

VAUADM-19.0

M-Vault Administration Guide

Isode

Table of Contents

Chapter 1	Overview.....	1
	The M-Vault Server has been designed and implemented to provide a foundation for a manageable, scalable and high-performance LDAP or X.500 (2008) Directory Service. The Directory can be accessed using a Directory User Agent (DUA), using either the Directory Access Protocol (DAP) or the Lightweight Directory Access Protocol (LDAP).	
Chapter 2	Setting up the Directory Service.....	8
	This chapter defines the tasks involved in setting up a Directory and the service to support it. It explains the information you need to correctly create a Directory Server.	
Chapter 3	Managing the Data.....	18
	This chapter describes how to add, modify and delete data using Sodium, either as individual entries or in bulk. This chapter should be read by the Data Manager, although the Server Manager will find some of the background information useful.	
Chapter 4	System Management.....	58
	This chapter explains how to use M-Vault Console to check and change the configuration of a Directory Service. It also covers standard operational tasks and performance tuning.	
Chapter 5	Authentication.....	72
	This aim of this chapter is to explain the authentication mechanisms that can be used with M-Vault Server.	
Chapter 6	Controlling Access.....	97
	This aim of this chapter is to explain how to define access to objects in the Directory. The M-Vault Server includes a number of security features to control access to and modification of Directory information.	
Chapter 7	Connecting Directories.....	112
	You may need more than one Directory Server to provide the Directory Service. This chapter cover creating additional servers, creating knowledge references (superior, subordinate and cross references) and specifying the authentication levels for connection.	
Chapter 8	Shadowing.....	121
	Making sure that Directory information is close to those who need it minimises access times. This is achieved by shadowing – or replicating – information so that more than one Directory Server holds a copy of the same information.	
Chapter 9	High Availability.....	134
	M-Vault provides three means of adding service resilience to system failure:	
	<ul style="list-style-type: none"> • Failover. Hot standby mode where a single master is active amongst a group of mirror servers. • Multimaster. All servers in a replication group accept changes and replicate them to all other members of the group. • Clustering. Hot standby mode where shared disks are used to support multiple nodes. 	

Chapter 10	HTTP And OCSP Services.....	144
	M-Vault incorporates a Web server that can be used to serve the following:	
	<ul style="list-style-type: none"> • PKI information - HTTP serving of PKI and CA related directory entry attributes (e.g. certificateRevocationList). • Web applications - currently a Web application (and underlying API) providing an account password modification user interface. • OCSP service - provision of OCSP (Online Certificate Status Protocol) services on the basis of stored CRLs. 	
	This chapter describes how to configure these services using M-Vault Console and Sodium.	
Chapter 11	Monitoring the Directory.....	150
	The Directory Service can be monitored in several ways. Logs can be inspected, and certain status information and statistics are kept by the system which can be displayed.	
Chapter 12	Synchronising Directories (using Sodium Sync).....	162
	This chapter explains how to use Sodium Sync for synchronizing data between directories, LDIF files, CSV files and SQL databases.	
Chapter 13	Managing Certificate Authorities (using Sodium CA).....	197
	This chapter describes the Sodium CA application, and explains how to use it to help configure and manage a PKI (Public Key Infrastructure) for Isode products.	
Chapter 14	OAuth2 Capabilities.....	214
	<p>OAuth2 is a Web based framework that provides secure authorization services for Web based applications. The system is based on a set of HTTP and token exchanges between the Web service, Web browser and OAuth2 server. M-Vault includes an OAuth2 server component that provides authentication and authorization services to Isode Web-based applications, a current example being Red/Black.</p> <p>Part of the OAuth2 configuration is intended to be managed by the Cobalt user and account provisioning application and the Cobalt manual should be consulted for this purpose. This chapter provides a lower level and more detailed view of the M-Vault OAuth2 service configuration, including instructions for how to view and manage the configuration using Sodium.</p>	
Chapter 15	SPIF Editor.....	219
	This chapter describes the SPIF Editor application and explains how to use it to create, edit and view a SPIF (Security Policy Information File) and various utility functions.	
Appendix A	Introduction to Directories.....	239
	This appendix provides a background to X.500 and LDAP Directories. It provides a context for Isode's implementation of the Directory, and should be read by anyone not familiar with the X.500 standard or who wants to understand how Isode has implemented it.	
Appendix B	Attributes.....	253
	This appendix provides details of the attributes associated with some of the common object classes, indicating which are mandatory and which are optional.	
Appendix C	Attribute Syntaxes.....	260
	Attribute values have an internal structure described by their syntax. When communicated over LDAP, or displayed in user agents like Sodium and M-Vault Console, the string representations associated with those syntaxes are used. This appendix describes all the currently recognized syntaxes and their LDAP string representations.	

Appendix D	Customising Sodium.....	276
	Sodium’s built-in templates can be modified to suit local needs.	
Appendix E	Advanced Configuration.....	288
	Once a Directory Server has been set up, you may wish to configure various attributes using a command line scripting interface, such as Teldish. This section describes the various attributes and entries which can be configured.	
Appendix F	Running as an OS Service.....	310
	This section describes how to configure and run Operating System services.	
Appendix G	Teldish – the Teldish and Lteldish DUAs.....	316
	This chapter describes the DAP and LDAP Directory management DUAs called Teldish and Lteldish.	
Appendix H	Dmish Scripting Interface.....	348
	This chapter describes the scripting interface Dmish . The Directory Management Shell (Dmish) is an extended Tcl (Tool Command Language) shell for use by Directory administrators and systems integrators.	
Appendix I	References.....	375
	Glossary.....	380

Isode and Isode are trade and service marks of Isode Limited.

All products and services mentioned in this document are identified by the trademarks or service marks of their respective companies or organizations, and Isode Limited disclaims any responsibility for specifying which marks are owned by which companies or organizations.

Isode software is © copyright Isode Limited 2002-2023, all rights reserved.

Isode software is a compilation of software of which Isode Limited is either the copyright holder or licensee.

Acquisition and use of this software and related materials for any purpose requires a written licence agreement from Isode Limited, or a written licence from an organization licensed by Isode Limited to grant such a licence.

This manual is © copyright Isode Limited 2023.

1 Software version

This guide is published in support of Isode M-Vault R19.0. It may also be pertinent to later releases. Please consult the release notes for further details.

2 Readership

This guide is intended for administrators who plan to configure and manage Directory Services using the M-Vault Server.

3 How to use this guide

You are advised to read through [Chapter 1, Overview](#), before you start to set up your Directory Service. If you are not familiar with X.500 Directories, you should also read [Appendix A, Introduction to Directories](#).

4 Typographical conventions

The text of this manual uses different typefaces to identify different types of objects, such as file names and input to the system. The typeface conventions are shown in the table below.

Object	Example
File and directory names	<i>isoentities</i>
Program and macro names	mkpasswd
Input to the system	cd newdir
Cross references	see Section 5, “File system place holders”
Additional information to note, or a warning that the system could be damaged by certain actions.	Notes are additional information; cautions are warnings.

5 File system place holders

Where directory names are given in the text, they are often place holders for the names of actual directories where particular files are stored. The actual directory names used depend on how the software is built and installed. All of these directories can be changed by configuration.

Certain configuration files are searched for first in (*ETCDIR*) and then (*SHAREDIR*), so local copies can override shared information.

The actual directories vary, depending on whether the platform is Windows or UNIX.

Name	Place holder for the directory used to store...	Windows (default)	UNIX
(<i>ETCDIR</i>)	System-specific configuration files.	<i>C:\Isode\etc</i>	<i>/etc/isode</i>
(<i>SHAREDIR</i>)	Configuration files that may be shared between systems.	<i>C:\Program Files\Isode\share</i>	<i>/opt/isode/share</i>
(<i>BINDIR</i>)	Programs run by users.	<i>C:\Program Files\Isode\bin</i>	<i>/opt/isode/bin</i>
(<i>SBINDIR</i>)	Programs run by the system administrators.	<i>C:\Program Files\Isode\bin</i>	<i>/opt/isode/sbin</i>
(<i>EXECDIR</i>)	Programs run by other programs; for example, M-Switch channel programs.	<i>C:\Program Files\Isode\bin</i>	<i>/opt/isode/libexec</i>
(<i>LIBDIR</i>)	Libraries.	<i>C:\Program Files\Isode\bin</i>	<i>/opt/isode/lib</i>
(<i>DATADIR</i>)	Storing local data.	<i>C:\Isode</i>	<i>/var/isode</i>
(<i>LOGDIR</i>)	Log files.	<i>C:\Isode\log</i>	<i>/var/isode/log</i>
(<i>CONFPDUSPOOLDIR</i>)	Large PDUs on disk.	<i>C:\Isode\tmp</i>	<i>/var/isode/tmp</i>
(<i>QUEDIR</i>)	The M-Switch queue.	<i>C:\Isode\switch</i>	<i>/var/isode/switch</i>
(<i>DSADIR</i>)	The Directory Server's configuration.	<i>C:\Isode\d3-db</i>	<i>/var/isode/d3-db</i>

6 Support queries and bug reporting

A number of email addresses are available for contacting Isode. Please use the address relevant to the content of your message.

- For all account-related inquiries and issues: customer-service@isode.com. If customers are unsure of which list to use then they should send to this list. The list is monitored daily, and all messages will be responded to.
- For all licensing related issues: license@isode.com.
- For all technical inquiries and problem reports, including documentation issues from customers with support contracts: support@isode.com. Customers should include relevant contact details in initial calls to speed processing. Messages which are continuations of an existing call should include the call ID in the subject line. Customers without support contracts should not use this address.

- For all sales inquiries and similar communication: sales@isode.com.

Bug reports on software releases are welcomed. These may be sent by any means, but electronic mail to the support address listed above is preferred. Please send proposed fixes with the reports if possible. Any reports will be acknowledged, but further action is not guaranteed. Any changes resulting from bug reports may be included in future releases.

Isode sends release announcements and other information to the Isode News email list, which can be subscribed to from the address: <http://www.isode.com/company/contact.php>

7 Export controls

Many Isode products use TLS (Transport Layer Security) to encrypt data in transit. This means that these products are subject to UK Export Controls.

For some countries (at the time of shipping this release, these comprise all EU countries, United States of America, Canada, Australia, New Zealand, Switzerland, Norway, Japan), these Export Controls can be handled by administrative process as part of evaluation or purchase. For other countries, a special Export License is required. This can be applied for only in context of a purchase order for those Isode products.

You must ensure that you comply with these Export Controls where applicable, i.e. if you are licensing or re-selling Isode products.

The TLS feature of Isode products is enabled by a TLS Product Activation feature. This feature may be turned off, and Isode products without this TLS feature are not export controlled. This can be helpful to support evaluation of Isode products in countries that need a special export license.

Isode products are used to administer sensitive data and so Isode strongly recommends that all operational deployments of Isode products use the export-controlled TLS feature.

All Isode Software is subject to a license agreement and your attention is also called to the export terms of your Isode license.

Chapter 1 Overview

The M-Vault Server has been designed and implemented to provide a foundation for a manageable, scalable and high-performance LDAP or X.500 (2008) Directory Service. The Directory can be accessed using a Directory User Agent (DUA), using either the Directory Access Protocol (DAP) or the Lightweight Directory Access Protocol (LDAP).

Note: If you need some background information about LDAP and X.500 directories in general, see [Appendix A, Introduction to Directories](#).

1.1 Roles

Directory Services are often administered by people with different functions or roles. Two roles are referred to throughout this manual:

- Server Managers

Someone with this role is responsible for managing the configuration and administrative tasks associated with the Directory Service (for example starting and stopping the server).

Someone in this role may not need access to confidential information stored in user entries.

- Data Managers

Someone in this role is responsible for the data content held within the Directory and is likely to have rights to access confidential information within that data.

Note: The roles may, in some organisations, be performed by the same people - but the responsibilities of each role are quite different.

1.2 Planning and preparing for the Directory

The areas to be considered when you are preparing to set up a Directory Service include:

- The logical structure of the contents of the Directory (and its associated schema)
- The initial naming context of your Directory Server (or of the first Directory Server, if you have more than one)
- Where the Directory fits in the overall hierarchy: are you only concerned with information within your own organization, or do you need a wider or deeper hierarchy, perhaps including country level
- How and whether the Directory is going to be distributed: is the information going to be shared across several Directories

This planning and preparation is carried out by the Server Manager, in consultation with the Data Manager.

Caution: If you are upgrading from an earlier version of M-Vault, you must read the release notes provided in case there are changes to the structure of the database between the versions. Failure to take appropriate action may result in loss of data.

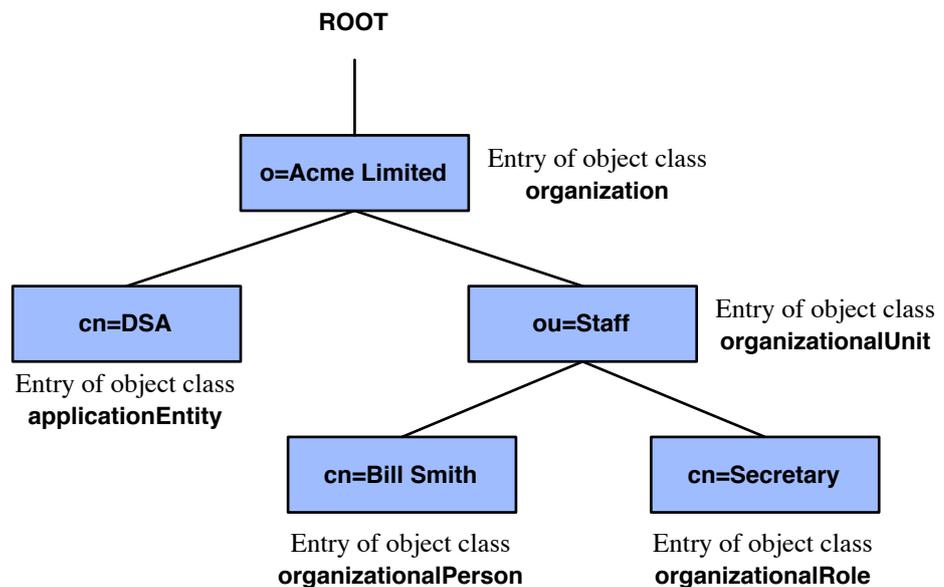
1.2.1 The logical structure of the Directory

The way the data is structured is dictated by its intended use. Some broad examples are:

- White Pages, which may be used to support address books or PKI.
- XMPP configuration (for M-Link).
- IMAP configuration (for M-Box).
- Messaging configuration (for M-Switch).
- Address book for AMHS.

A Directory is hierarchically structured, in the shape of a tree known as the Directory Information Tree (DIT). The width and depth of that tree are determined by your choices in structuring that hierarchy. For example if your Directory Server is not part of any bigger Directory infrastructure – perhaps it is just being used as a repository for user accounts for M-Link or M-Box – then you would typically create a very flat and wide structure.

Figure 1.1. Hypothetical local Directory Information Tree

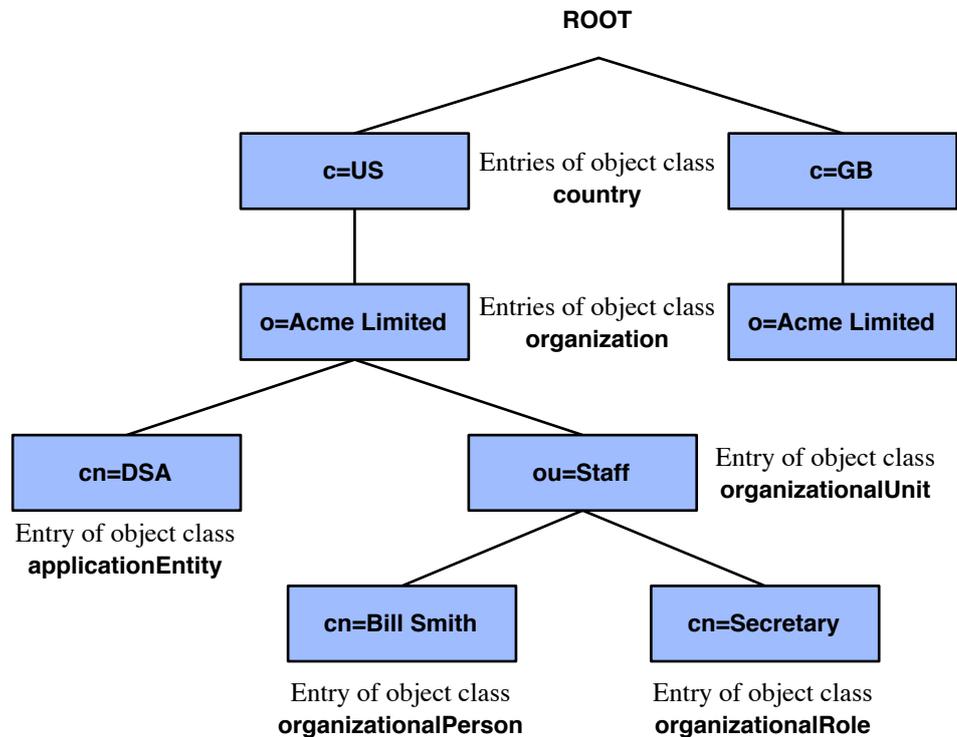


The Directory can be distributed over several Directory Servers, each one holding a subset of the information (see [Chapter 7, Connecting Directories](#)), and each server can use a different sub-structure. When you set up your Directory, you should consider the overall structure and where your Directory Server fits within it.

In a (hypothetical) world-wide Directory (as shown in [Figure 1.2, “Hypothetical world-wide Directory Information Tree”](#)), entries for countries, international administrative regions (such as Europe), and international organizations are listed just below the root of the tree. Below country entries are entries for localities (states and provinces) and organizations of national significance, such as government departments. Entries for other kinds of organizations are typically listed under the locality in which they are situated. At the leaves of the tree are entries for individuals, pieces of equipment (such as printers) and other simple objects (such as roles and applications).

You are allowed a considerable degree of flexibility with regard to the depth and contents of the DIT. Some levels may be omitted, while others, such as organizational unit, can also be used at one or several levels.

Figure 1.2. Hypothetical world-wide Directory Information Tree



1.2.1.1 What sort of structure do you need?

Many Directories are used for White Pages queries – searching for people in organizations using their names or other attributes. If this is the purpose of your Directory, keep the number of levels to a minimum. A flat tree means that Directory names (see [Glossary](#)) will be relatively short: a deep DIT leads to long Distinguished Names. (See [Section A.2.4, “Naming objects”](#), for an explanation of Distinguished Names.)

Two things determine the depth of the DIT:

- How far down the overall (international) DIT an organization is placed. Your organization will be placed in the locality in which it is situated in reality; for example, a country.
- The structure of the DIT within that organization.

Do not sub-divide your organization more than is necessary: two levels of organizational unit should be enough even for large organizations, and organizations with only a few tens or hundreds of employees should consider not using organizational units at all.

As part of creating the Directory Service, you may have specified the initial naming context (the top of your portion of the DIT) which the Directory Server will master. If you did not do this then, you will have to specify the initial naming context before you can start to place data in the Directory.

1.2.2 The initial naming context

The Distinguished Names of the object entries in the Directory will be determined not only by the schema, but also by the initial naming context selected for the Directory Server (see [the section called “The naming context”](#) for a definition of a naming context). When you set up a Directory Server you specify the Distinguished Name of the entry at the top of its initial naming context.

The Directory Server will master all entries from that point down the DIT hierarchy. All object entries to be mastered by this Directory Server are held within this initial context.

General guidelines on naming user information objects are given in [Section A.2.4, “Naming objects”](#).

1.2.3 Administration of the Directory

Administration of the Directory is mainly concerned with access control and the control of collective (shared) attributes.

After the Directory Server has been created, you will be able to connect to it using Sodium (see [Section 3.3.1, “Browsing the Directory”](#)) to create roles, define access points and control, and populate the Directory with data.

General guidelines on how to manage security in the Directory are given in [Chapter 5, Authentication](#) and [Chapter 6, Controlling Access](#).

1.2.4 Distribution of the Directory

The Directory can be distributed widely over many computer systems.

Note: If a request cannot be answered by a Directory Server, it may be passed to another using available references, with any answer being passed back along the same chain. This is the way the Directory Server operates when it is first set up. You may prefer to change this, so that the request is sent back with a referral to another Directory server that the requester should contact directly. Instructions for creating references and switching between ‘chaining’ and ‘referral’ are given in [Chapter 7, Connecting Directories](#).

- If you are *sure* that the information in your Directory is going to be held and updated on a single Directory Server and you are not going to need to change this later, the issues of distribution and communication with other Directory Servers are largely irrelevant.
- If you think you *may* want to distribute your Directory or connect to other Directories in the future, you need to consider this when locating your Directory Server within the DIT.
- If you know that you need to distribute your Directory across several Directory Servers, possibly on different machines in different locations, you need to consider how to divide your information among them.

1.3 Directory management tools

The management tools provided by Isode can be divided into tools used for:

- Setting up and managing the Directory Service: maintaining the operational information in the Directory.
- Accessing the Directory Information Base: maintaining the user information in the Directory.
- Monitoring the Directory Service.

1.3.1 Managing the Directory Service

Two tools are available for managing the Directory Service:

- M-Vault Console
- Dmish, which is a scripting interface that can manage the same operational information and that is useful for batched or repetitive tasks.

1.3.1.1 M-Vault Console

M-Vault Console is a graphical user interface for managing the M-Vault Server, and can be used for the following tasks:

- Creating and managing new Directory Servers.
- Starting and stopping local Directory Servers.
- Maintaining a manager's list of Directory Servers.
- Setting up and maintaining controls for authentication.
- Managing naming contexts.
- Maintaining references to Directory databases (GDAMs).
- Adding a superior reference to another Directory Server.
- Managing shadowing agreements between Directory Servers.
- Managing attribute indexes.

An overview of the interface is given in [Section 4.4, "Overview of M-Vault Console"](#), and details on how to use the tool to perform various tasks are given where the task is described.

1.3.1.2 Dmish

The Dmish command line scripting interface enables you to carry out many of the same tasks as are possible using M-Vault Console. This is a Tcl (Tool Command Language) application and is particularly useful for carrying out repetitive or regularly-run tasks, as it can be used in batch mode and has all the features of the Tcl interpreter available for writing scripts.

Dmish can be used to:

- Create a Directory Server.
- Start a Directory Server.
- Stop a Directory Server.
- Maintain managed objects.

Details on how to use the interface are given in [Appendix H, *Dmish Scripting Interface*](#).

1.3.2 Maintaining information in the Directory

The information in the Directory can be maintained using a Directory User Agent (DUA). Isode provides two tools:

- Sodium (Secure Open Data, Identity and User Manager), which has a graphical user interface.
- Tcldish, which has a command line scripting interface.

1.3.2.1 Sodium (Secure Open Data, Identity and User Manager)

Sodium is designed to handle complex data management tasks across multiple Directory Servers, using a graphical user interface. It allows you to:

- Manage information on multiple Directory Servers across the network from a single, remote client, using DAP or LDAP to access the Directory.
- Connect to (and disconnect from) individual Directory Servers.

- Browse the DIT.
- Display the entries using different views.
- Search for entries using filters.
- Add and modify entries using templates.
- Check the integrity of any Distinguished Name references.
- Manage complex data in the Directory.
- Manage administrative points and access controls.
- Import and export multiple entries (bulkload and dump) using LDIF files.

An overview of the Sodium interface is given in [Section 3.3.1, “Browsing the Directory”](#).

1.3.2.2 Tcldish

The general term, Tcldish, is used to refer to the DAP and LDAP Directory management DUAs Tcldish and Ltclldish.

- Tcldish uses DAP to access the Directory
- Ltclldish uses LDAP to access the Directory.

Tcldish is not intended as a general DUA. Its purpose is to provide a command line interface to the Directory. The commands, based on Tcl (Tool Command Language) allow you to move around, view and modify parts of the DIT, write and execute scripts and manage DSAs.

Specifically, you can:

- Connect to (and disconnect from) a Directory Server.
- Display the attributes of entries.
- Compare the attributes of entries.
- Search for entries matching given criteria.
- Add, delete and modify entries.

You can also use Tcldish to bulk load, delete and backup sections of the DIT, using LDIF files. You can bulk load from CSV files, if you require.

Details on how to use the interface are given in [Appendix G, Tcldish – the Tcldish and Ltclldish DUAs](#).

1.3.3 Monitoring the Directory Service

Various log files are available for monitoring the Directory service. By default, these files are located in (*LOGDIR*) and have the names:

- *dsa-audit.timestamp.log*, the Directory Server Audit Log
- *dsa-event.timestamp.log*, the Directory Server Event log
- *dua-event.timestamp.log*, the Directory User Agent Event Log

The locations of these log files, as well as the types of events logged and the level of detail contained in log messages can all be tailored to suit the needs of a specific installation. For details, see [Chapter 11, Monitoring the Directory](#).

Facilities are provided within M-Vault Console for monitoring:

- the Directory Server and connections to it (successful and unsuccessful, and analysed by type of operation)
- shadowing agreement status

- attribute indexes.

More details on monitoring the Directory Service are given in [Chapter 11, *Monitoring the Directory*](#).

Chapter 2 Setting up the Directory Service

This chapter defines the tasks involved in setting up a Directory and the service to support it. It explains the information you need to correctly create a Directory Server.

Note: This chapter assumes you have already downloaded and installed M-Vault.

Setting up the initial Directory Service involves, in sequence:

1. Initial planning and preparation (see [Section 1.2, “Planning and preparing for the Directory”](#)).
2. Setting up the first Directory Server.
3. Starting this Directory Server and opening a management connection to it.
4. Checking the initial configuration.
5. Setting up the initial Directory Information Base.
6. Setting up administration and security.

Note: This chapter describes the process using M-Vault Console, which uses a creation wizard to guide you. If you need, or prefer, to carry out the same tasks using the scripting interface, see [Appendix H, *Dmish Scripting Interface*](#).

2.1 Information required for setup

Before you can set up the first Directory Server (or any Directory Server) you need to have the following information:

- Where the information held by the Directory Server fits in the overall hierarchy – see [Section 1.2.1, “The logical structure of the Directory”](#) for guidance.
- What information the Directory Server is to master – the information it is going to contain.
- The Access Point for the Directory Server, so that it can be contacted by Directory User Agents (DUAs) and, possibly, other Directory Servers. This means you need to know or determine the following:
 - The location of the Directory Server’s entry within the Directory Information Tree (DIT) – in other words, the Distinguished Name (DN) of the Directory Server.
 - The location of the Directory Server on the network – its Presentation Address.
- Where the configuration files for the new Directory Server are to be stored. You need the name of an empty filestore directory.
- A password for the Server Manager.

2.1.1 The Directory Server’s role in the administration

Directory Servers are providers of a Directory Service within an administration. An individual Directory Server provides user information for a particular type of administration. It is thus said to *represent* that administration.

When you install a Directory Server, you need to give the server a DN – its location within the DIT. For example:

- If your Directory is going to hold information for an organization that is either an international organization (crossing country boundaries) or it does not need to specify the country in which it is located, the DN of the DSA could be something similar to: **cn=DSA,o=International Herald Tribune**.
- If your Directory is a national organization and you want to locate it within a country, you can include the country in its DN. For example, **cn=DSA,o=Singapore Straights Times,c=SG**.
- If you want to define the relationship between your Directory's entries and those held in another Directory more precisely, you may choose to add a location between the country and the organization. For example: **cn=DSA,o=Oxford Times,l=Oxford,c=GB**.

The DN of the server must begin with the **cn=** portion, but what you add after that depends on your particular implementation.

2.1.2 The information the Directory Server is to master

When a Directory Server is responsible for a set of user information entries, the Directory Server is said to master that information. The Directory Server may be used to master a complete Directory Information Tree (DIT), or only part of it. The part of the Directory which is mastered by a single Directory Server is termed a naming context. The entries within that naming context cannot be updated by any other Directory Server (see [Section A.3.2.1, "Directory Servers"](#) for a fuller explanation).

The Distinguished Name (DN) of the top entry of the naming context in the DIT is termed the context prefix. This is used to identify where in the DIT the naming context starts. The Directory Server masters user information from that point.

A single Directory Server can master more than one naming context. The one specified when the Directory Server is first set up is called the initial naming context, but you can add further naming contexts later. See [Section 3.5, "Adding single entries to the Directory"](#).

Note: A Directory Server may be used to hold only replicated information mastered by another Directory Server, and may not be responsible for any information of its own. In this case, you will not have to specify the naming context. If the Directory Server is used to master information, you must specify a context prefix; the initial naming context cannot commence at the root, although it can be immediately subordinate to it.

2.1.3 Using bind profiles

To make a connection to a server, you typically need to provide at least some of the following information:

- server address (for example, hostname and port number)
- authentication type (for example, `anonymous` or `simple`)
- extra authentication information (for example, username and password)
- extra connection options (for example, whether to impose a time limit on searches).

In some cases it may be appropriate to specify this information at the point of making a once-only connection to a server. But more often it is useful to specify this information once, and then re-use the same set of information whenever you want to refer to the same type of connection to the same server.

Isode tools provide a mechanism to do this, by wrapping up the collection of parameters in a Bind Profile. Bind Profiles are stored in a file on disk so that they are available whenever you run an Isode application which uses them. When you create a Bind Profile, you assign

it a name, which makes it easier to recognise. For example, you might have Bind Profiles called “UK Server” or “Management connection to backup server”.

Note: A bind profile is created automatically for any new Directory Servers you create using M-Vault Console.

Because Bind Profiles have user-defined names, they are also used as a way of making display of information more convenient. For example, when M-Vault Console shows information about a shadowing agreement, it may show a Bind Profile name (for example, **uk server**) in addition to displaying the presentation address of the peer Directory Server. In this case, it is not necessary for the Bind Profile to contain any authentication information.

A Bind Profile may contain sensitive data (such as passwords, which saves you having to enter them every time you establish a new connection). When saved on disk, any such data are protected by being encrypted. The whole set of Bind Profiles will be encrypted and protected by a single Bind Profile Passphrase, which you must provide before being able to use them.

2.2 Creating a Directory Server

Setting up the first (or any) Directory Server involves the following steps:

1. Creating the Directory Server system account on the network (see [Section 2.2.1, “Creating the Directory Server system account”](#)).
2. Starting M-Vault Console.
3. Creating the Directory Server.

2.2.1 Creating the Directory Server system account

The first step in creating a Directory Server configuration is to establish a system account for it on the host computer. Ideally, each Directory Server on the system should be assigned a special, non-privileged user identifier. Where there is only one Directory Server on the system, the user identifier could be `dsa`, for example. The user with this identifier is regarded as the owner of the Directory Server database.

On UNIX systems, you could create this account by editing `/etc/passwd`, unless there is an administration command on your system to create new accounts.

On a Windows system, the User Manager administrative tool should be used to create the account. You need administrator privileges to create this account. You also need to assign administrator privileges to the new account.

The directory (`LOGDIR`) must be writeable by this account. Other directories of the Directory Server installation, including (`SHAREDIR`), (`ETCDIR`) and (`LIBDIR`) and their files, must be readable by the account.

The Directory Server is created using M-Vault Console.

The same process is followed whether you are creating a new or a subsequent Directory Servers.

2.2.2 Starting M-Vault Console

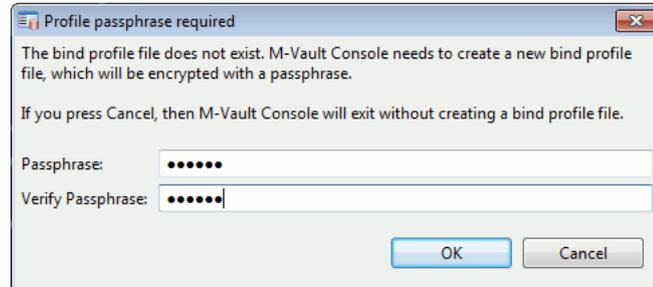
On Unix, run `/opt/isode/sbin/mvc`

On Windows systems, select **M-Vault Console** from within the **Isode** group on the **Start** menu.

2.2.2.1 Setting a passphrase

At start up, a window is displayed asking for a passphrase that will be used to encrypt the bind profile information for the currently logged-on user (see [Section 2.1.3, “Using bind profiles”](#)).

1. Type the passphrase you want to use to access the bind profile you are about to create.



You will be asked for this passphrase every time you start M-Vault Console.

Caution: This passphrase does not prevent another user from using tools to view or modify the Directory Server configuration files directly on disk.

2. Click **OK**. You will be prompted to create a DSA (Directory Server) or a profile for an existing DSA.
3. Click **OK**. The **M-Vault Console** window is displayed.

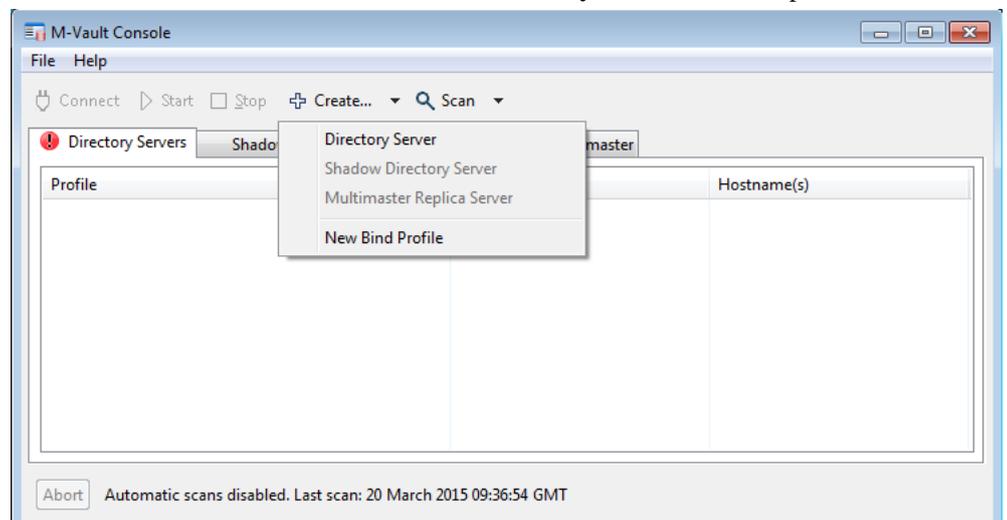
2.2.3 Creating a Directory Server

A wizard is used to create the Directory Server, prompting you for the required information:

- Click **Next** to store - and validate - your information and display the next page.
- Click **Back** to show the previous page so that you can check and amend the information, if necessary.
- Click **Cancel** to abandon the process.

Step-by-step instructions are provided for creating a Directory Server with the default configuration, with an overview of the variations available in subsequent sections.

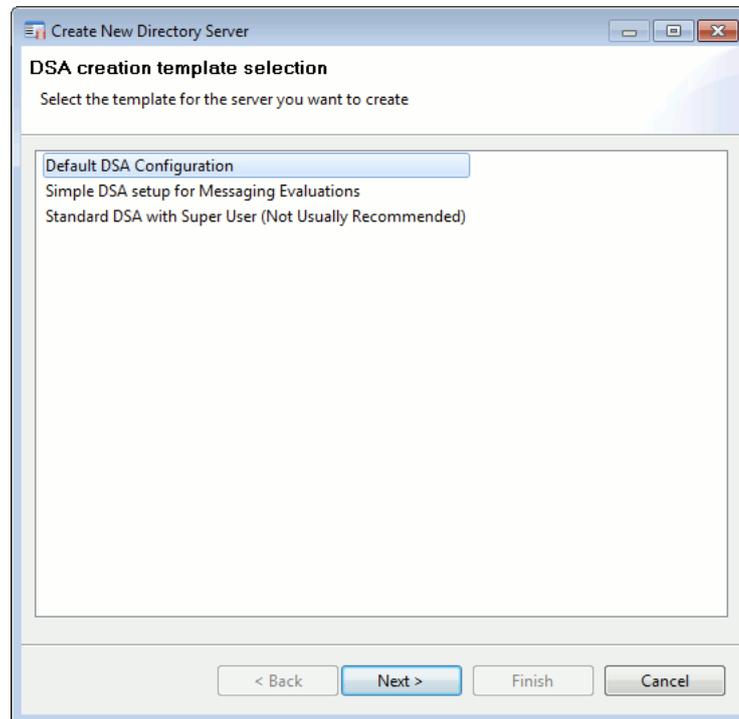
1. Click **Create** on the tool bar and select **New Directory Server** from the options shown.



2. The **DSA creation template selection** page opens.

Select the template that corresponds to the type of Directory Server you want to create:

- **Default DSA Configuration** creates a DSA containing a single user account, which you will specify. This account is a member of both the Server Manager and Data Manager groups, whose members have permissions relevant for their roles.
- **Simple DSA setup for Messaging Evaluations** creates a DSA with the structure necessary for an evaluation of a messaging solution - a simplified version of **DSA for Messaging Only**. See [Section 2.2.3.1, “Creating a simple DSA for messaging evaluations”](#) for an overview of this option.
- **Standard DSA with Super User** creates a DSA similar to the first (**Default DSA Configuration**) option, but with an additional user role outside of the Directory hierarchy. Select this option if you need to alter the structure of your Directory, removing the automatically created structure and replacing it with one of your own. See [Section 2.2.3.2, “Creating a DSA with super user account”](#) for an overview of this option.



Note: The remainder of these instructions assume you have selected **Default DSA Configuration**.

Click **Next**.

3. Set up the initial structure of your Directory, replacing any **xx** values with your own. If you change the **Base DN**, the change is carried down through the rest of the template.

Create New Directory Server

DIT structure configuration
Adjust the DNs provided by the template to suit your requirements

Base DN, for example: Country (c=gb), International Organization (o=isode), Organization (o=isode,c=gb), or Internet Domain (dc=isode,dc=com)

c=xx

DSA DN: This identifies the DSA when working with several DSAs. It must be unique within the group of DSAs that you plan to interact with using M-Vault Console. For a standalone DSA, you may simply call it cn=DSA. It is not necessary for there to be a real entry with this DN.

cn=DSA,c=xx

Users subtree location: This can be put either under the base DN (as below), or if you prefer, at the root (o=users)

o=users,c=xx

Groups subtree location: This can be put either under the base DN (as below), or if you prefer, at the root (o=groups)

o=groups,c=xx

Initial Directory User: This user is put into all the initial roles, and the bind profile created will bind as this user. Afterwards you can create more users and change which users are put in which roles. You should change the common name field to a suitable value for a real person.

cn=Thomas Atkins,o=users,c=xx ⓘ

< Back Next > Finish Cancel

- ❶ Change the name of this **Initial Directory User** to be a real person.

Click **Next**.

4. Depending on the template you have selected, you may be prompted with a page showing you a set of pre-defined access control rules that can be configured in the new Directory. Any rules you choose to configure will subsequently be visible when you use Sodium's **Global Access Control View**. See [Section 6.2, "Global access control"](#).

Click **Next**.

5. Select the groups that you want to be created. **CRL Writers' Group**, **Certificate Writers' Group** and **CA Managers' Group** all relate to the use of certificates to authenticate users. If you are not using certificates, you do not need these groups.

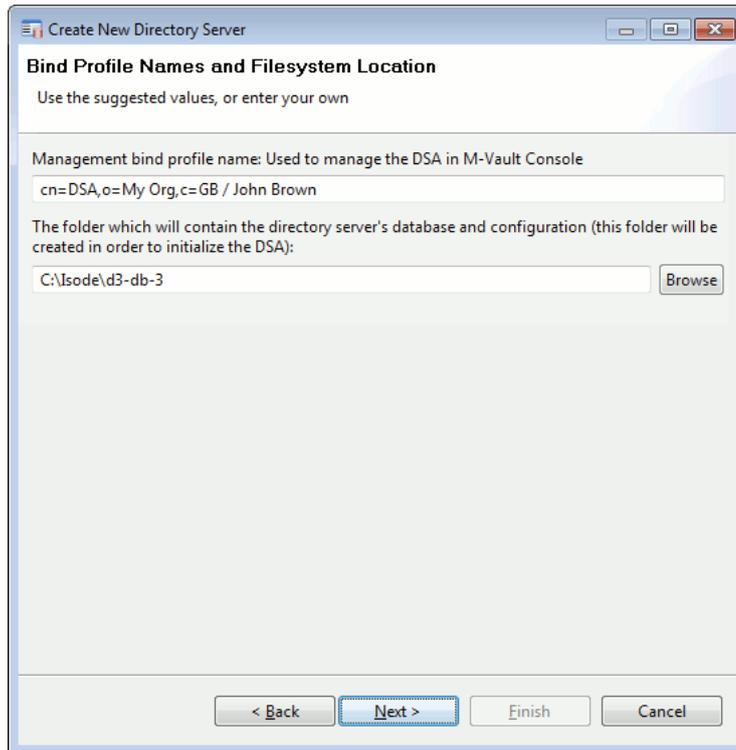
Click **Next**.

6. The **Passwords configuration** window is shown, with a system-generated password for the **Initial Directory User**. You can change this if you want to, but need to keep a copy of whatever it is, as you will need it to connect to the Directory.

Select **Show** to show the contents of the password field in plain text.

Click **Next**.

7. The **Bind Profile Names and Filesystem Location** window opens.

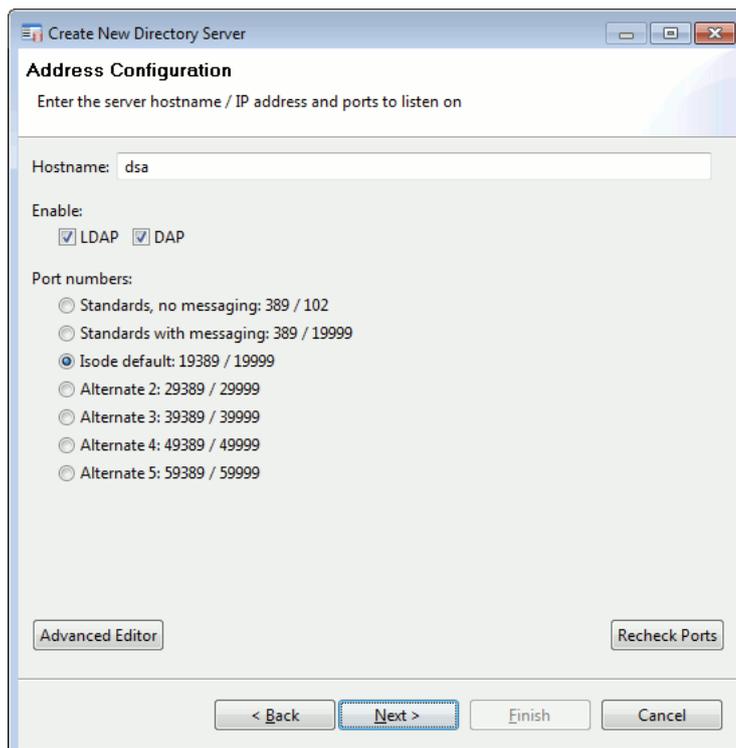


You can use any value for the bind profile name. The default is the DN of the Directory Server followed by the name of the person managing it.

Specify the location to store the Directory database (the **DSA folder**). The installation process creates the final directory in the path, and so requires write access to the parent folder. For example, if you specify a path of *J:\Isode\d3-db*, then *J:\Isode* must already exist and be writable: *d3-db* must *not* exist in that location and will be created.

8. The **Address Configuration** page opens.

You can specify basic connection information using this page, and enable both DAP and LDAP connections.



A selection of **Port numbers** are available: the **Alternate...** options will normally only be used if the **Standards-based** or **Isode default** port numbers are already in use.

Click **Full Presentation Address Editor** to specify more details or alternative ports: instructions are given in [Section 2.2.3.3, “Specifying a presentation address”](#).

Click **Next**.

9. The **Confirm Details** page is shown. Check that all the information it shows is correct. If anything needs to be changed, click **Back** to step back through the wizard.

Click **Finish**.

The wizard will then create and start the Directory Server. When the process is complete, you will be asked whether you want to connect to it now. If you click **Yes**, the M-Vault Console application opens, showing configuration information for the Directory Server on several tabbed pages. The interface is described in general terms in [Section 4.4, “Overview of M-Vault Console”](#).

2.2.3.1 Creating a simple DSA for messaging evaluations

The creation of a DSA for this purpose follows the basic wizard described above, but skips irrelevant pages.

The majority of the structure created with the option is predefined. You can specify your own **Base DN** and the **Initial Directory User** (highlighted below).

Create New Directory Server

DIT structure configuration
Adjust the DNs provided by the template to suit your requirements

Base DN
o=messaging

Initial Directory User: This user is put into all the initial roles, and the bind profile created will bind as this user. Afterwards you can create more users and change which users are put in which roles. You should change the common name field to a suitable value for a real person.
cn=Hermann Baumann,o=users,o=messaging

< Back Next > Finish Cancel

2.2.3.2 Creating a DSA with super user account

This option creates an account with a super user (**cn=DSA Manager,cn=config**) in addition to the **Initial Directory User**. The super user is able to create a complete Directory structure after the DSA has been created.

Two passwords are created, one for each user, and two profiles. This enables you to bind to the Directory either as the named person who is the **Initial Directory User** or as the DSA Manager.

2.2.3.3 Specifying a presentation address

A presentation address is one component of the Access Point for a Directory Server (the other being its Distinguished Name). In most cases, you will only need to provide the hostname (or IP address) of the server and the port number on which the application will be listening when either creating a Directory Server or connecting to one.

To help when specifying more complex presentation addresses, M-Vault Console includes a facility that splits them down into their component parts.

Note: Presentation address components are only briefly summarized here. See [Section C.2.28, “PresentationAddress”](#) and the *M-Switch Administration Guide* for a fuller description of string format presentation addresses.

The string representation of a presentation address can consist of transport, session and presentation selectors (up to three selectors), network addresses (up to eight), including an LDAP address. However, the only mandatory component is a single network address.

The detailed version of the **Presentation Address Configuration** page is available when you are asked to supply a presentation address, including:

- As part of the wizard when creating a new Directory Server (see [Section 2.2.3, “Creating a Directory Server”](#))
- As the **Directory Server Address** page when completing **Advanced** details (see [Section 3.2.2.2, “Creating or modifying a bind profile”](#)).
- As the **Server Address** page displayed in the **Configuration** section of M-Vault Console (see [Figure 4.1, “The M-Vault Console Managing window”](#)).

