The servers that are involved if the rule is selected
- Whether failover, load-balancing (user-routing only), and keep connection (operation-routing only) are in force

The request- and result-rewriting rules section shows the rules of these types that have been defined in the DLP server configuration file. For each rule, the output displays:

- The rule conditions that must be met and the actions to be performed.
- How many times the rule matched a request or a result to date. Note that a result matching can occur for each returned entry of a search rules. Consequently, if a search result contains *n* result entries, the selection count can increase by *n*.
- How many attributes and/or values the rule actions have actually changed to date. Note that if *m* actions are applied to *n* result entries of a single search request, the counter is increased by *n***m*.

The OID for this extended operation is **1.3.12.2.1107.1.3.2.11.61**.

5.4.3.4. Example

The following example shows how to apply the **ldap_proxy_status** LDAP extended operation on the local LDAP server (which is running with the complete database and the default ports on the local machine) with the **dirxextop** command:

```
dirxextop -D cn=admin,o=my-company -w dirx -t ldap_proxy_status
```

The LDAP extended operation returns output like the following:

```
================================================================================
++++ LDAP-Proxy-Configuration Status at:Thu Oct 11 10:12:34.576991
+++ Proxy-Name: ldapConfiguration
+++ Proxy-Mode: 2 (support for SSL/TLS target servers)
+++ Number of existing Proxy-Config Objects: 1
--------------------------------------------------------------------------------
Active Proxy-ID       : 1
CreateTime            : Thu Oct 11 10:10:58.144897
LB-Failover           : yes
LB-Balance            : yes
ConnectTimeout        : 5 sec
OfflineRetryTime      : 30 sec
JSON CodeSet          : UTF8
```

```
NotifyRewrite         : yes
NullParamStr          : NULL
Tracing               : 0
# Active Subscribers  : 2
LastSubscriber        : Conn:LDAP_Con1,
User:cn=richter,ou=sales,o=pqr, Time:Thu Oct 11 10:12:10.301399
------------------------------------------------------------------------
--------
--------------------[ Processed LDAP Requests
]-----------------------------
------------------------------------------------------------------------
--------
Total Requests : 16
Binds          : 2
Searches       : 13  (17 ResultEntries)
Adds           : 0
Modifys        : 0
Deletes        : 0
ModDNs         : 0
Compares       : 0
Unbinds        : 0
ExtOPs         : 1
------------------------------------------------------------------------
-----------------------------------[ Servers
]-------------------------------------
------------------------------------------------------------------------
--------
NumLdapServers        : 5 (SelCount:1)
Server                : Name=LDAP5, Status:AVAILABLE
                        Addr=10.93.25.72:3333, Sec=plain, ConnOK=0,
ConnFail=0, OpOK=0, OpFail=0
Server                : Name=LDAP4, Status:AVAILABLE
                        Addr=127.0.0.1:3333, Sec=plain, ConnOK=0,
ConnFail=0, OpOK=0, OpFail=0
Server                : Name=LDAP3, Status:AVAILABLE
                        Addr=127.0.0.1:3333, Sec=plain, ConnOK=0,
ConnFail=0, OpOK=0, OpFail=0
Server                : Name=LDAP2, Status:AVAILABLE
                        Addr=127.0.0.1:2636, Sec=TLSv1.2, ConnOK=0,
ConnFail=0, OpOK=0, OpFail=0
Server                : Name=LDAP1, Status:AVAILABLE
```

```
                        Addr=127.0.0.1:1636, Sec=TLSv1.2, ConnOK=1,
ConnFail=0, OpOK=1, OpFail=0
------------------------------------------------------------------------
--------
NumLBServers          : 3  (SelCount:0)
LBServers             : LDAP1->LDAP2->LDAP3
------------------------------------------------------------------------
--------
-------------------------[ UserRoutingRules
]------------------------------
------------------------------------------------------------------------
--------
NumUserRules          : 2  (TotSelCount:2)
 Rule-Name  : USERROUTING1
  Condition       :
(|(user=cn=richter,ou=sales,o=pqr)(wcuser=cn=IADM.*)(subuser=ou=sales
,o=pqr))
  Selection-Count : 1
  Failover        : yes
  Load-balance    : 0
  #Target-Servers : 2
  Target-Servers  : LDAP2->LDAP1
------------------------------------------------------------------------
--------