If the advanced version has been accessed from the **Simple Hostname / Port Editor**, it will contain details that were entered on that simpler page.

Type	Hostname or network address	Port number
X.500	dsa.example.com	19999
LDAP	dsa.example.com	19389

Selectors : (none)

Simple Hostname / Port Editor

< Back Next > Finish Cancel

- Click **Add...** to add other types of address information to those already shown (if any).
- Select an entry and click **Edit...** or double-click it to make changes.
- Select an entry and click **Remove** to delete it.

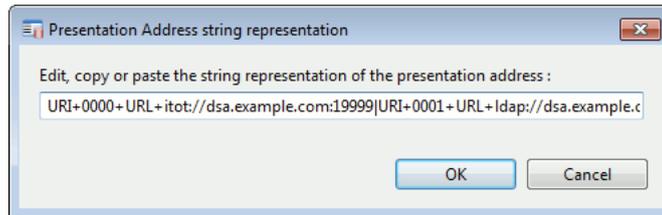
Click **Selectors...** to specify individual selectors. The selectors are **Presentation**, **Session** and **Transport**.



The selectors can be entered as IA5 (text) or hexadecimal:

- IA5 means an IA5 (ASCII text) string. For example, x500 may be a valid text **Transport** selector.
- Hexadecimal means a string of an even number of hexadecimal digits, for example, 001aff. The length of the selector is half the number of digits in the string. This is the most general form and can be used to represent any selector value.

Click **Advanced** to see a string representation of the address that can be edited directly.



Note: Any change to any part of the Directory Server's presentation address only takes effect after the Directory Server has been stopped and restarted.

2.3 What's next?

Depending on the role of the Directory Server, you may need to:

- Decide on the structure of the DIT it will master if this has not already been done – see [Section 1.2.1, “The logical structure of the Directory”](#).
- Delegate responsibility for management of the information stored within the Directory (see [Chapter 6, *Controlling Access*](#)).
- Specify security settings: the level of authentication required by Directory User Agents and secure connections (see [Section 5.3, “Configuring authentication for specific protocols”](#)).
- Prepare and load any existing data, either in bulk or as individual entries – see [Section 3.8, “Importing and exporting entries”](#) for guidance on preparing and uploading multiple entries, or [Section 3.5, “Adding single entries to the Directory”](#) for instructions on adding single entries.
- Connect to other Directories – see [Chapter 7, *Connecting Directories*](#). Determine shadowing (replication) requirements and create appropriate agreements between the Directories involved – see [Chapter 8, *Shadowing*](#).

Chapter 3 Managing the Data

This chapter describes how to add, modify and delete data using Sodium, either as individual entries or in bulk. This chapter should be read by the Data Manager, although the Server Manager will find some of the background information useful.

In particular, this chapter covers:

- Deciding where to store the information: choosing an appropriate location within the Directory structure enables you to control access to it more easily. This is covered in [Section 1.2.1, “The logical structure of the Directory”](#).
- Classifying the data you want to store: this determines what you must and what you can record about each entry. This is covered in [Section 3.1, “Classifying the data”](#).
- Using Sodium to add, modify and delete Directory entries.
- Customising Sodium to meet your organization’s needs.

[Appendix A, *Introduction to Directories*](#), contains useful background information about Directories that will help you to make appropriate decisions.

Note: When a Directory Service is created, only the Data Manager is allowed to add, modify or delete user information entries in the Directory. This management responsibility can be delegated using one of the management tools provided by Isode. See [Chapter 6, *Controlling Access*](#), for more information.

3.1 Classifying the data

Before you start adding data to the Directory, you need to know what sort of data it is. In Directory terms, this is defined by the object classes of that entry, which determine what information can be stored about it.

When adding an entry to the Directory, you need to specify a structural object class (**person**, **organization**, **organizationalUnit** and so on) and can optionally add multiple auxiliary object classes to enable additional information to be recorded.

Sodium simplifies this for you: you choose the type of entry you want to add, which specifies the structural object class, and then select from a list of pre-defined templates to enable additional information to be recorded (see [Section 3.4, “Modifying entries”](#), for information on modifying entries, and [Section 3.5, “Adding single entries to the Directory”](#), for instructions on adding entries).

Note: The number of available object classes is large. We suggest some time is spent familiarising yourself with the classes you might want to use for your organization’s data. Some of the object classes contain more attributes than are currently displayed on Sodium’s template pages. To view all the available attributes within an object class, select the **Schema view** (see [Figure 3.2, “Sodium’s Browse page”](#)).

3.1.1 What information can be stored?

The object classes selected for an entry define the set of attributes that the entry must (are mandatory) and may (are optional) contain. [Appendix B, *Attributes*](#), gives some examples of structural object classes and their associated attributes.

Note: While it is possible to define your own set of additional attributes and object classes (see [Section B.2, “Extending the schema”](#)), this should only be done when there is no alternative. Locally defined attributes may not be understood by remote systems.

3.1.2 Standard attributes

Many attributes available for use in White Pages and other applications are self explanatory – such as **Surname**. The list below gives more detail for those that require some explanation (these attributes are defined by the X.521 (2008) standard).

Note: It is important to use attributes to record the type of information they were intended to record. You should not, for example, use **Description** to hold a telephone number, as this will affect searching capabilities.

Note: The attributes below are listed as they are labelled in Sodium. The actual attribute name is the same, without the spaces, unless otherwise specified: variations are shown in square brackets.

See Also

Used to specify the DNs of other closely related objects. For example, when used as an attribute of a room object, **See Also** could be used to indicate the room’s usual occupant(s).

Postal Address Label

[**postalAddress**] Should be the full postal name and address, although it is limited to six lines of thirty characters each, which is to be used for the physical delivery of paper mail and parcels.

Phone

[**telephoneNumber**] Should be stored using the full international format defined in E.123, for example +44 123 456789. Extensions can be indicated by appending the letter “x” and the extension number to the telephone number.

Fax

[**facsimileTelephoneNumber**] Should be stored, like **Phone**, using the full international format, but can have additional parameters for controlling fax machines that normal phone numbers do not have.

Description

Can be used for arbitrary descriptive text about the object.

Title

Should be used to describe a job title or function within an organization.

City/Locality

[**localityName**] Used to identify the geographical areas of locality in which the object is physically located.

State/Province

[**stateOrProvinceName**] Describes the state in which the locality is found.

Aliased Object’s DN

[**aliasedObjectName**] The full Distinguished Name of an aliased object.

Group Members

[**member**] Specifies the DN of a member of a **Group of Names** [**groupOfNames**].

Owner

The DN of the person responsible for the associated object.

Role Occupant

The DN of the person who fills an organizational role.

3.1.3 Internet attributes

More contact and personal details can be added by selecting the **Internet Organizational Person** object class from the template. This makes a number of other attributes available, such as **E-mail** [**mail**], **Mobile**, **Pager** and **Home Phone**.

3.1.4 Collective attributes

Many entries within a subtree may share the same values. For example, all people within a small organisation might share the same telephone number and postal address. To manage this easily and avoid duplication of data there is a special kind of attribute called a collective attribute.

Collective attributes are stored in a subentry of an Administrative Point (see [Section A.3.1, “Administration of the Directory”](#)), but they appear in all the entries within the Administrative Area. Many of the standard attributes have a collective attribute subtype, which means that any queries will retrieve values from both the individual and the collective attributes. For example, **collectiveTelephoneNumber** is defined as a subtype of **telephoneNumber**.

Sodium provides a special view that allows you to manage collective attributes (see [Section 3.7, “Collective Attributes”](#)).

3.1.5 Unknown attributes

The M-Vault Server is configured with the most current, and commonly used attribute types, together with attributes defined for its own use. However, other Directory Servers may define their own local attribute types, and new standard attributes will continue to be published, all of which will have to be handled by the M-Vault Server. In most cases, the M-Vault Server will handle these attributes transparently by storing the attribute OID and the ASN.1 along with its value. This unknown attribute subsequently can be returned in future requests (hence a DUA/Directory Server which is configured with the attribute will be able to handle it correctly). All unknown attributes are treated as user-attributes in order for them to be handled appropriately by the M-Vault Server.

3.2 Data management using Sodium

Sodium is a Directory User Agent (DUA). It enables you to add, modify and delete Directory entries. Creating and modifying entries using Sodium ensures that they contain all the information required for the Directory to function correctly. You will not be able to create an entry without specifying values for any mandatory attributes.

There are two basic methods for getting data into a Directory:

- Add the data entries individually.
- Get the data into an acceptable form, and bulk load it.

Obviously, if you have large amounts of data and can get it into an acceptable form, it is much more efficient to bulk load it. See [Section 3.8, “Importing and exporting entries”](#), for more information. However you are going to add the data, you need to connect (bind) to the Directory Server using a user who has permissions to do so (see [Section 3.2.2, “Binding to the Directory using Sodium”](#)).

3.2.1 Profile passphrase

Bind profiles hold connection information for Directories. They can be protected by encrypting them using a passphrase (see [Section 2.1.3, “Using bind profiles”](#)). When you start Sodium, it will prompt you for this passphrase if your Bind Profile file is encrypted.

You can use Sodium without specifying a passphrase; however, you will be unable to perform any of the functions related to the management of identities (X.509 functions – see [Section 3.10, “Managing identities”](#)), and you will be unable to save passwords in your bind profiles (see [Section 3.2.2, “Binding to the Directory using Sodium”](#)). If you have not already created a passphrase for your bind profiles, you will be prompted to create one when you start Sodium.

If you want to set a passphrase at a later time or change your existing passphrase, select the **X.509** menu and then select **Set Sodium Passphrase...** If you are changing an existing passphrase, you will be prompted to enter it for verification. If you change your profile passphrase, then encrypted bind profile data and any deferred identities (see [Section 3.10.2, “Continuing to create a deferred identity”](#)) will be re-encrypted using the new one.

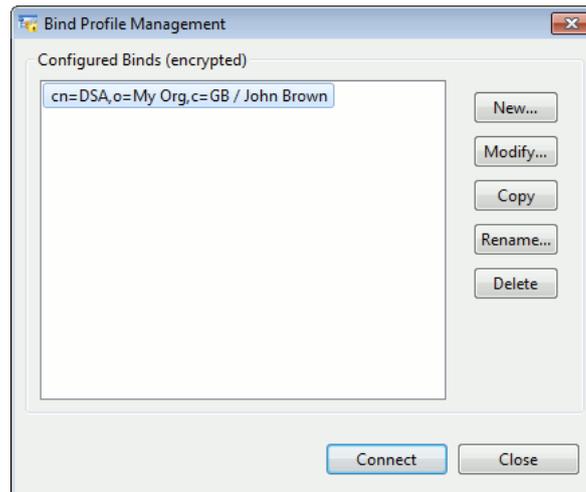
Note: You use the same file containing bind profiles whether you are connecting through Sodium or M-Vault Console. In Sodium you only see profiles for servers that you manage and profiles that you create in Sodium. You cannot see profiles that were created in M-Vault Console for ‘known’ servers (those you do not manage).

3.2.2 Binding to the Directory using Sodium

To connect to a Directory, Sodium uses a bind profile (see [Section 2.1.3, “Using bind profiles”](#)). Once a bind profile has been created, you can use it to create new connections to that Directory Server (DSA) whenever you want.

Note: For a one-off quick connection to a DSA, select **Session** → **Bind** → **Connect...** from the menus to create a temporary bind profile which will be used just to perform a single bind.

To see the list of available bind profiles at any time, open the **Bind Profile Management** window by selecting **Session** → **Bind** → **Manager...** from the menu.

Figure 3.1. Sodium's bind profile manager

A bind profile always includes the location of the server, the protocol (DAP or LDAP) it is using, and details about how to authenticate (for example, simple or strong authentication), along with any necessary credentials. Other information may also be held in a bind profile, such as the service controls to be used, or options which determine how Sodium displays information once a connection has been made.

You can create as many bind profiles as you want, and they are saved when you quit the application so that they will be available when you next run Sodium.

When bind profiles are saved to disk, Sodium will omit any sensitive data (such as passwords) unless the bind profiles are encrypted.

If your bind profiles are not already encrypted, an **Encrypt** button is shown on the **Bind Profile Management** window. If you click **Encrypt**, Sodium will prompt you for a passphrase, which it will use to encrypt your bind profiles. You will then have to give this passphrase whenever you run Sodium.

From the **Bind Profile Management** window, you can:

- Connect to a Directory using a bind profile
- Create a new profile or modify an existing one
- Delete profiles that are no longer required.

3.2.2.1 Connecting to the Directory

Select a profile from the list and click **Connect**. Once a connection has been made, Sodium displays a **Browse** tab in the main window.

Note: When you make a new connection, any previous connections will remain open (on separate tabbed pages). This means you may have simultaneous connections to multiple Directories (or to the same Directory).

Disconnect from a Directory using the **Session** → **Unbind** menu option. A connection will also be unbound if you close all the windows that were showing it.

3.2.2.2 Creating or modifying a bind profile

To create a bind profile, either click **New...** or **Copy**. Which you choose depends on how much of the information you need is in an existing profile. For example, if you are creating a second bind profile to connect to the same Directory but at a different access point or

using a different identity, choosing **Copy** will reduce the amount of information you need to enter manually. You can then modify the copy to make any necessary changes.

To make changes to a bind profile, select it and click **Rename** or **Modify**. If the only thing you need to change is the display name to make it more easily identified, select **Rename**. If you need to change other information as well, click **Modify**.

Note: The steps are the same whether you are creating a new bind profile or modifying an existing one.

1. Specify how the Directory will be contacted on the **Directory Server Address** page. Use this page to specify the location of the Directory Server and the protocol used to connect to it.

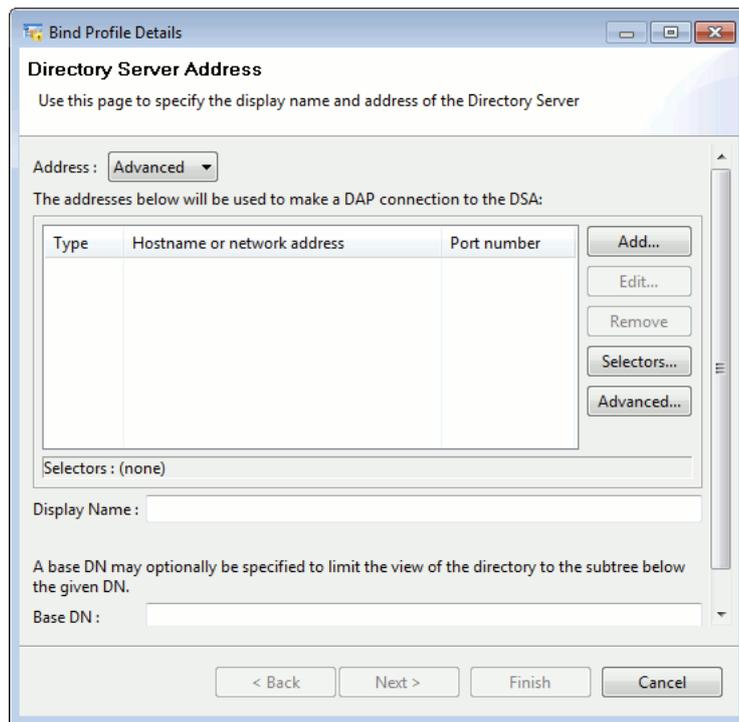
- The protocol you choose (**DAP**, **LDAP** or **LDAPS**) will change the **Port** number displayed to the default for that protocol, unless it has already been edited.
- The background of the **Hostname** box will change from red (invalid) to orange to clear as the hostname is entered, assuming it is a valid value that is reachable.

A hostname can be:

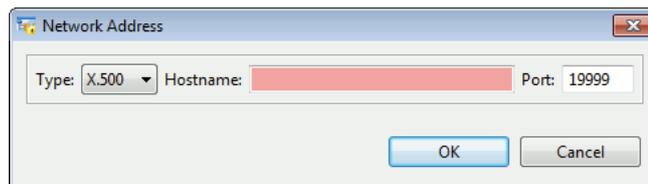
- A fully qualified domain name (for example, `server1.myorg.co.uk`)
- An IP address
- `localhost`.
- The **Hostname** is shown as the default **Display Name**, but this can be changed.
- Specify a **Base DN** if you want to connect to the Directory at a certain point in the tree.

If you chose **Advanced** in the first drop-down list, you can specify all the elements of the presentation address individually.

- The **Directory Server Address** page is shown in **Advanced** mode.



- Click **Add...** or select an existing entry and click **Edit...** to open the **Network Address** window.



- Click **Selectors** to open a window for specifying **Presentation**, **Session** and **Transport** details. See [Section 2.2.3.3, “Specifying a presentation address”](#), for more information on specifying the different selectors.

Click **Next** to move on to the **Authentication Type** page.

2. Choose the type of authentication you want to use when connecting to the Directory using this profile. Access to the Directory depends on the correct credentials being supplied when an attempt is made to bind.

Note: Authentication is the means by which the client and server prove their identities to one another once a connection has been established. Descriptions of the different levels of security are given in [Chapter 5, Authentication](#).

The available options are:

- **Anonymous:** No logon credentials (username or password) are required.

Click **Next** to go to the **Anonymous Bind** page, where the only option is to **Start TLS** and then (optionally) to select an identity to use if the LDAP Server requires a client certificate for TLS. See [Section 5.7, “TLS configuration”](#), for information on TLS.

- **Simple:** A DN is required, together with an optional password.

Click **Next** to go to the **Simple Bind** page. Select a **Bind DN** from the Directory and enter the password if you do not want to be prompted for it when binding. You also have the option to **Start TLS** (as with an anonymous bind).

- **Strong:** A strong bind uses a DN and a secure private key. For more information, see [Section 5.2.1, “Establishing identity”](#).

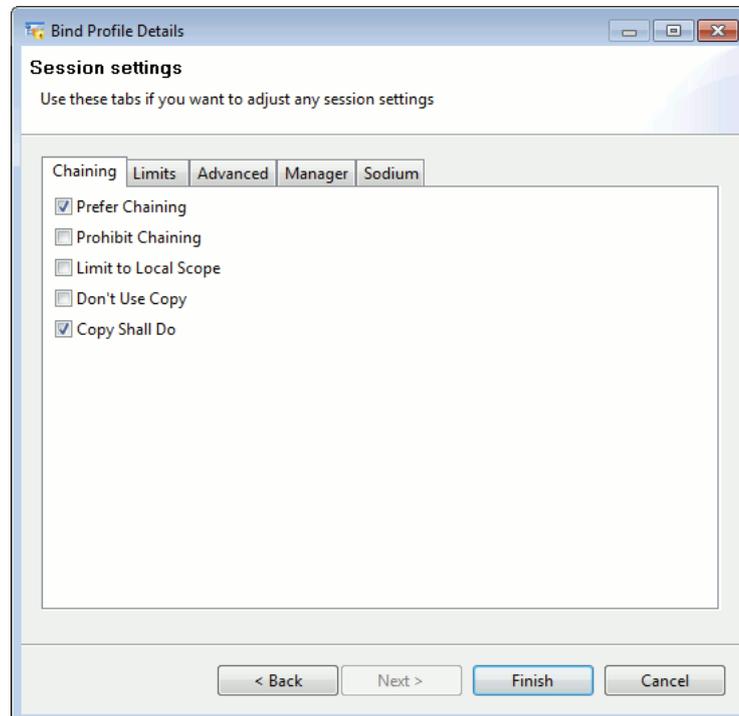
Click **Next** to go to the **Strong Bind** page. **Start TLS** is selected automatically, and you must provide an identity. You can also provide an optional **SASL Authorization ID** (see [Section 5.5, “SASL authentication”](#)).

- **Kerberos:** This type of bind uses a ticket issued by a trusted third party to authenticate you.

Click **Next** to go to the **Kerberos Bind** page. You need to specify the level of security (**Security Layer**) you require, whether **Active Directory compatibility** is required and whether you are going to use **Single sign-on** (the same credentials as used when logging on to the operating system) or will provide a **Username** and **Password**. See [Section 5.5.4, “SASL GSSAPI configuration”](#), for more information.

When you have set the bind credentials you require, click **Next** to go to the **Session settings** page.

3. This page contains five separate pages, accessed by tabs. You only need to make changes here if you want something other than the default settings.



Specify how you want to manage operations when results cannot be returned from the local Directory on the **Chaining** page.

- **Prefer Chaining** - if the objects are not on the local Directory Server, chain to another one if this is possible. Chaining passes the request to the other Directory Server and then passes the result back to the requester.

Note: This option is a request, and is irrelevant if **Prohibit Chaining** (see below) is selected.

- **Prohibit Chaining** - if the objects are not on the local Directory Server, do *not* chain to other servers.
- **Limit to Local Scope** - only use data stored on Directory Servers considered to be 'local'.
- **Don't Use Copy** - if this option is selected, the Directory Server will only consider master copies of information and will ignore shadow (replicated) information.

- **Copy Shall Do** - if this option is selected, a copy of information that may not be complete is acceptable. This may be sufficient to answer some end-user queries (such as obtaining an email address) and may be quicker than insisting only complete copies are acceptable.

Note: This option is irrelevant if **Don't Use Copy** is selected.

You can control the impact of operations using this bind profile using the **Limits** page.

- **Priority** - this is a numeric value that corresponds to a priority level associated with any operation started by binding using this profile. 0 = low priority, 1= medium priority, 2 = high priority.
- **Size Limit** - specify the maximum number of entries that are returned in a **list** or following a search. The server will enforce an upper limit regardless.
- **Time Limit** - the maximum time, in seconds, that a results from a **search** or a request to **list** entries can take. Any results that are not returned in this time are shown as errors.

Advanced options enable you to access and view the information in different ways.

- **Don't Dereference Aliases** is selected by default, which means that aliases that are either the starting point of an operation (e.g. the search base of a **search**, the parent of an **add**, or the object itself for other operations) are not automatically dereferenced. If this option is *not* selected, then the real entry referenced by the alias is used.
- **Request Password Policy Information** - see [Section 3.2.2.3, "Password policy"](#).
- **Search Aliases** - if selected, any aliases found in the search scope will automatically be dereferenced, and those entries searched instead.
- **Use Alias on Update** - if selected and **Don't Dereference Aliases** is also selected, updates made to an alias entry will be applied to the alias and not to the entry it references.
- **Subentries only** - when this option is selected, searches will only look at subentries (entries held immediately below administrative points that hold access control, schema or collective attribute details).
- **Manage DsaIT** - if not selected (the default), operations carried out during this session will apply to normal entries. Select this option to direct operations to other levels using knowledge references.

The **Manager** tab allows you to specify whether Sodium should present extra information that is useful when performing management operations. Of special note is the **Enable Directory Server Manager GUI Features** option: when this is selected, the **View** menu is updated to include options which allow the creation of views to allow the viewing and configuration of global and local access control ([Section 6.2, "Global access control"](#) and [Section 6.3, "Local Access Control Information \(ACI\)"](#)), knowledge references ([Section 7.3.2, "Subordinate and cross references"](#)) and collective attributes ([Section 3.7, "Collective Attributes"](#)).

Use the **Sodium** page to specify

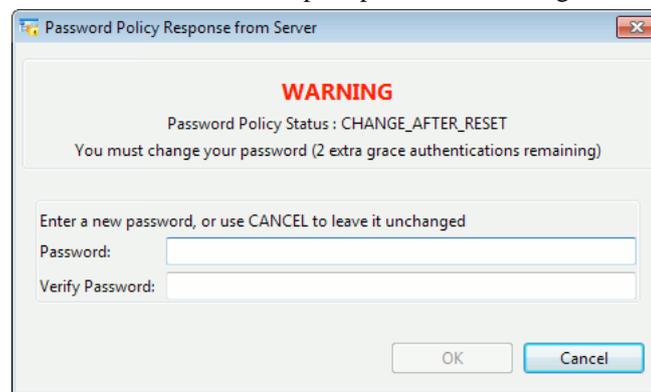
- Which views should be enabled (template, full, schema, raw)
- Whether the default view should be *Full* or *Template*
- Whether to issue a **Confirm** prompt before deleting any entries
- Whether to fill in **memberof** fields. By default, when displaying an entry whose template has a **Member of** attribute, Sodium will perform a search of the entire DIT in order to find groups that contain the entry's DN. This allows Sodium to show which groups contain the entry. For large directories where there are many groups, these searches can be slow and so this option allows you to disable this functionality.

- Whether DN verification is performed. When an entry contains attributes that are themselves distinguished names, DN verification means that Sodium will verify the attribute values and highlight any that do not correspond to actual entries in the directory. For entries where there are many attribute values of this type, you may want to limit the number of values which Sodium verifies, or disable verification altogether.
- Specify the entry in the DSA that contains security policy and catalogs that are to be used when Sodium displays and edits SIO information.

4. Click **Finish** to complete the bind profile.

3.2.2.3 Password policy

Sodium is able to request password policy information when performing a bind to a Directory Server. You can set this option using the **Session Settings** window, available in the **Session** menu, and also as the final stage of creating or modifying a bind profile). Directory Servers may respond to this request with information about password expiration, in which case Sodium will prompt the user to change his or her password.



A Directory Server may operate a password policy that imposes constraints on legitimate values for passwords (such as minimum length); errors caused by the entering of illegal passwords for the policy will result in Sodium requesting the user to try a different password. See [Section 5.6, “Password management”](#), for information on M-Vault’s implementation of password policy.

3.2.3 Changing your password

In the case of simple authentication, the Directory Server maintains a copy of your password (possibly in a hashed format; see [Section 5.6.3, “Storing passwords in the GDAM”](#)) which it uses to validate bind operations. When you are bound using simple authentication, the **Session** → **Change my Password...** menu option will be enabled, which lets you change your password on the Directory Server (any new password you supply will be subjected to whatever password policy checks are enabled on the Directory Server). If you change your password using a session which is associated with a bind profile, then Sodium will also update the bind profile to reflect the updated password, which means that the bind profile will continue to work with the new password.

3.3 Finding Directory entries

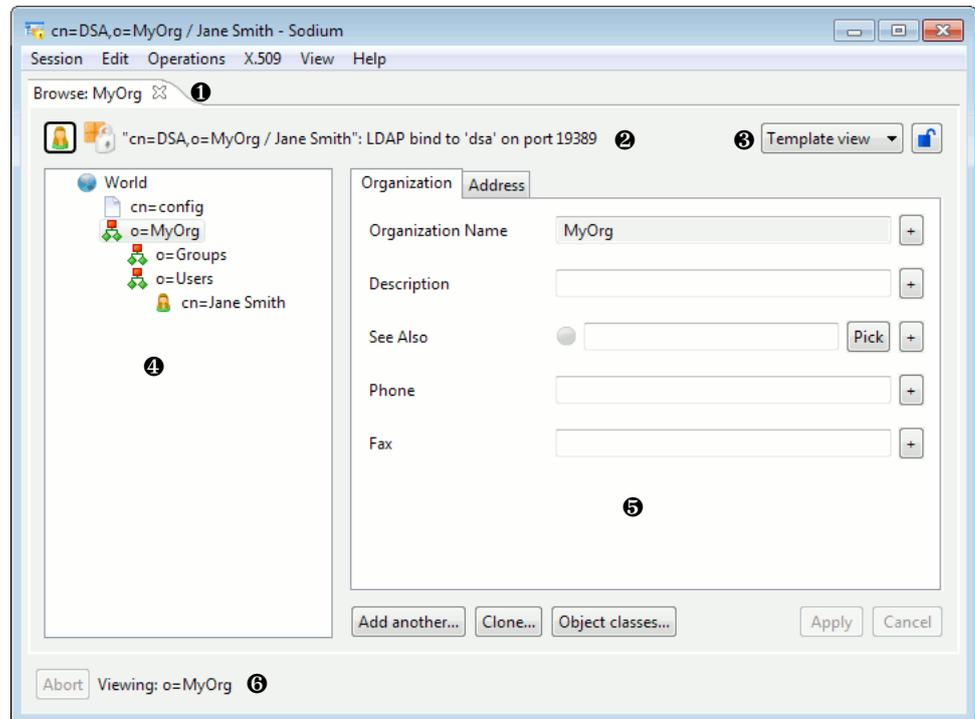
You can find Directory entries to check the values they contain or to make changes either by browsing the Directory (described below) or by searching for them (see [Section 3.3.2, “Searching the Directory”](#)).

3.3.1 Browsing the Directory

A **Browse** page (identified by a tab that is labelled **Browse:** followed by the name of the currently selected object) is opened when you connect to a Directory server. You can expand the tree (described in step 3, below) to see the contents of the Directory.

The different sections of the page have been numbered in [Figure 3.2, “Sodium’s Browse page”](#), for ease of reference: a description follows the image.

Figure 3.2. Sodium’s Browse page



Key to numbers:

1. The tab itself. This displays both the type of tab and the name of the entry currently being viewed.
Click the 'X' on the right of the tab to close that page.
2. The name and authentication level of the current connection are displayed at the top of the page, immediately below the tab.
3. Over to the right of this area is a drop-down list which changes the view of the contents on the right-hand side of the page, and a lock button that allows the current view and tab to be maintained whilst browsing between entries.
4. The left-hand side of the page shows a hierarchical tree view of the Directory.

Initially there will be only one entry in the view labelled **World** which represents the root entry of the connected Directory Server.

You expand each entry in the view to reveal the entries immediately below it in the DIT. Entries retrieved from the Directory will be named with their RDNs; for example, **o=Acme Limited** or **cn=DSA Manager**. Some entries will have no child entries and so cannot be expanded or contracted.

Right-click on an entry in this area to show a menu that mirrors the options available in the **Operations** menu.

5. The right-hand side of the page shows information related to the entry selected on the left-hand side in a series of tabbed pages. The way in which this information is displayed is determined by the option selected from the drop-down list at the top of the page.
 - **Template** – attributes are only displayed when they are defined in an appropriate template.
 - **Full** – this is the same as the Template view, except that any attributes returned that are omitted from the templates are displayed in an extra **Misc** tab.
 - **Schema** – this shows one tab per object class, and shows each attribute permitted in each object class using an appropriate editor. It does not use templates.
 - **Raw** – this shows a single tab, and shows each attribute using an appropriate editor. It does not use templates.
 - **Custom** – this shows a single tab listing each attribute that has failed Sodium’s referential integrity checking.
6. The full DN of the entry read is displayed at the bottom of the main window next to the **Abort** button.

The **Abort** button is used to cancel any currently active operations. This can be useful if a Directory Server is slow to respond.

Note: You can open multiple **Browse** pages: right-click on an object and select **Browse** or select **Operations** → **Browse** from the menu. The new page will open in a new window, but you can drag it to form a new tab in your existing window if you prefer.

3.3.2 Searching the Directory

In a Directory containing a large number of entries, it may be easier to find an entry by searching than by browsing.

Note: A search will start from whichever object you have selected. So, for example, if you start a search when you first bind to the Directory without doing anything else, you will be searching the whole Directory (from **World** level). If you know, for example, that the entry you want is inside a particular organization or organizational unit, select it before you start the search.

To start the search:

- Right-click on an object and select **Search**.
- Select **Operations** → **Search** from the menu.

You can choose either:

- **Simple Search** – type the text you are searching for into a text field and then click **OK**.

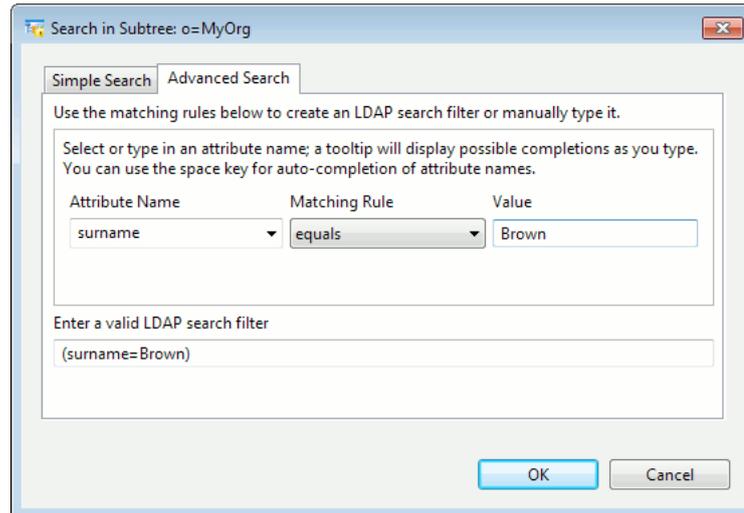
This option searches for entries containing your text in attributes commonly used for names (such as **commonName**, **surname**, etc.).

- **Advanced Search**, which allows you to build an LDAP-style search filter (see [Figure 3.3, “Sodium’s search window”](#)).

1. Select the **Attribute Name** you want to search from the list.
2. Select the **Matching Rule** you want to use.
3. Type the value you are searching for.

The search filter will be created by Sodium and will be displayed in the bottom field.

4. Click **OK**.

Figure 3.3. Sodium's search window

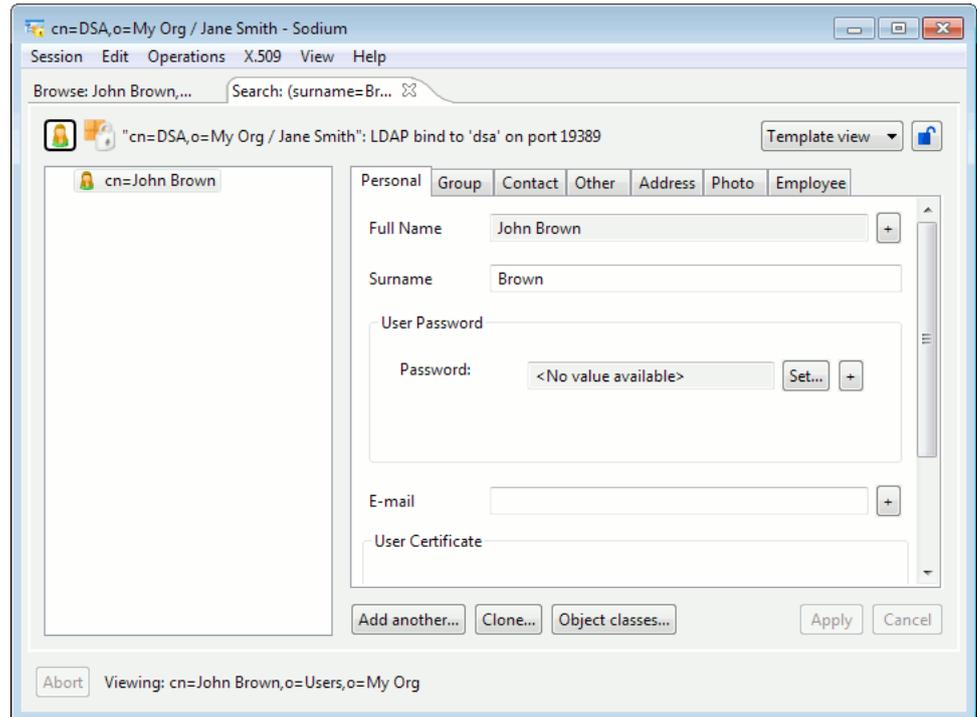
3.3.2.1 Viewing search results

Results from the search are displayed on a **Search** tabbed page, identified by the word **Search:** followed by search filter you used; for example, **Search: (surname=Bro*)**.

The results are displayed in the same kind of tree view as used in the **Browse** page (see [Section 3.3.1, "Browsing the Directory"](#) for an explanation of the page display), except that only the entries matching the search filter are displayed.

[Figure 3.4, "Results of a search"](#) shows the results of a search. The italicized entries in the tree view are not search results; they are present so that the real search results are displayed within the appropriate hierarchy.

Click an entry in the tree to display information about that object in the right-hand side of the page. Selecting **Refresh Search** from the DIT context menu or from the **Operations** menu requests fresh results from the Directory Server to refresh the view.

Figure 3.4. Results of a search

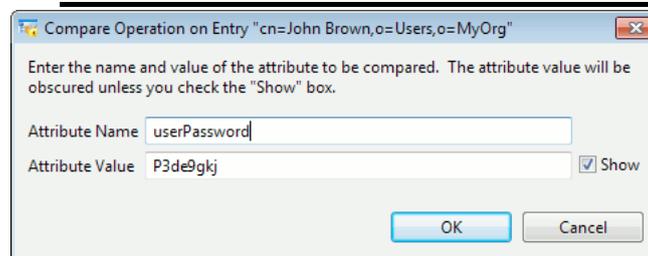
Note: You can have the results from several searches open at once. Use the information on the tabs to identify them.

3.3.2.2 Comparing attributes

To compare the value of an attribute with one you provide, right-click on an object in the tree on the left side of the **Search** page and select **Compare**. The **Compare** option is also available from the **Operations** menu. Enter an attribute name and value in the box displayed. Sodium will ask the Directory whether the entry contains the specified value for that attribute.

This is useful for checking attributes such as passwords in configurations where the attributes may not be read, but may be compared as a means of verification.

Note: The attribute value being compared is only visible if you select the **Show** option.

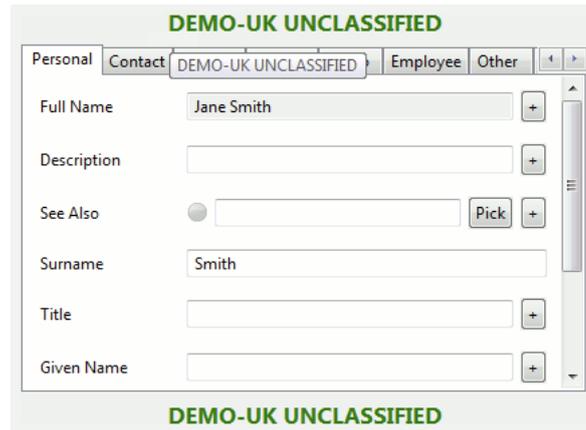


3.3.2.3 Security policy

An M-Vault server may be configured with a Security Policy ([Section 6.5, "Security labels and clearance"](#)), in which case Sodium will be subject to whatever constraints the policy specifies: for example, when you are browsing the Directory tree, the Directory Server will only return information about entries that your security clearance permits you to see.

For any entry that has a security label, Sodium attempts to validate the label against the DSA's security policy. Depending on the policy, this can result in Sodium displaying extra marking data above and/or below the entry's contents, as shown in [Figure 3.5, "Security policy warning message"](#). The Security Policy determines the value of the text, its colour, and its tooltip (in the example shown, "DEMO-UK UNCLASSIFIED", green, and "DEMO-UK UNCLASSIFIED", respectively). Should Sodium encounter an error when processing Security Policy information (for example, if an entry contains a Security Label which refers to a Policy other than that for which the DSA is configured), then a warning will be displayed before the entry is shown, and a warning label will be shown above and below the entry itself.

Figure 3.5. Security policy warning message



3.4 Modifying entries

Select an object in the tree on the left of the page to see display the values of its attributes in the area on the right. The way in which the information is displayed depends on the view that has been selected.

If an entry is successfully changed in some way (modified, renamed, or moved), Sodium will automatically update all the other views that also show that entry.

3.4.1 Changing the value of an attribute

Most of the attributes displayed in the right-hand part of the **Browse** page are editable. If Sodium determines (using the schema) that a particular attribute may not be changed by users, it will prevent it from being edited.

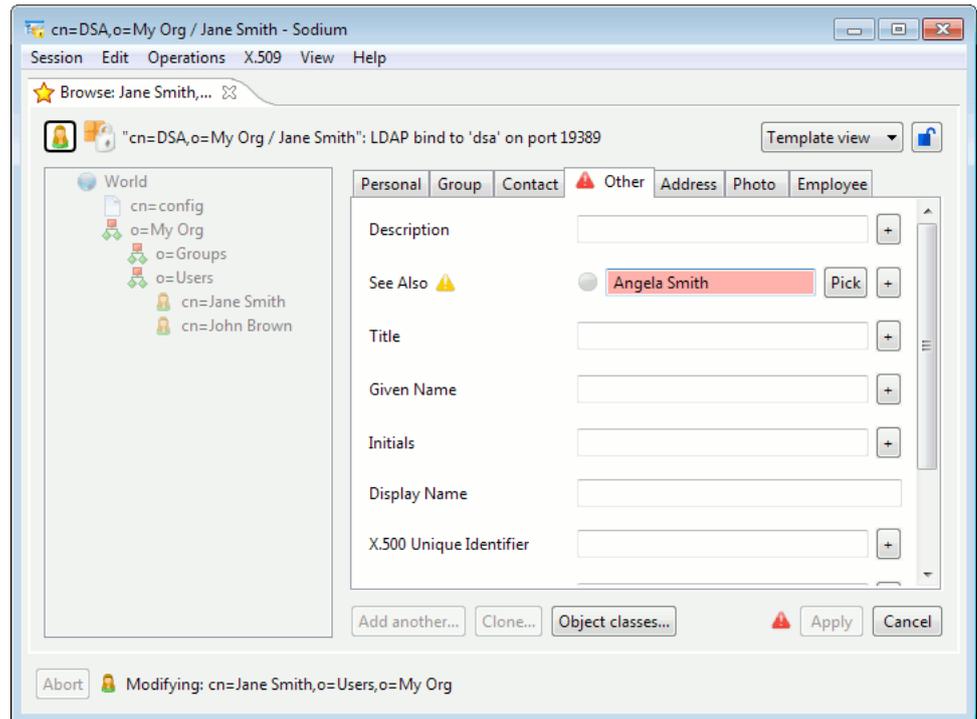
- The **Apply** button is disabled until a change is made to any of the values. If you click **Apply**, Sodium will attempt to modify the Directory entry with the changes.
- The **Cancel** button will discard all the changes made to an entry, and will re-read it from the Directory server.

3.4.1.1 Highlighting errors

You will not be able to apply changes (the **Apply** button will remain disabled) if mandatory attributes in the entry are empty, or if a value entered for an attribute is invalid. To enable you to identify the problem attribute(s) a red icon is displayed:

- On the tab of any page containing an affected attribute
- Alongside the attribute

- Alongside the **Apply** button.



Note: Sodium can perform additional checks on the contents of certain attribute values, and will draw yellow warning icons as well as change the background of any fields which fail these checks to yellow.

3.4.1.2 Undoing changes

You can undo changes to an attribute value by selecting one of the following from the **Edit** menu:

- **Revert attribute**, then **Revert attribute to last valid values**
- **Revert attribute**, then **Revert attribute to original values**

You can also see these options by right-clicking on the field.

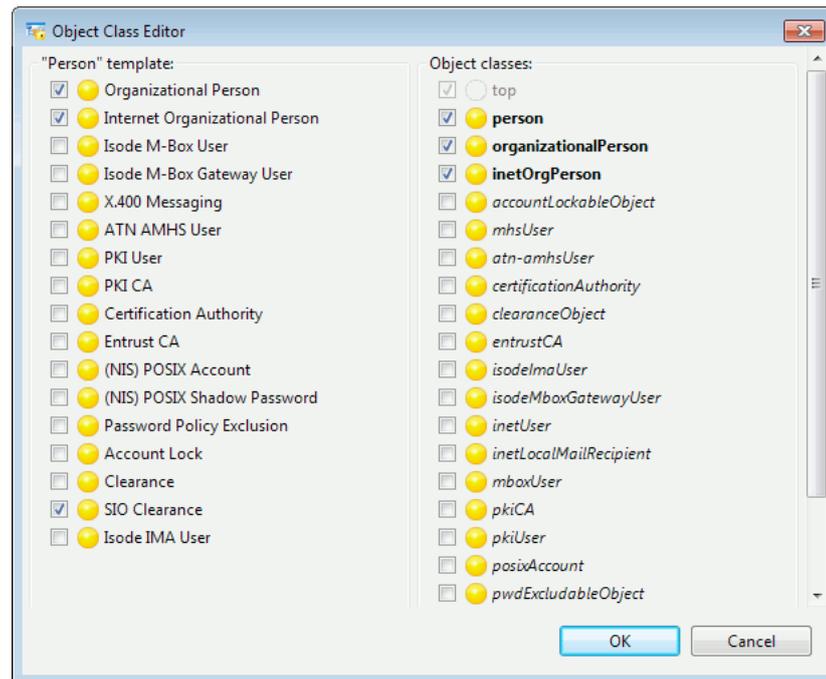
3.4.2 Changing an entry's object class

To modify the object classes associated with an entry, click **Object classes...** below the entry's attribute values.

The editor used to modify object classes can display two columns: a list of templates and a list of object classes.

- Both columns are shown in when you are using the **Full view** (see [Figure 3.6, "Object class editor showing both the templates and the object classes"](#)).
- Only the templates column is displayed in **Template view**.
- Only the list of object classes is displayed in all other views.

Figure 3.6. Object class editor showing both the templates and the object classes



A template or object class currently being used in the entry is indicated by a tick in the box to the left of its icon and name.

- To add an unused template or object class, click it. A '+' will appear on the icon, indicating it will be added when you apply the changes.
- To remove a currently used template or object class, click it. A '-' will appear on the icon, indicating it will be removed when you apply the changes.

Object classes are often related – for example the **inetOrgPerson** object class is a subclass of the **organizationalPerson** object class. Sodium is aware of these relationships and will automatically add any required object classes when necessary.

To see which object classes are related to one you want to use, hover the mouse over it:

- Required object classes are shown with a blue icon.
- Optional object classes are shown with a green icon.

In the **Full view** (as shown in [Figure 3.6, “Object class editor showing both the templates and the object classes”](#)) a list of object classes is shown on the right. In this list, structural object classes are displayed in bold type and auxiliary object classes are displayed in italic type. Selecting object classes will automatically cause the appropriate templates to be selected.

3.4.3 Entering data containing line breaks

Most string entry fields in the entry editor start off as a single-line field, but many of them can be switched to a multiple-line mode. To do this, either:

- Paste multi-line text into the field.

- Select **Switch to multiple-line field** from the **Edit** menu, or from the menu shown when you right-click on the field.

Note: To enter a line-break in a multi-line field, use the Return key on your keyboard.

Multi-line mode is also used if the existing field contents contains unprintable characters or line breaks. In the multiple-line mode, escape sequences are used: “\” for backslash, “\r” for carriage-return, “\n” for line-feed, and “\” plus two hex digits for undecoded UTF-8 bytes. The hex form may be used both for invalid UTF-8 fragments and also for valid UTF-8 characters that are unprintable.

Buttons to the right of the field allow switching between UNIX (LF) and DOS (CR-LF) line-ending views and also conversion of line-endings between the two systems. (DOS is the convention used on Windows systems.)

Note: Switching the view does not change the underlying data in any way; it only determines which control sequence is shown as a line-break in the field. Any remaining control sequences are shown using escapes. Conversion does modify the underlying data, mapping LF to CR-LF or CR-LF to LF depending on the direction of conversion.

3.4.4 Moving and renaming

When you rename an entry, you change its RDN – the name that uniquely identifies it in its current location. When you move an entry, you change its DN – the name that uniquely identifies it within the tree. This means you cannot:

- Move an entry to a location where there is already another entry with the same name.
- Rename an entry if there is already an entry in the same location with that name.

Caution: If you move or rename an entry, you may invalidate the values of some attributes if they are based on a DN that included the RDN of that entry. Renaming or moving anything other than a leaf entry – one that does not have any subordinate entries – is likely to affect a number of other entries. You can identify which values have been affected by checking referential integrity (see [Section 3.9, “Checking the referential integrity of attributes”](#)).

3.4.4.1 Moving an entry

To move an entry to a new location in the tree:

1. Select the entry (either from the **Browse** page or following a search).
2. Drag and drop it into its new location.

Note: If you are moving an object in a large well-populated Directory, you may find it easier to drag and drop the entry you are moving between two different **Browse** pages in different windows.

3.4.4.2 Renaming an entry

To rename an entry, either:

- Right-click on the entry and select **Rename** from the menu displayed

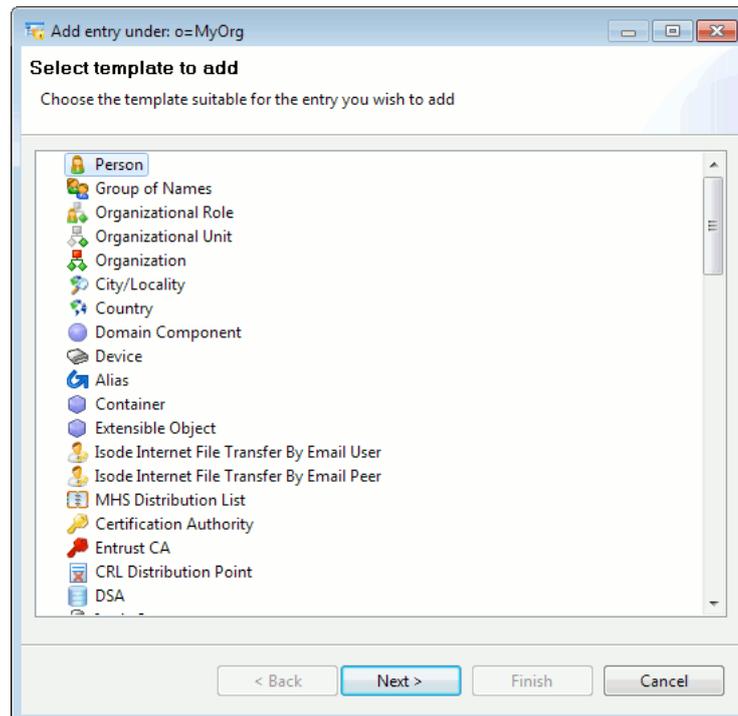
- Select **Operations** → **Rename** from the menu.

Note: If you rename an entry from a **Search** page, you will see a red warning triangle telling you that the entry cannot be found – that is because that search page was based on the object’s old name.

3.5 Adding single entries to the Directory

There are three ways to add new entries using Sodium:

- The **Add another...** button will create a new entry as a sibling of the current one, using the same template and options as the current entry. This conveniently allows multiple similar entries to be created.
- The **Clone...** button will create a new entry as a sibling of the current one, using the same template and options, with all the attributes (except the naming attribute) initially set to values copied over from the current entry.
- The **Add below...** menu option (displayed by right-clicking on an object) invokes a wizard which lists all the templates configured in Sodium, which generally represent the structural object class of the desired entry. Some templates have some related (optional) parts which can also be selected; these optional parts generally represent the auxiliary object classes of the desired entry and are shown when you click **Next**.



Sodium prompts for the value(s) to use as the name for the new entry.

Sodium can check for some data entry errors and will not allow you to save your new entry until you have corrected them: see [Section 3.4.1.1, “Highlighting errors”](#), for details.

If you need to enter information containing line breaks, see [Section 3.4.3, “Entering data containing line breaks”](#), for instructions.

You can undo changes: see [Section 3.4.1.2, “Undoing changes”](#), for details).

Note: If the entry you are adding is to be a context prefix, you must select the **Create as a context prefix** option: you cannot change an existing entry to make it a context prefix.

3.6 Deleting entries

You can either delete a single entry or a whole section of the Directory tree.

3.6.1 Deleting a single entry

To delete entries that do not contain any other entries:

1. Select the entry in either a **Search** or a **Browse** window.
2. Either right-click and select **Delete** or select **Operations** → **Delete** from the menu.

Note: You will be asked to confirm that you want to delete the selected entry. You cannot delete entries that have subordinate entries in this way. You would have to either delete all the subordinate entries or move them to another location first.

3.6.2 Deleting an entire subtree

To delete an entire subtree (an entry and all the entries it contains):

1. Select the entry in either a **Search** or a **Browse** window.
2. Either right-click and select **Bulk tools** → **Delete subtree** or select **Operations** → **Bulk tools** → **Delete subtree** from the menu.

A window is displayed enabling you to delete the selected entry and all its subordinate entries.

A warning message is shown, which also gives you the option of deleting the selected entry and any subordinate entries or just deleting its subordinate entries.

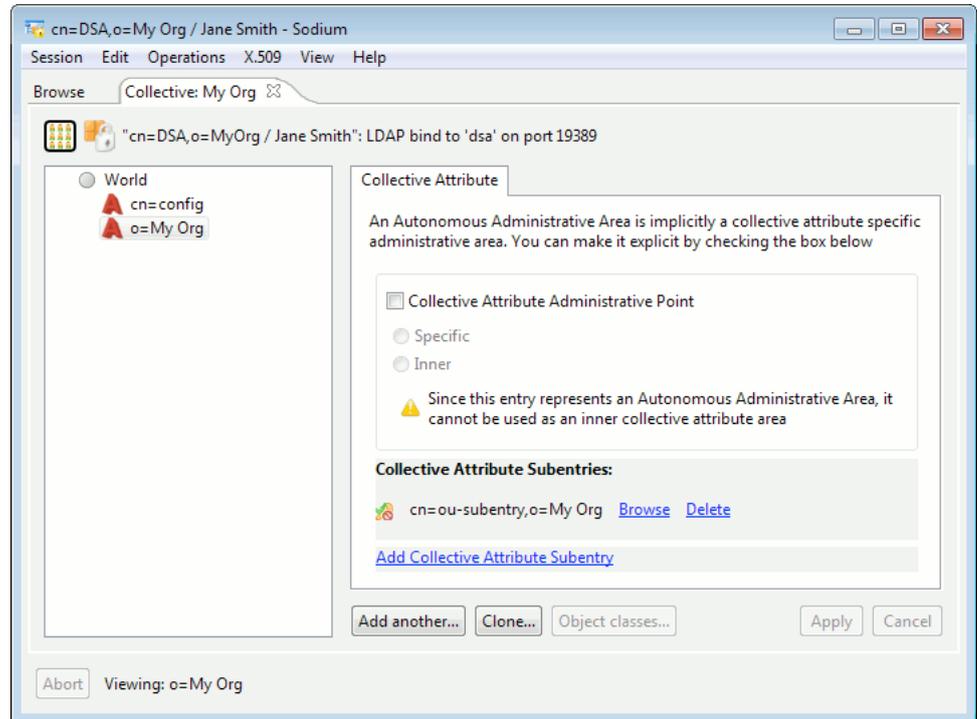
3.7 Collective Attributes

Sodium provides a Collective Attributes View which can be used to view or modify collective attributes. This view is enabled for any session where you have enabled Sodium with GUI Management Features enabled (see [Manager tab of Session settings](#)).

Collective Attributes View is associated with Administrative Points, and so the Collective Attributes View highlights existing Administrative Points in the DIT. An Administrative Point may have an arbitrary number of Collective Attribute subentries, each of which has a unique name. It is also possible to exclude collective attributes from appearing in a particular entry through use of the **collectiveExclusions** operational attribute using the

User View in Sodium. Such entries with exclusions are also highlighted in the Collective Attributes View.

Each Collective Attribute subentry has a *subtree specification*, which determines the scope of the collective attribute subentry within the Administrative Area, and user attributes of the subentry which define the collective attributes of the entry collection.



Existing Collective Attribute subentries are shown on the right-hand side of the window. Each existing subentry is identified by the subentry name, and you can view, modify, delete or add new Collective Attribute subentries by clicking on the appropriate options.

3.7.1 Finding a suitable Administrative Point

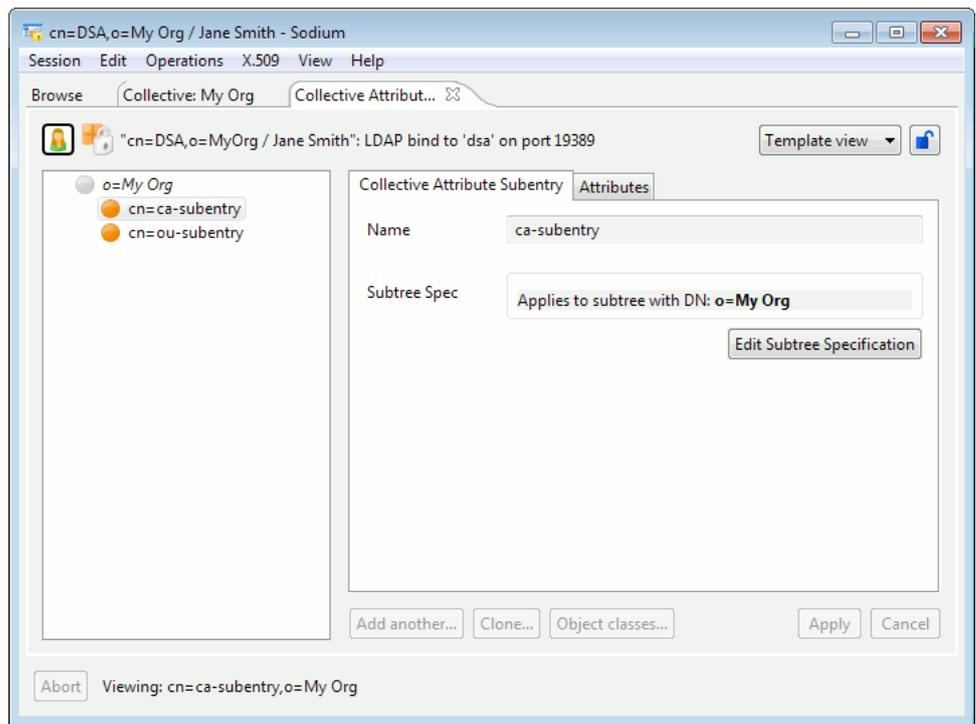
An Autonomous Administrative Point may define collective attributes without explicitly making it a Collective Attribute Administrative Point. It can be partitioned in order to deploy and administer collective attributes by defining Collective Attribute Specific and Inner areas. This can be done by selecting the **Collective Attribute Administrative Point** option. Note that a Specific Administrative Area defined for the purpose of collective attribute administration may be further divided into nested inner areas for the same purpose.

If you want to set up collective attributes at a location in the DIT which is not currently an Administrative Point, then you can select the appropriate entry in the DIT and use the **Create an Admin Point at this entry** option. Note though that you may be able to achieve a similar effect by creating a Collective Attribute subentry on an Administrative Point at a higher level in the DIT, and using a *subtree specification* to specify that it should affect only entries in a portion of the subtree.

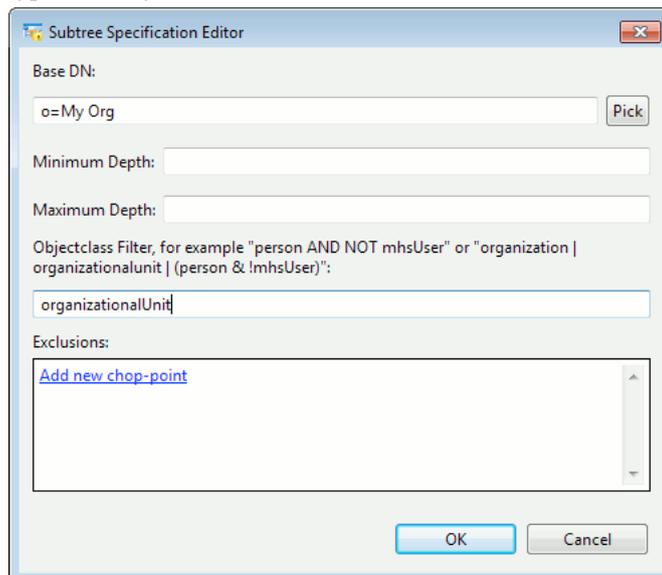
3.7.2 Managing Collective Attribute Subentries

Once a suitable Administrative Point has been selected (see [Section 3.7.1, “Finding a suitable Administrative Point”](#)), Collective Attribute subentries can be created.

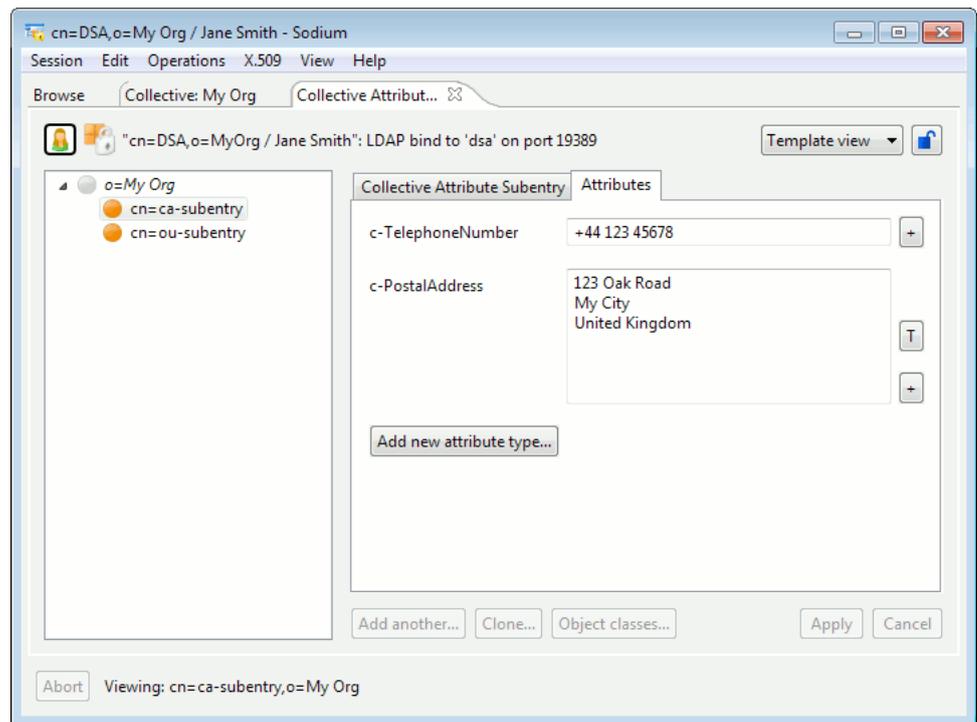
To create a new subentry, click on the **Add Collective Attribute Subentry** link. Once you have provided the name of the new subentry, Sodium will display a subentry view that allows you to configure its subtree specification and attributes.



By default, the subtree specification will be set to include *all* entries in in the subtree, but you are able to edit the subtree specification to restrict which parts of the subtree, or which types of entry in the subtree are affected.



The right-hand side of the window for the subentry view shows a list of collective attributes and their values. You can add new attributes and values; Sodium will check that that any new attributes you add are collective attributes (for example, you may add **c-TelexNumber**, but not **telexNumber**, since the latter is not a collective attribute).



Use **Apply** to commit changes and create the new collective attribute.

An existing Collective Attribute subentry may be modified by using the **Browse** link, which opens the subentry view where you can remove or modify existing attributes, add new ones, or edit the subtree specification.

3.8 Importing and exporting entries

Sodium provides a number of tools that operate on a potentially large number of Directory entries, that is, in bulk.

Note: This section only covers the use of Sodium to update or retrieve data over protocol from running DSAs. Other tools exist for importing and exporting data directly to and from the on-disk databases. Those tools are described in [Section 4.8.2, “Exporting and Importing Data”](#).

You can populate your Directory with data extracted from another system, or export data for backup purposes or to provide information to another system using these tools.

Note: The tools may result in large numbers of entries being sent to or from the Directory Server, and so may be constrained by server-defined limits. In this event, Sodium will attempt to complete the requested operation and will also log an appropriate warning message in a **Log** tab.

Sodium imports data from and saves data to the LDIF format.

3.8.1 LDIF files

This section gives you a brief overview of the format of LDIF files, so you can view and understand a file’s contents. For a detailed description of the format, consult *RFC 2849*.

3.8.1.1 Contents of an LDIF file

An LDIF file consists of a series of records separated by carriage return/linefeed, and a record consists of a sequence of lines describing a Directory entry. These may take one of the following forms:

```
attribute name:[ value]
attribute name::[ base64-encoded value]
attribute name:<[ url]
attribute name;binary::~<[ base64-encoded ASN.1-encoding]
```

Note: A space is required before the value. For details of the permitted format of LDAP attribute names, consult *RFC 4512*.

Comment lines must start with #, and lines which continue from the previous line (wrapped lines) must begin with a space. A line can be wrapped by inserting carriage return/linefeed.

3.8.1.2 Sample of an LDIF file

The example below is a simple LDIF file which adds two entries to the DIT. The entry for Adam Alexander is added at the level **o=Widget Ltd, c=GB** and the entry for Harry Hanson is added at the level **ou=Accounting, o=Widget Ltd, c=GB**.

This example starts with a comment line and includes a wrapped line in the second record.

```
#New entries for July 2007

dn: cn=Adam Alexander, o=Widget Ltd, c=GB
objectclass: top
objectclass: person
objectclass: organizationalPerson
objectclass: inetOrgPerson
cn: Adam Alexander
sn: Alexander
userid: aa
telephonenumber: 555-0442
rfc822Mailbox: aa@Widget.com

dn: cn=Harry Hanson, ou=Accounting, o=Widget Ltd, c=GB
objectclass: top
objectclass: person
objectclass: organizationalPerson
objectclass: inetOrgPerson
cn: Harry Hanson
sn: Hanson
userid: hh
telephonenumber: 444-6053
rfc822Mailbox: hh@Widget.com
description: Harry joined the LM project in July 1997. His
  manager is J. Johnson.
```

The following is an example of a line which contains a base64-encoded value.

```
description:: SGFycnkgam9pbmVkiHRoZSBMTSBwcm9qZWN0IGluIEp1bHkg
  MTK5Ny4gSGlzIG1hbmFnZXIgaXMgSi4gSm9obnNvbi4=
```

The following illustrate lines which include a URL to an external file. URLs in LDIF files are expected to be `file:` URLs without specifying a host.

External file references in LDIF files on Unix

```
jpegphoto:< file:///file/path/to/harry.jpg
```

External file references in LDIF files on Windows

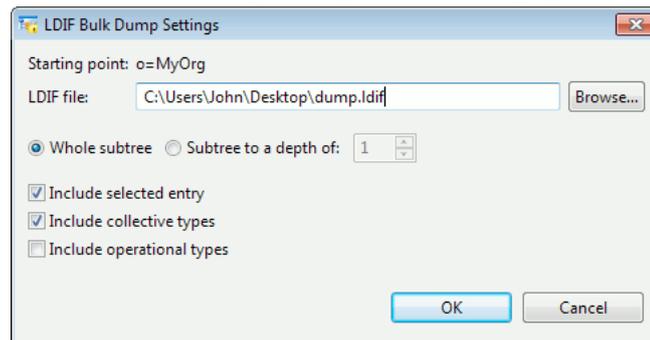
```
jpegphoto:< file:///Program Files/Isode/myphoto.jpg
```

3.8.2 Exporting entries to an LDIF file

Note: You should back up the data in the Directory regularly. A simple way of doing this is to export all of the data to LDIF files. More sophisticated backup techniques are described in [Section 4.8, “Backup and recovery procedures”](#). You must connect (bind) to the Directory Server as Data Manager in order to have the permissions required to backup the data, unless authority has been delegated.

1. In either a **Browse** or a **Search** page, select the entry in the DIT that contains all of the information you want to export.
2. Either:
 - Right-click object and select **Bulk tools** → **LDIF dump...** from the menu displayed
 - Select **Operations** → **Bulk Tools** → **LDIF dump...** from the menu.

The **LDIF Bulk Dump Settings** window is displayed.



3. Specify the location to save the LDIF file containing the information.
4. Choose whether to export the whole subtree or whether to limit it to a certain depth. If the **Subtree to a depth of** option is selected, Sodium will limit the dump to just immediate children of the selected entry (depth is 1) or grandchildren (depth is 2), and so on. Otherwise all subordinate entries are included.
 - If the Directory supports paged results, then there is no limit on the size of the subtree that may be dumped.
 - If the Directory does not support paged results, then the operation may be limited by administrative limits imposed by the Directory Server. A warning will be generated if an administrative limit has been hit.
5. Specify the types of entry to be included.
6. Click **OK**.

Caution: The resulting LDIF file may contain readable passwords. You need to ensure the LDIF file is given adequate file system protection.

3.8.3 Importing entries from an LDIF file

Entries must be in the correct format in the file – any errors will be displayed on a **Logs** page at the end of the process.

1. In either a **Browse** or a **Search** page, select the entry in the DIT that you want to contain the information you want to import.
2. Either:
 - Right-click object and select **Bulk tools** → **LDIF load...** from the menu displayed
 - Select **Operations** → **Bulk Tools** → **LDIF load...** from the menu.
3. Click **Browse...** and locate the LDIF file containing the entries you want to upload.

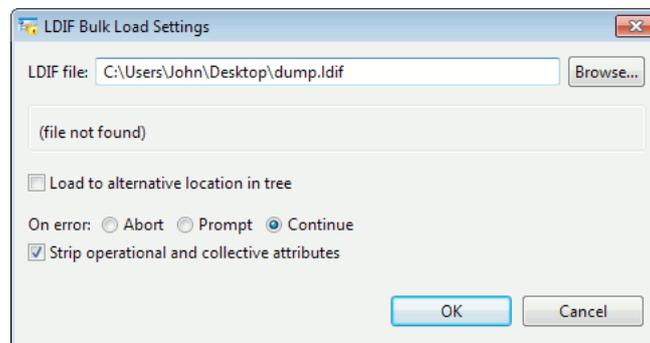
Examples of the DNs of the entries in the file are shown.

4. You have the option of modifying the structure of the subtree(s) containing the data being imported.
 - a. Select **Load to alternative location in tree** and choose to substitute some of the RDNs with alternative values.
 - b. Select the number of levels you want to change.
 - c. Type the new values.
 - d. Select **Correct loaded DN attribute values** to convert any values referencing the original DNs to the new DNs within the LDIF file.

The effect of your substitutions is shown immediately below this section.

Note: The area containing the fields for you to specify substitution values is not visible unless **Load to alternative location in tree** has been selected.

5. Decide how you want errors to be handled.
6. Click **OK** to begin importing the data.



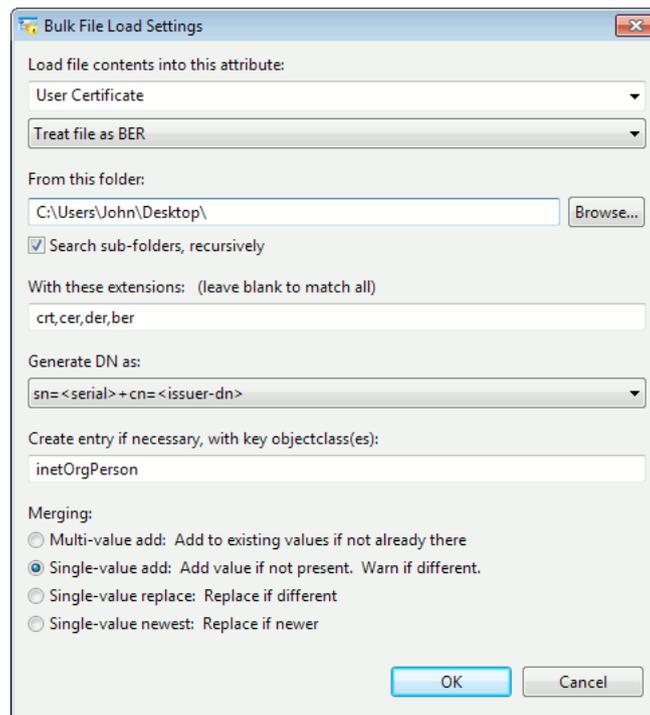
3.8.4 Loading files into specific attributes

You can load files into specific attributes within entries. These may be certificates, CRLs, JPEG images, text files, or any file you wish to load. For example, you may add new photos to a large number of existing person entries, or bulk-load certificates into the Directory, creating entries automatically if they do not already exist.

Sodium needs to be able to determine which entry each file should be associated with. For example, if you are uploading photographs of your staff, ensure the filenames match the common name (the identify value) of the person.

There is special support for extracting information from certificates and CRLs.

To specify the settings to be used, select **Operations** → **Bulk Tools** → **Bulk file load...** from the menu.



1. To specify which attribute will contain the file after import, either select it from the preset list or type the LDAP attribute name directly into the field.
2. Specify the type of data held in the file. This may be BER or binary data, or text in one of a number of different character sets. In the case of text, the characters will be converted to UTF-8, as used in the Directory.
3. In **From this folder**, browse to the folder containing the files. You can optionally choose to include sub-folders if the files are organized in that way.
4. List the file extensions of files that contain the data you want to upload.

If no file extensions are specified, then all files will be read, and any that contain a valid value for the selected attribute type will be loaded.

5. Specify how the DN of the entry whose attribute will be loaded with the data will be identified or created by selecting an option from the list. Choose a method that is appropriate for the type of data you are uploading.
6. If an entry with the generated DN does not exist, one will be created. Specify the object class to use for this newly-created entry.
7. Select the preferred merging behaviour to be used when there are already values for the attribute in the entry.

Single value newest is only applicable when loading certificates or CRLs. It determines whether the existing or imported version is newer by comparing issue issue dates on certificates or CRLs. It does not work for other types of data.

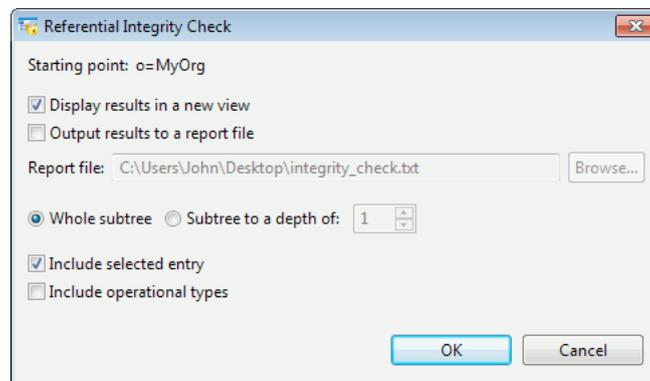
8. Click **OK** to upload the files.

3.9 Checking the referential integrity of attributes

Some attribute values may contain DNs, referencing another Directory entry. For example, an entry representing a person that is using the **Internet Organizational Person** template may have values for **Manager** and **Secretary**, which will be DNs of other entries in the Directory.

When entries are added individually, these DNs are selected from those available so are valid, but when entries are added in bulk these DNs are written directly to the attribute value. To check that DNs held in attribute values actually exist in the Directory:

1. In either a **Browse** or a **Search** page, select the entry in the DIT that is at the top of the subtree you want to check.
2. Either:
 - Right-click object and select **Bulk tools** → **Referential integrity...** from the menu displayed
 - Select **Operations** → **Bulk Tools** → **Referential integrity...** from the menu.
3. Specify how you want the results to be presented.
4. Specify how much of the Directory you want to check, starting from the selected entry.
5. Click **OK** to start the integrity check.



If the **Subtree to a depth of** button is selected, Sodium will limit the check to just immediate children of the selected entry (depth is 1) or grandchildren (depth is 2), etc. Otherwise all subordinate entries are checked.

3.10 Managing identities

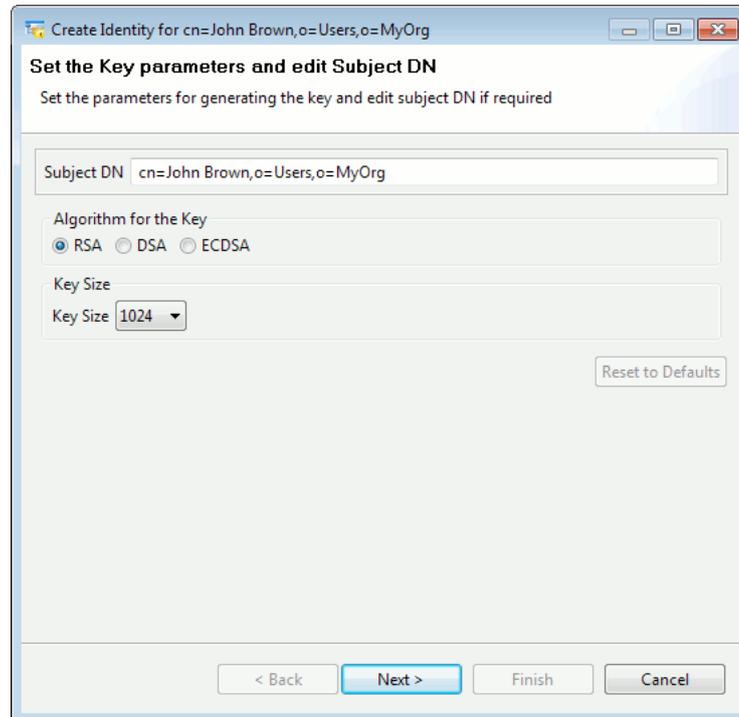
An X.509 certificate and the corresponding key are required to perform strong authentication using Sodium and other applications. This combination of certificate and private key is called an “identity” by Sodium. Identities are stored on disk in a standard file format called PKCS#12, which is encrypted.

3.10.1 Generating a certificate request

X.509 certificates are issued by a Certificate Authority (CA) in response to a Certificate Signing Request (CSR). Sodium provides a convenient way to generate CSRs and keys, and to create identities from certificates that are returned from a CA.

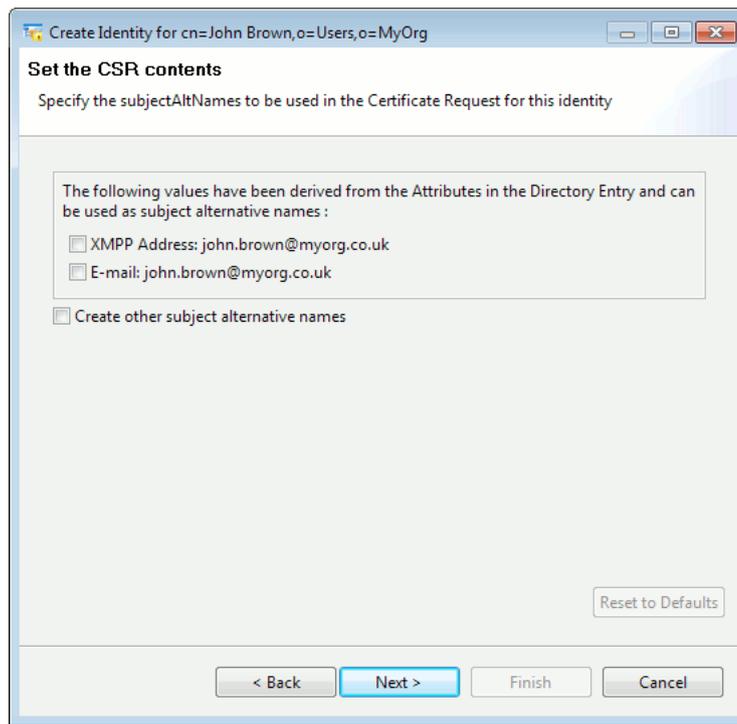
To generate a certificate signing request:

1. Select an entry in the tree view.
2. Select **X.509** → **New X.509 Identity** from the menu.



The wizard starts a **Create Identity** wizard. The wizard provides default values for the key parameters and CSR subject name, based on the selected entry. The subject name is derived from the selected Directory entry, although you can change this to any legal DN.

3. Depending on what attributes are found in the entry, various **subjectAltName** values may be suggested.

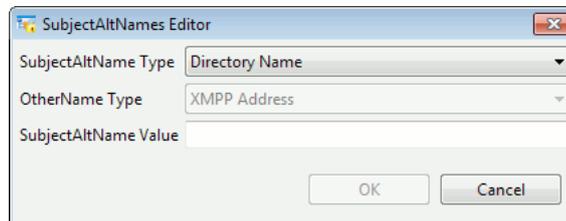


You may also add arbitrary **subjectAltName** values as appropriate.

- a. Select **Create other subjectAltNames**.

A new area is displayed, with **Add...**, **Edit...** and **Remove** buttons to the right of it. If there are no entries in the area, the **Edit...** and **Remove** buttons are disabled.

- b. Click **Add...**

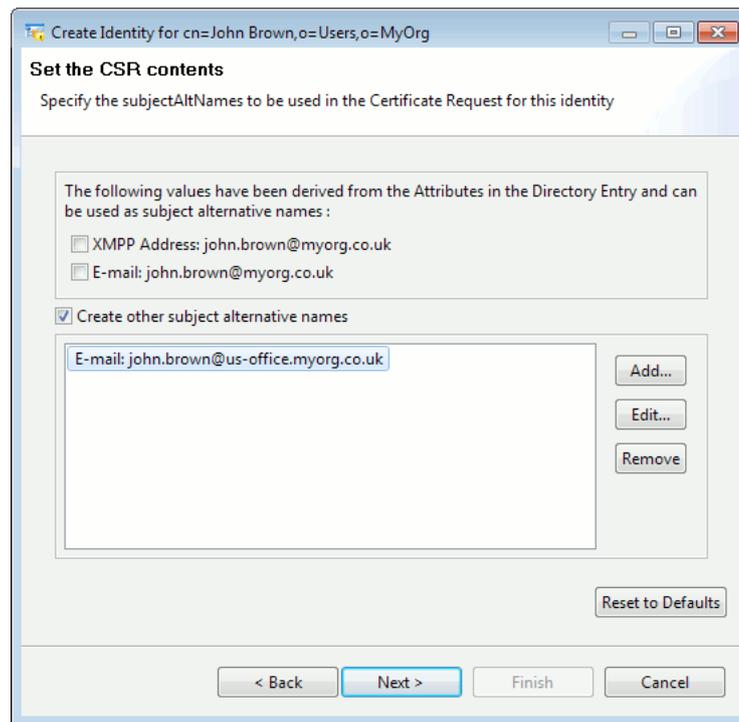


- c. Select the **SubjectAltName Type** you want to add from the list.

If you select **OtherName**, the **OtherName Type** list is enabled for you to choose which other name.

- d. Type a value for the chosen attribute. Click **OK**.

The **subjectAltName** you have created is now displayed and you can edit or remove it if necessary.



4. Click **Next**.

5. You are now shown the **Certificate Request Contents**. Click **Details** to make sure everything is as required.

Click **Next**.

6. The certificate request now needs to be passed to a Certificate Authority (CA) for signing. You have three options:

- Click **Desktop** to save the request as a file on your desktop, which can then be passed to a CA.
- Click **Save** to save the request as a PKCS#10 format file in a location of your choice. This is useful if you are using Isode Sodium CA to issue certificates and have created a default location for the CA to collect CSRs and return certificates (see [Section 13.3, “Issuing certificates”](#)).
- Click **Copy** to copy the request to the clipboard. This can then be pasted into an email message.

Click **Next**.

7. You now have two options:

- **The CA has provided a certificate:** choose this option if the CA is able to issue the certificate while you are waiting.

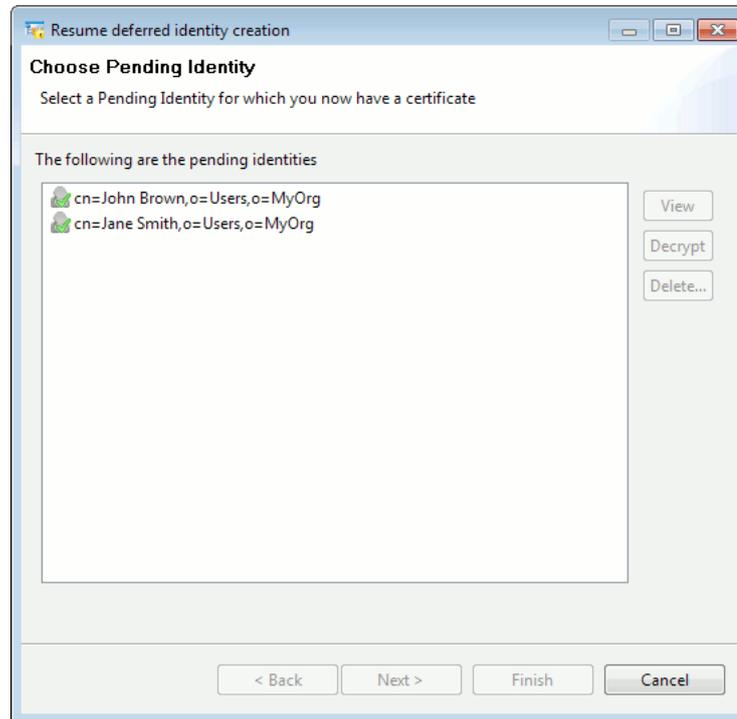
Click **Next**. The wizard will continue as described in [Section 3.10.3, “Linking a certificate to a Directory entry”](#).

- **I will complete this operation later:** choose this option to save the information you have recorded as a “deferred identity”. Deferred identity information is protected by your passphrase (see [Section 3.2.1, “Profile passphrase”](#)) and preserved between Sodium sessions. You will be able to link the certificate to the appropriate Directory entry when it is available.

Click **Finish** to close the wizard and complete the operation later (see [Section 3.10.2, “Continuing to create a deferred identity”](#)). Click **OK** to acknowledge the message.

3.10.2 Continuing to create a deferred identity

To collect certificates that were requested earlier, select **X.509** → **Deferred Identities** from the menu. Any outstanding certificate requests are shown.



Select the identity that is awaiting a certificate. The filename containing the certificate request and the date and time it was generated are shown at the bottom of the window. You can choose to **View** more details, or to click **Next** to continue.

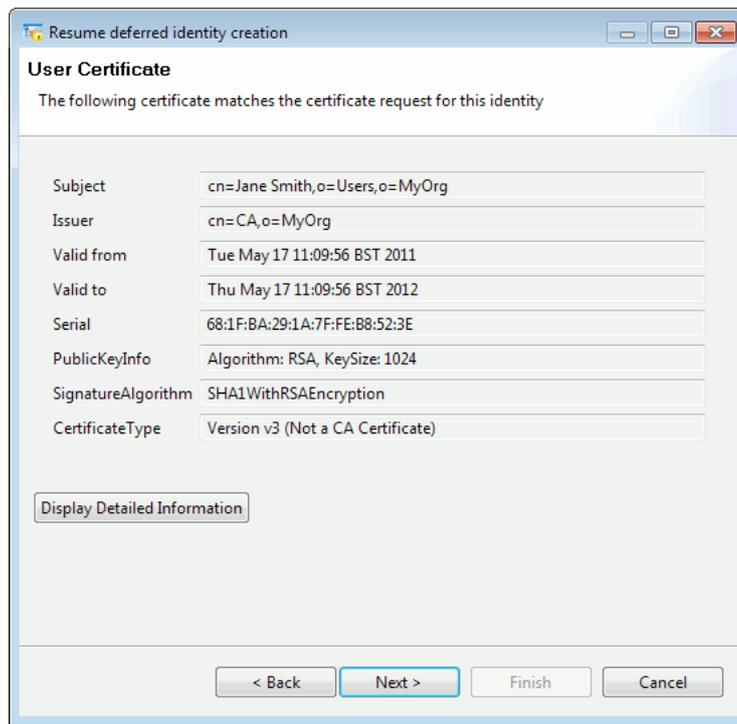
You will then begin the process of linking the generated certificate to the entry as described in [Section 3.10.3, “Linking a certificate to a Directory entry”](#).

3.10.3 Linking a certificate to a Directory entry

After either creating the certificate signing request or locating the deferred identity, you can link the generated certificate to an identity.

1. Sodium will look for the certificate on your desktop. Click the **Change** button to look in another location.

If a certificate is found that matches the request, Sodium will display details of it.



2. To complete the creation of an identity, two certificates are required: the one issued by the CA in response to the CSR, and the CA's own certificate. If the wizard is unable to find these certificates, then you will be prompted for their locations. If multiple (non-CA) certificates are found, you will be prompted to select which one you want to use.
 - If you are using Isode Sodium CA then its default behaviour is to write issued certificates (and a copy of its own certificate) to the same disk directory where it found the CSR.
 - If you are using a third-party CA, or a CA on a different system, you need to copy the certificates to a disk directory which you then direct Sodium to read.
3. Once both certificates have been found, the wizard will be ready to create the PKCS#12 file representing the identity. How this is done depends on who the identity is for.
 - If the identity is one that you (**the currently logged in user on this system**) are going to use yourself (for example, to bind using strong authentication to a DSA), then the PKCS#12 file will be saved in your *x509* directory, and protected using your passphrase.
 - If the identity is one which you are creating on behalf of another user, then you are prompted for the location and name of the PKCS#12 file to be created, as well as the passphrase to be used to encrypt it. The wizard will prompt you with a suggested passphrase, which will be displayed on the screen. You will need to give this to the user whose identity you are creating as it is needed to use the identity after you have sent or stored the PKCS#12 file.

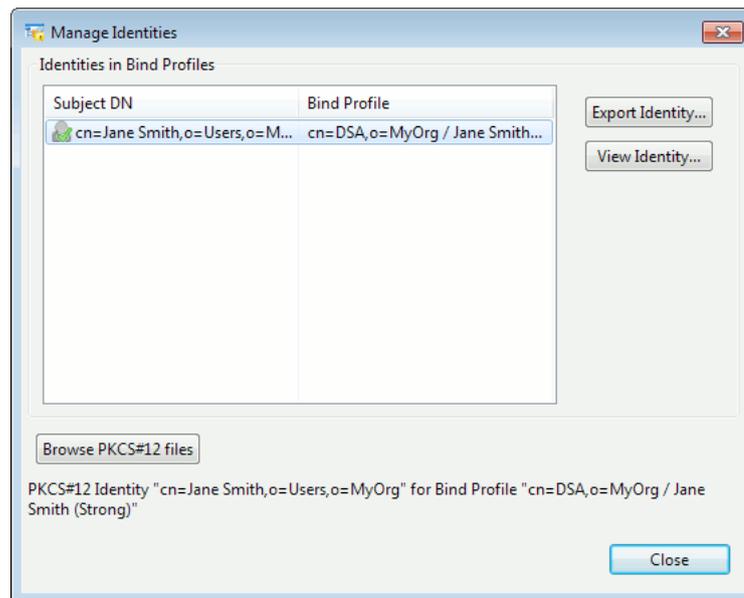
3.10.4 Creating an Identity for the local DSA

To create an identity for a Directory Server, you should use M-Vault Console. See [Section 5.4, "Configuring the Directory for X.509"](#).

3.10.5 Managing identities

Sodium allows you to view and delete identities by selecting **X.509** → **Manage Identities...** from the menu.

A list of identities associated with bind profiles is displayed, enabling you to view details of an entry or export it to a PKCS#12 file. When exporting an identity, you must provide a passphrase that will be used to encrypt the PKCS#12 file.



You can also **Browse PKCS#12 files** anywhere on the file system, which may be useful to examine files which you have previously exported, or to re-encrypt existing PKCS#12 files with a different passphrase.

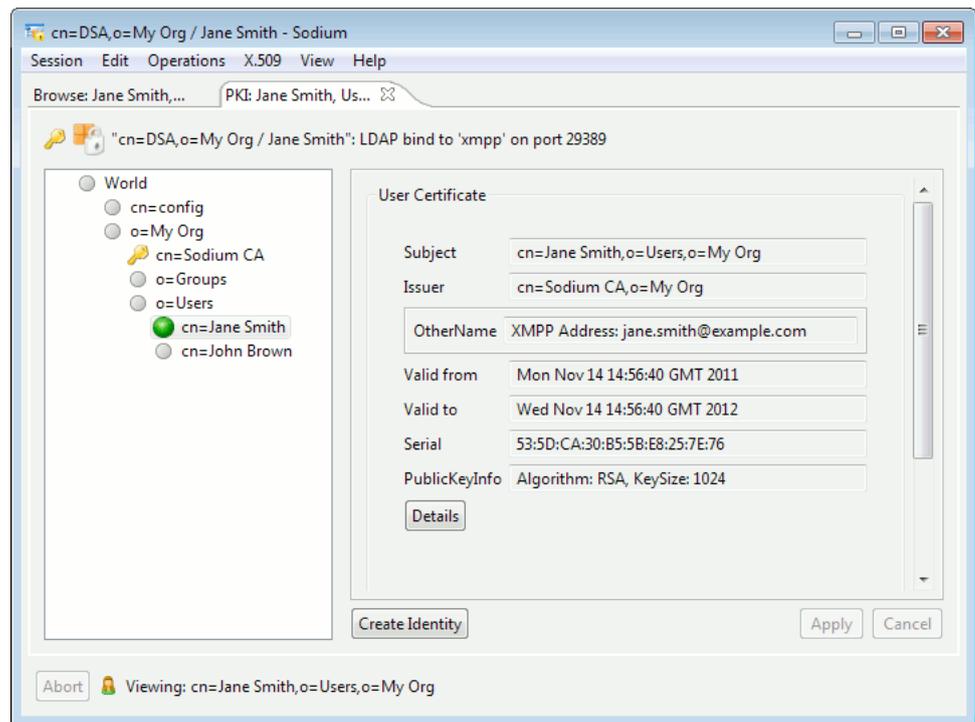
When browsing PKCS#12 files, you can view the contents of any file by providing its passphrase (unless it is protected by the current Sodium passphrase, in which case it will automatically be viewable). An icon is shown next to each file to indicate whether it is valid or invalid, or that it contains an expired certificate.