 Rule-Name  : USERROUTING2
  Condition       :
(|(user=cn=admin,o=pqr)(wcuser=^cn=ad.*)(subuser=ou=sdevelopment,o=pq
r))
  Selection-Count : 1
  Failover        : yes
  Load-balance    : 0
  #Target-Servers : 2
  Target-Servers  : LDAP3->LDAP1
------------------------------------------------------------------------
--------
--------------------------[ OprRoutingRules
]------------------------------
------------------------------------------------------------------------
--------
NumOprRules           : 2  (TotSelCount:5)
 Rule-Name  : OPRROUTING1
```

```
  Condition       :
(&(opr.req.type=search)(search.req.control=simplePagedResult))
  Selection-Count : 5
  Failover        : no
  KeepConn        : yes
  #Target-Servers : 2
  Target-Servers  : LDAP2->LDAP1
-------------------------------------------------------------------
--------
 Rule-Name  : OPRROUTING2
  Condition       :
(&(opr.req.type=search)(search.req.baseObject=o=pqr))
  Selection-Count : 0
  Failover        : yes
  KeepConn        : yes
  #Target-Servers : 2
  Target-Servers  : LDAP1->LDAP3
-------------------------------------------------------------------
--------
--------------------[ RequestRewriteRules
]---------------------------------
-------------------------------------------------------------------
--------
NumReqRewriteRules  : 5
 Rule-Name  : REQREWRITE0
  Condition       : (opr.req.type=search)
  Selection-Count : 13
  Change-Count    : 0
  Action          : SEARCH.res.entry.hide(o=my-company,NULL,NULL)
-------------------------------------------------------------------
--------
 Rule-Name  : REQREWRITE1
  Condition       : (&(opr.req.type=search)(user=anonymous))
  Selection-Count : 0
  Change-Count    : 0
  Action          : search.req.filter.replace(mycn,cn,NULL)
  Action          : search.req.filter.replace(*,abele,richter)
  Action          : search.req.filter.replace(mycn,abc,NULL)
-------------------------------------------------------------------
--------
 Rule-Name  : REQREWRITE2
```

```
  Condition       : (opr.req.type=modify)
  Selection-Count : 0
  Change-Count    : 0
  Action          : modify.req.object.replace(my-company,pqr,NULL)
  Action          : modify.req.changes_add.replace(abc,23,45)
-------------------------------------------------------------------
--------
 Rule-Name  : REQREWRITE4
  Condition       : (opr.req.type=modDN)
  Selection-Count : 0
  Change-Count    : 0
  Action          : modDN.req.newrdn.replace(myxx,cn,NULL)
  Action          : modDN.req.newsup.replace(xxx,sales2,NULL)
-------------------------------------------------------------------
--------
 Rule-Name  : REQREWRITE5
  Condition       : (&(opr.req.type=compare)(compare.req.attr=road))
  Selection-Count : 0
  Change-Count    : 0
  Action          : compare.req.attr.replace(road,street,NULL)
-------------------------------------------------------------------
--------
--------------------[ ResultRewriteRules
]----------------------------------
-------------------------------------------------------------------
--------
NumResRewriteRules  : 1
 Rule-Name  : RESREWRITE0
  Condition       : (opr.req.type=search)
  Selection-Count : 18
  Change-Count    : 1
  Action          : search.res.entry.hide(Abele72,NULL,NULL)
  Action          : search.res.attributes.replace(street,road,NULL)
  Action          :
search.res.attributes.replace(road,Schönweg,Schoenweg)
  Action          :
search.res.attributes.replace(road,Schoenweg,Hausweg)
  Action          :
search.res.attributes.replace(description,jensen,#4DC3BC6C6C6572)
  Action          :
search.res.attributes.replace(description,Der\,Hund,Der Wolf)
-------------------------------------------------------------------
```

```
--------------------------------------------------------------------------
--------
--------------------------------------------------------------------------
--------
+++LDAP-Proxy Status finished at:Thu Oct 11 10:12:34.577093
==========================================================================
========
```

5.4.3.5. See Also

(Configuration chapter) DLP Server Configuration Objects

(Proxy Rules chapter) User-routing Rules, Operation-routing Rules, Rewriting Rules

ldap_proxy_server_enable

ldap_proxy_server_disable

5.4.4. ldap_proxy_update

5.4.4.1. Synopsis

ldap_proxy_update

5.4.4.2. Purpose

Activates changes made to the DLP server configuration file for a DLP server without having to re-start the server.

5.4.4.3. Description

This operation is used to update the settings from the DLP server configuration file while the DLP server is running. Use this extended operation when you've added new rules or servers to the configuration file and want to update the DLP server without interrupting the service.

If an update is performed, chances are that there are existing connections that were established against the previous DLP server configuration settings. To maintain a clean relationship between existing users and the DLP server settings, the old settings are preserved as long as there are users that were bound when these old settings were active. As a result, multiple DLP server configurations can coexist and multiple users can be subscribed to these different configurations. To which configuration a user is subscribed depends solely on the time of its connection to the DLP server. When the DLP server configuration is updated via an **ldap_proxy_update** call, a new configuration instance is created to which all subsequent users will subscribe. Users that were subscribed before the update remain attached to the old configuration and will thus follow the old rules and settings until they unbind from the DLP server. When the last user unbinds from a configuration, the entire configuration is deleted

The OID for **ldap_proxy_update** is **1.3.12.2.1107.1.3.2.11.60**.

5.4.4.4. Example

The following example shows how to apply the **ldap_proxy_update** LDAP extended operation on the local LDAP server (which is running with the complete database and using the default ports on the local machine) with the **dirxextop** command:

```
dirxextop -D cn=admin,o=my-company -w dirx -t ldap_proxy_update
```

The LDAP extended operation returns output like the following:

```
================================================================================
========
+++LDAP-Proxy-Configuration Update Started at:Thu Oct 11
13:54:05.016999
+++Proxy-Name:ldapconfiguration
Proxy-Update suceeded. Old Proxy-Mode: 1, New Proxy-Mode: 1
Number of existing Proxy-Configuration Objects: 2

  Proxy-Object-ID: 9, Subscribers: 0  (active)
    CreateTime          : Thu Oct 11 13:54:05.016999
    NumAllServers       : 4
    NumLBServers        : 2  (sel-count:0)
    NumUserRoutingRules : 1  (sel-count:0)
    NumOprRoutingRules  : 2  (sel-count:0)
    NumRewriteRules     : 3 (2 Req, 1 Res)

  Proxy-Object-ID: 8, Subscribers: 1
    CreateTime          : Thu Oct 11 09:24:32.085999
    LastSubscriber      : LDAP_Con33 (Thu Oct 11 13:50:19.832999)
    NumAllServers       : 4
    NumLBServers        : 2  (sel-count:2)
    NumUserRoutingRules : 1  (sel-count:0)
    NumOprRoutingRules  : 2  (sel-count:0)
    NumRewriteRules     : 3 (2 Req, 1 Res)
--------------------------------------------------------------------------------
--------
+++LDAP-Proxy Update finished at:Thu Oct 11 13:54:05.296999
================================================================================
========
```

This example shows that the DLP server configuration was updated for the eighth time since server startup (**Proxy-Object-ID: 9**) It also shows that one user is still subscribed to the previous configuration settings (**Proxy-Object-ID: 8**) while 0 users are subscribed to the new settings.

The reason for these values is that the update was performed via **dirxextop**, which performs a **bind** and then the extended operation; as a result, at the time of the configuration update, the user of the **dirxextop** tool was subscribed to the settings established by the Proxy-Object-ID: 8. Instance. The update created Proxy-Object-ID: 9 but no new **bind** operation has since been initiated.

For each DLP server configuration instance, the number of servers and rules are listed along with their total selection count per rule group.

If the update fails, the existing configuration remains active.

5.4.4.5. See Also

(Configuration chapter) DLP Server Configuration File

ldap_proxy_status

6. Examples and Considerations

Generally, there are two types of proxy rules:

- Rules that define target servers
- Rules that can change requests/results

This chapter provides some examples to illustrate the use of both types of rule with a special focus on operation-routing rules and the rewriting rules.

As described in the chapter “Proxy Rules”, all rules have a **condition** key. Rules that define target servers specify target servers while rules that define request/result rewriting rules have **action** keys.