3.10.6 Managing PKI attributes

Sodium provides a special view that allows you to manage PKI (Public Key Infrastructure) attributes. This view is enabled for any session where you have enabled Sodium with GUI Management Features enabled (see [Manager tab of Session settings](#)).

The PKI view highlights the entries with certificates, or that represent Certification Authorities. The right-hand side of the window displays the PKI attributes (**userCertificate** and **userSMIMECertificate** for an entry; and **caCertificate**, **certificateRevocationList**, **authorityRevocationList** and **crossCertificatePair** for a CA entry). These editors provide the option for loading, saving and removing the PKI attributes.

Note that these editors and the corresponding functionality is also available in the User View of Sodium, but the PKI View makes it easy to manage the PKI information by allowing you quickly to locate and view just the relevant entries and attributes.



The **Create Identity** button starts a "Create Identity" wizard which facilitates generation of X.509 Identity for the selected user entry (see [Section 3.10.1, "Generating a certificate request"](#)).

3.11 Security Information Objects

You can use Sodium to display, create and edit Security Information Objects (SIO): Security labels and clearances, and the corresponding catalogs.

3.11.1 Setting up a Security Policy

The security policy is represented as an SDN.801c SPIF in the Open XML SPIF format. Sodium needs to know the name of the Directory entry containing the policy. This entry will also hold additional security information, such as catalogs of labels and clearances.

The default value for the entry is **cn=core,cn=config**, which is the M-Vault Directory Server's core configuration entry. Although using this entry allows an appropriately authorized Sodium user to display and edit security information used by the Directory Server, it is not generally accessible by users: you may therefore choose to use a different and more accessible, entry.

Note: Policy information for Directory applications, such as M-Link, will typically be stored in separately entries, such as the main M-Link configuration entry.

The name of the entry holding the SIO Information is configured on by either:

- Specifying it on the **Session settings** screen (on the Sodium page) when creating or modifying a bind profile

- Modifying the settings for the current session when bound to Sodium. Open **Session Settings** by selecting **Session** → **Session settings...**. The entry is specified on the **Sodium** page.

If you are not using the default entry, you will need to create an appropriate entry to hold the SIO information; for example, an M-Link Server entry is a suitable entry for holding SIO information associated with the M-Link Server. To create the entry, right-click a suitable entry in Sodium and choose **Add Below** from the menu displayed, then select **Isode M-Link Server** from the list of templates. Follow the steps in the wizard to create an M-Link Server whose DN can be used as the alternate entry for SIO objects.

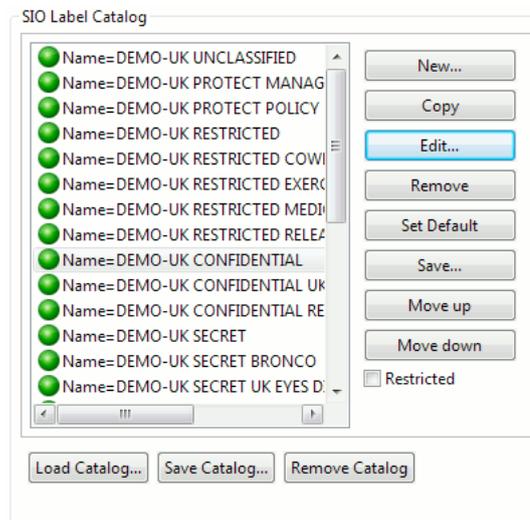
To configure the security policy, select the entry that is being used to hold the SIO information and click the **SIO** tab. Load the security policy XML file using the **Load** button. Isode provides sample security policies which are called *policy.xml*. These can be found in */opt/isode/share/security-label/example-data* (on Unix systems) and *C:\Program Files\Isode\share\security-label\example-data* (on Windows systems, by default).

Click **Apply** to save the security policy in the entry. You will now be able to create Security Labels and clearances. The above folders also contain sample clearance and label XML files.

Note: Until you have configured the security policy and clicked **Apply** to apply your changes to the entry, the **New** buttons are disabled.

3.11.2 Setting up catalogs

Security Label and Clearance catalogs are collections of security labels and clearances. Sodium enables you to create and manage catalogs using the catalog editors on the **SIO** tab. If a catalog is available as an XML file, it can be loaded using the **Load Catalog...** button. Once loaded the editor displays the catalog as a collection of labels or clearances, each of which can be displayed and edited.

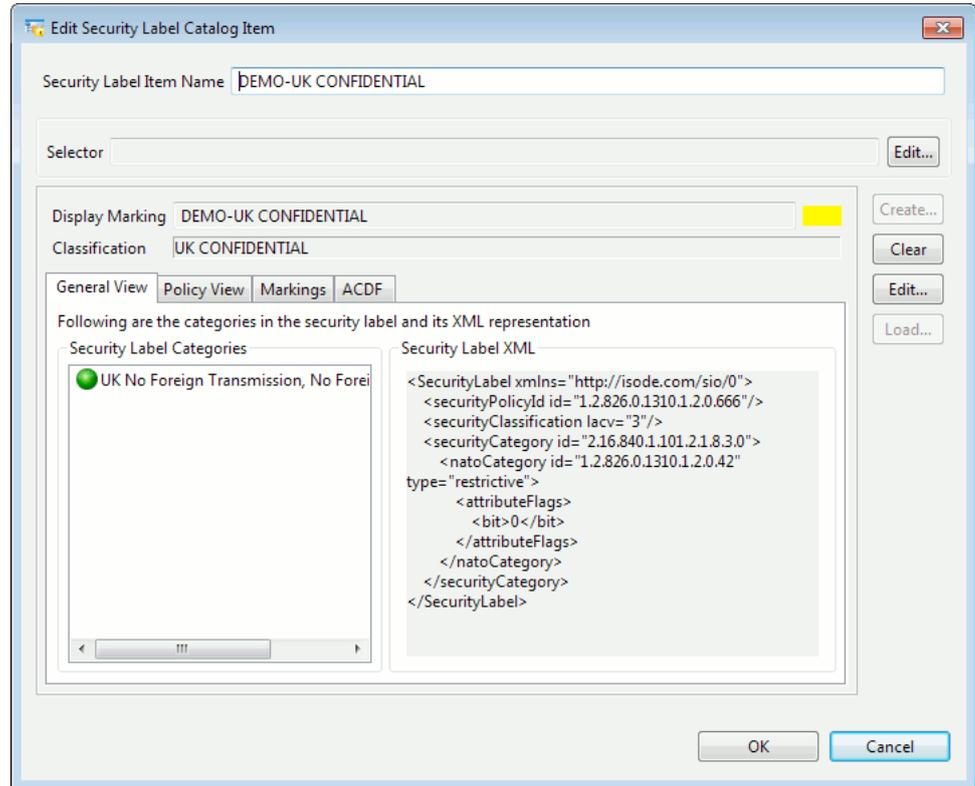


Icons to the left of each catalog name indicate its status:

- Labels and clearances issued under the configured policy are shown with green discs to the left of their names. If an issued label or clearance contains obsolete elements, a white cross is displayed inside the green disc.
- Labels and clearances that are issued under a different policy are displayed with an orange disc to the left of their names.
- Any invalid labels or clearances are marked with a yellow warning triangle.

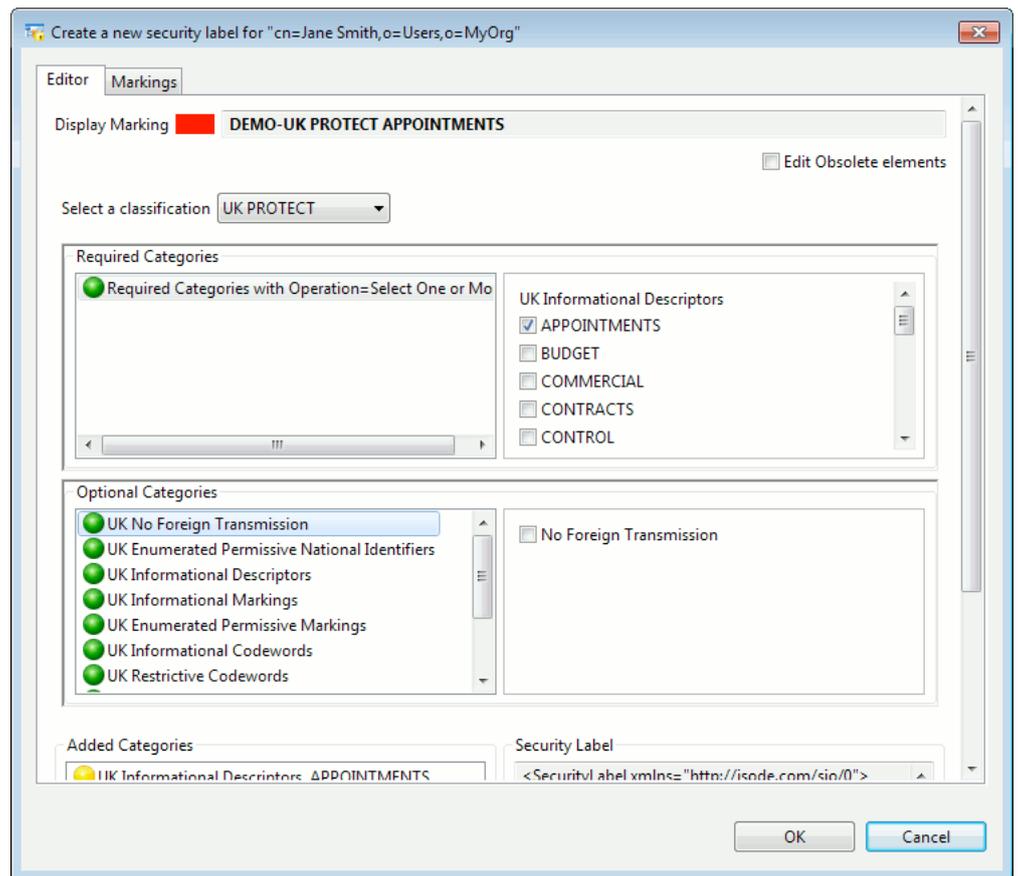
3.11.3 Editing catalogs

Once a label or clearance catalog is stored into a suitable entry, it can be edited. Click **New...** to add a new label or clearance to the catalog, or **Edit...** to edit an existing label or clearance in the catalog.



3.11.4 Applying a label to an entry

Once a security policy has been configured, a label can be applied to an entry. Labels are normally held in an operational attribute, so it will normally be necessary to modify the Sodium session settings to show operational attributes in the **Op-Attrs** tab. Click **Load...** to load a new label from a file (XML or BER formats), **New...** to create a new label using the configured security policy or **Catalog...** to select a label from the label catalog. The following editor will appear when either the **New...** or **Edit...** button is selected:



First select a classification from the drop-down list. Selecting a classification may or may not require inclusion of certain categories. The required categories if any will be displayed as a list in the **Required Categories** pane. The **Optional Categories** pane lists all the categories in the configured policy from which the user can select certain categories to be added to the label. The categories which are disallowed based on the selection of a certain category or the classification will be disabled automatically on the editor. The obsolete categories will be allowed for editing based on whether **Edit obsolete elements** is selected or not.

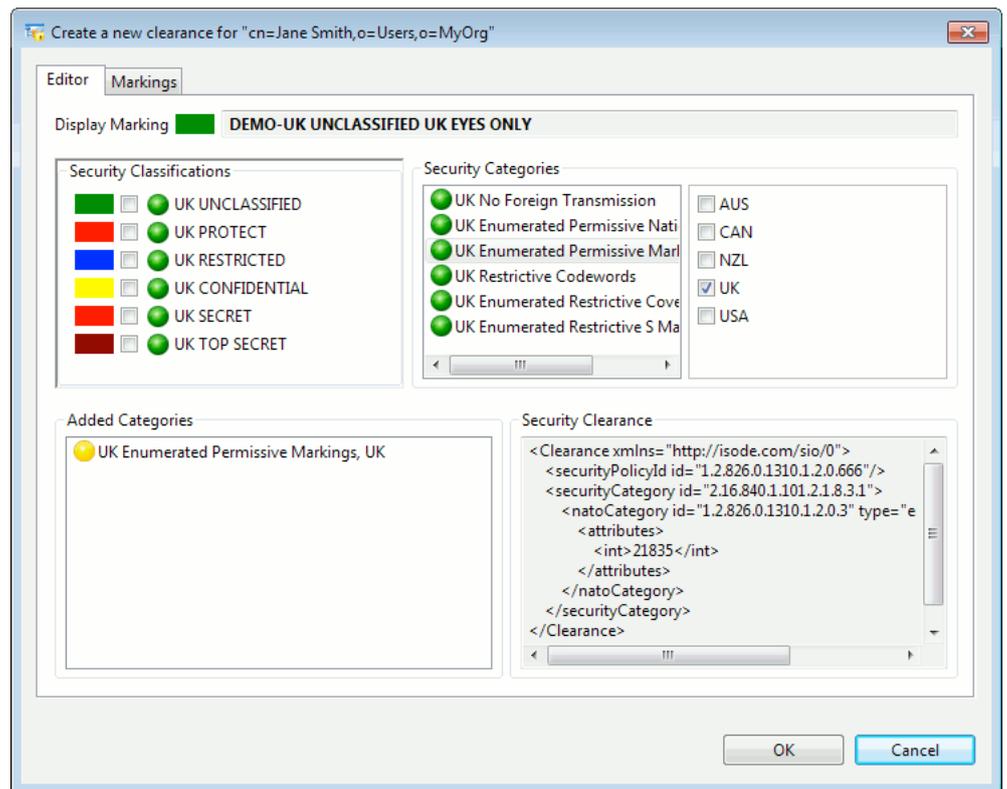
Selection rules of a category group determine whether it allows selection of single or multiple categories in the group. For single category selection, the categories are displayed using radio buttons and for multiple category selection they are displayed as check-boxes.

The markings get updated on the **Markings** tab when a valid combination of categories has been selected.

Note: Rules for label editing are based on SDN.801c and are configured in the security policy.

3.11.5 Applying a clearance to an entry

Once a security policy has been configured, a clearance can be applied to an entry. Clearances are normally held in an attribute that is part of the **sioClearance** auxiliary object class, so it will normally be necessary to add that object class to the entry using Sodium's **Object classes...** button. Then click **Load...** to load a new clearance from a file (XML or BER formats), **New...** to create a new clearance using the configured security policy or **Catalog...** to select a clearance from the clearance catalog. The following clearance editor will appear when either the **New...** or **Edit...** button is selected:



One or more classifications can be selected from the **Security Classifications** pane to be added to the clearance. The **Security Categories** pane lists all the categories in the configured policy from which the user can select certain categories to be added to the clearance. The markings get updated on the **Markings** tab as and when the clearance is edited.

3.12 Displaying warnings and errors

Sodium will display warnings and details of errors it finds on a special **Log** page (shown in the same area as the **Browse** and **Search** pages). If there are no warnings or errors to show, the **Log** tab is not be displayed.

Sodium will also append any warnings or errors to the configured log stream(s), which by default will include the file `(LOGDIR)/dua-event.timestamp.log`. See [Section 11.1, “Logging”](#) for details on configuring this.

3.13 Customizing Sodium

You can make changes to the way Sodium operates and displays information, and to the templates that Sodium uses when adding or modifying entries.

3.13.1 Changing settings

You determine how Sodium operates – how it searches for entries, whether or not some entries are displayed and the views that are available – as part of a bind profile.

You can modify any of these options on a temporary basis by editing them within a session. Select **Session** → **Session Settings** from the menu, make any changes, and click **OK**. The next time you start Sodium, your settings will be back to the way they were.

If you want your changes to become the new defaults, you have to make them to the bind profile. You can do this when starting Sodium or you can select **Session** → **Bind** → **Manager** from the menu, then select and modify the profile. The steps are the same as those shown in [Section 3.2.2, “Binding to the Directory using Sodium”](#).

3.13.2 Creating and modifying templates

When installed, Sodium provides all the templates that you will need for most Directory applications. You may decide that you need to modify or supplement these, especially if you have defined your own object types or attributes. Full instructions for configuring Sodium templates are provided in [Appendix D, *Customising Sodium*](#).

Chapter 4 System Management

This chapter explains how to use M-Vault Console to check and change the configuration of a Directory Service. It also covers standard operational tasks and performance tuning.

Note: If you prefer, you can use the Dmish scripting interface to carry out the tasks described in this chapter (see [Appendix H, *Dmish Scripting Interface*](#)). This may be useful if you have a lot of repetitive work to do. However, if you are new to the M-Vault Server you are advised to read this chapter first, as it gives you general advice on maintaining the Directory Service and refers at specific points to other relevant chapters.

4.1 Starting M-Vault Console

To carry out any of the changes, you must first:

1. Log in to the local Directory Server account.
2. Start M-Vault Console.
3. Connect to the Directory Server to which the changes should be applied, as described in [Section 4.3, “Opening a management connection”](#), giving the Server Manager’s name and password.

Note: If the Directory Server is not already started, you can start it using M-Vault Console (see below).

4.1.1 M-Vault Console’s use of Bind Profiles

M-Vault Console maintains a set of Bind Profiles (see [Section 2.1.3, “Using bind profiles”](#)) corresponding to all the DSAs that it knows about. These fall into two categories:

- **Managed DSAs** - these are servers which M-Vault Console is involved in managing, and knows how to connect to.

A bind profile for a managed DSA will contain its address as well as information about on how to connect and authenticate to it. Bind profiles for managed DSAs are visible in Sodium’s list of bind profiles because Sodium can use the same information to connect to the DSA.

- **Known DSAs** - these are servers which M-Vault Console doesn’t manage but needs to know about because they are referred to by one or more managed servers.

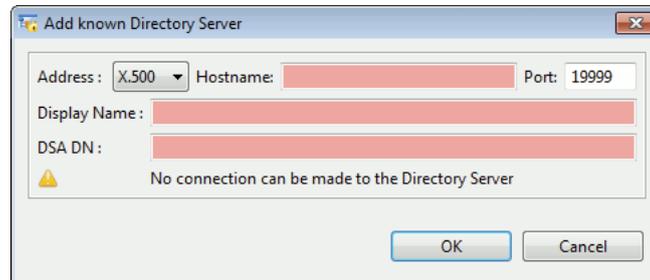
A bind profile for a known DSA contains the DSA’s address, but does not contain information on how to connect and authenticate to it. Known DSAs are not visible when running Sodium.

M-Vault Console always assumes that you are managing at least one DSA, so you must encrypt your Bind Profile file (because the bind profiles for managed DSAs contain authentication information). This means that whenever you run M-Vault Console, you have to give the passphrase you used to encrypt the Bind Profile file.

When performing tasks that require you to specify the address of a Directory Server, M-Vault Console displays a drop-down box that lets you choose from a list of all bind profiles (both managed and known) that might be appropriate for the operation in question.

For example, when creating a new shadowing agreement, you will be asked to choose which DSA the shadow agreement is with.

For situations where there is no bind profile for the DSA in question, choosing the option <**Other directory server...**> opens the **Add known Directory Server** dialog box, which allows you to create a new “known” bind profile.



4.2 Starting the Directory Server

4.2.1 Platform Specific Service Management

M-Vault services are managed using systemd on Linux and the platform system service management tools on Windows. See [Appendix F, Running as an OS Service](#) for details.

4.2.2 Isode Management Tools

You can start or stop a local Directory Server, that is a Directory Server which is resident on the same host as M-Vault Console, using:

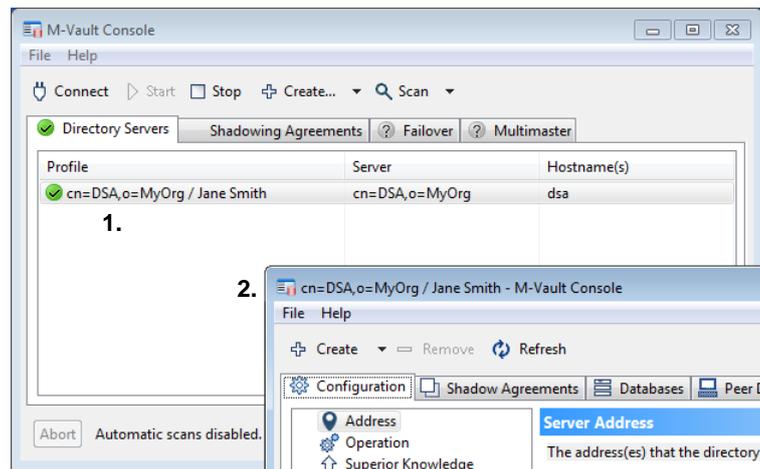
- M-Vault Console – select the server you want to start and click the **Start** (or **Stop**) button on the toolbar.
- The Isode Services Manager (Windows only).

From the **Start** menu, open the **Isode** program group and select **Isode Services Manager**. You will have to run this option as an administrator. Select the server you want to start and click **Start**.

4.3 Opening a management connection

Before you can perform operations (management or otherwise) on a Directory Server, you must connect to it.

1. From the **M-Vault Console** window, select the profile name associated with the server you want to manage and click **Connect**.
2. The <**Profile Name**> – **M-Vault Console** window opens (see [Figure 4.1, “The M-Vault Console Managing window”](#)), showing the name of the profile in the title bar.



Note: To close the connection to a Directory Server, you select **Close** from the **File** menu.

4.3.1 Creating a new management connection

If the Directory Server you want exists but is not listed, you can create a profile for it.

1. Click **Create...** and then select **New Bind Profile** from the menu. The **Bind Profile Details** window opens and a wizard will guide you through creating a new bind profile.
2. Choose whether this a **Managed server** (you want to be able to configure it using M-Vault Console) or a **Known server** (your Directory Server needs to connect to it, but you do not manage it).

Click **Next**.

3. As you work through the wizard, you need to specify:
 - The **Hostname** of the Directory Server

The default ports used for X.500 (DAP) and LDAP connections are shown – these can be edited if necessary.

If you need to specify the full presentation address, see [Section 2.2.3.3, “Specifying a presentation address”](#).

- The **Directory Server DN** (see [Section 2.2.3, “Creating a Directory Server”](#))
 - A **Display Name** to enable you to recognise this profile.
4. Click **Finish**.

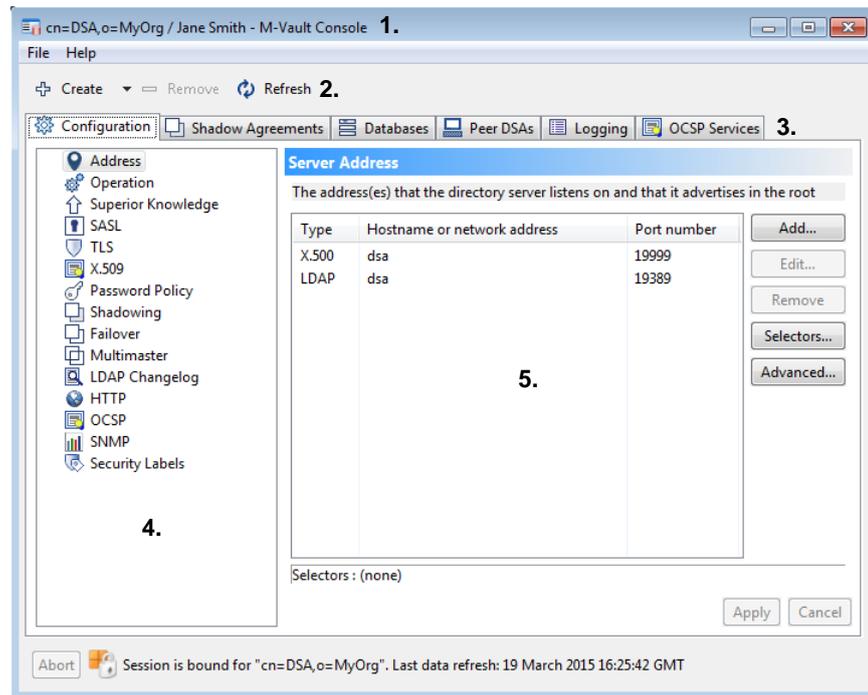
4.4 Overview of M-Vault Console

Note: This section contains a general overview of M-Vault Console. Instructions for carrying out specific tasks using it are given in the appropriate sections.

The **<Profile Name> – M-Vault Console** window opens when you connect to a Directory Server. The full name of the window includes the name of the profile; for example, if the profile is called “MyProfile”, then the window will be called **MyProfile - M-Vault Console**.

A typical view of M-Vault Console is shown below. The numbers are used in the explanation that follows.

Figure 4.1. The M-Vault Console Managing window



1. The title bar at the top of the window tells you which profile is currently being managed.
2. The three options in the tool bar are:
 - **Create** - this contains six options, each of which launches a wizard or opens a box to create the appropriate item:
 - **Supplier Agreement** – see [Chapter 8, Shadowing](#).
 - **Consumer Agreement** – see [Chapter 8, Shadowing](#).
 - **Failover Configuration** - see [Chapter 9, High Availability](#).
 - **Failover Mirror** - see [Chapter 9, High Availability](#).
 - **Database** – see [Section 4.6, “Database configuration”](#).
 - **Peer Configuration** – see [Section 7.4, “Securing connections between Directory Servers”](#).
 - **Log Stream** – see [Chapter 11, Monitoring the Directory](#).
 - **Remove** - enables you to delete a configuration item that is no longer required.
 - **Refresh** – repopulates the pages with the most recent details from the Directory Server.
3. The main areas for which configuration information is displayed, corresponding to the **Create** options listed above plus **Configuration** (general configuration of this Directory Server).
4. The individual configuration items in the selected area.
5. Details of a selected individual configuration item. Depending on the item selected, this area may contain a set of tabbed pages.

4.5 General configuration of the Directory Server

The **Configuration** page holds general configuration information about the Directory. The only configuration items described in this section are **Address** and **Operation**. All others are described in relevant chapters as listed below:

- **Superior Knowledge** is described in [Chapter 7, *Connecting Directories*](#).
- **SASL, TLS, X.509** and **Password Policy** are described in [Chapter 5, *Authentication*](#).
- **Shadowing** is described in [Chapter 8, *Shadowing*](#).
- **Failover** is described in [Chapter 9, *High Availability*](#).

4.5.1 Changing address information

Current address information for the Directory Server is displayed on a page very similar to that used when creating a Directory Server or creating a bind profile to connect to one.

The hostname or IP address of the server holding the Directory is shown, along with the port numbers it is listening on.

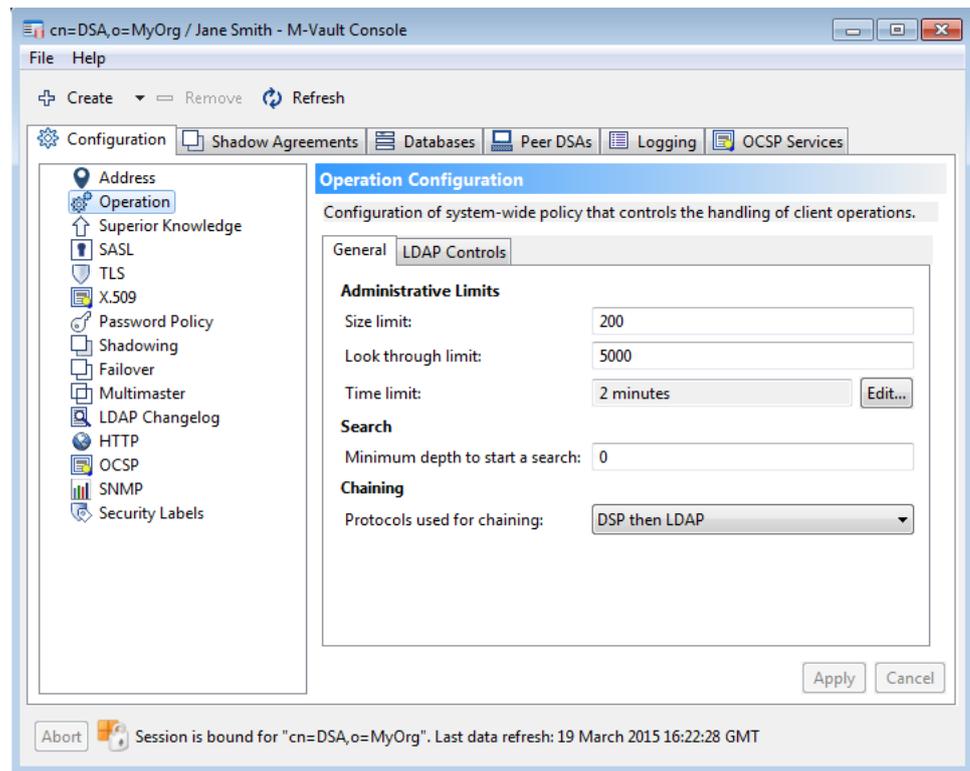
- Existing connection details can be changed by selecting them and clicking **Edit...** or by double-clicking them.
- Connection details for other protocols can be added by clicking **Add...** and completing relevant details.
- Unwanted connection details can be removed by selecting them and clicking **Remove**.
- **Presentation, Session** and **Transport** selectors can be specified by selecting an address type and clicking **Selectors...** (see [Section 2.2.3.3, “Specifying a presentation address”](#) for details).
- **Advanced...** enables you to view or edit the presentation address as a string.

Caution: Changing this information is changing the details of the actual Directory Server. Modifying port numbers, for example, will mean that any bind profiles using previous numbers will no longer function.

Note: Changes made to the Directory Server’s address will only take effect after the Directory Server has been stopped and re-started.

4.5.2 Operation configuration

The configuration in this section defines some of the operational parameters for the Directory Service.



The following options can be viewed and changed using this page:

- **Administrative limits**
 - **Size limit:** the maximum number of entries to return in response to a **List** or a **Search** request, if the requesting Directory does not specify a limit.
 - **Look through limit:** the maximum number of entries to be considered for **List** or **Search** requests. Use this to balance time against how comprehensive a search should be.
 - **Time limit:** the maximum elapsed time (in seconds) within which the results of a **List** or a **Search** request must be returned. Set this to a value that will prevent an unreasonable amount of time being spent on one operation.
- **Search**

The minimum depth at which a search may be started defaults to 0 (the root of the Directory). It may be set to a lower level if you know that DUAs are unlikely to require information from the higher levels: this will save search time.

- **Chaining**

You can prohibit chaining at all (**No chaining**), can specify that it is only to be carried out using either the LDAP or DSP protocols, or that either protocol can be used but there is a preferred order.

4.6 Database configuration

Entries held by the server are stored in one or more databases. Databases are also referred to as GDAMs, where GDAM stands for Generic Database Access Module. A Directory instance can contain entries in more than one naming context. Multiple locally mastered

naming contexts can be stored in a single database. However, this is not true of shadowing where a shadowed naming context must make exclusive use of the storing database.

M-Vault's current implementation is **IMGDAM**, an in-memory approach where all directory information is loaded into memory at start up. The information consists of a set of snapshots of the data, together with a log of changes made. The server can create new snapshots at configurable times (periodically, when a certain number of changes have been made and/or at server shutdown time).

The **Databases** page shows details of all databases currently configured for this Directory Server. A list of databases is shown on the left side of the page, and the configuration information for the selected database is shown on the right.

4.6.1 Creating a database

To create a new database:

1. Click **Create** and select **Database** from the menu displayed.
2. Give the database a name and then click **OK**.

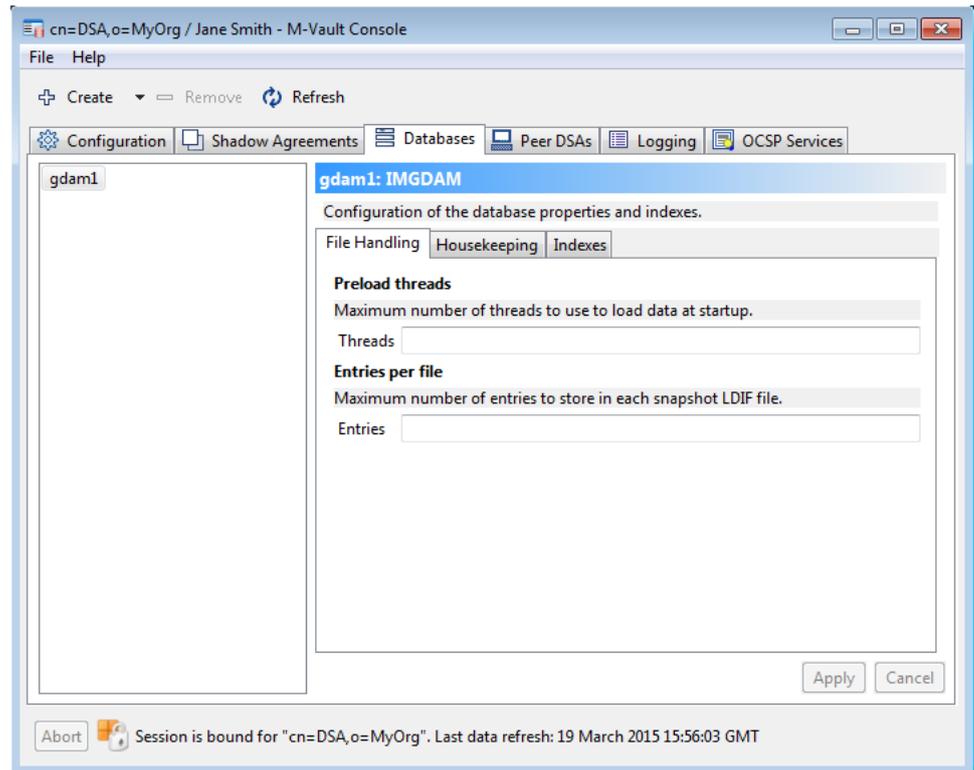
You can subsequently modify the database configuration, and the options available are described next.

4.6.2 Configuring the in-memory database (IMGDAM)

This GDAM holds all data in main memory. Data is persisted on disk by holding a number of recent snapshots of the data and a set of transaction logs. Snapshots consists of a number of data files, each comprising some number of directory entries. At startup each data file is loaded to build up the full in-memory representation. The changes held in the transaction log are then applied to the in-memory snapshot to bring the data up to date. Due to this there may be some delay between the server being invoked and becoming available, though there are configuration options that permit the manager to optimize initial loading for the system in use.

Configuration of this database type is held on three sub-pages: **File handling**, **Housekeeping** and **Indexes**.

[Figure 4.2, "Configuring the in-memory database"](#) shows the in-memory database configuration screen.

Figure 4.2. Configuring the in-memory database

- The **File handling** page is used to specify how data is stored in a snapshot and also how it is loaded at startup.
 - **Preload threads** enables you to specify the number of threads used to perform data loading at startup. If the full data consists of multiple data file then parallelisation will reduce the time it takes the server to load the data to memory and so the time it takes to come online. If this value is not set the server will use a number corresponding to the number of online CPUs.
 - **Entries per file** option specifies the number of entries stored in each data file. The fewer entries stored in each file the greater the scope for initial load parallelisation.
- The **Housekeeping** is used to specify how often database snapshots are written to disk. The process of creating new snapshots is known as checkpointing. Checkpointing can be configured to take place periodically, at specific times or when the server is shutting down. Note that if a checkpoint interval is provided then the specific schedule is ignored. Checkpointing is explained in more detail in [Section 4.6.3, “Checkpointing”](#)
- The **Indexes** page lets you manage the way that the data is indexed. By default, a single index is created on the **Attribute Type** of **mail** (email address). Indexing is explained in more detail in [Section 4.6.4, “Database indexes”](#).

4.6.3 Checkpointing

Careful consideration should be given to how often snapshots will be generated (checkpointing). Important factors are:

- The checkpoint operation places additional load on the server and this can be significant if the number of entries stored in the directory is large and/or the data is subject to large numbers of changes. Thus where server performance is a priority checkpoints should be configured to take place at times of least load.
- Large numbers of outstanding changes will cause a delay in server restart, as the DSA has to process all changes before it can start serving the up-to-date data. Checkpointing more often will alleviate this.

The default configuration is for changes to be incorporated into a new snapshot at 1 a.m. local time. Backup scheduling should also take the regularity of snapshot generation into account, i.e. it would make sense to back the server up soon after a new snapshot has been generated.

4.6.4 Database indexes

The Directory uses indexes managed by M-Vault Console (or Dmish – see [the section called “Adding one or more index objects”](#)) to make single-level and subtree searches faster. An index holds a list of entries which match each possible value for a given attribute and search type. Without indexes, for each search request the Directory Server needs to check each entry in the search scope to determine whether it matches the search criteria.

The value of indexes should be weighed against the cost of maintaining them. In the current in-memory GDAM implementation all indexes must also be held in memory, and so there is a trade off between the additional search speed that an index might provide against the overall process size.

4.6.4.1 Index search types

You can configure one or more attributes to be indexed for equality, approximate and substring matches. The type of attribute index and the algorithms used to search the database restrict the filters that can be used to make the search operation more efficient. Using non-indexed filters means that a search has to look through all of the entries.

- **Equality** – these indexes match the unique values of an attribute to the identifier of every entry with that particular value stored in that attribute.

For example, if the database contains four entries with the following values in the **surname** attribute: Emmit, Smith, Smith and Jones, then the corresponding index file will contain Emmit, Smith and Jones, with the Smith index entry mapping to both Directory entries with an attribute value of Smith.

- **Substring** – these indexes match substrings to the identifier of entries where the value of the specified attribute contains that substring. The files named with *attr.sub*. (e.g. *sn.sub.db*) contain mappings for substrings of all values of that attribute used in entries in the GDAM onto the entry identifiers of the entries which hold those values.

This index is only created if the attribute being indexed has a string syntax (**CaseIgnoreString**, **IA5String** and so on – see [Appendix C, Attribute Syntaxes](#), for more details).

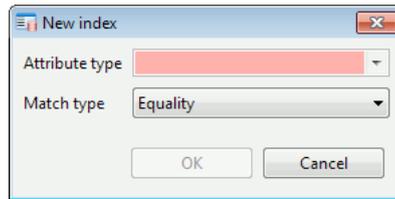
Initial and final substrings are two characters long; all others are 3 characters long.

- **Approximate** – these indexes contain unique Soundex mappings of the values of the specified attribute and match them to the entry identifiers of entries whose values match the same Soundex code.
- **Presence** – these indexes map any values for an attribute. The files named with *attr.pres*. (e.g. *sn.pres.db*) contain mappings for any values of that attribute used in entries in the GDAM onto the entry identifiers of the entries which hold those attributes

4.6.4.2 Adding an index

To add an index:

1. If you are not already connected, connect to the appropriate Directory Server.
2. Click the **Databases** tab.
3. Select the database to be indexed, then click the **Indexes** tab.
4. Click **Add**. The **New index** window opens.



5. Enter the **Attribute type** that you want to index.
6. Select the **Match type** you want to use (see [Section 4.6.1, “Creating a database”](#)).
7. Click **OK**.

Note: The index must be built before it can be used. To build the indexes for the database, click **Build indexes**.

4.7 Managing configuration files

The operation of the Directory Server is affected in part by a number of configuration files. This section describes how to maintain these files in a way that enables simpler upgrades to later releases.

The Directory Server reads in a number of configuration files when starting up. This means that for changes in these files to take effect, the Directory Server must be stopped and restarted.

1. The Directory Server initially attempts to read each file from the (*ETCDIR*) directory.
2. If a file is not found in (*ETCDIR*) the Directory server next looks in the (*SHAREDIR*) for the file.
3. If the file is still not found, the Directory Server will use built-in defaults if appropriate.

This use of two directories permits multiple machines to share common configurations by sharing the (*SHAREDIR*) directory, and yet to have local configuration files for each machine in its (*ETCDIR*) directory. For example:

- A licence file is local to the machine so is in (*ETCDIR*).
- Schema shared between machines can be stored in (*SHAREDIR*).

To avoid problems with M-Vault upgrades overwriting configuration file changes, no actual configuration files are installed. Instead sample files are installed, which you can copy and rename to replace the actual configuration file in the appropriate directory. The sample files all have a *.sample* extension.

4.8 Backup and recovery procedures

A Directory Server may have one or more database GDAMs. Each of these is held in a subdirectory of the filestore directory (configuration path) which was specified when the Directory Server was created (see [Section 2.2.3, “Creating a Directory Server”](#), for how this is specified in M-Vault Console and [Section E.4.1, “GDAM Files”](#), for details of the

files). These database GDAMs should be backed up for security purposes and so that recovery can be carried out if there is a problem with the Directory.

4.8.1 Backup and recovery of the in-memory GDAM database

The in-memory database produces snapshots of the data at times configured by the server manager and the server will retain a number of the most recent snapshots generated. The backup strategy is to copy the most recent snapshot of the data as well as the index configuration, which is read prior to the user data at startup. Recovery involves rebuilding the GDAM from the backed up snapshot(s) and configuration files.

4.8.1.1 Backup procedure

Snapshots are held in sub-directories of the *snapshots* directory and are identified by a 64-bit change sequence number. The snapshot directories are named using the hexadecimal encoding of that number, e.g. *000000000006d78*. The most recent snapshot is the one identified by the greatest change sequence number (the filesystem timestamps on the snapshot directories are also a place to look).

The user data backup procedure is to copy at least the latest snapshot to the backup area. The simplest strategy is just to copy all available snapshots by performing a recursive copy of the *snapshots* sub-directory to the backup area.

It is also necessary to backup the index configuration. This is held in the *config* sub-directory of the GDAM directory. Again this should be backed up by performing a recursive copy of the sub-directory to the backup area.

4.8.1.2 Recovery procedure

First recreate the GDAM directory structure, which is the following directories:

- *gdam-dir*.
- *gdam-dir/config*.
- *gdam-dir/snapshots*.
- *gdam-dir/changelog*.

The contents of the *snapshots* and *config* sub-directories should be copied from the backup area to the reconstructed GDAM tree. The server can then be restarted.

4.8.2 Exporting and Importing Data

Sometimes it may be necessary to make copies of or load data into the directory using the LDIF exchange format (see *RFC 2849*). The following tools exist for importing and/or importing LDIF data:

- *dbulk* - Dump or load LDIF data to a database. Single entries or subtrees can be dumped. The tool cannot currently process LDIF change records.
- *dsnapdump* - Dump LDIF data from a database snapshot (as found in the *<gdam-dir>/snapshots* directory).
- *dlogdump* - Dump the GDAM database changelog. The database changelog consists of the recent changes made to the database. The *dlogdump* tool can dump the sequence of changes to a file using an Isode specific format or as a sequence of LDIF change records.

4.8.2.1 Using dbulk to Import and Export Data

The direct to disk bulk data handling facility (**dbulk**) has three operating modes:

- **load** Load entries from an LDIF file into a Directory Server's database.

- `dump` - Dump a portion of DSA database into an LDIF file.
- `clean` - Remove a subtree from a DSA database.

Caution: Note that **dbulk** modifying modes (`load` and `clean`) - must not be used while the DSA is running. The exporting mode (`dump`) can be used while the server is running.

General limitations to note:

- Any LDIF data used as a source of bulk loading must conform to the schema as `dbulk` does not check for schema errors.
- The database GDAM must already exist before bulk loading can be carried out.
- There is no support for file URLs.
- All RDN values are assumed to be present as attributes in LDIF records.
- Only the binary option is supported in LDAP attribute descriptions.

Logging, including errors, goes to the `dbulk-event.log` log file.

4.8.2.1.1 Using dbulk load

The `load` option loads LDIF content (noting that change records aren't currently supported) into the database. To load data enter the following command:

```
dbulk load -db_directory pathname -ldif filename -user DN
[-maxfail n] [-skip n] [-count n] [-overwrite]
```

The meanings of the parameters are as follows:

`-db_directory pathname`

This gives the location of the GDAM database, for example: `/var/isode/dsa-db/gdam1`

`-ldif filename`

This gives the name of the LDIF file.

`-user DN`

This gives the Distinguished Name assigned to creator and modifier attributes in the database. It is recommended that this be the name of an account with appropriate write permissions for the entries being loaded, although any syntactically valid DN is accepted.

`-maxfail n`

This specifies the maximum number of failed records which should be tolerated. When this number is exceeded the program aborts. The default is 9. Use 0 to abort on the first error.

`-skip n`

This tells the program to skip the specified number of records at the beginning of the LDIF file.

`-count n`

This tells the program to process only the specified number of records from the LDIF file.

Note: The `-skip` and `-count` options can be used to process an LDIF file in chunks, or to process part of an LDIF file which previously failed.

-overwrite

This option allows existing entries to be replaced. By default, existing entries in the database are not overwritten. You will get an error entry exists (nooverwrite): <dn> for each entry that exists if -overwrite is not set.

Caution: It is possible to overwrite Directory Server configuration entries using this option. Care must be taken not to overwrite essential configuration data found in the cn=config subtree.

4.8.2.1.2 Using dbulk clean

The dbulk tool can be used in clean mode to delete entire subtrees. To remove a subtree from the database, enter the following command line:

```
dbulk clean baseDN -db_directory pathname [-maxfail n]
[-descendants]
```

The meanings of the parameters are as follows:

baseDN

This identifies the subtree to be removed (cleaned). The root entry may not be removed, hence the base may be " " to indicate the root, only in conjunction with -descendants.

-db_directory pathname

This specifies the location of the GDAM database, for example, */var/isode/dsa-db/gdam1*

-maxfail n

This gives the maximum number of failed records which should be tolerated. When this is exceeded the program aborts. The default is 9. Use 0 to abort on the first error.

-descendants

This removes all the subordinates of the base entry down to the bottom of the tree, but does not remove the base entry itself.

4.8.2.1.3 Using dbulk dump

The dump mode is used to export single entries or subtrees of entries to LDIF. The command line is:

```
dbulk dump baseDN -db_directory pathname [-maxfail n]
[-descendants] [-operational] [-subentries] [-ignore
attribute-type] [-maxthreads n]
```

The meanings of the parameters are as follows:

baseDN

This identifies the subtree to be dumped (exported). The empty string ("") can be used to direct **dbulk** to dump *all* entries in the database.

-db_directory pathname

This specifies the location of the GDAM database, for example, */var/isode/dsa-db/gdam1*

-maxfail n

This gives the maximum number of failed records which should be tolerated. When this is exceeded the program aborts. The default is 9. Use 0 to abort on the first error.

-descendants

This dumps all subordinates of the base entry down to the bottom of the tree, excluding the base entry.

`-operational`

Whether to include operational attributes in the output LDIF.

`-subentries`

Whether to include subentries, e.g. access control subentries, in the output LDIF.

`-maxthreads`

The maximum number of threads to use when processing the in-memory database. By default dbulk will use all available processors.

4.8.2.2 Using dsnapdump to Export Historical Snapshots

While **dbulk** is used to retrieve the current data, **dsnappedump** is used to dump one of the historical snapshots stored in a directory database (see [Section 4.6, “Database configuration”](#) for details of the GDAM database structure). The command line for **dsnappedump** is:

```
dsnappedump -s|--snapshot pathname [-l|--ldif filename]
```

The meanings of the parameters are as follows:

`-s | --snapshot pathname`

This specifies the location of the snapshot within a GDAM database, for example, `/var/isode/dsa-db/gdam1/snapshots/0000000001f381b`.

`-l | --ldif filename`

The filename to output LDIF to. If this argument is not provided then **dsnappedump** will output to the terminal.

4.8.2.3 Using dlogdump to export changes

While **dbulk** and **dsnappedump** export the directory content as is, **dlogdump** exports recent *changes* as stored in a directory database changelog (see [Section 4.6, “Database configuration”](#) for details of the GDAM database structure). The command line for **dlogdump** is:

```
dlogdump -c|--changelog pathname [-l|--ldif filename]
```

The meanings of the parameters are as follows:

`-c | --changelog pathname`

This specifies the location of the changelog within a GDAM database, for example, `/var/isode/dsa-db/gdam1/changelog`.

`-l | --ldif`

Output in LDIF format to the terminal.

`-f | --full`

Output raw changes to the terminal. This outputs a full diagnostic dump of the changelog. Note that the output is *not* in LDIF format.

Chapter 5 Authentication

This aim of this chapter is to explain the authentication mechanisms that can be used with M-Vault Server.

Note: These mechanisms do not by themselves provide any guarantees of Directory security but rely on security of the operating systems and hosts on which the Directory Server(s) and any Directory User Agents (DUAs) run, and in some cases also on the networks connecting them.

5.1 Security in the Directory

The M-Vault Server provides two kinds of security services associated with network applications:

- **authentication:** determining the identity of a communications partner

M-Vault enables authentication of:

- A Directory User Agent (DUA) to the Directory Server, and of the Directory Server to a DUA, when a DAP or LDAP association is made
- Peer Directory Servers to each other when a DSP or DISP association is made.
- **authorization/access control:** once the identity has been established, determining what data and operations may be accessed by that identity.

This is discussed in [Chapter 6, *Controlling Access*](#).

Authentication level, which relates authentication to access control, is discussed in [Section 5.8, “Authentication levels”](#).

5.1.1 General security issues

As the Directory Server can be started automatically (i.e. without an operator), there are certain issues regarding security which you need to consider; for example, it is generally not possible to request a passphrase at startup, or to obtain random number data from operator input.

As a result, decisions about the following issues need to be made carefully:

- account selection
- file system access control
- server keys.

The security of the Directory Server relies on file system access controls. This means that the Directory Server security is only as strong as the login security of the system (which is usually password-based), including the ease with which a user can acquire the privileges of another user.

As the data managed by the Directory Server is held unencrypted in the file system, there is little point in protecting the cryptographic keys any more securely, as a successful attempt to obtain the cryptographic keys by subverting the file system access control mechanisms would also allow access directly to the data protected by the cryptographic keys. However, it is important that the cryptographic keys are not used for any other purpose.

5.2 Introduction to authentication

Authentication is about proving who you are: verifying the credentials of both parties when establishing a connection between them.

All Directory protocols have a similar binding phase when establishing an association:

1. The initiator (a DUA in DAP and LDAP, or a Directory Server in DSP or DISP) opens an association and sends its credentials.
2. The responder (always a Directory Server) detects a new incoming association and receives these credentials.
 - If the credentials are invalid, the responder returns an error code and may close the connection. The only exception is for a DSP initiation, in which case a referral is returned.
 - If the credentials are valid, the responder may send its own credentials to the initiator. The initiator may wish to check these credentials to ensure it is communicating with the correct Directory Server.
3. Once the authentication process is complete, an authentication level is assigned (see [Section 5.8, “Authentication levels”](#)).

Note: Authentication provides an assurance at time of use only. Attackers will be able to subvert an authentication mechanism if they have access to the network and are able to insert packets which appear to be from the DUA or Directory Server.

5.2.1 Establishing identity

One of seven kinds of credentials may be used to establish identity, each useful under different circumstances. The five supported types form the basis of the authentication modes that can be configured in M-Vault (the authentication mode value associated with each one is given below).

anonymous

(Authentication mode 0) No information is provided, which means the Directory Server will not be able to identify the DUA at all. This kind would typically be used when connecting to a Directory Server which provides public information such as a white pages service, where modification or access to sensitive data is not possible.

name only

(Authentication mode 1) The Distinguished Name (DN) of the person or service requesting connection is sent, but without any proof (no password). This kind of authentication has limited application, but may be useful for providing a name to the Directory Server.

simple unprotected

(Authentication mode 2) The DN and a password string are sent in cleartext; for exceptions refer to [Section 5.6.3, “Storing passwords in the GDAM”](#). This method is widely implemented in DUA and Directory Server products, and provides a level of security similar to that in FTP and other Internet protocols. It is most suited for authentication inside a single enterprise, where the Directory Server holds a copy of all users' passwords and the network is trusted or is inaccessible to attackers.

simple protected

[Not currently supported by M-Vault.] The DN is sent in cleartext, but the password is sent in an encrypted form.

strong

(Authentication mode 4) A digitally signed request and response, using X.509 certificates, is exchanged. X.509 certificates are digitally signed with the private key of a Certificate Authority (CA), which the responder should be able to verify with the CA's public key. A token is transmitted with the certificate that contains a timestamp and a random number, in order to prevent reuse of the transmission. The responder must be configured to trust the CA that has signed the certificate.

Configuration of X.509 is discussed in [Section 5.4, “Configuring the Directory for X.509”](#). The advantage of strong credentials is that the initiator does not reveal any sensitive information by signing, and the certificate can be verified by any Directory server which has knowledge of the certificate hierarchy. The disadvantage of strong credentials is that they rely on algorithms which are patented or export controlled in many countries, and thus are not widely implemented.

external

[Not currently supported by M-Vault.] The credentials are transmitted via some external protocol.

SASL

Multiple mechanisms are permitted, and some of these mechanisms use a challenge-response system to avoid passing credentials over the network in the clear. SASL uses userids instead of DNs, and these must be mapped to DNs using a number of configurable mapping rules.

SASL is only supported for LDAPv3 DUAs, and provides multiple mechanisms that can be used to authenticate. Some mechanisms are very weak (e.g. PLAIN and LOGIN) while others are considered strong (e.g. DIGEST-MD5). Users' SASL credentials (“secrets”) are normally held in their Directory entries, but can also be held in external databases.

Note: These credentials may be carried over an underlying confidentiality transport layer (for example, TLS) which may affect their treatment. In particular, LDAP simple credentials over a TLS confidentiality connection may be treated differently from LDAP simple credentials over a connection without an underlying confidentiality transport layer.

5.3 Configuring authentication for specific protocols

Different protocols have different requirements. Many of these attributes can be set using Sodium by modifying the specified attributes in **cn=core,cn=config**. More information is provided in [Section E.1.6, “Chaining”](#), and [Section E.1.5, “Shadowing”](#).

For strong authentication (and signed operations) six more attributes are relevant (**isodeDAPIncludeCertificationPath**, **isodeDSPIncludeCertificationPath**, **isodeDISPIncludeCertificationPath**, **isodeDAPStrongTokenExpiry**, **isodeDSPStrongTokenExpiry**, **isodeDISPStrongTokenExpiry**) which may appear either in **cn=core,cn=config** or (for DSP and DISP) in peer-specific entries. These are described in [Section E.1.3, “X.509 Strong Authentication”](#).

5.3.1 DAP (as responder)

The Directory Server supports anonymous, name only, simple, and (when configured) strong authentication. To disable some modes, change the **isodeDAPAuthModesIExpect** attribute in **cn=core,cn=config**.

Authentication levels may be degraded from strong to simple by setting **isodeDAPDegradeStrong** attribute, and from simple to none by setting **isodeDAPDegradeStrong** in **cn=core,cn=config**.

Passwords are compared against **userPassword** attributes, possibly in conjunction with password policy, see [Section 5.6, “Password management”](#).

If strong authentication is configured, the Directory Server can require that all modification operations be signed using the **isodeDSPSignModify** attribute in **cn=core,cn=config**.

5.3.2 LDAP v3 (as initiator)

The Directory Server sends only anonymous binds when initiating LDAP chained connections.

5.3.3 LDAP v3 (as responder)

The Directory Server supports anonymous, name only, simple, and (when configured) SASL authentication. To disable some modes, change the **isodeLDAPAuthModesIExpect** attribute in **cn=core,cn=config**. Passwords are compared against **userPassword** attributes, possibly in conjunction with password policy, see [Section 5.6, “Password management”](#).

5.3.4 DSP (as initiator or responder)

The Directory Server supports anonymous, name only, simple and (when configured) strong authentication. To disable some modes, change the **isodeDSPAuthModelSend** and **isodeDSPAuthModesIExpect** attributes in **cn=core,cn=config**.

Two passwords are used in simple authentication. By default the other Directory Server’s password is compared against the **isodeDSPPasswordIExpect** attribute in the **cn=core,cn=config** entry. By default the password sent to the other DSA is the **isodeDSPPasswordISend** attribute in **cn=core,cn=config**.

If strong authentication is used, signed operations can be required by setting the **isodeDSPSignArg** attribute in the **cn=core,cn=config** entry, and signed results can be required by setting the **isodeDSPSignRes** attribute in the **cn=core,cn=config** entry.

In addition, chained operation authentication levels can be degraded (reduced) from simple to none by setting the **isodeDSPDegradeSimple** attribute in the **cn=core,cn=config** entry, and from strong to simple by setting the **isodeDSPDegradeStrong** attribute in the **cn=core,cn=config** entry.

Chained operations will also be degraded by analysing the DSP trace information. Peers can be marked as trusted by setting the **isodeDSPTrusted** attribute in the peer entry; provided all peers in the trace are so trusted this degrading will not happen.

If a peer-specific entry for the other Directory Server is present under **cn=config** the attributes are read from there instead.

Note: A Directory Server can receive chained operations even if it is a DSP initiator. Initiator and responder just indicate which Directory Server created the connection, not which sends the operations.

5.3.5 DISP (as initiator or responder)

The Directory Server supports name-only (only as initiator), simple and (when configured) strong authentication. To disable some modes, change the **isodeDISPAuthModelSend** and **isodeDISPAuthModesIExpect** attributes in **cn=core,cn=config**.

Two passwords are used in simple authentication. By default the other Directory Server's password is compared against the **isodeDISPPasswordIExpect** attribute in the **cn=core,cn=config** entry. By default the password sent to the other Directory Server is the **isodeDISPPasswordISend** attribute in **cn=core,cn=config**.

If strong authentication is used, shadow coordinate operations can be signed by setting **isodeDISPSignCoShaUp**, shadow update operations can be signed by setting **isodeDISPSignShaUp** and shadow update requests can be signed by setting **isodeDISPSignSrShaUp**.

If a peer-specific entry for the other Directory Server is present under **cn=config** the attribute are read from there instead.

5.4 Configuring the Directory for X.509

For a M-Vault Server to be able to make use of X.509 based strong authentication for DAP, DISP and DSP operations, it must be configured to have access to at least:

- Its own X.509 certificate, representing the Directory Server itself
- One or more certificates that can be used to form a chain to a “trust anchor” certificate, which will be that of a trusted Certificate Authority.

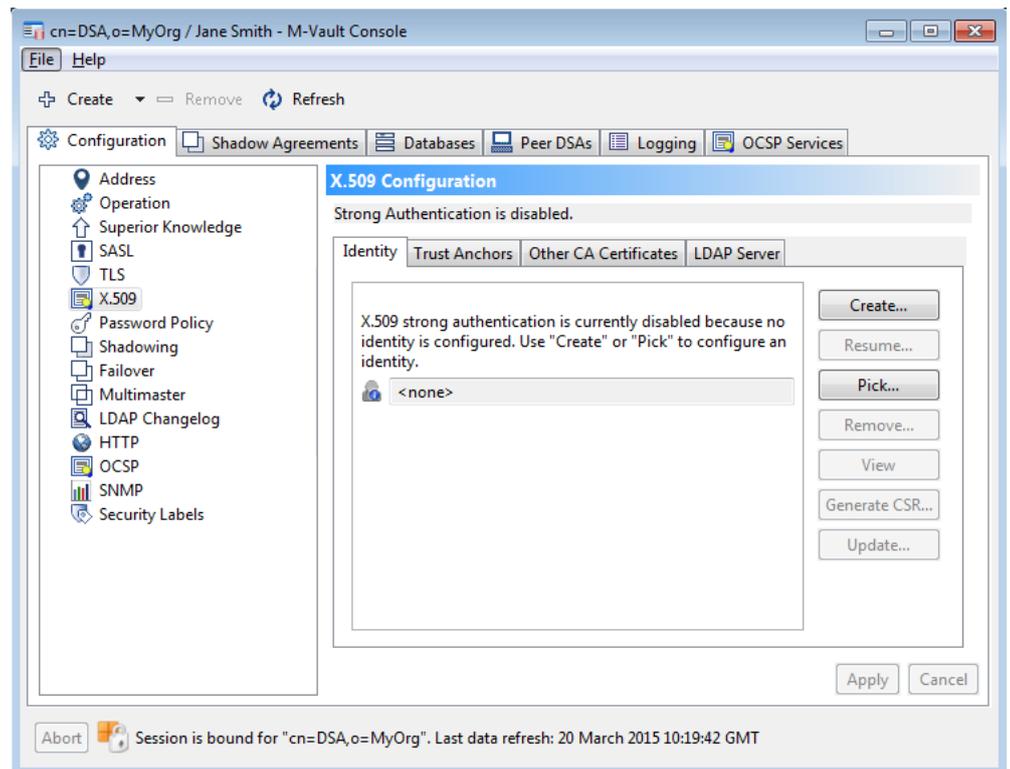
When *initiating* a strong bind (DSP or DISP), the Directory Server includes its certificate as part of the bind request. The response to that request will include a certificate from the other Directory Server. In order for authentication to succeed, both Servers must validate each other's certificate, using their own trust anchors.

When *responding* to a strong bind (DSP, DISP or DAP), the certificate received in the bind request must be validated against the Directory Server's trust anchor. The bind response sent by the Directory Server includes its own certificate.

5.4.1 The Directory Server's own certificate

You use M-Vault Console to create an identity for a Directory Server. The certificate from the identity does not contain sensitive data, and will be published by the Directory Server. But an identity also contains a private key, which must not be disclosed. Identities for the Directory Server are stored in encrypted form on the system where the server is running. M-Vault Console therefore needs to be running on the same system as the Directory Server when creating an identity for it, and will not offer the option of creating an identity for any remote Directory Server that you are connected to.

1. Start M-Vault Console and bind to the Directory Server.
2. Select **X.509** from the list of options on the left.



3. Click **Create** to create an identity for this Directory Server.

Note: You can also click **Pick** to select an identity that already exists and associate it with this DSA.

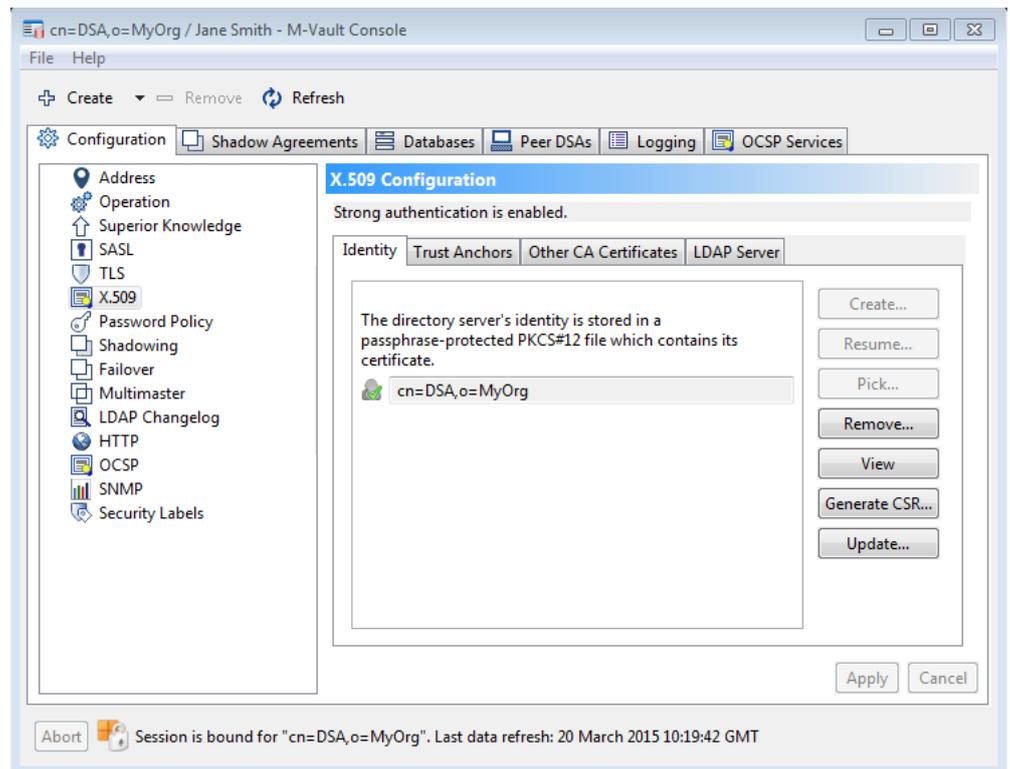
4. Follow the steps outlined in [Section 3.10.1, “Generating a certificate request”](#).

You cannot change the subject DN but all other steps are the same.

5. At the end of the wizard, you are asked if you want to use this identity for TLS as well.
6. Click **Finish**.

5.4.2 Additional X.509 configuration

The **Trust Anchor** page automatically shows the trust anchor derived from the Directory Server’s identity, which cannot be removed. You can, however, add further trust anchors by clicking the **Add** button and selecting certificates from the file system. You can also use **Pick** to select identities from within the Directory.



Any other certificates that may be used during the certificate verification process are entered on the **Other CA Certificates** page, using the same **Add** or **Pick** options as before.

Finally, the details of any LDAP Server used to look up certificates and revocation lists during verification should be listed on the **LDAP Server** page. You can choose to use the local server's details and also whether you want to use the same values for X.509 and TLS.

5.5 SASL authentication

SASL is an Internet standard (*RFC 4422*) which defines a Simple Authentication and Security Layer and it is used in several Internet protocols such as SMTP, IMAP, BEEP, and LDAP. It provides a means of supporting different authentication mechanisms in an easily extensible fashion.

SASL distinguishes authentication from authorization. This means that it is possible for some intermediate system to authenticate as one user, but act as some other entity. This is often called “proxy authentication”.

SASL does not generally use DNs to identify users to the Directory Server. Because DNs are vital for the Directory Server in the processing of access controls, the use of SASL with the Directory requires some additional configuration to map SASL userids into DNs. The M-Vault server supports several different mapping schemes.

In general then, binding using SASL involves the following logical steps:

1. Authentication

The LDAPv3 client specifies a SASL mechanism, some credentials and optionally a SASL userid, to the M-Vault server. Some mechanisms may require further information from the client.

If the authentication is successful the connection is associated with a SASL userid.

2. Authorization

The LDAPv3 client may specify that it wants to act as another entity. This is known as proxy authentication. M-Vault verifies that the authenticated entity is allowed to act as the other entity.

Normally the authorization identity is the same as the authentication identity.

3. Userid mapping

The authorization identity is then mapped algorithmically into an authorization DN.

4. Verification

The Directory Server verifies that the entry described by the authorization DN exists.

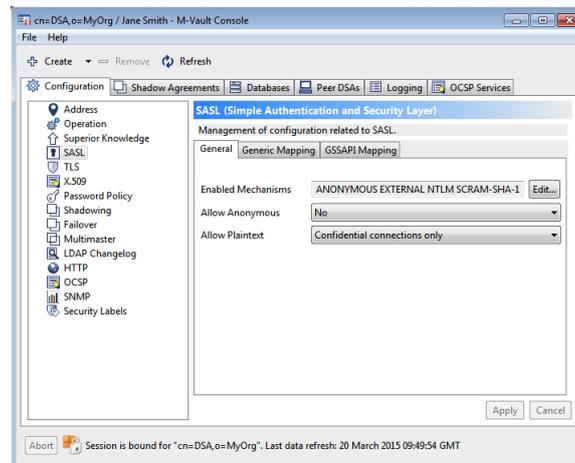
5.5.1 Configuring SASL

Instructions for configuring SASL are given in this section. Reference is made to subsequent sections for background explanation.

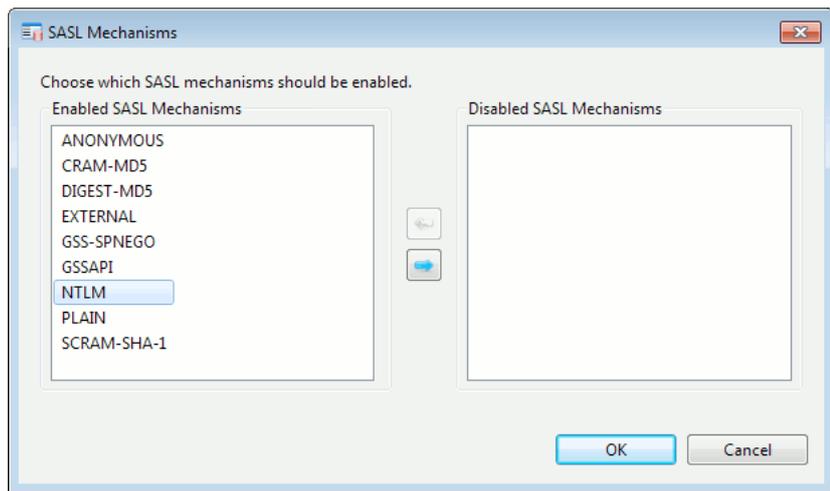
SASL is configured using M-Vault Console.

1. Bind to the Directory using M-Vault Console.
2. On the **Configuration** page, select **SASL** from the list of options on the left.

The **SASL Configuration** pane is displayed, which contains three pages: **General**, **Generic Mapping** and **GSSAPI Mapping**.



3. M-Vault supports multiple SASL mechanisms, although none is enabled when a Directory is first installed. To specify which of the available mechanisms are to be enabled and disabled:
 - a. Click **Edit...**



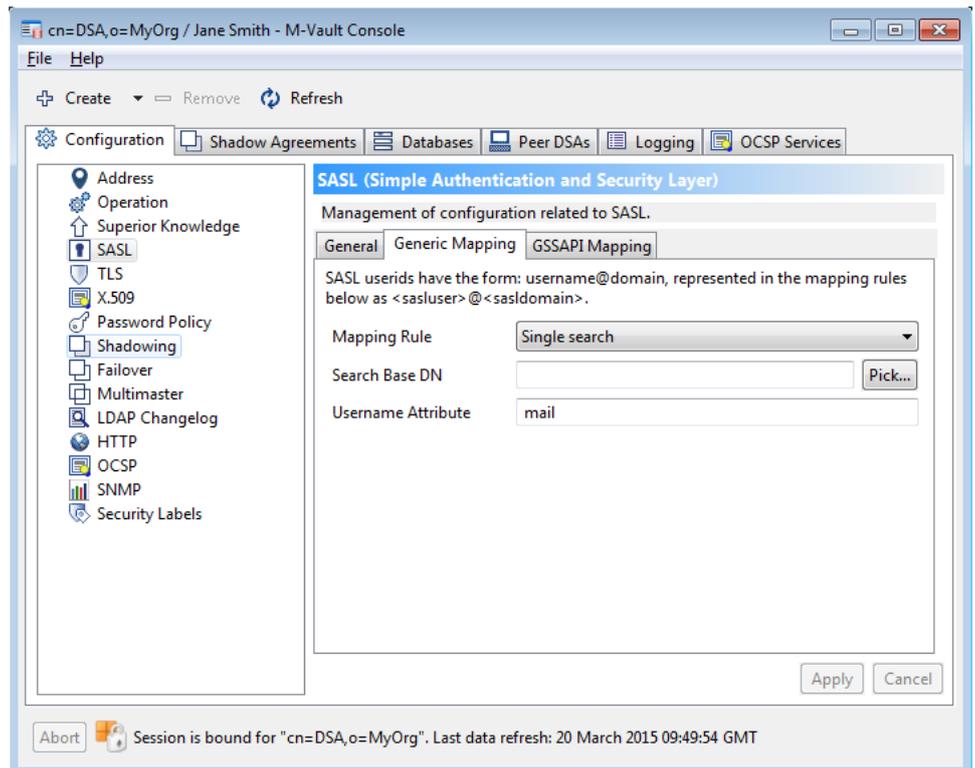
The **SASL Mechanisms** window opens.

- b. Select at least one mechanism from one of the boxes (**NTLM** is selected in the **Enabled SASL Mechanisms** box) and the appropriate arrow button between the boxes is enabled. See [Section 5.5.2, “SASL mechanisms”](#) for information on the strengths of the different mechanisms.
 - c. Click the arrow button between the boxes to transfer the selected mechanisms between the enabled and disabled lists. Click **OK**.
4. Choose whether anonymous and plaintext SASL connections are allowed or not.

Note: These options only apply to non-TLS connections.

5. Click the **Generic Mapping** tab if you are going to use generic mapping rules to generate a SASL user id (see [Section 5.5.3, “SASL userid mapping”](#)) or the **GSSAPI Mapping** tab if you are going to use those mapping rules (see [Section 5.5.4, “SASL GSSAPI configuration”](#)).

The **Generic Mapping** page is shown below. On the **GSSAPI Mapping** page, the first option is called **GSSAPI Rule**.



6. Select the mapping rule you want to use.
 - Information on the generic mapping rules can be found in [Section 5.5.3.1, “Generic mapping rule – Active Directory compatible”](#); [Section 5.5.3.2, “Generic mapping rule – domain part search”](#); [Section 5.5.3.3, “Generic mapping rule – two searches”](#); and [Section 5.5.3.4, “Generic mapping rule – single search”](#).
 - Information on the GSSAPI mapping rules can be found in [Section 5.5.4.1, “GSSAPI mapping rule - Active Directory Compatible”](#); [Section 5.5.4.2, “GSSAPI mapping rules - Domain Part and Two Searches”](#); and [Section 5.5.4.3, “GSSAPI mapping rule - Single search”](#).
7. Select a DN from the Directory to use as the **Search Base DN** when mapping user IDs to Directory entries (see [Section 5.5.3, “SASL userid mapping”](#), and [Section 5.5.4, “SASL GSSAPI configuration”](#)). For **Generic Mapping**, this is stored in the **isodeGenericBase** attribute, and for **GSSAPI Mapping**, it is stored in the **isodeSASLGSSAPIBase** attribute.
8. Type the name of the attribute that holds the value that you are going to use to look for a match to the user part of the SASL id (see [Section 5.5.3, “SASL userid mapping”](#), and [Section 5.5.4, “SASL GSSAPI configuration”](#)). For **Generic Mapping**, this is stored in the **isodeSASLGenericFullMatchAttr** attribute, and for **GSSAPI Mapping**, it is stored in the **isodeSASLGSSAPIFullMatchAttr** attribute.
9. Click **Apply** to save your changes.

5.5.2 SASL mechanisms

M-Vault supports multiple SASL mechanisms via a plugin system. When the Directory Server starts up it loads all the plugins installed in `(LIBDIR)/sas2`. This makes it simple to disable certain mechanisms completely (by removing the plugin file and restarting the Directory Server), or to add additional mechanisms (by copying in the new plugin and restarting the Directory Server). The mechanisms may be enabled from the Directory server properties screen using M-Vault Console, as described in [Section 5.5.1, “Configuring SASL”](#).

Each mechanism supplied has different characteristics that might make it more or less useful for a given Directory Server.

Table 5.1. SASL mechanisms

Mechanism	Approach	Security
PLAIN	Sends plaintext passwords across the network.	Very weak
LOGIN		
CRAM-MD5	Basic challenge/response, but vulnerable to server spoofing attacks.	Weak
NTLM	Basic challenge/response, using a Microsoft-specific algorithm.	Weak
DIGEST-MD5	Challenge/response.	Good
SCRAM-SHA-1	Challenge/response.	Stronger
GSSAPI	Trusted third party (for example, Kerberos)	Stronger
TLS + EXTERNAL	Client uses an X.509 certificate	Best

5.5.3 SASL userid mapping

The SASL bind operation passes a DN, which is always ignored, to the Directory Server: instead, a userid is used by many of the mechanisms. SASL userids have the form:

username@domain, which are represented in the mapping rules below as <sasluser>@<sasldomain>.

The Directory Server then maps the SASL userid to a DN using the mapping rule specified by the Directory Server Manager. The following sections describe the parameters shown by M-Vault Console for each of the different mapping rules.

Note: If a search returns either no results or more than one result, or the entry corresponding to the constructed DN does not exist, the mapping fails.

Note: The GSSAPI and EXTERNAL mechanisms do not use SASL userids; for more information see [Section 5.5.4, “SASL GSSAPI configuration”](#) and [Section 5.5.5, “SASL EXTERNAL configuration”](#).

5.5.3.1 Generic mapping rule – Active Directory compatible

This rule can be used to construct a DN that is compatible with DNs used in Active Directory (AD). The DN is constructed as follows:

<attr>=<sasluser>, <ADglue>, dc=<subdomain>, dc=<subdomain>, <suffix>

where:

- <attr> is the naming attribute used to identify entries in the Directory (for example, **uid**) and is specified in **Username Attribute**
- <sasluser> is the original user portion of the SASL userid
- <sasldomain> is the domain portion of the SASL userid
- <ADglue> is a glue entry (such as **cn=Users**) to bridge the gap between the information provided in the <sasldomain> and where AD stores the relevant entries - it is specified in **Container**.
- <subdomain> is a sub-domain of the <sasldomain> (the element dc=<subdomain> is repeated as often as necessary to incorporate all sub-domains)
- <suffix> is optional and provides the remainder of the DN if the sub-domains are not sufficient on their own to construct an appropriate DN - it is specified in **AD DN Suffix** attribute.

For example, if your system is configured so that:

- **Username Attribute** is set to **uid**
- **Container** is set to **cn=Users**
- **AD DN Suffix** is set to **o=MyCorp, c=US**

then the SASL userid of barabash@example.net will map to **uid=barabash, cn=Users, dc=example, dc=net, o=MyCorp, c=US**

5.5.3.2 Generic mapping rule – domain part search

This mapping supports holding users with multiple SASL domains in multiple separate subtrees.

- If <sasldomain> is absent or is the same value as **Default Domain**, form the DN as follows:

<attr>=<sasluser>, <search base>

where

- <sasldomain> is the domain portion of the SASL userid

- `<attr>` is specified in **Username attribute** and is the naming attribute used to identify entries in the Directory (for example, **uid**)
- `<sasuser>` is the original user portion of the SASL userid
- `<search base>` is specified in **Search base DN**
- If `<sasldomain>` is not the default domain:
 1. Search from `<search base>` for a domain entry (`<search result>`) where the value of the `<domain match attribute>` (specified in **Domain attribute**) matches `<sasldomain>`
 2. Use the template `<attr>=<sasuser>, <search result>`

For example, if your system is configured so that:

- **Username attribute** is set to **cn**
- **Search base DN** is set to **o=My Corp, c=US**
- **Default Domain** is set to **example.net**

then the SASL userid of `barabash@example.net` will map to **cn=barabash, o=My Corp, c=US**

5.5.3.3 Generic mapping rule – two searches

This mapping rule is similar to the domain part mapping rule, except that instead of forcing all user entries to be directly below the domain suffix, the second option performs a subtree search under the domain suffix for the user.

- If `<sasldomain>` is absent or is the default domain, then search from `<search base>` for a user entry with `<user match attribute>=<sasuser>`

where

- `<search base>` is specified in **Search base DN**
- `<user match attribute>` is the attribute used when searching for matches in the values of `<attr>`
- `<attr>`, specified in **Username Attribute** is the naming attribute being used to identify entries in the Directory (for example, **uid**)
- `<sasuser>` is the original user portion of the SASL userid
- If `<sasldomain>` is not the default domain:
 1. Search from `<search base>` for a domain entry (`<search result>`) where the value of the `<domain match attribute>` matches `<sasldomain>`
 2. Search from `<search base>` for a user entry where `<user match attribute>=<sasuser>`

The resulting match is used as the DN.

5.5.3.4 Generic mapping rule – single search

This mapping rule is the most flexible, as it allows users in the same subtree to have different SASL domains. It does this by searching for the complete SASL userid.

- If `<sasldomain>` is absent, search from `<search base>` for a user entry with `<full user match attribute>` matching `<sasuser>@<default domain>`
- If `<sasldomain>` is present, search from `<search base>` for a user entry with `<full user match attribute>` matching `<sasuser>@<sasldomain>`

where:

- `<sasldomain>` is the domain portion of the SASL userid

- `<search base>` is specified in **Search base DN** attribute
- `<full user match attribute>` is the value of **isodeSASLGenericFullMatchAttr**

The resulting match is used as the DN.

For example, if your system is configured so that:

- `<full user match attribute>` is specified in **Username attribute**
- the default domain is set to `example.net`

then the Directory Server will:

- search `<full user match attribute>` for a single entry matching `uid=barabash@example.net` if the provided SASL id is `barabash`
- search `<full user match attribute>` for a single entry matching `uid=barabash@myorg.co.uk` if the provided SASL userid is `barabash@myorg.co.uk`

5.5.4 SASL GSSAPI configuration

RFC 1964 defines the Kerberos v5 GSSAPI (Generic Security Service Application Program Interface). This is used by the SASL GSSAPI mechanism.

SASL userids for GSSAPI are therefore Kerberos principals, which take the form: `username@realm` where `realm` is a Kerberos v5 realm (normally written in uppercase.) The realm is somewhat analogous to the domain used in other SASL mechanisms. The default Kerberos realm must be specified in the Directory Server's **isodeSASLGSSAPIRealm** attribute.

In order to use the GSSAPI mechanism, an administrator first has to:

1. Configure Kerberos v5 on each machine running M-Vault
2. Create the LDAP service Kerberos principal for each Directory server, of the form: `ldap/hostname@realm`

The hostname is the fully qualified hostname of the machine, and the realm is the Kerberos realm. GSSAPI implementations vary in this area so for full details consult your GSSAPI vendor's documentation; but generally you would generate a random password and you should also define that the principal is a service as opposed to a user.

Although the realm often looks like a network domain name, it may not be. In this example the server is running on `demo1.example.net` yet is a member of the `EXAMPLE.ORG` realm: `ldap/demo1.example.net@EXAMPLE.ORG`

3. Export the principal's key created in step 2 on each machine running M-Vault into a keytab file. Again, different GSSAPI implementations vary in this area so for full details consult your GSSAPI vendor's documentation.

The GSSAPI configuration in M-Vault uses the Kerberos schema defined for MIT Kerberos 5 (see [Kerberos: The Network Authentication Protocol \[377\]](#)). The main difference from the non-GSSAPI mapping rules is that by default searches for the full Kerberos principal name will use the **krbPrincipalName** attribute.

These rules are defined in the following sections.

5.5.4.1 GSSAPI mapping rule - Active Directory Compatible

This rule can be used to construct a DN that is compatible with DNs used in Active Directory. The DN is constructed as follows:

<attr>=<k-user>, <ADglue>, dc=<subrealm>, dc=<subrealm>, <suffix>

where:

- `<attr>` is the naming attribute used to identify entries in the Directory and is specified in **Username Attribute**
- `<k-user>` is the original user portion of the provided userid
- `<ADglue>` is a glue entry (such as **cn=Users**) that bridges the gap between the information provided in `<realm>` and where AD stores user entries - the value of this is set is **Container**
- `<subrealm>` is a sub-realm of `<realm>` (the element **dc=<subrealm>** is repeated as often as necessary, once for each sub-realm)
- `<realm>` is the realm portion of the provided userid
- `<suffix>` is optional and provides the remainder of the DN if the sub-realms are not sufficient on their own to construct an appropriate DN - the value of this is set in **AD DN Suffix**.

For example, if your system is configured so that:

- **Username Attribute** is set to `uid`
- **Container** is set to `cn=users`
- **AD DN Suffix** is set to `o=My Corp, c=US`

then the Kerberos principal of `barabash@EXAMPLE>NET` will map to `uid=barabash, cn=Users, dc=EXAMPLE, dc=NET, o=My Corp, c=US`

5.5.4.2 GSSAPI mapping rules - Domain Part and Two Searches

These mappings support holding users with multiple Kerberos realms in multiple separate subtrees, and is a two-stage process:

1. Search from `<search base>` for entries where the value of `<realm match>`. The DN of the matching entry is used as the value of `<realm suffix>`, the starting point for stage 2. **krb5RealmName=realm**.
 - The attribute represented by `<realm match>` is specified in **Username Attribute**.
 - The attribute represented by `<search base>` is specified in **Search base DN**.
2. Perform a subtree search from `<realm suffix>` for entries where `<user match>` match the provided Kerberos principal. The resulting match of this second search is used as the DN.
 - `<realm suffix>` is the result of the first part of the search.
 - `<user match>` is specified in **Username attribute**

For example, if your system is configured so that `<search base>` is set to `o=My Corp, c=US` and the provided Kerberos principal is `barbash@EXAMPLE.NET`, the Directory Server will:

1. Search from `<search base>` for an entry where `<realm match> = EXAMPLE.NET`
2. Use the DN of that entry (for example, `o=My Corp, c=US`) as the value of `<realm suffix>`.
3. Search from `<realm suffix>` for an entry where the value of `<user match>` is a match for the provided Kerberos principal.
4. Use the DN of the entry containing the match as the matching DN. For example, `cn=R Barabash, ou=R&D, o=My Corp, c=US`

5.5.4.3 GSSAPI mapping rule - Single search

This mapping rule is the most flexible, as it allows users in the same subtree to have different Kerberos realms. It does this by searching for the complete Kerberos principal.

It searches from `<search base>`, checking the values of `<full match attribute>` for a match to the Kerberos principal, where:

- `<search base>` is specified in **Search Base DN**
- `<full match attribute>` is specified in the **Username attribute**.

5.5.5 SASL EXTERNAL configuration

If the LDAPv3 DUA has previously set up a confidential connection using SSL or TLS, and presented the Directory Server with a client X.509 certificate, then a SASL bind using the EXTERNAL mechanism can be attempted.

X.509 certificates contain a subject name, which is a DN. Certificates purchased from commercial CAs such as Verisign usually contain non-useful subject names which include information like email addresses, and the vendor's context prefix. Because the Directory Server uses the subject name directly from the certificate, and since the names in commercial certificates are unlikely to exist in your DIT, this means that commercially obtained certificates typically cannot be used with the SASL EXTERNAL mechanism. The only practical way to use SASL EXTERNAL is when there is a locally controlled CA that can issue certificates with locally valid DNs.

Before using the subject name the Directory Server must make additional checks on the certificate to ensure that the subject name should be trusted.

To configure these extra checks, configure TLS as described in [Section 5.7, "TLS configuration"](#). Set the `tlsVerifyClient` attribute to either optional or required and set the list of trusted CAs to include the CA that signs the certificates issued to SASL EXTERNAL users.

5.6 Password management

This section describes how passwords are used, transmitted and stored.

5.6.1 Password policy

Normally M-Vault allows DAP and LDAP binds using simple authentication to succeed based on the equivalence of a stored password and the presented password. This is not always appropriate, and it is sometimes required to impose some additional constraints on how users are able to authenticate to their accounts in the Directory.

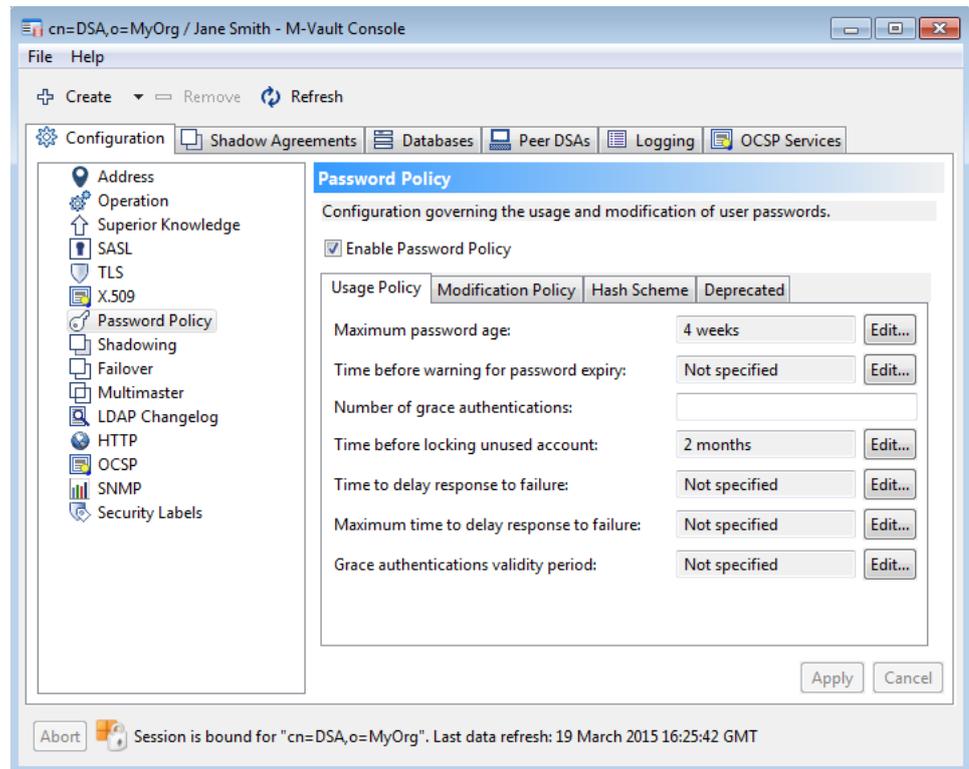
M-Vault can be configured to enforce a password policy, based on *draft-behera-ldap-password-policy-09*. Entries containing `userPassword` attributes are all subject to this policy, with the sole exception of the DSA Manager account.

The policy controls:

- if and when passwords expire
- whether failed attempts to bind cause the account to be "locked", and for how long
- if and how users are able to change their passwords.

To enable a password policy, connect to a Directory Server using M-Vault Console. **Password Policy** is an option in the **Configuration** group.

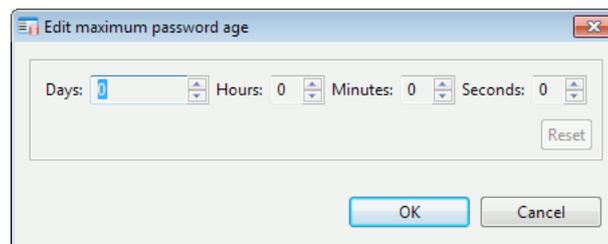
Figure 5.1. Password Policy page



Before you can specify any other options, you must select **Enable Password Policy**.

There are four separate pages in the **Password Policy** section:

- **Usage Policy:** This page is used to specify general information about passwords, most of them time-based.



You can specify times using whichever units are most appropriate and they will be converted to seconds for storing as part of the password policy.

- **Modification Policy:** This page is used to specify details about who is allowed to change passwords, and what restrictions should be imposed on any password values.
- **Hash Scheme:** For an installation where password values are hashed (determined by whether **Enable Password Hash Scheme Policy** is selected), the parameters on this page determine what kind of hash scheme to use (see [Section 5.6.3, “Storing passwords in the GDAM”](#)).
- **Deprecated:** This page is provided for backward-compatibility. If this is a new installation that is not using previous configuration files, nothing on this page is required.

5.6.2 Changing passwords

Passwords may be changed using the normal DAP/LDAP **Modify** operation, and also using the LDAPv3 extended **Password Modify** operation defined in *RFC 3062*.

Successful attempts to modify a password will cause the entry's **pwdChangedTime** attribute to be updated, and potentially its **pwdHistory** and **pwdFailureTime** attributes.

The Directory Server can be configured to check the quality of new passwords. The Directory Server first has to be configured with **Enable Password Policy**. The **Check Password Quality** option then determines what kind of checking is performed:

- **Never** Do not perform any checking. This is also the default if password policy is not enabled.
- **If Possible** Check passwords if possible. If a check is not possible (for example, the new password has been hashed by the DUA) allow the modification.
- **Always** Always check passwords. If a password cannot be checked, prevent the modification.

Users with LDAP clients can use a special control defined by *draft-behera-ldap-password-policy-09* when modifying their own entry, which will cause additional details of any password quality check failures to be returned.

The following checks are made on the presented password (some of the related parameters here appear on the **Deprecated** tab):

Minimum length

If the Directory Server's **Minimum password length** is set and non-zero, the presented password must be at least that length.

Invalid reuse

If the Directory Server's **Maximum number of passwords in history** is set, the presented password must not be in the list of historical passwords in the entry. The list of historical passwords is maintained in each entry's **pwdHistory** operational attribute.