6.1. Operation-Routing Rules: Examples

The following examples illustrate the impact of different settings of the **keepconn** and **failover** keys in operation-routing rules. The examples include failover and non-failover scenarios.

6.1.1. Example 1: All Target LDAP Servers Up and Running

In this example, the following rules exist:

- User-routing rule: Forward **User_A** to **LDAP_Server_1**.
- Operation-routing rule: Forward all **add** operations to **LDAP_Server_2**.

No operation routing rules are defined for **search** operations.

The following tables illustrate where the operations are routed to based on the **keepconn** key.

keepconn=1

Step #	Operation executed	Target server selected for operation
Step 1	**User_A** binds.	**LDAP_Server_1**.
Step 2	**User_A** performs an **add** operation.	**LDAP_Server_2**.
Step 3	**User_A** performs a **search** operation	**LDAP_Server_2** due to **keepconn=1** from Step 2.

keepconn=0

Step #	Operation executed	Target server selected for operation
Step 1	**User_A** binds.	**LDAP_Server_1**.

Step #	Operation executed	Target server selected for operation
Step 2	**User_A** performs an **add** operation.	**LDAP_Server_2**.
Step 3	**User_A** performs a **search** operation	**LDAP_Server_1** due to **keepconn=0** from Step 2.

These tables show that **keepconn=1** overrules the target server from a previous user-routing rule for all subsequent operations that do not have their own rule. A **keepconn=1** can also be seen as a "switch-connection" operation. (Of course the switch is only applied if the operation was successful).

6.1.2. Example 2: Target Server Failure, no Failover Servers Defined

In this example, the following rules exist:

- User-routing rule: Forward **User_A** to **LDAP_Server_1**.
- Operation-routing rule: Forward **add** operations to **LDAP_Server_2**.

No operation-routing rules are defined for **search** operations.

The following tables illustrate where the operations are routed to based on the **keepconn** settings in the operation-routing rule:

Note that a server failure means that the server cannot be reached via the network for the rest of the test sequence; that is, it is not available again until end of the sequence.

Sequence 1: Keepconn=1 or keepconn=0

Step #	Operation executed	Target server selected for operation
Step 1	**User_A** binds.	**LDAP_Server_1**.
!	***LDAP_Server_2 fails.***	
Step 2	**User_A** performs an **add** operation.	Operation fails, due to no failover servers defined.
Step 3	**User_A** performs a **search** operation	**LDAP_Server_1** due to no connection available from Step 2.

Sequence 2: keepconn=1

Step #	Operation executed	Target server selected for operation
Step 1	**User_A** binds.	**LDAP_Server_1**.
Step 2	User_A performs an **add** operation.	**LDAP_Server_2**.
!	***LDAP_Server_2 fails.***	
Step 3	**User_A** performs a **search** operation	Operation fails due to **LDAP_Server_2** failure after Step 2 and **keepconn=1** in Step 2.

Sequence 2: keepconn=0

Step #	Operation executed	Target server selected for operation
Step 1	**User_A** binds.	**LDAP_Server_1**.
Step 2	**User_A** performs an **add** operation.	**LDAP_Server_2**.
!	***LDAP_Server_2 fails.***	
Step 3	**User_A** performs a **search** operation	**LDAP_Server_1** due to **keepconn=0** for Step 2.

As the example shows, the reaction towards the client depends on when the target server drops out, although the client executes the same sequence. As the target servers are transparent to the client (the client is unaware of being physically connected to a DLP server and not to an LDAP server) this might look strange, because the client never “lost” the LDAP-connection but might still receive different results.

6.1.3. Example 3: Target Server Failure, Failover=1, Multiple Targets

In this example, the following rules exist:

- User-routing rule: Forward **User_A** to **LDAP_Server_1**.
- Operation-routing rule: Forward **add** operations to **LDAP_Server_2** or **LDAP_Server_3**.

No operation-routing rules are defined for **search** operations.