Too recently changed

If set, the **Minimum password age** prevents users from changing their passwords too quickly.

Insufficiently mixed

The password must use characters from at least 3 of the 4 sets: upper-case, lower-case, digits, and other characters.

If an administrative user with appropriate access control permission - such as a user in the Password Manager group - changes another user's password, that password is said to have been *reset*. If **Force user password change** is set to **YES** in the **Modification Policy** page, this will automatically set the modified entry's **pwdReset** operational attribute to **TRUE**. The user will be allowed to authenticate using the password set by the administrator, but will then be forced to change the password immediately. When the user successfully changes their password the **pwdReset** operational attribute will be automatically removed.

5.6.3 Storing passwords in the GDAM

Passwords are normally held in the GDAM as-is, i.e. as unprotected plaintext. This mode allows the use of simple binds and all password-based SASL mechanisms such as DIGEST-MD5.

However if an attacker is able to obtain a copy of the database files, they will be able to steal all of the passwords held in the Directory.

To protect against this kind of attack, the server can be configured to hash passwords before storing them. To enable this, use M-Vault Console's Password Policy section (see [Section 5.6.1, "Password policy"](#)). The choice of the hashing algorithm used for storage affects what kinds of binds will subsequently work:

Value	Algorithm	Simple	SASL
(none)	Plaintext	Yes	All mechanisms
MD5	MD5 Digest	Yes	No mechanisms
SHA	SHA-1 Digest	Yes	No mechanisms
SHA2	SHA-2(256) Digest	Yes	No mechanisms
CRYPT	Traditional UNIX crypt	Yes	No mechanisms
SMD5	Salted MD5 Digest	Yes	No mechanisms
SSHA	Salted SHA-1 Digest	Yes	No mechanisms
SSHA2	Salted SHA-2(256) Digest	Yes	No mechanisms
SCRAM-SHA-1	Iteratively salted SHA-1 Hash	Yes	SCRAM-SHA-1, PLAIN and LOGIN only

If passwords are hashed in the GDAMs, a DUA using simple binds *must* provide the equivalent plaintext password. The server will refuse attempts to compare a hashed password from a DUA with a hashed password in the GDAM, regardless of the hashing algorithms being used.

As a consequence, hashing passwords does not protect against attackers able to read arbitrary packets from the network (e.g. the public Internet, or a wireless network). To protect against that kind of attack the use of SASL security layers, or TLS, or some other network-level confidentiality mechanism is recommended.

5.7 TLS configuration

The M-Vault Server supports LDAP and LDAPS over TLS/SSL. Your specific functionality may also vary depending on the sorts of algorithms that your Directory clients can support.

This section explains the security issues to consider before implementing TLS, and then describes how to configure the following in M-Vault Console:

- common TLS parameters, including cipher suites accepted by the Directory Server, personal key information, and other options
- TLS configuration when using LDAP.

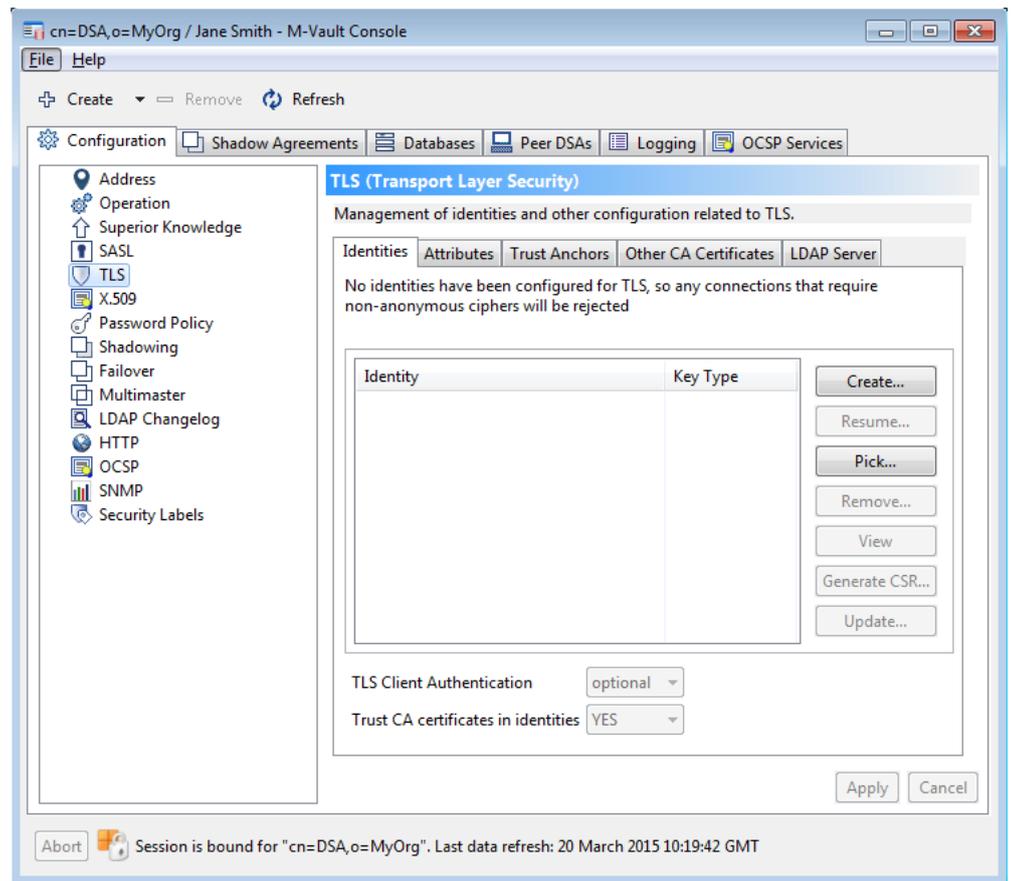
Caution: If LDAPS support is configured, but the TLS functionality is unavailable, any new connections on the LDAPS port will be closed. This happens if TLS is misconfigured or disabled.

5.7.1 Configuring TLS

TLS is configured using M-Vault Console.

1. Bind to the Directory using M-Vault Console.
2. On the **Configuration** page, select **TLS** from the list of options on the left.

The **TLS Configuration** pane is displayed, which contains the pages identified by tabs: **Identities**, **Attributes**, **Trust Anchors**, **Other Certificates** and **LDAP Server**.



3. On the **Identities** page, you need to create or select at least one identity to be used when connections are made that are not anonymous:
 - **Create...** starts a wizard to create a new identity. This invokes the same wizard used when a new identity is created in Sodium (see [Section 3.10.1, “Generating a certificate request”](#)) and initialises the wizard using the DN of the Directory Server itself. Once an identity has been created, you will be given the option to use it for both X.509 and TLS if you want.
 - **Resume...** is enabled if you have already generated a certificate request and now need to finish creating the identity (see [Section 3.10.3, “Linking a certificate to a Directory entry”](#)).
 - **Pick...** enables you to browse the filesystem for an identity which has been created previously.
 - **Remove...** enables you to remove an existing identity so it will no longer be used by the server.
 - **View...** enables you to see details about the identity, to be sure it is the one that you want to use.

For more information on identities, see [Section 3.10, “Managing identities”](#).

4. **TLS Client Authentication** and **Trust CA certificates in identities** are enabled once an identity has been selected. See [Section 5.7.2.1, “Identity information”](#) for more details.
5. Click the **Attributes** tab.

On the **Attributes** page, you can either set all the values yourself, or you can click **Set Defaults** (as has been done in the example above).

Both **TLS Support Flags** and **TLS Configured Cipher Suites** are set using the associated **Edit...** buttons. For more information on supported cipher suites, see [Section 5.7.3, “Supported TLS cipher suites”](#).

6. The **Trust Anchors** and **Other Certificates** page are used to specify certificates that are used during certificate verification.

7. The **LDAP Server** page is used to specify the address of an LDAP server. The LDAP server may be used as a source of certificates (if a trust chain refers to certificates which are not otherwise available). Additionally, if **Check CRLs** is specified, then all certificates will be checked to make sure that they have not been revoked. You can check **Use this directory server's LDAP address** to have the certificate verification process read this information from the local directory.

5.7.2 Server keys

TLS can operate without either the server or client having a key (using the `DH_anon_*` suites). However, generally you will want at least the server to have a key, since the anonymous suites offer no authentication.

The public key algorithm key types which may be used are:

- RSA key pairs
- DSA key pairs (DSA is sometimes referred to as DSS)
- ECDSA key pairs

Client support for the RSA algorithm is far more widespread than for DSA, ECDSA or the anonymous DH suites. It is also difficult to obtain a commercially signed certificate using the DSA or ECDSA algorithms, so if in doubt install an RSA key pair.

5.7.2.1 Identity information

Identity Information refers to the information used to identify the server cryptographically to the client (usually by digital signature).

To support a public key algorithm suite (for example, `RSA_with_DES_CBC_SHA`), the information comprises a private key (used for signature but never divulged to the client), a certificate containing the corresponding public key bound to the server's name, plus any additional certificates which form a certificate path to a well-known trust point.

The Directory Server expects the Identity Information to be held on its own filesystem. Note that the Identity Information is not stored or exposed inside a Directory entry, and so it is never visible to clients.

Updating or viewing the Identity Information therefore requires access to the file system used by the Directory Server, and so when using M-Vault Console to configure identities, you must be running M-Vault Console on the same system as the Directory Server itself.

It is possible to use the same identity for both TLS and X.509 strong authentication, either by selecting this option when creating it, or by using the **Pick...** button to choose the same identity from both in both TLS and X.509 tabs.

5.7.2.2 Client authentication

TLS also supports authenticating clients using certificates installed on the clients. There are three choices:

none

No client certificate is requested, and if one is presented then verification failure will have no effect.

optional

A client certificate is requested and if one is presented it must verify, otherwise the connection is aborted. If no certificate is presented, then the connection succeeds.

require

A client certificate is requested and if one is not presented or fails to verify, the connection is aborted.

The certificate that a client presents is verified against a list of trusted CA certificates. This list is provided to M-Vault as a single PEM encoded file containing one or more certificates. The name of this file is configurable (it is a pathname relative to the server's filestore directory.)

TLS also permits clients to present chains of certificates, with these connecting the end client certificate to a CA that the server trusts. The length (the depth of verification) of such chains which will be permitted is configurable. The default length is 1, which means that the certificate is signed by a single CA (which has a self-signed certificate). If certificates are being used from a commercial CA a larger value will probably need to be set.

5.7.2.3 Mandating TLS in LDAP

If the **Require TLS in LDAP** configuration option is set the server will enforce use of encrypted communication over LDAP access points by aborting any connection where communication is attempted when no previous successful StartTLS operation has taken place. Note that this configuration option does not affect LDAPS access points, as LDAPS enforces encryption implicitly and by definition.

5.7.3 Supported TLS cipher suites

Note: Isode does its best to ensure that regulations governing use and export of cryptographic algorithms are not broken, but ultimately it is the licensee's responsibility to ensure that the appropriate regulations are obeyed with respect to the use of cryptography.

The Directory Server supports a number of TLS cipher suites. Some of the suites are defined by Internet standards (*RFC 2246* and *RFC 3268*), while others are only defined by OpenSSL (and therefore primarily only useful with OpenSSL clients). It supports RSA, DSA, and ECDSA private keys.

Presuming an RSA key is being used, `TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384` from *RFC 5289* is a good choice. (When more than one cipher suite is enabled, the server will select what OpenSSL considers to be the strongest from those that are enabled and which are offered by the client.)

5.7.4 Revocation checking

Certification Authorities (CAs) occasionally wish to revoke a certificate, for example to indicate that the certificate owner has reported that the private key has been compromised. There are two common ways for a CA to communicate this: OCSP and CRLs.

OCSP (Online Certificate Status Protocol) provides a protocol for requesting the status of specific certificates. A CRL (Certificate Revocation List) is a signed list of serial numbers of revoked certificates.

Ordinarily, the CA indicates the location of CRLs and OCSP by adding extensions to issued certificates. For CRLs, they use the `id-ce-cRLDistributionPoints` extension, and for OCSP `id-pe-authorityInfoAccess`, with `id-ad-ocsp`.

When `*CheckCRLs` is set (either `dsaStrongAuthCheckCRLs` or `tlsCheckCRLs`) then URLs in such extensions may be used to retrieve CRLs. If the OCSP choice in the `AuthorityInfoAccessSyntax` extension (as above) is set then OCSP will be used to determine status (and CRLs may not be retrieved). Similarly, if `*OCSPuri` is set then that provides a URI which will be used for checking all certificates.

Notice that these happen regardless of whether the relevant LDAP options are set. Those options enable lookups when distribution points contain `directoryName`, and for lookups of certificate entries even when no suitable extensions are set. Similarly, M-Vault itself will use native lookup, so if it holds the relevant CRLs then they can be used.

OCSP requests are always sent unsigned. When `*OCSPnonce` is set the requests will have the nonce extension, otherwise no extensions will be present. The response must normally be signed by either the certificate's issuer or by a CA Designated Responder (with a certificate issued by the certificate issuer specifically for OCSP). If it is signed by a designated responder then that certificate must have the `id-pkix-ocsp-nocheck` extension (indicating that that certificate's revocation status need not be checked). `*OCSPresponder` can be set to indicate an alternative certificate that will be accepted as an OCSP response signer; ordinarily that would be used along with `*OCSPuri`. If status checking with OCSP fails (because the server fails to respond or the above constraints aren't satisfied) then CRLs will be used instead. Ordinarily, short requests are made using HTTP GET (as allowed in Appendix A of RFC 2560); this can be prevented (forcing use of HTTP POST) by setting `*LookupAvoidOCSPHTTPGET` to `TRUE`.

There are also `*CheckLeaf` attributes which just check the revocation status of the leaf certificate rather than the whole chain. In some environments this may be desirable (it may be that the CAs are carefully managed and do not have revocation information, for example), but usually the more general checking should be preferred. These attributes only have effect when the relevant `*CheckCRLs` attribute is not `TRUE`.

Various kinds of retrieval can be disabled using ***LookupAvoid*** attributes. (This affects both certificate retrieval and revocation checking.)

5.8 Authentication levels

To facilitate access control decisions, all Directory operations on the server take place in the context of an assigned access control Authentication Level; this is normally (but not always) derived from the form of credentials used to authenticate the originator of the requested operation. This authentication level is also passed between Directory Servers when the operation is chained over DSP.

5.8.1 Levels supported

The Directory Server supports the three basic authentication levels of X.501: *None*, *Simple* and *Strong*. These represent increasing levels of authentication of the originator of the operation, and are used by access control to decide whether the originator is sufficiently authenticated to perform part or all of the operation.

Table 5.2. Authentication levels

		Authentication required by Access Control		
		None	Simple	Strong
Originator's authentication level	None	Y	N	N
	Simple	Y	Y	N
	Strong	Y	Y	Y

Authentication level is just one of the inputs into the access control decision function; access may be denied on other grounds even when permitted by authentication level. See [Chapter 6, *Controlling Access*](#) for further details.

5.8.2 Derivation of authentication level

The authentication level of an operation is usually derived from the form of credentials (authentication mode) used by the last **Bind** operation on the association. However, any Directory Server may choose to lower the authentication level associated with an operation, especially one passed to it by a remote Directory server it knows little about. This section describes how the authentication level of a Directory operation (originated by a DAP or LDAP user, not necessarily connected to the local Directory Server) is affected by the authentication mode of the association over which the operation arrives in the server.

5.8.2.1 DAP

The authentication level of incoming DAP operations is derived directly from the authentication mode used in the most recent DAP **Bind** operation on the association.

Bind Mode	Authentication Level
Anonymous	None
Nameonly	None
Simple	Simple
Strong	Strong

5.8.2.2 LDAP

The authentication level of incoming LDAP operations is derived directly from the authentication mode used in the most recent LDAP **Bind** operation. You may configure the local Directory Server to modify this authentication level, to distinguish between a **Bind** in simple mode and a **Bind** in simple over TLS mode.

LDAPv3 may request operations without binding. Such requests are treated as if they had used an anonymous mode **Bind** on the first operation received.

Bind Mode	Authentication Level
Anonymous	None
Nameonly	None
Simple	Simple
Simple over TLS	Simple
SASL	Simple
SASL over TLS	Strong

5.8.2.3 DSP

Operations chained to the local Directory Server over DSP arrive with an authentication level assigned by a remote Directory Server. The local Directory Server may (and usually does) choose to modify this authentication level for the purposes of local processing of the operation, depending on various criteria, both of implementation and policy.

There are a number of factors affecting the local authentication level:

- The authentication level of the DSP association over which it arrived.
- Whether all the Directory Servers it has passed through are “trusted”.
- DSP operation configuration.
- Whether the original DAP operation was signed.

These are described in more detail below.

The DSP-association authentication level is normally derived directly from the authentication mode used in the most recent DSP **Bind**. However if the DAP operation is signed and the signature can be verified, then the authentication level will always be **Strong**, even if the operation was chained over an anonymous DSP association.

Bind Mode	Authentication Level
Anonymous	None
Nameonly	None
Simple	Simple
Strong	Strong

The local authentication level of an incoming operation may never be greater than the authentication level of the DSP association, except if the operation is signed. Signed operations have their signature verified by each Directory Server and (on successful verification) the authentication level for such operations is **Strong**.

The local Directory Server may choose which of its peers to trust to assign an authentication level appropriate to the authentication mode used. Since any intervening peer may modify an authentication level in a chained request, it is unsafe to trust any authentication level higher than **None** if it has come from or passed through a peer which is not trusted by the local Directory Server. The local Directory Server therefore modifies the local authentication level of the operation to **None** if an untrusted peer has been involved.

Absent or adverse DSP operation configuration may also result in the local authentication level being lower than the incoming authentication level. By default, if no DSP operation configuration information is supplied, all incoming operations are assigned a local authentication level of `None`. See [Section 5.3.3, “LDAP v3 \(as responder\)”](#) for further details.

Chained operations which pass through the local Directory Server (that is, incoming operations which are chained on to another peer) keep the authentication level they arrived with. The local authentication level is only used if the operation is processed locally.

Authentication level is not significant to DISP operations. LDAP chaining The M-Vault Server has the ability to “LDAP chain” incoming LDAP operations onto other LDAP Servers. When it does this, the Directory Server is acting as an LDAP client using anonymous authentication; consequently the authentication level of the original LDAP request is not passed on to the new server. Incoming LDAP-chained requests are treated as if they come from an LDAP client; see [Section 5.8.2.2, “LDAP”](#). Note also that LDAP operations may also be chained using DSP, depending on the type of knowledge reference configured for the chained-to peer; in this case normal DSP chaining considerations apply.

5.8.2.4 DISP

Authentication level is not significant to DISP operations.

5.8.2.5 LDAP chaining

The M-Vault Server has the ability to “LDAP chain” incoming LDAP operations onto other LDAP Servers. When it does this, the Directory Server is acting as an LDAP client using anonymous authentication; consequently the authentication level of the original LDAP request is not passed on to the new server.

Incoming LDAP-chained requests are treated as if they come from an LDAP client; see [Section 5.8.2.2, “LDAP”](#).

Note: LDAP operations may also be chained using DSP, depending on the type of knowledge reference configured for the chained-to peer; in this case normal DSP chaining considerations apply.

Chapter 6 Controlling Access

This aim of this chapter is to explain how to define access to objects in the Directory. The M-Vault Server includes a number of security features to control access to and modification of Directory information.

Note: These services do not by themselves provide any guarantees of Directory security but rely on security of the operating systems and hosts on which the Directory Server(s) and any DUAs run, and in some cases also on the networks connecting them.

6.1 Overview of access control

Access control protects entries, attributes and their values against disclosure or modification. Access control regulates what type of operation can be performed on an entry and on an attribute or value. Access control is supported by authentication (see [Chapter 5, Authentication](#)), as access control itself is not concerned with proving identity.

Before performing any Directory operation, access controls are checked by the Directory Server to ensure the requesting DUA has permission to perform the operation. If not, the Directory Server may be permitted to perform only part of the operation (for example, some attributes may be excluded from being returned in a Read), or a security error may be returned, or, in some cases, the existence of the target of the operation may be denied.

There is a special setting that causes all modifications to require signed operations, **isodeRequireSignedModify**. If that is set, then all modifications require signed DAP operations.

A Directory Server may be configured to use a security policy, which in which case operations must also satisfy the checks imposed by that policy in order to succeed (see [Section 6.5, “Security labels and clearance”](#)). It is also possible to require that all write (add, remove, modify, rename) operations must be signed by the DUA, and this requirement takes precedence over the normal access control mechanism.

Access to entries in the Directory can be controlled using either Global Access or Local Access. These are configured separately in Sodium: see [Section 6.2, “Global access control”](#) and [Section 6.3, “Local Access Control Information \(ACI\)”](#).

- Global Access Control uses the Simplified Access Control scheme.
- Local Access Control can use either of the two available access control schemes: Basic Access Control or Simplified Access Control.

They both operate in the same way but the information is stored differently.

Access Control Information (ACI) settings are found in several places in the DIT.

6.2 Global access control

Caution: As the Data Manager and Server Manager are just other users, changes made here can prevent them from binding to the Directory Server, and so can also prevent the Data Manager from undoing erroneous access control changes. Complex access control changes should not normally be attempted on a live production Directory Server. You would have to enable a ‘super user’ mode to resolve this issue.

Configuring Simplified Access Control can be quite complex. The “Global Access Control” mode provided by Sodium abstracts much of this complexity into a view which makes it easier to manage which users are permitted access to which parts of the directory tree. Changes made using the Global Access Control view are translated by Sodium into Simplified Access Control.

6.2.1 Roles, Rules, Items and Precedence

Global Access Control is based on *roles* and *rules*.

Roles are used to specify a set of users and the authentication level that they have. For example, you may have a role which is called “System Manager” which applies to a single specific user when bound using strong authentication, or a role called “France” which applies to all the users below **o=Acme,c=FR** regardless of what authentication they have used.

Rules define a set of access controls which either permit or deny access to certain entries or attributes in the directory. So you might have a rule called “Can modify own telephone number” which permits modification of the **telephoneNumber** attribute by the owner of the entry (but not in entries owned by other people), or a rule “Cannot see devices” which prevents the reading of any entries that have the **device** object class.

Any role may be combined with any rule, and the combination applied to a particular area of the directory tree. This combination appears as a single item line in the Global Access Control editor. For example, you might create an item consisting of the role “France” and the rule “Cannot see devices”, and apply this to all entries below **o=Acme**.

There will be many items combining roles and rules, and each one has a *precedence*, where higher precedence items override lower precedence ones. For example, a low precedence item which denies access to anyone trying to read a **telephoneNumber** attribute will be overridden by a higher precedence item which allows a specific user to read **telephoneNumber** attributes. The standard installation includes predefined roles and rules which can be copied and/or modified to suit your own needs.

6.2.2 Sodium’s Global Access Control View

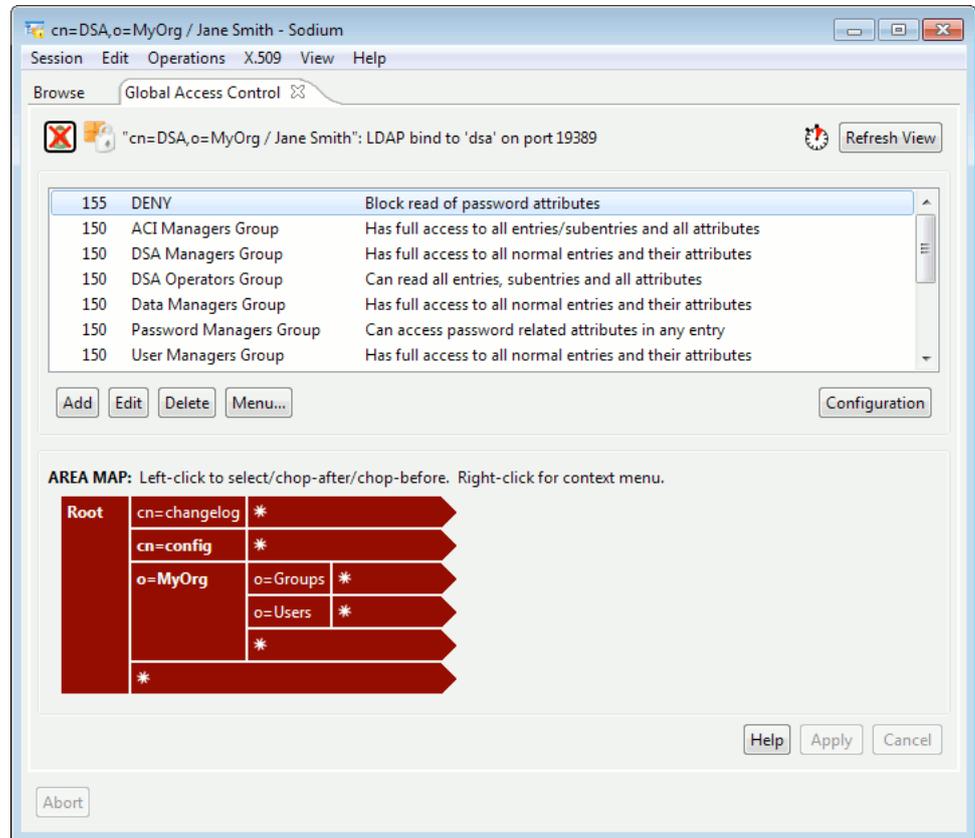
Global access control is set in Sodium, using the **Global Access Control View** from the **View** menu.

This view shows a list of items in order of precedence, each one containing a role/rule pair. The default behaviour for the directory is to allow only access when specifically permitted, and to reflect this, a special fallback item marked “DENY” will always appear at the bottom of the list (below precedence 0). In the lower half of the window, a schematic display of the directory tree is displayed, which reflects the scope of whichever item is selected.

Colours are used to indicate which area the selected item affects, and whether it denies (red) or grants (green) access.

Guidance on using the interface to set access levels is available from the integrated help, displayed by clicking the **Help** button.

Figure 6.1. The Global Access Control view



Note: At the top-right of the window is a small clock icon. Hover your mouse over this clock icon to see how long it has been since the information displayed in this window was refreshed.

6.2.3 Using predefined roles and rules

Figure 6.1, “The Global Access Control view” shows the default values for global access control: these can be modified as required. The upper half of the window shows the roles and rules currently in use. Click a role to see its area of application. Figure 6.1, “The Global Access Control view” shows the effect of the first rule in the list: Block read of password attributes.

- **Add** enables you to add more items using the roles and rules available.
- Select an item and click **Edit** to change its role, rule or precedence, or click **Delete** to remove it altogether.
- Select an item and click **Menu** for a range of options appropriate for your selection. **Item Notes** displays information about the roles and rules referenced in the selected item. You can also (where appropriate) choose to view the role entries in a **Browse** view.

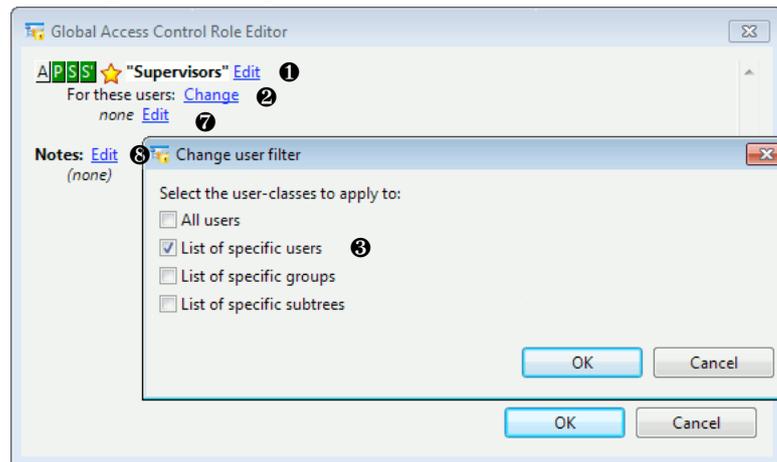
When you have made your changes, click **Apply**.

6.2.3.1 Creating and modifying roles

To create a new role, open the **Global Access Control View** from the **View** menu and click the **Configuration** button.

Click **New Role**, or select a role and click **Modify** to change an existing one.

When you create a role, you have to specify the name of the role and the minimum level of authentication required. The next steps are the same whether you are creating a role or modifying one. The interface guides you through the process, and the following example is shown to help you to understand the steps involved. The numbers in the images correspond to the steps in the example.



1. A new role is created called **Supervisors**. The name can be changed if necessary by clicking the **Edit** link to its right.

This role requires at least simple binds, shown by the APSS icons. A green background means that level applies:

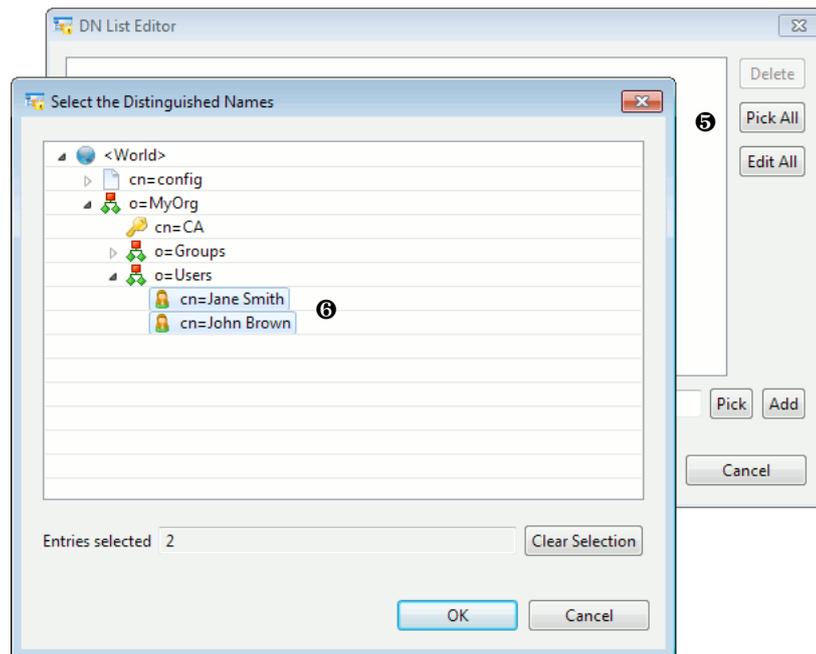
- **A** represents anonymous binds
- **P** represents password protected (simple)
- **S** represents strong binds but unsigned operations
- **S'** (S with a prime sign) represents strong binds with signed operations.

As the level is set to at least simple binds, **P**, **S** and **S'** are all green.

2. By default, the role applies to all users. For this example, the role will incorporate specified individuals: click **Change**.
3. The **Change user filter** box is shown. You can choose to apply the role to users, groups or subtrees. For this example, select **List of specific users**.

Note: The next step is the same for all options other than **All users**, which was the default.

4. The **Global Access Control Role Editor** now shows that the role will apply to specific users.
 - To return to the previous step and select another option, click **Change**.
 - To specify the users, click **Edit**. The **DN List Editor** opens.

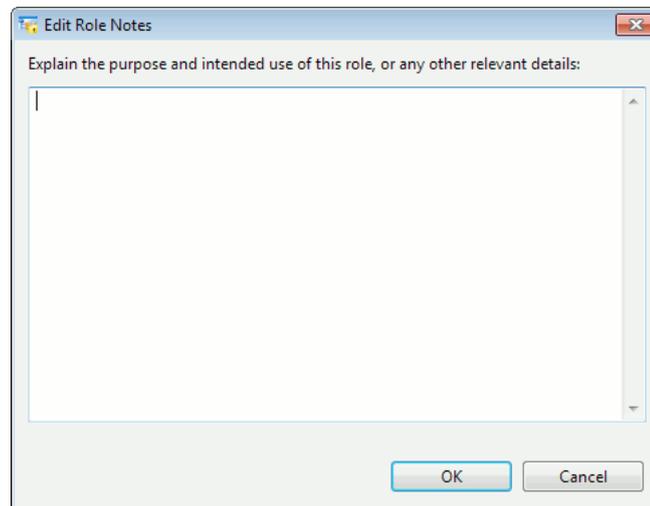


5. Click either **Pick All** or **Pick** to open a view of the Directory tree.
 - **Pick All** enables you to select several names at the same time using the standard keyboard options (**Ctrl** and click on Windows).
 - If you choose **Pick**, you have to select the entries one at a time.

Alternatively, type the DN of an entry in to the field at the bottom of the **DN List Editor** and click **Add** to add this DN to the list of those selected.

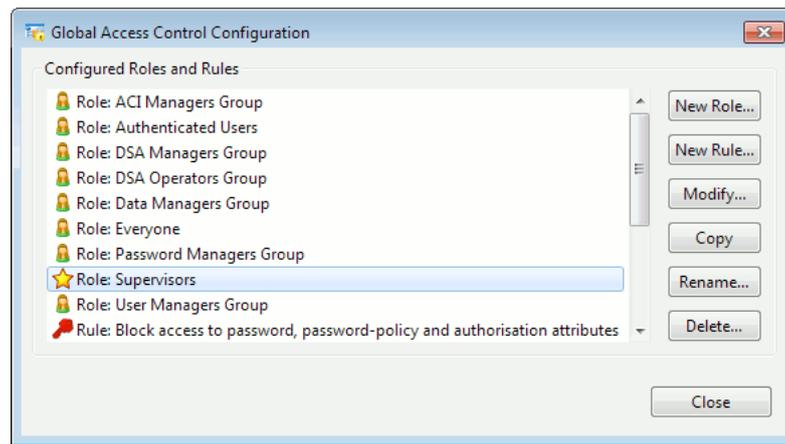
6. When you have chosen your entries, click **OK** to close the selection box. In this example, two person entries have been selected.

Figure 6.2. Creating notes for the new role



7. The selected users are shown. Click the **Edit** link to make changes.
8. Click **Edit** to the right of **Notes**. The **Edit Role Notes** box opens, in which you should provide information that would be helpful to anyone reviewing the purpose of this role in the future.

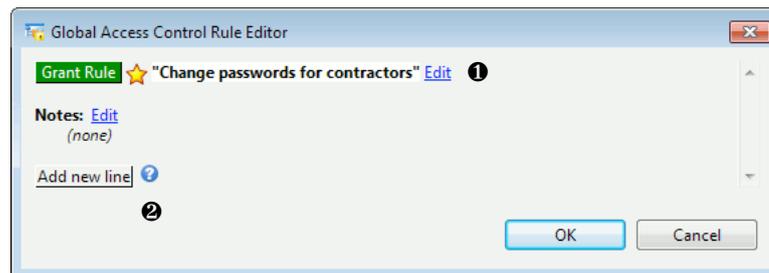
The new role is displayed in the list of roles and can be used to configure access.



6.2.3.2 Creating and modifying rules

Create and modify rules following the guidance on screen in a similar way to the way that roles are created and modified.

1. When first creating a rule, it must be given a name and you must specify whether the rule is going to grant or deny access. All other information is specified once the rule exists. You can change its name later.
2. The details of the rule are specified by adding restrictions one line at a time, clicking **Add new line** to do so. Detailed guidance can be obtained by clicking the question mark icon.



6.2.3.3 Making library roles and rules available

Before starting to create your own roles or rules, it is worth seeing if any of the rules in the library are suitable. Some of the standard roles and rules may have been removed if **Cleanup** has been run in the past. To reinstate them:

1. Click **Library**.
2. Select any roles or rules you want to add to the list of those available.
3. Click **OK**.

6.2.3.4 Importing and exporting roles and rules

Roles and rules can be imported from and exported to xml files. You can only export your entire global access control configuration, but when importing you can specify that you only want to import roles and rules for a subtree specified by a DN. This provides you with a simple mechanism for backing-up your global access configuration before making changes and, if necessary, reverting just a single subtree back to its original state.

6.2.3.5 Removing unused roles and rules

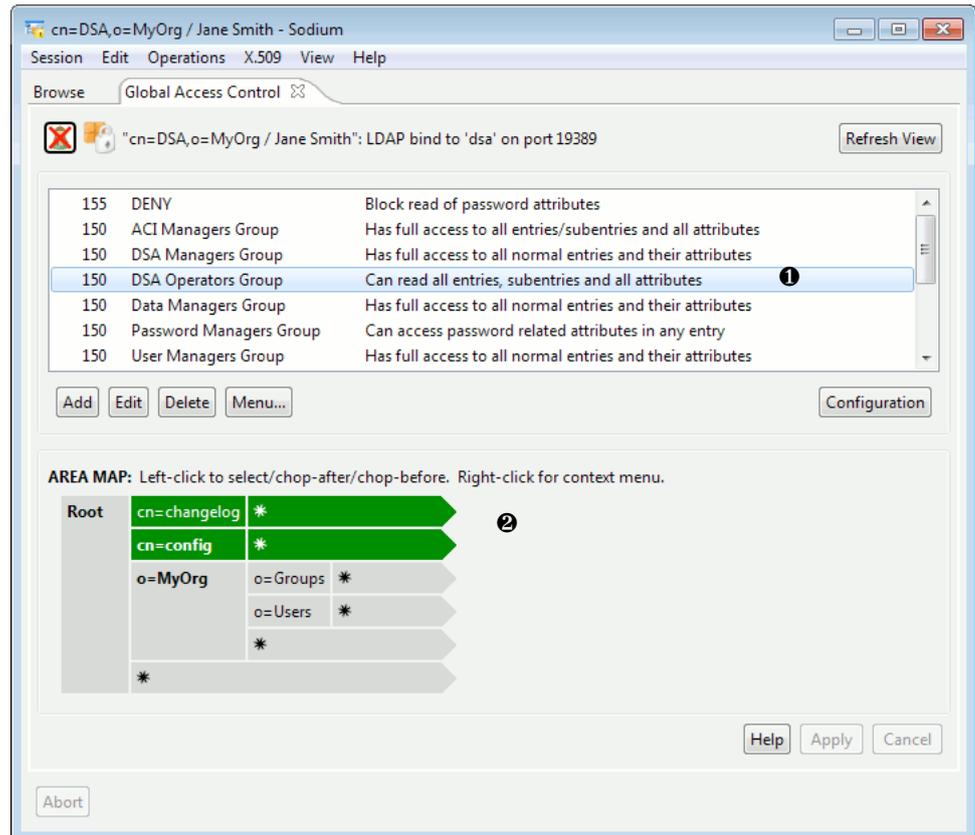
The **Cleanup** option streamlines the display by telling you which roles and rules are unused and letting you remove them.

6.2.4 Modifying the area map

The area map shows the structure of the DIT rotated through 90 degrees, with the root on the left and leaf entries (where shown) on the right.

The coloured background of the map indicates the area where the currently selected rule applies. For example, when the Global Access Control view is first opened, the rule at the top of the default list (a DENY rule, preventing password attributes from being read) is selected and the whole of the map is red.

If you select the 4th rule from the top - the rule that grants permission to members of the DSA Operators Group to read all entries and all attributes - you will see that it applies to the **cn=changelog** and **cn=config** branches of the DIT (shown in green below).



Note: It is possible to add DN's to the diagram that do not yet exist.

To apply a rule to another part of the DIT, click the section of the map representing it. You can also reduce the scope of a rule by clicking on a coloured block to de-select it. Clicking on an entry in the map toggles through three possibilities: selection, chop after and chop before.

To limit the complexity of the display, the wildcard character (*) represents all entries in a portion of the DIT - you can, however, add a specific entry if you want to include it in or exclude it from a rule. A yellow star to the left of a rule indicates that its scope has been changed - and you can revert to the previous setting by right-clicking on a rule and selecting that option from the menu.

Detailed instructions on navigating the map are provided in the online help, as previously described.

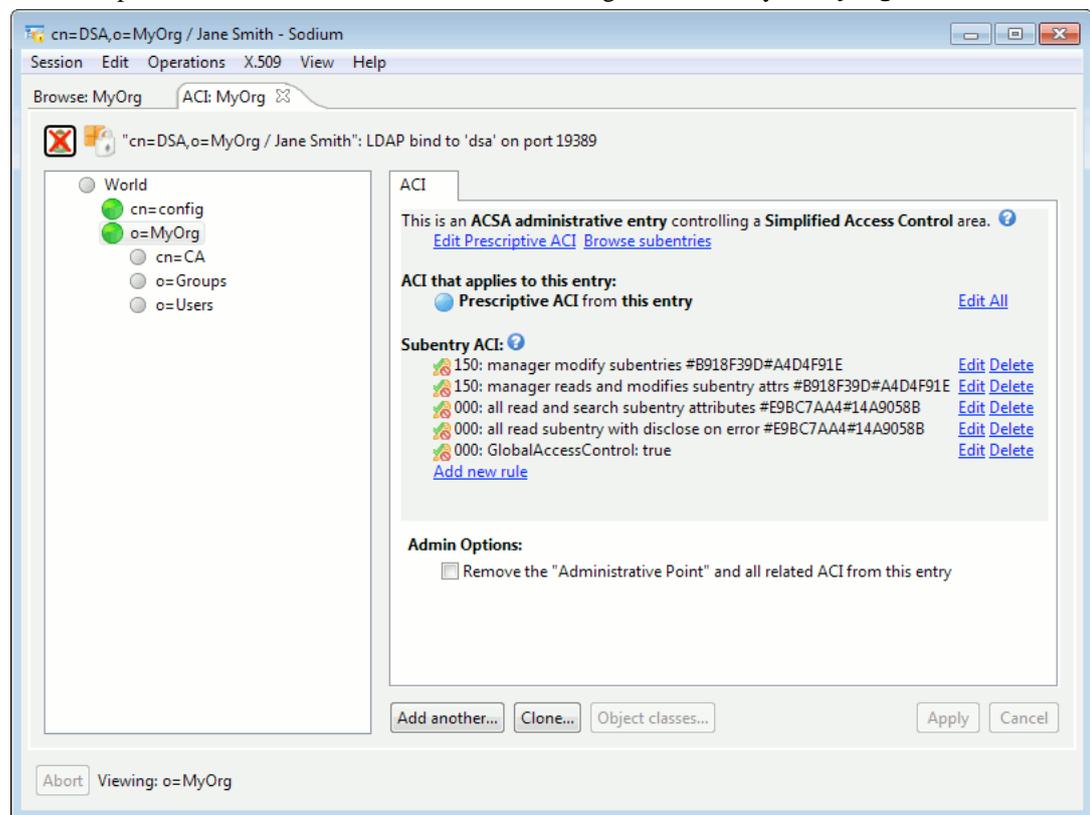
6.3 Local Access Control Information (ACI)

M-Vault supports Basic Access Control (BAC) as defined in X.501, and Sodium allows the Access Control Information (ACI) rules for BAC to be viewed and edited.

For convenient reference, a ‘quick guide’ help-text is available within Sodium by clicking on the help icon.

ACI information can be displayed for each entry using a separate ACI page. To open an ACI page, first select the entry in the tree, then select **New Local ACI view** from the **View** menu.

The example below shows the ACI information for a organization entry, **o=MyOrg**.



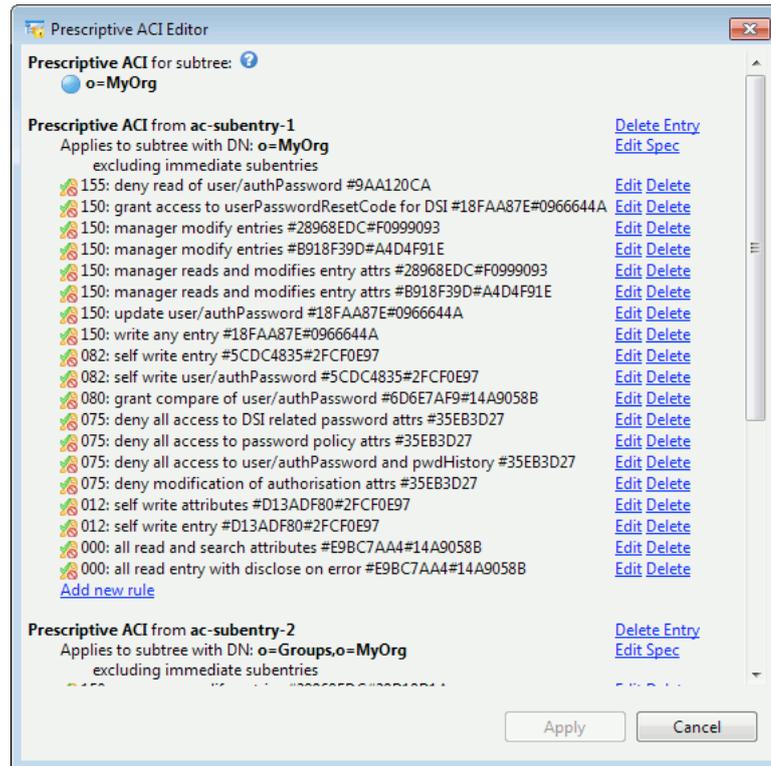
Note: The ACI shown in the example above has been created by the GAC editor. The GAC editor detects when the local ACI editor has been used to manipulate ACI and appends hex values to the information.

An ACI page has two purposes:

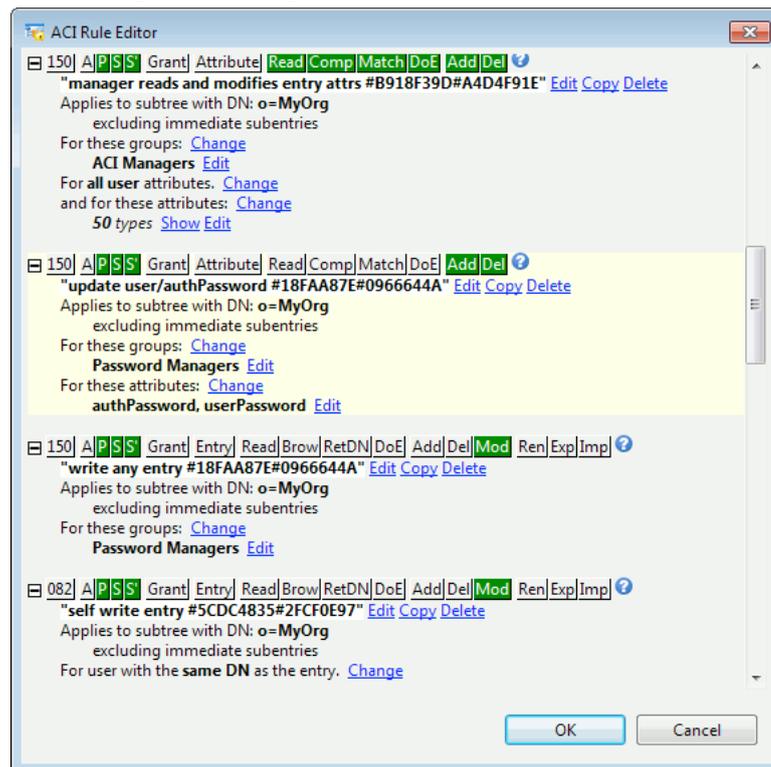
- To show the list of sources of ACI rules that affect operations on this entry, and provide navigation to the entries containing those rules to view or edit them.
- To allow editing of the ACI rules contained *within* this entry. These will typically have an effect on entries elsewhere, except in the case of Entry ACI.