The following tables illustrate where the operations are routed to based on various **keepconn** settings and **failover=1** in the operation-routing rule:

Sequence 1: Keepconn=1

Step #	Operation executed	Target server selected for operation
Step 1	**User_A** binds.	**LDAP_Server_1**.
!	***LDAP_Server_2 fails.***	
Step 2	**User_A** performs an **add** operation.	**LDAP_Server_3** due to **failover=1**.
Step 3	**User_A** performs a **search** operation	**LDAP_Server_3** due to **keepconn=1** from Step 2.

Sequence 1: Keepconn=0

Step #	Operation executed	Target server selected for operation
Step 1	**User_A** binds.	**LDAP_Server_1**.
!	***LDAP_Server_2 fails.***	
Step 2	**User_A** performs an **add** operation.	**LDAP_Server_3** due to **failover=1**.

Step #	Operation executed	Target server selected for operation
Step 3	**User_A** performs a **search** operation.	**LDAP_Server_1** due to **keepconn=0** from Step 2.

Sequence 2: Keepconn=1

Step #	Operation executed	Target server selected for operation
Step 1	**User_A** binds.	**LDAP_Server_1**.
Step 2	**User_A** performs an **add** operation.	**LDAP_Server_2**.
!	***LDAP_Server_2 fails.***	
Step 3	**User_A** performs a **search** operation.	**LDAP_Server_3** due to **keepconn=1** from Step 2 and **failover=1** for **LDAP_Server_2** down.

Sequence 2: Keepconn=0

Step #	Operation executed	Target server selected for operation
Step 1	**User_A** binds.	**LDAP_Server_1**.
Step 2	**User_A** performs an **add** operation.	**LDAP_Server_2**.
!	***LDAP_Server_2 fails.***	
Step 3	**User_A** performs a **search** operation	**LDAP_Server_1** due to **keepconn=0** from Step 2.

6.2. Rewriting Rules: Examples and Considerations

This section supplies some examples of rewriting rules and describes considerations to keep in mind when using rewriting rules.

6.2.1. Examples of Rewriting Conditions and Actions

Here are some condition/action pair examples.In these examples, **C:** refers to a condition and **A:** refers to an action.

6.2.1.1. Example 1: Enforce SSL/TLS

Enforce all users to connect via SSL/TLS only by denying any operation that comes from plain/unsecure socket:

C: (&(opr.req.type=*)(security=plain))

A: denyreq

6.2.1.2. Example 2: Deny Requests from Local Host

Do not allow any requests from the local host (this rule does not apply to extended operations)

C: (&(opr.req.type=*)(ip=127.0.0.1))

A: denyreq

6.2.1.3. Example 3: Reject binds from a User

Reject binds from a specific user:

C: (&(opr.req.type=bind)(bind.req.name=cn=hohner,ou=sales,o=my-company))

A: denyreq

6.2.1.4. Example 4: Reject Anonymous Users

Reject all operations from anonymous users:

C: (&(opr.req.type=*)(user=anonymous))

A: denyreq

6.2.1.5. Example 5: Replace a Base Object String in a Request

Replace the substring “o=pqr” with “o=my-company” in the base object of incoming search requests:

C: (opr.req.type=search)

A: search.req.baseObject.replace(o=pqr,o=my-company,NULL)

6.2.1.6. Example 6: Remove a Base Object String

Remove the string “ou=sales” from the incoming BaseObject

C: (opr.req.type=search)

A: search.req.baseObject.replace(ou=sales,NULL,NULL)

6.2.1.7. Example 7: Add/Remove Requested Attributes (Two Actions)

Add the requested attribute **mycn** and remove an existing requested attribute **cn** in a search request:

C: (opr.req.type=search)

A: search.req.attributes.add(mycn,NULL,NULL) ← first action

A: search.req.attributes.del(cn,NULL,NULL) ← second action

6.2.1.8. Example 8: Add/Remove Requested Attributes (One Action)

Add the requested attribute **mycn** and remove an existing requested attribute **cn** in a search request with one action:

C: (opr.req.type=search)

A: search.req.attributes.replace(cn,mycn,NULL)

6.2.1.9. Example 9: Change a Filter Attribute Name

Change the attribute name **cn** to **mycn** in all filter attribute-names:

C: (opr.req.type=search)

A: search.req.filter.replace(cn,mycn,NULL)

6.2.1.10. Example 10: Change a String in One Attribute Filter Value

Change the substring **pqr** to **my-company** in all filter values for the attribute **organization**:

C: (opr.req.type=search)

A: search.req.filter.replace(o,pqr,my-company)

6.2.1.11. Example 11: Change a String in All Attribute Filter Values

Change the substring **pqr** to **my-company** for all attributes in the filter:

C: (opr.req.type=search)

A: search.req.filter.replace(,pqr,my-company)*

6.2.1.12. Example 12: Deny Anonymous User Subtree Searches

Deny subtree searches for anonymous users:

C: (&(opr.req.type=search)(search.req.scope=wholeSubtree)(user=anonymous))

A: denyreq

6.2.1.13. Example 13: Deny Unlimited Searches

Deny unlimited (**sizelimit=0**) searches:

C: (&(opr.req.type=search)(search.req.sizelimit=0))

A: denyreq

6.2.1.14. Example 14: Set a Size Limit for Anonymous User Searches

Set a size-limit of **1000** for any search from anonymous users:

C: (&(opr.req.type=search)(user=anonymous))

A: search.req.sizeLimit.set(1000,NULL,NULL);

6.2.1.15. Example 15: Change a String in all Entry DNs of a Search Result

Change the string **o=my-company** to **o=pqr** in all resulting entry DNs of a search request:

C: (opr.req.type=search)

A: search.res.objectName.replace(o=my-company,o=pqr,NULL)

6.2.1.16. Example 16: Hide an Attribute from Returned Search Result Entries

Hide the attribute **secret** (with all its values) from any returned search result entry:

C: (opr.req.type=search)

A: search.res.attributes.del(secret,NULL,NULL)

6.2.1.17. Example 17: Remove an Attribute from a Request List

Remove the attribute **secret** from the request list:

C: (opr.req.type=search)

A: search.req.attributes.del(secret,NULL,NULL)

6.2.1.18. Example 18: Change an Attribute Value in Returned Search Results

Change the string **MainStreet** to **Hauptstrasse** in all values of the attribute **street** in any returned search result entry:

C: (opr.req.type=search)

A: search.res.attributes.replace(street,MainStreet,Hauptstrasse)

6.2.1.19. Example 19: Escape a Special Character

Replace a substring that contains a comma using the LDAP escape character \ (backslash). The original output is **description=The,dogs bark loud**. The desired output is **description=The wolves bark loud**.

the double escape is necessary because '\' (backslash) is defined as a meta-character in JSON.

C: (opr.req.type=search)

A: search.res.attributes.replace(description,The\\,dogs,The wolves)

6.2.1.20. Example 20: Deny Renaming or Moving Entries

Deny renaming or moving entries in the DIT:

C: (opr.req.type=modDN)

A: denyreq

6.2.1.21. Example 21: Prevent Attribute Value Creation for New Entries

Prohibit creation of any values for the attribute **strange** for newly created entries:

C: (opr.req.type=add)

A: add.req.attributes.del(strange,NULL,NULL)

6.2.1.22. Example 22: Prevent an Attribute Modification

Never modify the attribute **strange**:

C: (opr.req.type=modify)

A: modify.req.changes_add.del(strange,NULL,NULL)

A: modify.req.changes_del.del(strange,NULL,NULL)

A: modify.req.changes_replace.del(strange,NULL,NULL)

6.2.2. Considerations for Rewriting Rules

When search results are changed/re-written, it is usually necessary to add additional rules in order to maintain a consistent approach to the database in subsequent calls.

To illustrate this concept, let's look at the following simple example.Suppose there is a group of legacy clients that are accessing a DIT tree via LDAP calls.The DIT tree consists of a simple structure with a context prefix (CP) of **o=companyX**.The company decides to rename itself from **companyX** to **companyZ** and all the entries in the DIT need to reflect this change to the calling clients.

One way of achieving the result is to rename all entries in the DIT by changing the entry **o=companyX** to **o=companyZ**.This solution also implies that all clients must use **o=companyZ** as new target addresses; for example, for **baseObjects** in **search**es or in entry DNs for **add**s or **modify**s.But what if some of the legacy clients cannot be changed so easily?

A better approach is to use the DirX Directory LDAP proxy server with a simple rewriting rule like this:

C: (opr.req.type=search)

A: search.req.baseObject.replace(companyX,companyZ,NULL)

This rule changes any occurrence of **companyX** on-the-fly into **companyZ** in the **baseObject** of any search, making the search successful again for clients that are only aware of **o=company**.