In the case of Prescriptive ACI, the ACI rules are stored in subentries of the administrative entry (see [Section A.3.1.2, "Security"](#) for an explanation of administrative entries), but they are considered to be logically associated with the administrative entry. To view Prescriptive ACI rules, open a **Local ACI View** of the administrative entry, then click the

Edit Prescriptive ACI link at the top of the page. This shows all the Prescriptive ACI rules together, along with their subtree specifications. This dialog also provides a way to add new access control subentries, via the **Add new subentry** link.



The **ACI Rule Editor** window is where the actual editing takes place. The header line of each rule shows its precedence, the levels of authentication that it applies to, whether it grants or denies access, its scope (attribute or entry), and the specific permissions that are granted or denied. A green color is used to indicate aspects that have been granted, and a red color for aspects that have been denied. All of these buttons have tooltips to help explain their meaning in context. Below the header are additional restrictions or limitations that apply to this rule. All link text can be clicked to make immediate changes or to pop up editor dialogs for those aspects.



The whole list of rules may be expanded or shrunk by using the plus/minus icons or by clicking on the background of the currently-selected rule. The list may be sorted by **Scope** (entry or attribute), **Precedence** and **Name** using controls at the top of the list.

6.3.1 Access control held in the entries

In Basic Access Control, access control items can be stored as values of the **entryACI** operational attribute in the entries themselves. Each value is a single item, and there may be multiple values to this attribute. Access controls held in this attribute only affect the individual entry itself and its attributes, and have no effect on other entries.

However, the **AddEntry** operation checks that **Add** permission has been granted to the DUA to add the entry and all its attributes. As the entry does not already exist, the **entryACI** attribute cannot be used, nor can a supplied **entryACI** attribute be used, as that would allow anyone to add entries.

6.3.2 Access control held in subentries

Access control subentries solve this problem. Subentries are special entries located immediately subordinate to an access control Administrative Point, and are not visible to normal operations. They have an attribute **subtreeSpecification** which defines the subset of entries over which they are effective, typically to the leaf entries or to a subordinate naming context or another administrative point. The access control subentry has an attribute **prescriptiveACI** whose values are used in the access control decision function for all entries in that subtree. Thus, items in **prescriptiveACI** must grant the manager **Add** permission if new entries are to be added to the subtree.

Access control subentries can also be used to reduce the problem of managing **entryACI** in a large number of entries. If all entries in a subtree have the same access control, then the common items can be placed as values of the **prescriptiveACI** attribute in an access control subentry, and those values can be removed from the **entryACI** attributes in all the entries.

This is taken further by the Simplified Access Control variant of Basic Access Control. In this scheme, **entryACI** is ignored, the only access control used is that in **prescriptiveACI** or **subentryACI**.

In order for a subentry to be used, the parent entry must be an Administrative Point with an administrative role of Autonomous Administrative Area, Access Control Specific Area, or Access Control Inner Area.

6.3.3 Access control held in the administrative point

Items from the **prescriptiveACI** attribute do not affect subentries themselves. In order to add a new subentry via protocol, an **Add** permission must be granted in a **subentryACI** attribute in the administrative point itself.

6.4 How access control is determined from ACI

The mechanism used to determine access control from local access control information is known as the Access Control Decision Function (ACDF).

6.4.1 Access Control Decision Function inputs

The inputs to the Access Control Decision Function are:

- The identity of the requester: its Distinguished Name (DN)
- The authentication level associated with the requester, either None, Simple or Strong (see [Section 5.2.1, “Establishing identity”](#)), and whether the operation is signed
- The target entry: the Distinguished Name (DN) of the entry to which the requester is trying to gain access
- The set of Access Control Information (ACI) items which affect that entry (see [Section 6.3, “Local Access Control Information \(ACI\)”](#))
- Optionally, a specific type or value in the entry
- The type of permission requested.

For an entry the type of permission can be one of the following:

Disclose on error	Read	Remove
Return DN	Add	Import
Browse	Modify	Export

For an attribute or value it can be one of the following:

Disclose on error	Compare	Add
Filter match	Read	Remove

6.4.2 Access Control Information ACI items

Access Control Information (ACI) items define the access control policy for an entry. They are located either in the entry itself ([Section 6.3.1, “Access control held in the entries”](#)), in a parent administrative point ([Section 6.3.3, “Access control held in the administrative point”](#)) or in a subentry ([Section 6.3.2, “Access control held in subentries”](#)).

The most useful fields are:

- The Item name, which identifies this ACI item to the administrator
 - This is ignored by the Access Control Decision Function.
- The Precedence level (0-255) of this item. If there are multiple items, some of which grant permission and some deny, the higher level will override.
- The minimum Authentication Level to be considered as part of the User Class (see below), either None (anonymous), Simple or Strong.
- The User Class to which this item grants or denies access: all users. the user with the same name as the implied entry. explicitly named users. members of a group. users within a subtree of the DIT.
- Protected items, which can be:
 - The entry itself (the identity of the entry is implied by the location of this Access Control Item in the Directory, see [Section 6.3, “Local Access Control Information \(ACI\)”](#)).
 - All user attribute types and their values.
 - Explicitly listed types and values.
 - A value of a DN-valued attribute which is the same name as the requester (so that a requester can add or remove their own name from a membership list without being able to affect the names of any other members on that list).
- Grants or denials of Permissions (see [Section 6.4.1, “Access Control Decision Function inputs”](#) above for the permission types).

A typical set of items are created automatically when M-Vault is installed. Before making changes, we suggest you familiarise yourself with these and make sure you understand their effects and how they have been configured.

6.4.3 Access control decision function (ACDF) rules

The ACDF processes the information items and arrives at an answer: either grant permission or deny permission. In doing so, it makes use of a number of rules:

- Permissions must be explicitly granted; if there are no items which grant access, permission will always be denied.
- Only the item with the highest precedence relevant for this type of permission is used.
- Items which specify the most specific user class or protected value are preferred. For example, there could be two items with the same precedence, one of which denies Modify permission of an entry for all users, the other which grants Modification permission of the entry for a specific named user. The latter will be honoured for that particular user.
- If an item requires signed and the operation is not signed, the item does not apply.
- If a conflict remains that cannot be resolved by following the above rules, permission is denied.

To prevent unexpected grants or denials, it is recommended that items which could conflict be given different precedences, and that explicitly denying permission be avoided, except in certain cases (for example, allow reads of all attributes except the **userPassword**).

6.4.4 The effects of ACL on operations

Table 6.1, “Permissions required for various operations” summarizes what permissions are required in order for operations to be performed.

Table 6.1. Permissions required for various operations

Operation	Required permission: entry	Required permission: attributes
Bind	Read	Compare userPassword , unless a SASL bind is being attempted
SASL bind	Read	Read userPassword attribute.
Read	Read and ReturnDN	Read for each requested type and its values
Compare	Read	Compare of the type and a matching value
List	Browse and ReturnDN of each matching entry	None required
Search	Browse and ReturnDN of each matching entry	FilterMatch for types and values used to match the filter, Read for types and values returned in the result
AddEntry	Add	Add for each type and value supplied
RemoveEntry	Remove	None required
ModifyEntry	Modify	Add or Remove for each value specified in the entry modification
ModifyDN (same parent)	Modify	Add or Remove for each value in the RDN

Operation	Required permission: entry	Required permission: attributes
ModifyDN (new parent)	Export on the original entry, and Import on the new parent	Add or Remove for each value in the RDN, otherwise none required
Modify passwords	Modify	Add and Remove for the userPassword attribute

6.5 Security labels and clearance

This section covers access and administrative controls based upon security labels, clearances, and associated security policy.

6.5.1 Introduction to security labels and clearances

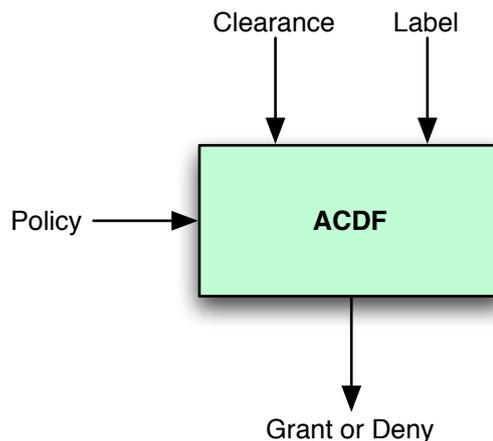
Security labels and clearances may be used, in conjunction with a security policy, to determine whether a user’s access to an object is to be granted or denied.

The determination is made based upon a label (or lack thereof) which indicates the sensitivity of the object (e.g., object is “SECRET”) and a user’s clearance to access sensitive information (e.g., user holds “SECRET” clearance).

6.5.2 Interaction with simplified and basic access controls

Security Label/Clearance Access Controls complement Simplified and Basic Access Controls (see [Section 6.4.3, “Access control decision function \(ACDF\) rules”](#)). Both subsystems’ access control decision functions (ACDF) must yield “grant” for access to be granted.

Figure 6.3. Security label/clearance Access Control Decision Function



If the Security Label/Clearance ACDF yields “deny”, then access is denied regardless of the outcome of the Simplified or Basic ACDF. Likewise, if the Simplified or Basic ACDF yields “deny”, then access is denied regardless of the outcome of the Security Label/Clearance ACDF.

6.5.3 Security policy configuration

The security policy governing security label and clearance access and other administrative controls is held in the **rbacSecurityPolicy** attribute in the Directory Server’s entry. This

attribute holds an XML representation of a Security Policy Information File (SPIF), as defined by SDN 801c.

The XML representation may be created through the use of an XML editor or text editor, or by BER to XML conversion tools which are available by request from Isode.

Changes to the **rbacSecurityPolicy** attribute do not become effective until the Directory Server is restarted.

In the absence of a valid security policy at startup, Security Label/Clearance access and administrative controls are disabled.

6.5.4 User clearances and object security labels

A user's clearance may be specified by the **clearance** attribute held in the user's entry. The **clearanceObject** auxiliary object class is provided to allow user entries to be augmented by this attribute. A change to a user's clearance becomes effective upon their subsequent Bind to the Directory Service. Sodium may be used to add clearance attributes to users, see [Section 3.11.5, "Applying a clearance to an entry"](#).

A default clearance (for users whose clearance is not specified in the **clearance** attribute) may be specified via policy using the default security policy data mechanism. Commonly, the default clearance is either 'Unclassified' or none (all access is denied).

An object's security label is specified with the **unsignedSecurityLabelInfo** attribute of the object. This attribute is operational. A change to an object's security label becomes effective immediately. Sodium may be used to add security labels to objects, and displays relevant marking data when entries are shown. See [Section 3.11.4, "Applying a label to an entry"](#).

A default security label (for objects whose security label is not specified in the **unsignedSecurityLabelInfo** attribute) may be specified via policy using the default security policy data mechanism. Commonly, the default security label is either 'Unclassified' or none (all access is denied).

Note: The DSA Manager is exempt from these access controls.

6.5.5 Directory Server clearance and security labels configuration

The Directory Server may be configured to restrict how objects may be labeled. For instance, a Directory Server may be configured such that all objects must have "SECRET" labels. This is accomplished by adding a **clearance** attribute to the Directory Server entry. In addition to other access control checks, the ACDF must return "grant" for the Directory Server clearance and the object's label for access to be granted. Changes to this attribute are effective upon restart of the Directory Server.

The Directory Server may also be configured to require that users hold a suitable clearance to utilize the Directory Service. For instance, a Directory Server may be configured such that all users must have clearance granting access to "SECRET" data. This is accomplished by adding a **securityLabels** attribute to the Directory Server's entry, specifying one or more security labels. If the user clearance fails to grant access to at least one of these

security labels, the user's Bind request will be rejected. Changes to this attribute are effective upon restart of the Directory Server.

Note: The Security Label mechanism does not preclude anonymous access. If anonymous access is not desired, it may be disabled via Basic or Simplified Access Controls.

Note: The Data Manager is exempt from these access and administrative controls.

Chapter 7 Connecting Directories

You may need more than one Directory Server to provide the Directory Service. This chapter cover creating additional servers, creating knowledge references (superior, subordinate and cross references) and specifying the authentication levels for connection.

7.1 Overview

Communication between Directory Servers is required for:

- Chaining and/or referral of operations (for background information, see [Section A.3.3.1, “Interactions between Directories and DUAs”](#)).
- Shadowing (for background information, see [Section A.3.4, “Shadowing”](#)).

In both cases, you have to make changes for each Directory Server involved in communication with other Directory Server(s).

When a Directory Server handles an operation, it needs to resolve where the target entry (for example, the start of a search, the entry being modified or the parent of an entry being added) is located. It does this using knowledge of the naming contexts that it holds.

The Directory Server can also have references to remote naming contexts that are subordinate to ones that it holds (subordinate references) and to remote naming contexts that are not directly subordinate to ones that it holds (cross references).

If subordinate and cross references are present that can be used to progress an operation, the Directory Server will chain using information in those references. To create a subordinate or cross reference you need to know:

- the DN of the naming context held on the remote Directory Server
- the DN of the remote Directory Server, and its presentation address
- whether the remote Directory Server masters the naming context being referenced, or simply holds a copy.

If the Directory Server does not have the target entry in a local naming context, and does not have any relevant subordinate or cross references, the Directory Server will chain to the Directory Server using information held in its superior reference.

Directory Servers mastering naming contexts directly below the root are called *first level Directory Servers*, and have no superior reference. Other Directory Servers do.

References are not bi-directional: a Directory Server used as a target for a subordinate reference may *not* have any direct knowledge of the naming contexts held on the Directory Server chaining to it.

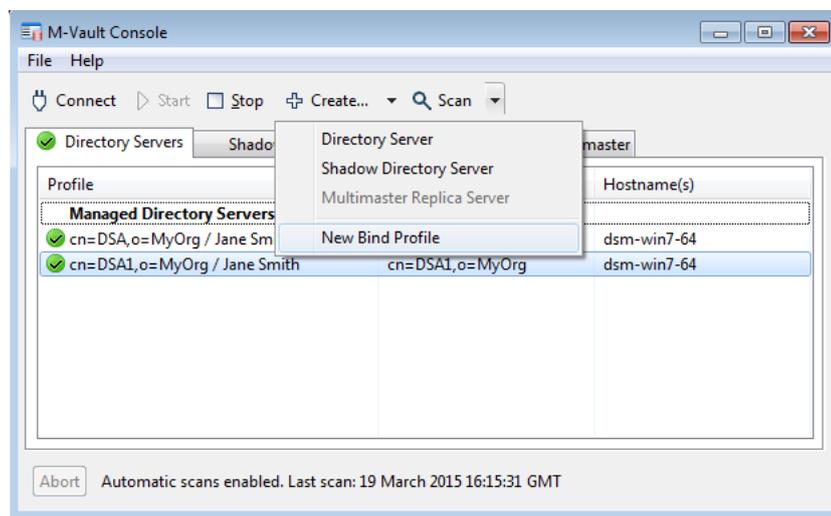
You can make references to Directory Servers that you do not manage.

7.2 Connection details for Directory Servers

Before you can connect to another Directory Server, whether it is one you are managing or not, your Directory Server needs to know connection details: the presentation address of the other server, its DN and other connection information. This information is recorded using M-Vault Console. For servers you have created using M-Vault Console and are managing, this information is already stored in a bind profile.

For servers you have not created and are not managing:

1. Start M-Vault Console and click **Create** on the tool bar.
2. Select **New Bind Profile** from the list.



3. Select:
 - **Managed Server** if you want to create a new access point for a server you manage.
 - **Known Server** if you want to record details of a server that you do not manage.

For the remainder of this section, the instructions assume you have selected **Known Server**. For information relating specifically to a **Managed Server**, see [Section 2.2.3, “Creating a Directory Server”](#).

4. Enter the **Hostname** of the Directory Server and click **Next**.
5. Check that the port numbers are correct. If they are not, select them and click **Edit**.

See [Section 2.2.3.3, “Specifying a presentation address”](#) for information on specifying a full presentation address, including selectors.

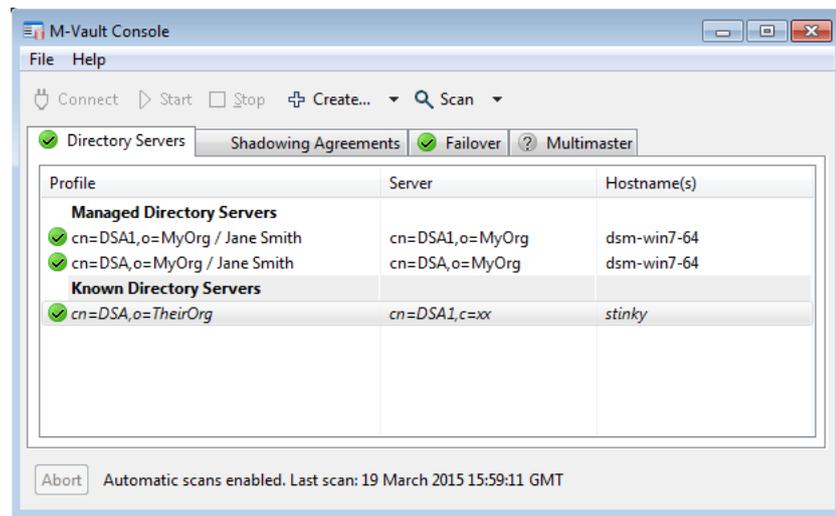
Click **Next**.

6. Type the DN of the Directory Server.

Click **Next**.

7. Give the profile a name so it can be recognised in a subsection of the list of **Directory Servers**.

Click **Finish**. A new bind profile will now be shown in the list of **Known Directory Servers**.



7.3 Configuring knowledge references

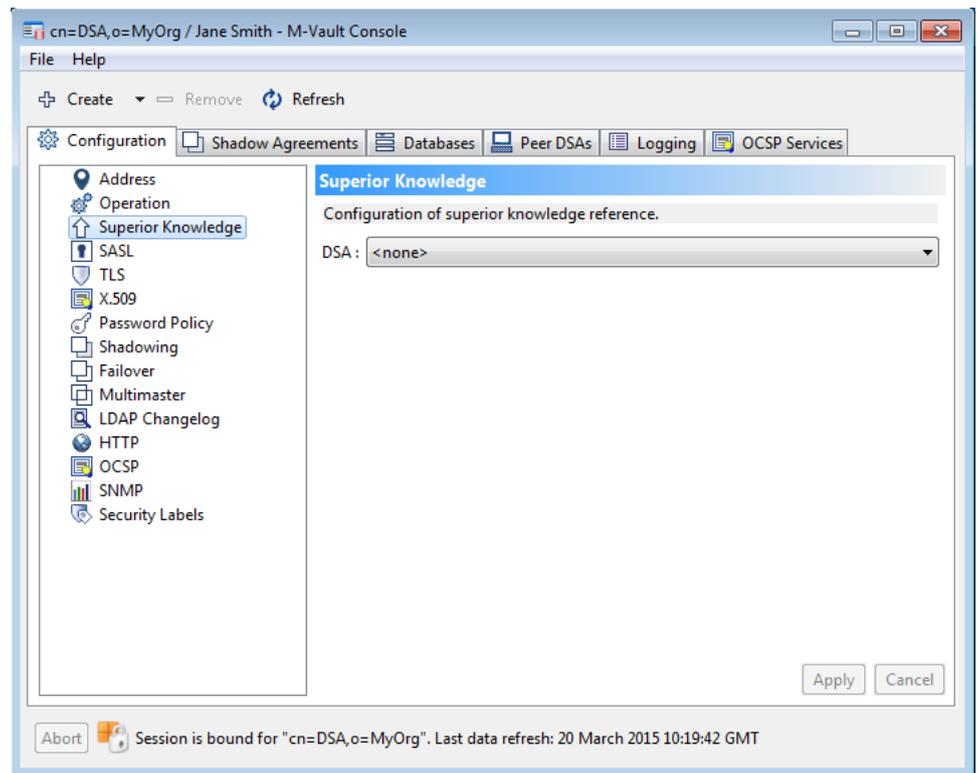
Once you have bind profiles for more than one Directory Server, you have to configure the references between them so that they can communicate with one another.

7.3.1 Superior reference

A superior reference is used to search other Directory Servers, higher in the hierarchy, for information that cannot be found in the area covered by your Directory Server. (See [Section A.3.3.1, “Interactions between Directories and DUAs”](#) for an explanation of superior references.)

You can create a superior reference within M-Vault Console.

1. Bind to a Directory that you are managing using an appropriate level of access.
2. Select **Superior Knowledge** in the list of options on the left side of the **Configuration** page.



3. Select the **DSA** that holds information at a higher level in the tree than this Directory Server.

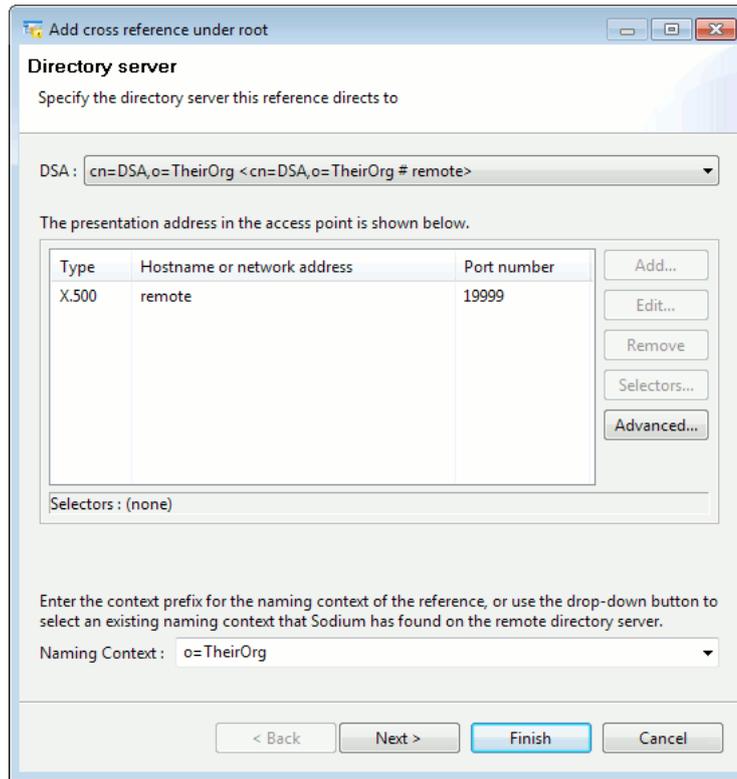
A bind profile must exist for the Directory Server (either as a **Managed** or a **Known** server) for it to be shown in this list. See [Section 7.2, “Connection details for Directory Servers”](#) for information on creating bind profiles for **Known** servers.

4. Click **Apply** to save the changes.

7.3.2 Subordinate and cross references

Subordinate and cross references are created within Sodium: Sodium automatically determines whether a reference should be a cross-reference or a subordinate reference depending on the naming context of the data on the remote server. As with a superior reference, bind profile information must be recorded for the Directory Server you are going to reference.

1. Open Sodium and bind to the Directory Server from which you are going to create a reference.
2. Switch to **Admin** view using **View** → **New Knowledge Reference view**.
3. Select the root of the Directory tree (**World**).
4. Right-click and select **Add reference...** from the menu shown.
5. Select the DSA holding the information you want to reference. The address of that DSA is displayed automatically.

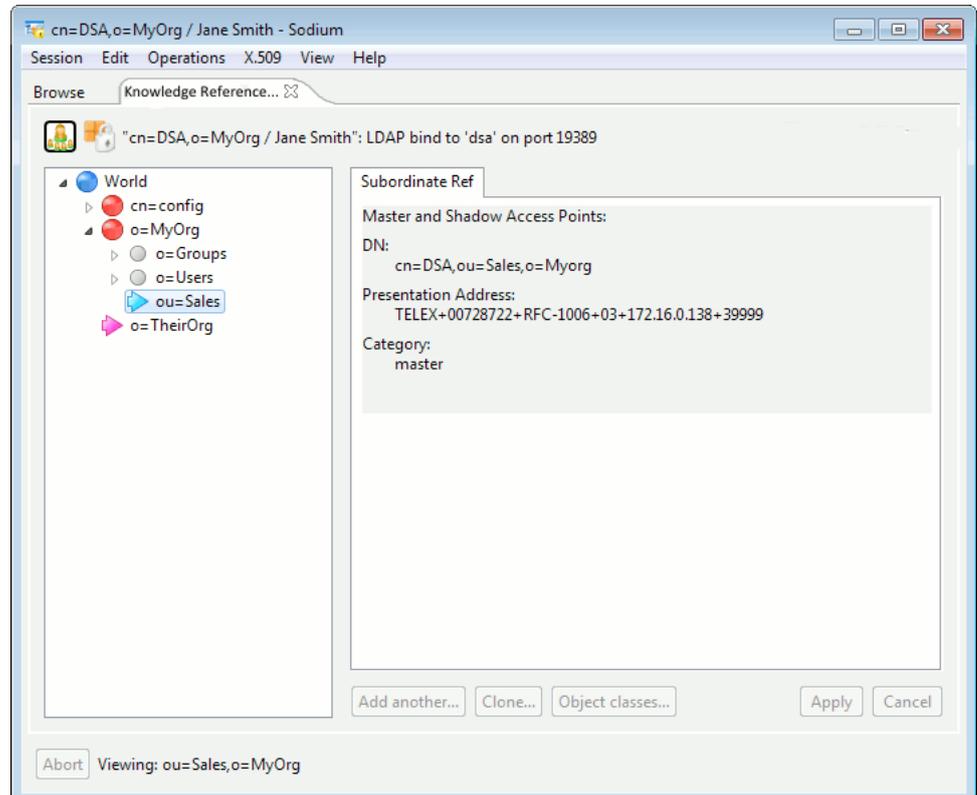


Select or enter the **Naming Context** of the information on the remote server. The naming context chosen does not have to be directly below the root of the Directory.

Click **Next**.

6. Select the database that will hold the reference information. Click **Finish**.

Sodium now shows the existence of the new cross reference or subordinate reference. The example below shows a subordinate reference to **ou=Sales,o=MyOrg**.



Cross references, such as the one to **o=TheirOrg**, are shown with a pink icon.

7.3.2.1 Viewing and changing reference properties

1. In Sodium, open an **Knowledge Reference** view and select the reference.
2. The right hand pane contains summary information about the reference. Note that this pane is read-only. To change these values, you can use Dmish - see [Section H.7.6](#), “[Modify managed object](#)”.

7.3.2.2 Deleting a reference

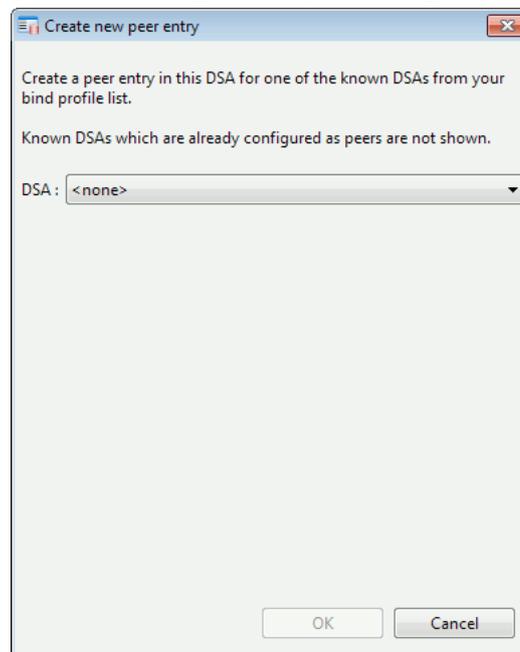
To delete a reference, view the reference (using **Knowledge Reference** view). Right-click on it and select **Delete** from the menu. You are asked to confirm the deletion.

7.4 Securing connections between Directory Servers

When one Directory Server connects with another Directory Server, it does using either *DSP* (for chaining) or *DISP* (for shadowing).

To specify authentication information for two Directory Servers that are communicating with each other, you need to create a **Peer Configuration**.

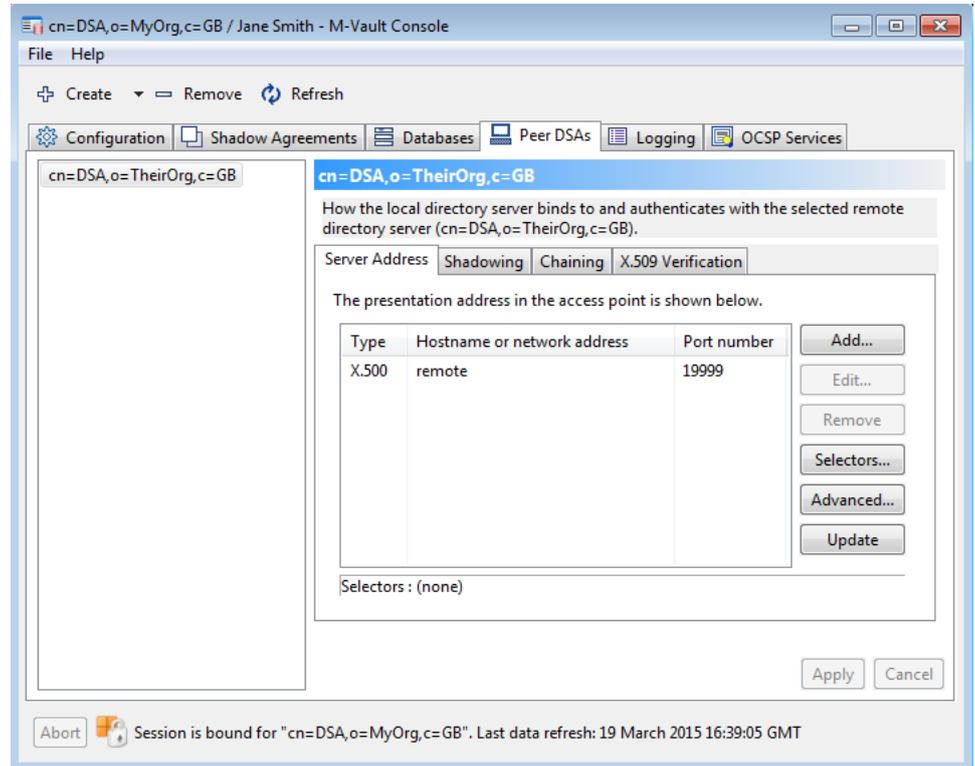
1. Start M-Vault Console and connect to the **Managed** server for which you want to specify connection details.
2. Choose **Create** → **Peer Configuration** from the toolbar.
3. The **Create new peer entry** window opens.



Select the Directory Server that you want to create a peer entry for from the list and click **OK**.

Note: Only Directory Servers for which you have created a bind profile (either **Managed** or **Known**) that are not already configured as peers are shown in this list.

- The selected Directory Server is now shown on the left side of the **Peer DSAs** page. If several peer DSAs configurations have been created, they will be listed here: make sure that the DSA you are providing details for is selected.

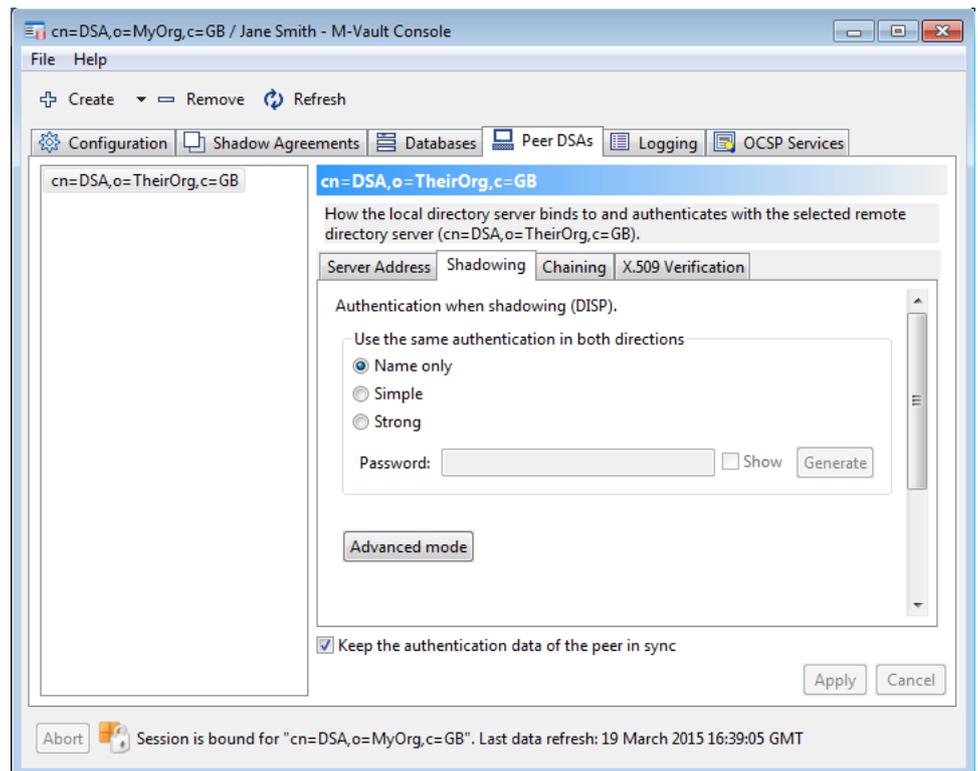


The **Server Address** of the Directory Server is shown in the list on the right. It can be changed here if necessary.

7.4.1 DISP authentication

The DISP protocol is used when shadowing, and you can specify the authentication mode that the server you are managing will use when it initiates or responds to a bind request from a remote DSA. See [Section 5.3.4, “DSP \(as initiator or responder\)”](#) for information on the effects of choosing modes of authentication for the DISP protocol.

- From the **M-Vault Console** window, click **Peer DSAs**. Select the peer that you are configuring from the list on the left.
- Click **DISP** to configure the authentication level for DISP connections (when shadowing) between the two Directory Servers.



3. The default is to use the same authentication mode for initiating and responding. You can choose to specify them individually using the **Advanced mode** button. The available modes are:
- **Name only** – no other details are required.
 - **Simple** – you must specify a password. Either type one into the field or click **Generate** to create one automatically. To see the characters of the password, select **Show**.
 - **Strong** – the Directory Server must have an X.500 identity (a certificate).

Note: M-Vault only accepts *Simple* or *Strong* bind requests, so do not choose *Name only* if the peer DSA is another Isode DSA.

7.4.2 DSP authentication

The DSP protocol is used when chaining or referring. You configure the authentication mode to use in exactly the same way as you do for the DISP protocol (see [Section 7.4.1, “DISP authentication”](#)), using the **DSP** page. See [Section 5.3.4, “DSP \(as initiator or responder\)”](#) for information on the effects of choosing modes of authentication for the DSP protocol.

You have an extra option when choosing authentication mode:

- **Anonymous** – no identifying information is passed between the Directory Servers.

7.5 Distributive changes

To distribute the Directory Service over more than one Directory Server, you must:

1. Set up a new Directory Server following the steps in [Section 2.2, “Creating a Directory Server”](#). If the naming context mastered by this new Directory Server is not immediately beneath the root, you will need to supply a superior reference to another Directory Server (see [Section 7.3.1, “Superior reference”](#)).

Note: It is possible to set up Directory Server which will only hold a copy of a naming context mastered by another Directory server.

2. Check the new Directory Server’s initial configuration. See [Section 4.5, “General configuration of the Directory Server”](#) for how to check the default settings in M-Vault Console. For example, you may wish to change:
 - Policies with regard to searching, chaining or referral and shadowing.
 - The authentication requirements when communicating between Directory Servers.[Section 5.2.1, “Establishing identity”](#) gives background information on authentication.
3. Configure the references between servers:
 - Set up subordinate references to the naming context(s) of the new Directory Server. This has to be done on all the Directory servers with naming contexts superior to that of the new Directory server. Set up any required cross references between servers. See [Section 7.3.2, “Subordinate and cross references”](#).
 - Set up any required shadowing agreements. See [Chapter 8, Shadowing](#).
4. Load the Directory user information for the new Directory server, if this has not yet been done. See [Section 3.8, “Importing and exporting entries”](#).

7.6 Using a Directory Server as a connection

You can create a Directory Server to use as a connector between two other Directory Servers. To do this:

1. Create a Directory Server, choosing the option that incorporates a superuser account (see [Section 2.2, “Creating a Directory Server”](#)).
2. Create bind profiles for the Directories you are going to connect (as known servers) using M-Vault Console (see [Section 7.2, “Connection details for Directory Servers”](#)).
3. Connect to the Directory using Sodium. Delete all existing entries.
4. Create cross-references to the two servers you are connecting (see [Section 7.3.2, “Subordinate and cross references”](#)).

Chapter 8 Shadowing

Making sure that Directory information is close to those who need it minimises access times. This is achieved by shadowing – or replicating – information so that more than one Directory Server holds a copy of the same information.

8.1 Overview

Replication of Directory information is called shadowing. Two Directory Servers establish a shadowing agreement with each other, where one Directory agrees to supply a copy of some or all of the information it holds.

A single Directory Server may act in the role of a shadow supplier, a shadow consumer, or both, depending on the shadowing agreements between it and other Directory Servers.

Many copies of a part of the DIT (a naming context or part of it) may be held by a number of different Directory Servers; however, only one is considered to be the master copy. When information changes, it is the master copy which is updated, and then the changes are propagated to the shadow copies, following the rules laid down in the various shadowing agreements.

Depending on the shadowing agreement, a Directory Server in a shadow consumer role may request an update from a supplier Directory Server, or the shadow supplier may initiate the update. The update can take place at a defined frequency or on demand, and the actual update can be either incremental or total. An incremental update includes only those modifications since the last update. In both cases, a timestamp uniquely identifies the update transaction.

The protocol used for replication is the Directory Information Shadowing Protocol (DISP).

8.1.1 Replicating parts of the Directory

To set up a shadowing agreement between two Directory servers:

1. The two Server Managers must agree:
 - The context prefix identifying the naming context to be shadowed under the agreement. You can limit the replication to a certain number of levels below the context prefix, you can prevent certain subtrees from being included (see [Section 8.1.1.1, “Chop shadowing”](#)), and you can exclude certain attributes from the shadowed information (see [Section 8.1.1.2, “Attribute filtering”](#)).
 - Whether the shadowing is to be initiated by the supplier or the consumer.
 - The agreement identifier and version number.
 - The frequency with which to supply or request updates, depending on whether the agreement is supplier or consumer initiated, and when the updates should occur.
2. Each Server Manager needs to have the following information about the other Directory Server:
 - Its access point; that is, the Distinguished Name and Presentation Address.
 - The security credentials.

Note: Both Directory Servers involved in a shadow agreement need to have bind profiles in M-Vault Console.

3. If secondary shadowing is to take place – neither the supplier nor the consumer masters the specified naming context – the Distinguished Name of the mastering Directory Server and the security credentials associated with it must be known.
4. If an Administrative Point for the naming context to be shadowed does not exist, the supplier Server Manager should set one up. Configure it as an access control specific area and a collective attribute specific area. How to do this is described in [Chapter 6, Controlling Access](#).

Note: An Administrative Point for this naming context must not exist in the consumer Directory Server.

5. You may wish to decide the authentication details for the initiating and responding Directory Servers.
6. Set up the responder end of the agreement on the Directory server that will respond to a request. This could be the shadow supplier or consumer.
7. Set up the initiator end of the agreement on the Directory server that will initiate the request. This could be the shadow consumer or supplier.

Details on how to set up a shadowing agreement are given in [Section 8.3, “Creating shadow agreements”](#).

8.1.1.1 Chop shadowing

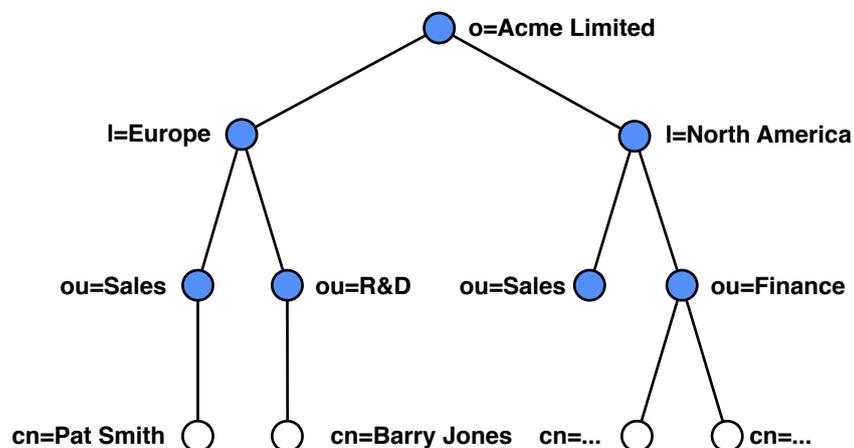
M-Vault Console permits the configuration of shadowing agreements which either cover full naming contexts, or a specified number of levels in the DIT below the context prefix, or which excludes particular subtrees. This section describes two ways in which subtrees can be used to control the information which is shadowed. These methods are called:

- Chop before: information below a subordinate naming context is excluded from a shadow update of a superior naming context.
- Chop after: the context prefix of a subordinate naming context is replicated, but information below the context prefix is excluded from a shadow update of a superior naming context.

In both methods, the subtrees are not shadowed. In the case of chop after, the top of the subtree is shadowed.

This section uses the example in [Figure 8.1, “Example Directory Information Tree”](#) to explain these methods.

Figure 8.1. Example Directory Information Tree



Using the example in [Figure 8.1, “Example Directory Information Tree”](#), chop after shadowing would be useful when a user performs a single-level search on all departments

in the locality of Europe (**l=Europe**). If the department represented by **ou=R&D** has been excluded as per chop before shadowing, a one-level search for all departments below **l=Europe** involves chaining to the Directory Server holding the context prefix **ou=R&D**. However, if you have used chop after shadowing, this context prefix could be included in the shadow update on the consumer Directory Server. A one-level search on all departments in the Europe locality would then not require chaining, as all of the required information would be shadowed on the consumer Directory Server.

Note: The consumer Directory Server in a shadow agreement knows that entries are missing from its copy of the naming context, and it may chain back to the supplier Directory Server to locate information if required.

If you want to exclude a subtree from a shadow update (for example, **ou=R&D**), you need to modify the shadowing agreement on the supplier, and add the subtrees to the list of specific exclusions.

If this agreement has already sent an update to the consumer Directory Server, you must then force the supplier Directory Server to send a total update. This is the only way to force the consumer Directory Server to remove the subtree.

8.1.1.2 Attribute filtering

In certain environments it may be useful to restrict the contents of entries that are replicated. This may be done for reasons of privacy (excluding sensitive personal information), security (excluding passwords), or to conserve network bandwidth (excluding large attributes).

The attributes being filtered from the replication may be selected on the basis of the object class of each entry in the replicated area. For example, telephone number attributes could be excluded from all entries using the **person** object class, but included in **organization** entries.

Consumer Directory Servers have knowledge about which entries are incomplete, but have no knowledge regarding *which* attributes are missing from these entries. As a consequence, if a DUA requests attributes (this includes searching on those attributes) that are not present from an incomplete entry, the consumer Directory Server will attempt to chain the operation back to the supplier Directory Server. This is often inappropriate, so X.500 defines the `copyShallDo` service control to disable this behaviour.

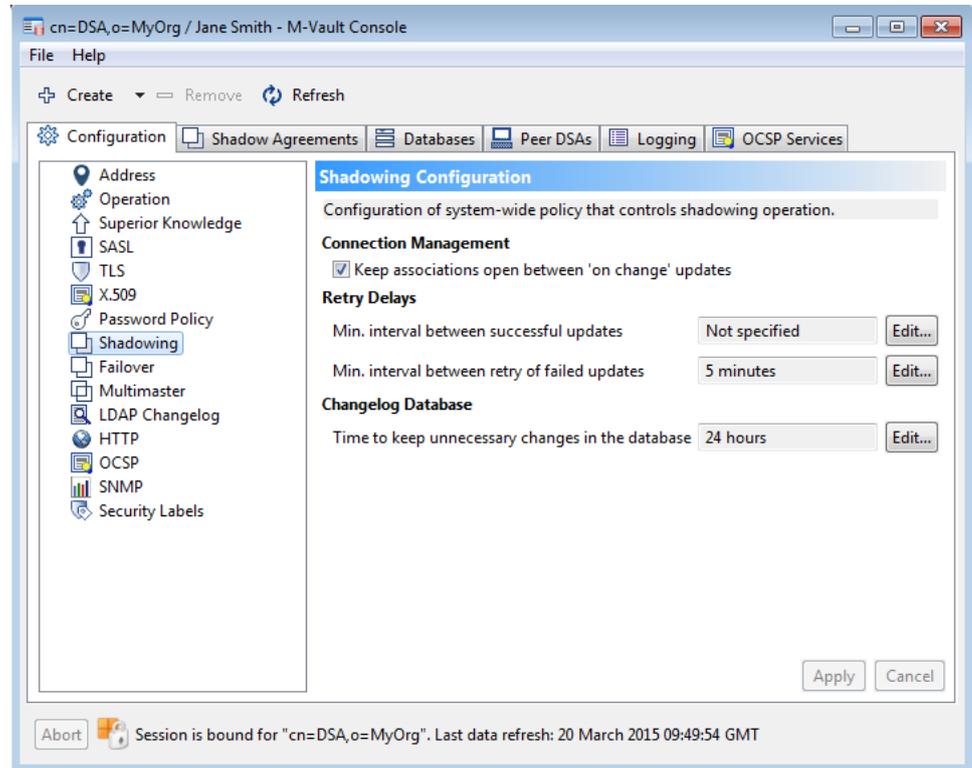
There is no means in LDAP to specify the `copyShallDo` service control. Because not setting this service control causes surprising behaviour, M-Vault treats requests from LDAP DUAs as always having the `copyShallDo` service control set.