Well, but what is returned now? The returned entries from the LDAP server will contain the new name **o=companyZ** again. Clients may now struggle to receive entries containing **o=companyZ** in their DNs although they have given a **baseObject** of **o=companyX** and therefore expect such resulting DNs.

Again, the DLP server can help by adding another simple result-rewriting action to another rule like:

C: (opr.req.type=search)

A: search.res.objectName.replace(companyZ,companyX,NULL)

Note that it must be another rule because the rule will now operate on *results* and so the **ruleType** must be **ResRewrite**. The baseObject change was performed by a **ReqRewrite** rule type.

By adding this result-rewriting rule, a client now receives the resulting entries as usual with the well-known old name **companyX**.

What if the client now decides to create a new group-of-names entry containing all received DNs as members from the previous search result? The client will surely issue an **add** request. As the client is only aware of **o=companyX**, the target DN of the **add** operation may again contain **companyX** and the member attributes will also contain the **companyX** name.

Clearly this **add** operation is bound to fail, as the LDAP servers no longer recognize **companyX**. Thus, it will be necessary to define further rules like

C: (opr.req.type=add)

A: add.req.entry.replace(companyX,companyZ,NULL)

A: add.req.attributes.replace(member,companyX,companyZ)

where the first action corrects the DN and the second action changes the members to the new name.

If you plan to deploy result rewriting, you need to have a full understanding of client behavior. You may need to add many more additional rules to manage the changes in subsequent **add**, **modify**, **delete**, **moddn** or **compare** operations properly.

DirX Product Suite

The DirX product suite provides the basis for fully integrated identity and access management; it includes the following products, which can be ordered separately.

DirX Identity

DirX Identity provides a comprehensive, process-driven, customizable, cloud-enabled, scalable, and highly available identity management solution for businesses and organizations. It provides overarching, risk-based identity and access governance functionality seamlessly integrated with automated provisioning. Functionality includes lifecycle management for users and roles, cross-platform and rule-based real-time provisioning, web-based self-service functions for users, delegated administration, request workflows, access certification, password management, metadirectory as well as auditing and reporting functionality.

DirX Directory

DirX Directory provides a standards-compliant, high-performance, highly available, highly reliable, highly scalable, and secure LDAP and X.500 Directory Server and LDAP Proxy with very high linear scalability. DirX Directory can serve as an identity store for employees, customers, partners, subscribers, and other IoT entities. It can also serve as a provisioning, access management and metadirectory repository, to provide a single point of access to the information within disparate and heterogeneous directories available in an enterprise network or cloud environment for user management and provisioning.

DirX Access

DirX Access is a comprehensive, cloud-ready, scalable, and highly available access management solution providing policy- and risk-based authentication, authorization based on XACML and federation for Web applications and services. DirX Access delivers single sign-on, versatile authentication including FIDO, identity federation based on SAML, OAuth and OpenID Connect, just-in-time provisioning, entitlement management and policy enforcement for applications and services in the cloud or on-premises.

DirX Audit

DirX Audit provides auditors, security compliance officers and audit administrators with analytical insight and transparency for identity and access. Based on historical identity data and recorded events from the identity and access management processes, DirX Audit allows answering the “what, when, where, who and why” questions of user access and entitlements. DirX Audit features historical views and reports on identity data, a graphical dashboard with drill-down into individual events, an analysis view for filtering, evaluating, correlating, and reviewing of identity-related events and job management for report generation.

For more information: support.dirx.solutions/about

Legal remarks

On the account of certain regional limitations of sales rights and service availability, we cannot guarantee that all products included in this document are available through the Eviden sales organization worldwide. Availability and packaging may vary by country and is subject to change without prior notice. Some/All of the features and products described herein may not be available locally. The information in this document contains general technical descriptions of specifications and options as well as standard and optional features which do not always have to be present in individual cases. Eviden reserves the right to modify the design, packaging, specifications and options described herein without prior notice. Please contact your local Eviden sales representative for the most current information. Note: Any technical data contained in this document may vary within defined tolerances. Original images always lose a certain amount of detail when reproduced.