The agreement contains a number of rules defining which attributes should be included, and which attributes should be excluded. By default, an agreement will have a rule specifying that all attributes are included from all object classes.

If this agreement has already sent an update to the consumer Directory Server, you must then force the supplier Directory Server to send a total update. This is the only way to force the consumer Directory Server to remove the filtered attributes (see [Section 8.4.1, “Viewing the status of shadow agreements”](#)).

8.2 System-wide shadowing settings

Some shadowing settings apply to all agreements across the whole system. These are specified using the **Shadowing** option on the main **Configuration** page when managing the Directory Server using M-Vault Console.



The **Shadowing** page allows you to specify:

- Whether you want to keep connections open when performing ‘on change’ updates.
- The minimum interval between successful updates, in seconds. If you do not specify a value, a default of 600 is used.
- The minimum delay before retrieving unsuccessful updates, in seconds. If you do not specify a value, a default of 1800 is used.
- How long to keep unnecessary changes in the database. Unnecessary changes are those that are earlier than the last time that all the agreements (from this GDAM) were last updated, so they are not required for any incremental updates.

Click **Apply** to save your changes. Changes you make here apply to every shadowing agreement.

8.3 Creating shadow agreements

A shadowing agreement must be set up on both the supplier and the consumer Directory Servers.

Make sure that a bind profile exists for both the consumer and supplier Directory Servers: if you do not manage both servers, one of the profiles will be for a 'known' server. See [Section 7.2, "Connection details for Directory Servers"](#).

Note: The Directory Server initiating the shadowing should be set up last, regardless of whether this is the supplier or the consumer.

8.3.1 The supplier's end of a shadowing agreement

A supplier's shadow agreement is set up using M-Vault Console. A bind profile for the consumer Directory Server must exist in M-Vault Console (see [Section 7.2, "Connection details for Directory Servers"](#)).

1. Using M-Vault Console, connect to the Directory Server that will be supplying the data.

Note: If this Directory will be initiating the shadow request, make sure the responder has already been configured.

2. Select **Create** → **Supplier Agreement** from the toolbar. The **Create Supplier Agreement** window opens.
3. Select the **Naming context** that forms the top of the tree to be replicated.

Click **Next**.

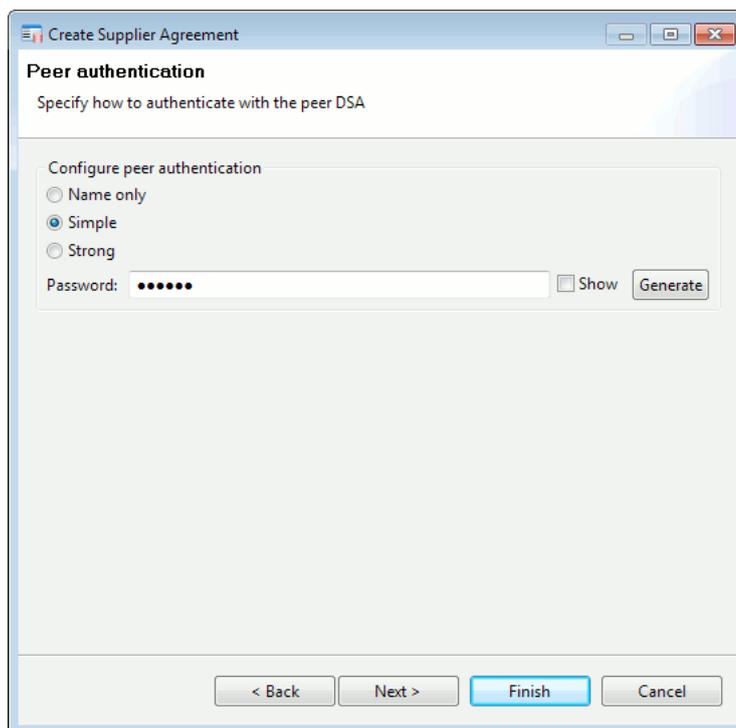
4. Select the Directory Server that will be the consumer.

Note: If you cannot see the consumer Directory Server, create a bind profile for it. See [Section 7.2, "Connection details for Directory Servers"](#).

Click **Next**.

5. If you are managing the consumer, and it is online, you will be asked to specify a database to hold the supplied data. This page is not shown if the consumer database is not online, or if you are not managing it.
6. Specify the authentication requirements for the initiating and responding Directory Servers.

Note: If you have created a peer entry for consumer Directory and have specified authentication details there (see [Section 7.4.1, "DISP authentication"](#)), this page is skipped.



The options are:

- **Name only:** the Directory Server's DN is passed for identification purposes
- **Simple:** a password is required – select **Show** to see the characters in the password and click **Generate** if you want the system to create a random password for you
- **Strong:** the server must have an X.509 identity (a certificate)

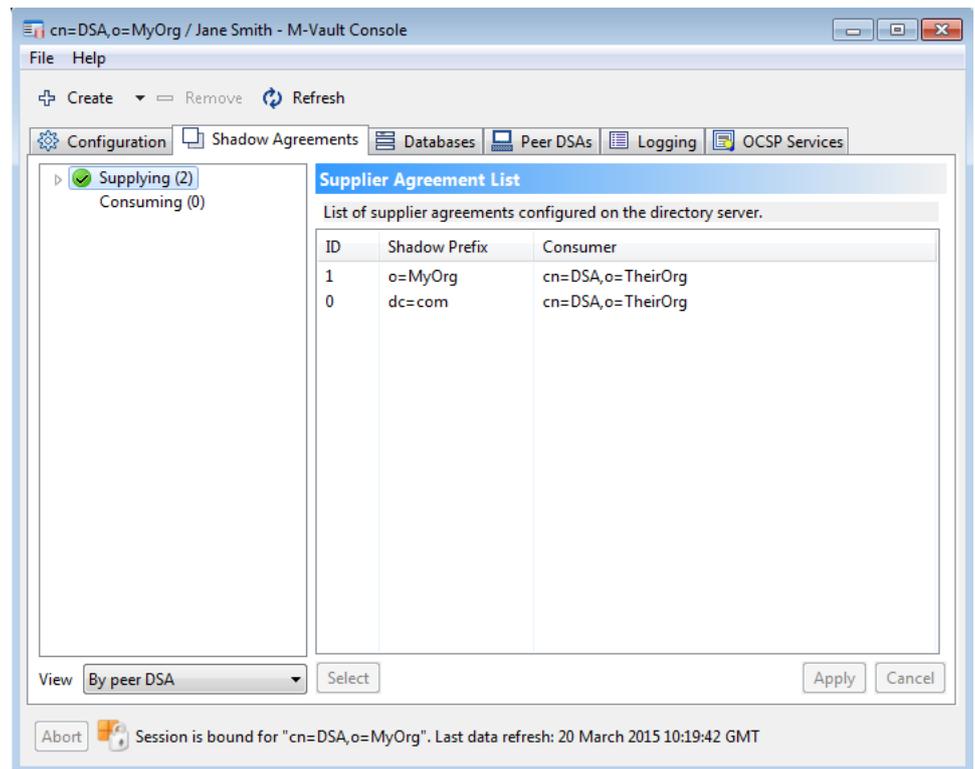
Your choice must match that on the consumer Directory Server. This will happen automatically if the consumer Directory Server is online and is managed by you; otherwise, you will have to share connection details with the manager of the other Directory server.

Click **Next**.

7. The **Summary** page is displayed.

- If the consumer Directory Server is online and is managed by you, the consumer version of the agreement is created automatically in that Directory Server's configuration.
- If the consumer Directory Server is not online or is not managed by you, the consumer version of the agreement must be created manually.

Click **Finish**.



8.3.2 The consumer's end of a shadow agreement

The wizard pages for creating the consumer end of a shadow agreement are very similar to those for creating the supplier end, with the following differences:

- You will have to select the server that is supplying the information. If the server is not listed, you must create a bind profile for it. See [Section 7.2, “Connection details for Directory Servers”](#) for more information.
- If the supplier cannot be contacted, or is not advertising its naming contexts, you may have to enter the context prefix manually.
- You will always be asked to specify a database for storing the shadowed entries.
- The supplier end of the agreement will be created automatically if the supplying Directory Server is managed by you and is online.

8.4 Configuring shadow agreements

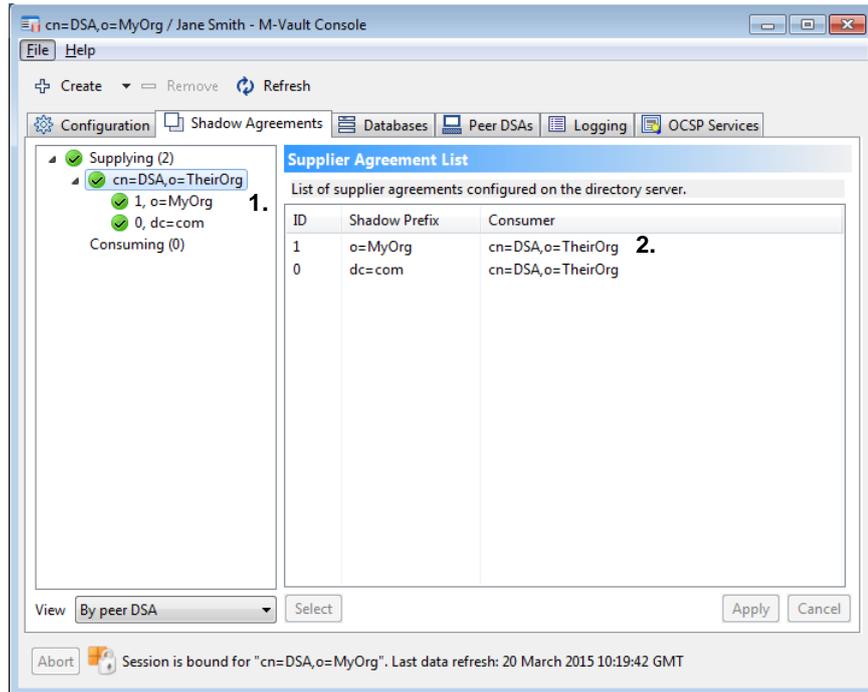
The shadow agreements that exist for a Directory Server are shown on the **Shadow Agreements** page of M-Vault Console. Details for consumer and supplier agreements are very similar, so a supplier agreement is shown as an example.

1. Using M-Vault Console, connect to the Directory that is the supplier for an agreement and select the **Shadow Agreements** page.
2. Select **Supplying** in the list on the left. A list of shadow agreements is shown on the right.

The **Supplying** entry on the left can be expanded to show the individual shadow agreements.

3. To view details of a shadow agreement, either:

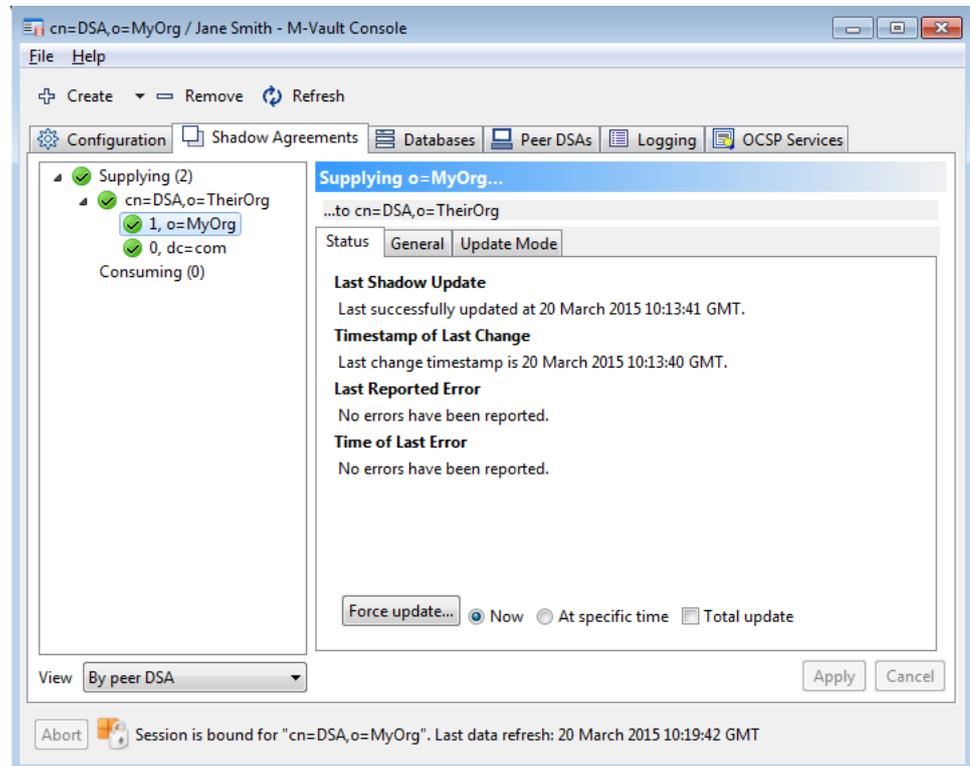
- Expand the list on the left and click the agreement you want to view.
- Select the agreement in the list on the right and click **Select**.



The **Supplier Agreement List** is replaced by a set of pages relating to the selected agreement.

8.4.1 Viewing the status of shadow agreements

The **Status** page shows statistical information relating to the agreement. In the example below, the last shadow update took place on 25 Nov 2010. Details of the most recently reported error are also displayed. This page is also used to initiate an update.



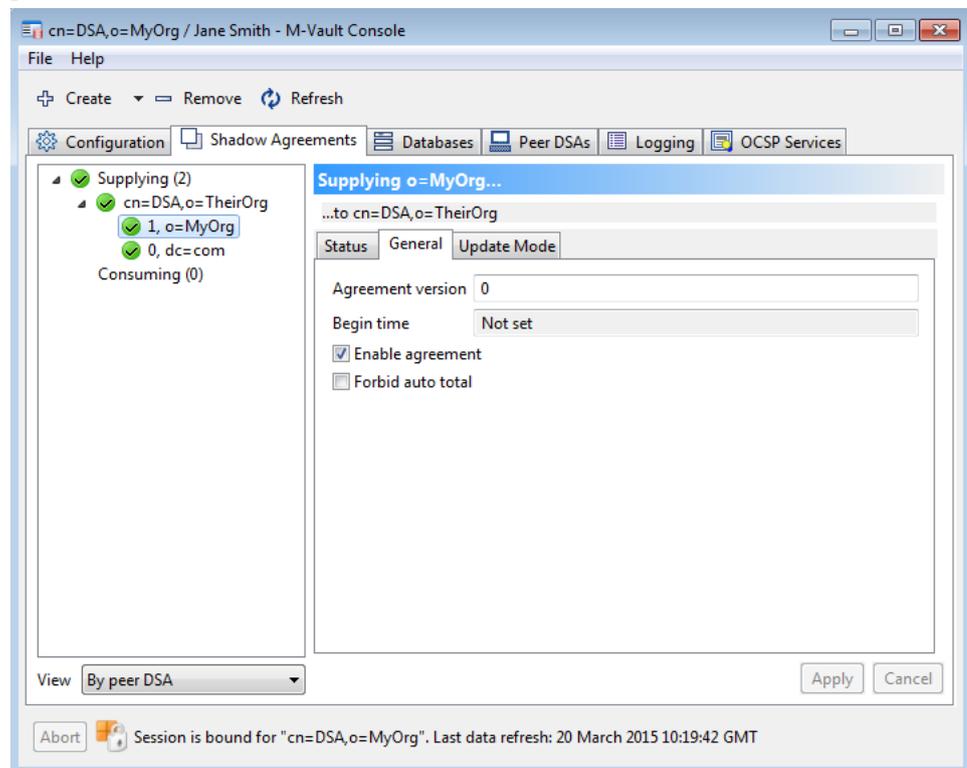
- **Refresh** updates information on this page. For example, if you force an update to take place instantly, you have to refresh the page to show the success of that update and any errors that resulted.
- **Force update...** initiates a shadow update, either immediately (**Now**) or at a set time.

Choose when you want the update to happen *before* you click **Force update...**:

- **Now** is the default: an update will take place as soon as **Force update...** is clicked.
- If you choose **At specific time**, a calendar is shown when **Force update...** is clicked, enabling you to select a date and time.

8.4.2 Enabling agreements

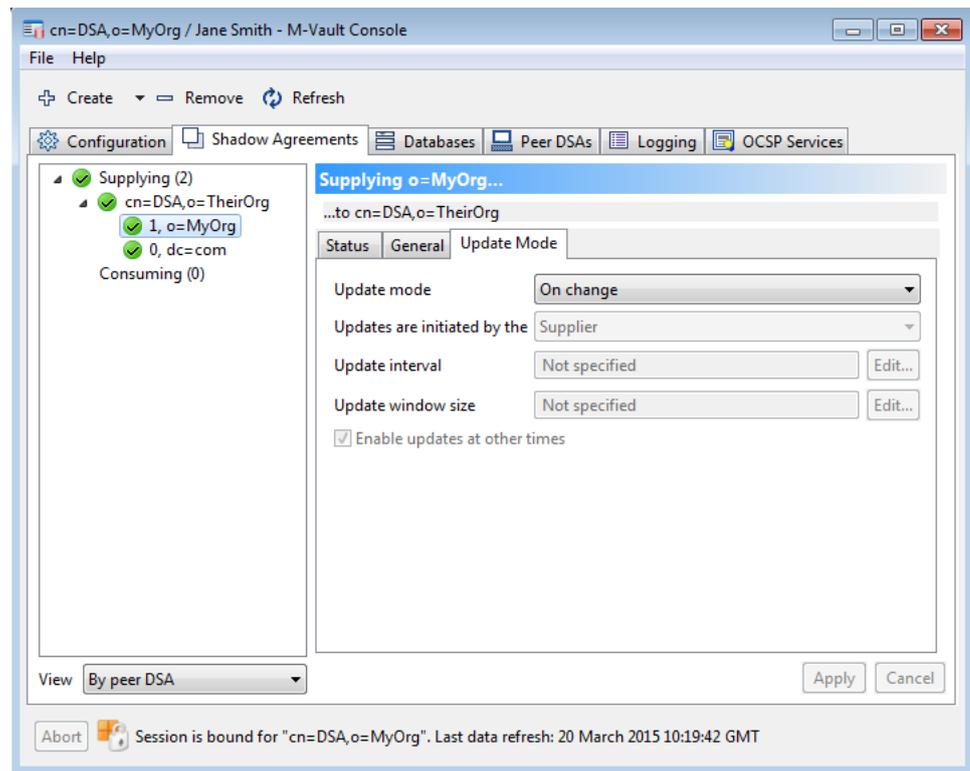
You can enable and disable agreements using an option on the **General** page. You may need to do this if, for example, you are making changes on the Directory Server at one end of the supplier–consumer chain and do not want updates to take place during the change process.



8.4.3 Specifying update mode

You can choose to update information – send it from supplier to consumer – at regular intervals or when there is a change in the data. This is specified using the **Update Mode** page.

The default is to update information whenever a change is made: all other options on this page are disabled. If this is not a suitable option for your environment, you can change the **Update mode** to **Periodic**, and specify related details.



For periodic updates, you need to specify:

- Whether the update will be initiated by the **Supplier** or the **Consumer**.
- How frequently the update will take place (**Update interval**)
- How long the update can last before the connection is terminated (**Update window size**)
- Whether you will allow updates to take place outside of the specified time period.

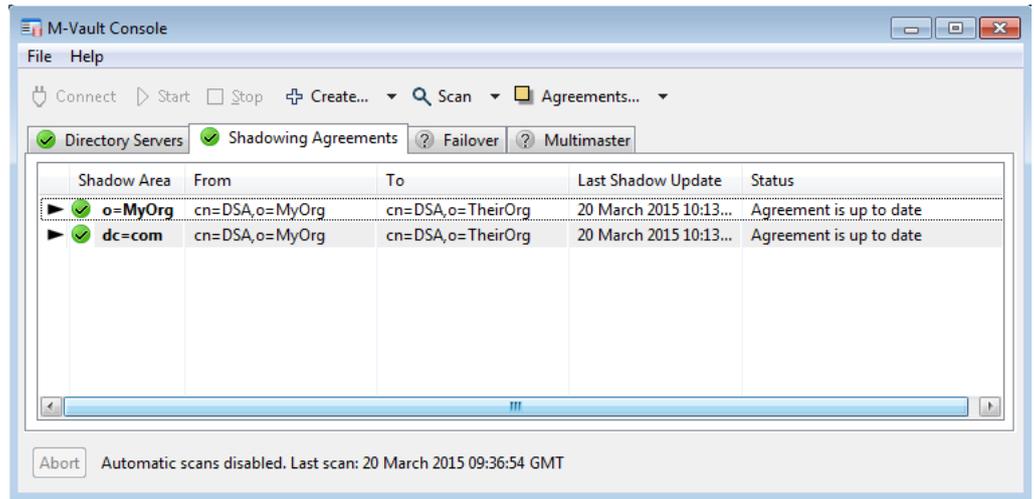
8.4.4 Monitoring shadow agreements

When shadowing is configured, both Directory Servers (supplier and consumer) maintain information about the state of each agreement that they know about.

Typically a Directory Server will log errors if an agreement is not working correctly, but there may be situations where even though the agreement is not working properly, one or both servers will not report an error. For example, if the supplier initiates an update which never reaches the consumer, then the consumer may not report any error. If supplier and consumer each believe that it is the other Directory Server's responsibility to initiate updates, then neither of them will report an error, even though the agreement will never be updated.

M-Vault Console's **Shadow Agreements** view is intended to make it easy to keep track of the status of shadowing agreements. It does this by combining information from both supplier and consumer of an agreement, and displaying it in a dynamic view that allows an operator to see at a glance whether things are working properly, and where there may be problems that require investigation.

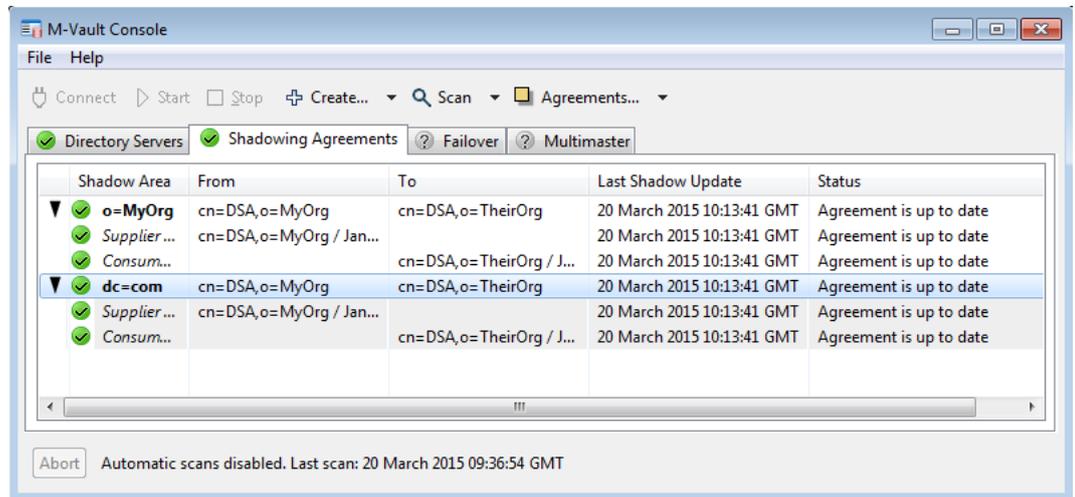
M-Vault Console gathers information about shadowing agreements from each managed Directory Server, combining information from both sides of the agreement where possible, and using this information to present a view that summarises the status of all known agreements.



Initially, the **Shadowing Agreements** tab displays one row per shadowing agreement: each row contains an agreement status summary. The tab will only display information about agreements when there is a management connection to at least one of the Directory Servers (either supplier, consumer, or both).

If M-Vault Console has a management connection to only one of the Directory Servers (for example, you are managing a server that is consuming a shadow agreement which is supplied by a non-managed Directory Server in a separate organisation), then the information shown in the **summary** row is derived solely from the managed server: i.e., in the absence of information from one of the peers, M-Vault Console assumes that whatever information it *can* see is authoritative.

Each of the **summary** rows may be expanded or collapsed by using the **Agreements...** menu (or by double-clicking on the triangle in the table) to show or hide information reported from supplier and consumer. When expanded, the individual rows for supplier and consumer contain information from the Directory Server in question.



Information displayed in each column of the table is as follows:

- An icon represents the state of the agreement, as determined by M-Vault Console.

A green icon is used to show that no errors are present; a red icon shows that errors are being reported. An agreement is considered to be in an error status if it has reported an error since the last successful shadow update.

In situations where M-Vault Console detects inconsistencies between the supplier and consumer view of the agreement, an orange icon is used.

If M-Vault Console has a connection to only one of the Directory Servers, then the "expanded" view of the agreement will display a grey icon to indicate that no management connection is active to the non-managed server.

- The name of the shadow area is displayed in the **summary** row. When the **summary** row is expanded, this column contains whether the information in each row comes from the supplier or consumer Directory Server.
- **From** and **To** columns show the identities of the supplier and consumer Directory Server, respectively. In the **summary** row, the Directory Server DNs are shown. When supplier and consumer rows are displayed, the display name of the Directory Server (if available) is used.

If the supplier is a member of a failover group, then the **summary** row will contain a failover icon, and the **From** column will show the DN of the failover group.

- The **Last Shadow Update** column displays the last time when the shadow agreement was last updated successfully. The information shown is that which was reported by supplier and consumer, and so there may be differences in the timestamps if the clocks for both servers are not exactly synchronized.
- The **Status** column contains a brief description of the agreement state

To see more information on any of the agreements in the table, use the **View Agreement...** option in the **Agreements** menu. This will invoke a dialog that provides more detail about the selected agreement.

Shadow Agreement Details

Shadow area :

Supplier

DSA DN :

Bind profile of DSA :

Consumer

DSA DN :

Bind profile of DSA :

Summary
 Supplier
 Consumer

Supplier and consumer both report that the agreement is operating correctly.

Current Error :

Error Time :

Last shadow update :

Update mode :

Agreement ID/ver :

OK

The Shadow Agreement Details dialog contains tabs which show summary, supplier and consumer information. Any inconsistencies between supplier and consumer will be listed on the **Summary** tab.

Shadow Agreement Details

Shadow area :

Supplier

DSA DN :

Bind profile of DSA :

Consumer

DSA DN :

Bind profile of DSA :

Summary Supplier Consumer

Supplier and consumer agreement are inconsistent

onChange Supplier reports onChange = "true", but consumer reports "false"

updateInterval Supplier reports updateInterval = "null", but consumer reports "20"

OK

The supplier and consumer tabs contain a **Go to agreement...** button which will be enabled for any agreement on a managed Directory Server. Clicking this button will cause M-Vault Console to open the management window that allows you to make changes to the configuration of the agreement.

Chapter 9 High Availability

M-Vault provides three means of adding service resilience to system failure:

- Failover. Hot standby mode where a single master is active amongst a group of mirror servers.
- Multimaster. All servers in a replication group accept changes and replicate them to all other members of the group.
- Clustering. Hot standby mode where shared disks are used to support multiple nodes.

9.1 Failover

Failover is a hot-standby approach where there is a single master DSA and a number of read only mirror DSAs, where any of the mirror servers can be promoted to become the master as required. Each mirror server holds copies of all user data held in the master as well as the configuration required to provide regular shadow consumers with shadow updates. In the event of required operational downtime or catastrophic failure the system can fail over to one of the mirror servers. When failover is complete the nominated mirror server becomes the new master and can accept updates to the data and then supply any changes to the other mirror servers and also to any regular shadow consumers.

The unit of replication in failover servers is the GDAM. This means that there is one replication agreement per GDAM that is held in the master DSA. As GDAMs are created on the master DSA replication of the configuration will mean that an equivalent GDAM is automatically created on each failover mirror. As GDAMs are populated with entries, or when any changes are made to them, this information is automatically replicated to each of the mirrors. This means that once a set of failover servers is created and configured all data stored on the master server (and changes to that data) will be automatically replicated to each mirror.

9.1.1 Fundamentals and limitations

The server that currently accepts update operations and distributes changes is referred to as the failover (or current) master. The full set of servers within the failover configuration is referred to as the failover group. Servers in the group which are receiving updates from the master are referred to as mirrors. Mirror servers will reject any client update requests as these can only be directly handled by the current master. Servers receiving standard shadow updates are simply referred to as shadows.

The failover mechanism uses an extended form of X.500 DISP to replicate data between mirrors. The extensions allow shadowing to recover better from loss of synchronization between servers - in particular if a mirror server has gone offline for a long period of time. There is one failover shadowing agreement per-GDAM stored in the master server and the agreement covers all data held in that GDAM database.

There are some important points and limitations that should be noted prior to deploying a failover configuration:

- The failover implementation does not currently allow failover servers to contain consumer agreements, i.e. they must not receive shadow information from any external server.
- Failover mirrors do not contain copies of all configuration. For example, SASL, TLS and X.509 configuration is not copied to mirror servers. Thus, if the master server is to provide a wider directory service, then this configuration must be created manually on each server.

- When failing over it is always possible that the failover master will deem it necessary to send a total update to mirror or shadow servers. This can happen if the level of synchronization between servers cannot be established. Thus, if a very large amount of data is involved then there may be some delay in resynchronizing servers, i.e. the length of time it takes to perform a total update of all data.

9.1.1.1 Failover modes

There are two modes of master failover; normal and forced. In the normal mode the master DSA is still running and M-Vault Console notifies it, upon operator request, of failover initiation. Forced mode is used when the current master has suffered a failure or is unreachable, and in this case M-Vault Console notifies each of the mirror servers which should act as the master. These modes are described in detail in the following sections.

9.1.1.1.1 Normal failover mode

In this mode the master server must be running and be available. M-Vault Console informs the current master of the server to fail over to. The current master then performs the following steps:

1. The master waits for any outstanding user update operations to complete. Further update operations are then rejected until failover has completed.
2. The current master sends the latest data changes to the nominated master until it is fully synchronized.
3. The current master passes master status to the nominated master, which will then begin to accept updates.
4. The current master notifies other mirror servers that master status has passed to the nominated server.

9.1.1.1.2 Forced failover mode

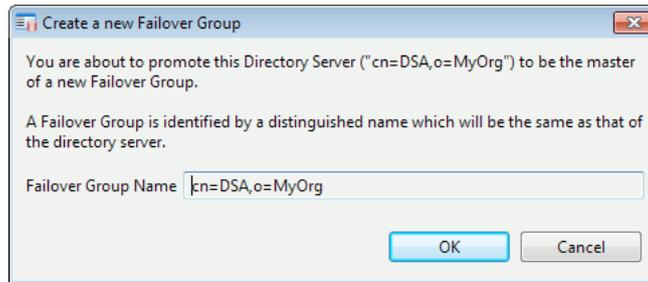
In this mode at least one server in the failover group must be running. Here M-Vault Console updates each running mirror with the value of the new master. The new master will then update each of the mirror servers. Total updates will be pushed to any mirror that cannot be updated with an incremental update. This may occur if, for example, a mirror server contains changes that are not present on the new, and now current, master.

9.1.2 Creating a failover configuration

To create a failover configuration open a management window to the base directory server, i.e. that you wish to mirror. Mirror servers will copy all data from this server and the selected base server will be the initial master. Next, select **Create** → **Failover Configuration** via the toolbar.

Note: The base directory server must not contain any consumer agreements.

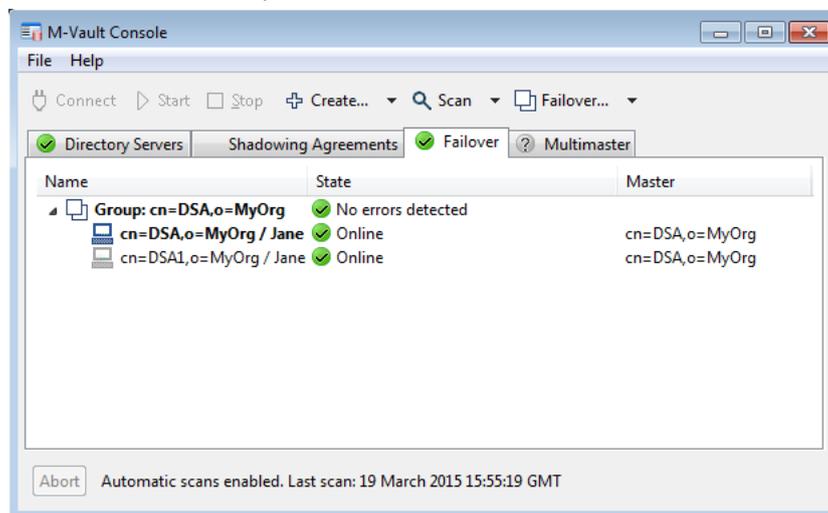
You will then be required to enter a DN used to indicate the failover identity, or the failover group name. When replicating data to shadow consumers, a supplier DSA must provide an identity. This identity is commonly the DN of the base DSA. In the failover case different DSAs can supply to a given shadow consumer at different times (depending on which is the master at any given point in time). In order for this to work the failover servers must establish a common identity when communicating with shadow consumers, i.e. with Directory Servers that are not part of the failover group. This is always the DN of the first DSA that was created in the failover group.



Once the failover configuration has been created you can go and create one or more failover mirror servers.

9.1.3 Managing the failover group

The failover group is the set of Directory Servers comprising the current master server and the set of mirror servers. It is possible to remove mirror servers from a failover group as well as creating new failover group members. To view a failover group go to the **Failover** tab of the M-Vault Console main window and select the group (or a member of the failover group). The current master is displayed in bold. Servers are indicated as being up or down as in the main **Directory Servers** tab.



9.1.3.1 Adding failover mirrors

To add a failover mirror open a management window to the current master server. Next, select **Create** → **Failover Mirror** from the toolbar.

Note: When creating a new failover server, as with creating any new server, M-Vault Console must be running on the same system as the server that is to be created.

You will then be presented with a wizard which asks the following:

1. The unique server name, the Distinguished Name, of the new Directory Server. This is the DN used by the server to identify itself to other members of the failover group.
2. The password of the new manager. There are two cases here:
 - a. If the manager bind profile of the master is *not* **cn=DSA Manager,cn=config**, the password stored in the bind profile is used for the mirror bind profile.
 - b. Otherwise, a password is requested which does *not* have to be the password of the current manager.

3. The bind profile name, where the bind profile will connect to the new failover mirror using the manager's credentials.
4. The file-system directory in which the new server will be created.
5. The presentation address that the server will listen on.

After the server has been created and started M-Vault Console will ask the current master to add the new mirror to the failover group. The master will then go on to send initial updates (of the configuration and all mastered information) to the mirror.

9.1.3.2 Removing failover mirrors

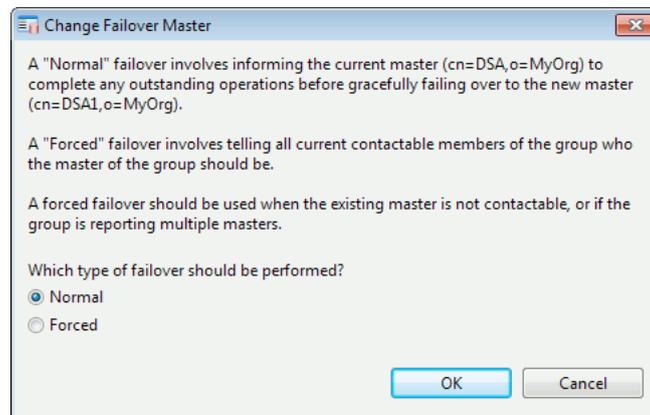
To remove a mirror from a failover group:

1. Open the M-Vault Console main window.
2. Click on the **Failover** tab.
3. Highlight the mirror server that you wish to remove.
4. Select **Failover** → **Remove DN from the group...**

Once a mirror has been removed from the group it is not possible to add it back in to the group. To add a mirror of the same name back in to the group you should remove the old instance and create a new server from scratch.

9.1.4 Failing over to a new master

Failover is initiated from the M-Vault Console main window. To nominate a Directory Server as the new master go to the **Failover** tab, highlight the target Directory Server and then select **Failover...** → **Make DSA the master ...**. You will then be presented with a dialog offering a choice of failover modes.



If the current master is running and contactable then the default choice, **Normal**, should be made. This mode would be used when the master server is to be later brought down, perhaps for an upgrade.

Forced mode should be chosen in the following cases:

- The current master is not running.
- The current master is not reachable.
- The current master cannot, for some reason, make contact with the proposed master.

9.1.4.1 Errors and recovery

If a failover group member is unavailable at the time failover is performed then it is likely to be in an error state when brought back online as that server's understanding of the current master may be inconsistent with the other servers in the failover group. To illustrate this consider a scenario in which there are three servers (DSA1, DSA2 and DSA3) in a failover

group. If DSA1 is the initial master and it goes down, the operator may then initiate a forced failover to DSA2. At this point DSA2 and DSA3 believe that DSA2 is the master. When DSA1 is brought back online it will still believe that it, DSA1, is the master. As a consequence it will reject any changes sent to it by the current and actual master (DSA2 in this example). To recover from this the following steps should be performed:

1. Bring DSA1 back online.
2. Initiate a forced failover, selecting again the current master (DSA2 in the example scenario). This will mean that DSA1 will start accepting updates from DSA2. If a total update is required then this may take a non-trivial period of time.
3. To make DSA1 the master again initiate a normal failover from DSA2 to DSA1. DSA1 will not be made the current master until it has fully resynchronized with DSA2.

9.2 Multimaster

Multimaster replication allows writes to happen on any node at anytime. Changes are propagated to all other members of the group in synchronous fashion. Servers that lose synchronization or lose connectivity with other members of the group are regarded as degraded until such time as that server can reconnect and resynchronize with the other members of the group.

The unit of replication in multimaster is the GDAM. GDAM configuration is replicated between group members. Thus if a GDAM is created on one server, the corresponding configuration is replicated and an identically configured GDAM is created on the other servers. The contents of GDAM database are then replicated once a replication agreement has been created for it.

9.2.1 Fundamentals and limitations

In the multimaster model employed by M-Vault the server processing a client update operation will not send a positive notification to the client until all multimaster group members have accepted the update. Thus replication is synchronous. The advantage to this is that at any given time all servers will return the same result to a query for information. If a password change happens on one server, then all other servers should report the same password value immediately. This contrasts with other multimaster models, such as eventual convergence, where the server processing an update can return a positive response to the client before any of the other replica servers have been updated. The downside is that update throughput is reduced, due to the network overhead of the additional synchronization protocol and by being limited to the speed of the slowest processing server. The upside of M-Vault's approach is that, in contrast to the eventual convergence model, inconsistent changes cannot be applied simultaneously to different servers in the network, and so the view of the data is consistent across the service.

Limitations to the use of multimaster are as follows:

- Multimaster servers cannot share shadow supplier duties. A single multimaster can provide X.500 DISP updates to shadow consumers, but no other server in the multimaster network can. This is due to DISP's requirement for a single logical and sequence of changes being the basis of the synchronization algorithm. This property is not present across replicas in a multimaster as the order of stored changes can vary between servers.
- Security related configuration, e.g. password hashing, SASL and TLS configuration is copied into new replica DSAs by MVC at creation time. However, this configuration is not subsequently replicated between servers and so any changes made to the core configuration on one server should be made manually on all other servers as necessary.

Note that password hashing configuration, in particular, must always be consistent across all replicas. Failure to do so may result in password based authentication failing on some servers (due to different treatment of passwords in line with the local configuration).

- Changes are distributed if quorum is achieved. Specifically this means changes will only be replicated if a given number of replica servers signal that they are able to accept a change. By default the quorum level is set to 0, meaning any server will process an update irrespective of the ability of other servers to receive and process the change. This confers the highest level of service resilience, though means that network partitioning may result in the possibility of inconsistent changes being applied to different replica servers. In general it is recommended that the quorum level be set to half the number of replica servers, which ensures that inconsistencies cannot enter the system. If the number of available servers is reduced to a level below this then the configured quorum level can be reduced as necessary.

9.2.2 Creating a multimaster configuration

To create a multimaster configuration open a management window to the server that you wish to create a replica of. Next, select **Create** → **Multimaster Configuration** from the toolbar.

Note: The source directory server cannot contain any consumer agreements or be configured for failover.

You will first be required to enter a label for the multimaster replication group. This is then used to label the group in M-Vault Console's main window. The next piece of information you must provide is the port number used to listen on for multimaster protocol.

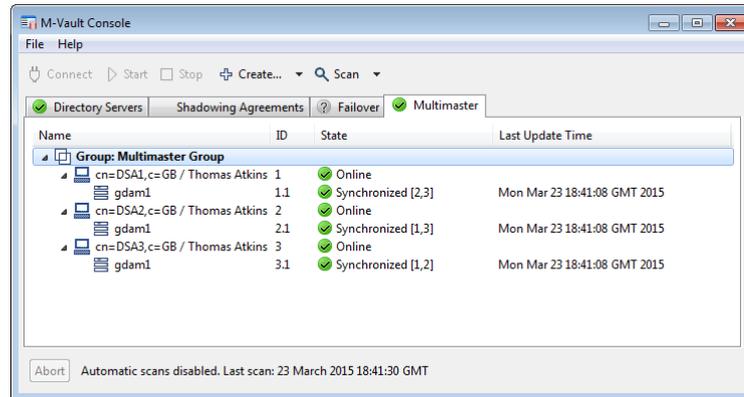
Note: The multimaster protocol (named MESH) is proprietary, and so must listen on a distinct and specific port. The default port is 20000, though you are free to select any other available port for this purpose.

9.2.3 Managing the multimaster group

A multimaster group is a set of Directory Servers that are replicating directory information between them. Any information that is replicated between any two members of this group must be replicated between all members of the group, i.e. there is no support for partial replication within a group. Individual servers may master data, in a particular GDAM, that is not shared between the group. Such locally mastered may be added to the set of replication agreements at a later date.

The status of the set of known multimaster groups can be seen by selecting the **Multimaster** tab of the main window, as shown in figure [Figure 9.1, “Multimaster group status”](#) below.

Figure 9.1. Multimaster group status



The view provides the status information below:

1. Per-server state, specifically whether the server is on or offline. Servers are listed by their DN (in the **Name** column), as well as by an integer server identifier (in the **ID** column).
2. Per-agreement state. Whether the agreement is synchronized with any other server. The **State** column for agreements also shows which other servers have been synchronized for a particular agreement by listing the set of synchronized server identifiers. In addition the time of the last change received is reported. Note that if the last update received was a total update then **Unknown** is reported.

9.2.3.1 Adding multimaster replica servers

To add a multimaster replica select a source DSA from the main window and then select **Create** → **Multimaster Replica Server**. The server that is selected in the main window will be the initial source for the new replica, i.e. it will configure and provide initial updates to the new replica.

Note: When creating a new multimaster replica server, as with creating any new server, M-Vault Console must be running on the same system as the server that is to be created.

You will then be presented with a wizard which asks the following:

1. The unique server name, the Distinguished Name, of the new Directory Server. This is the DN used by the server to identify itself to other members of the multimaster group.
2. The password of the new manager. There are two cases here:
 - a. If the manager bind profile of the source server is *not* **cn=DSA Manager,cn=config**, the password stored in the bind profile is used for the mirror bind profile.
 - b. Otherwise, a password is requested which does *not* have to be the password of the current manager.
3. The bind profile name, where the bind profile will connect to the new failover mirror using the manager’s credentials.
4. The file-system directory in which the new server will be created.
5. The presentation address, including MESH address component, that the server will listen on.

Once the new server has been created it will connect to the source server to get the multimaster configuration and the body of replicated data. Once this is complete the server will be ready for operation.

9.2.3.2 Removing multimaster replica servers

It is not currently possible to remove servers from a multimaster group, though this feature will be implemented shortly.

9.3 Hot-standby clusters

This section describes how to configure a Directory Server in a hot-standby cluster, which is a mode where the Directory Server databases are on a disk shared between all the nodes in the cluster, and where the clustering software arranges for only a single instance of the Directory server to be running at any one time. When the clustering software detects a hardware or software failure on the active node, it fails over the active node and activates the Directory Server on another node.

When configuring a cluster, it is important to note that there are multiple network addresses being used, and it is critical to use particular network addresses at particular points. Mistakes here will cause DSP and DISP connections from other Directory Servers to fail.

9.3.1 On Unix systems

1. On each node in the cluster-to-be, install M-Vault on the node's local disks.
2. On the node which has the shared disk mounted:
 - a. Move the `/etc/isode` and `/var/isode` directories onto the shared disk, and make symbolic links from `/etc/isode` and `/var/isode` to the new locations.
 - b. Copy `/etc/isode/dsa.rc.sample` to `/etc/isode/dsa.rc`.
 - c. Configure the **DSADIR** value, pointing at the shared disk.
 - d. Configure the **DSACHECK** value, using the localhost address.
 - e. Use M-Vault Console to create a new Directory Server on the shared disk. Make sure the presentation address uses the service's network address, not the node's local network address.

Note: You will not be able to connect to the Directory Server using M-Vault Console until the clustering service has been created

- f. Verify that the Directory Server can be started, stopped, and its status checked using the normal startup scripts (make sure the server is stopped at the end):

```
/etc/rc.d/init.d/dsa start
```

```
/etc/rc.d/init.d/dsa status
```

```
/etc/rc.d/init.d/dsa stop
```

- g. Change the `DSACHECK` variable to the service's shared network address.
- h. Unmount the shared disk.
3. On every other node, replace the `/etc/isode` and `/var/isode` directories with symbolic links to the locations on the unmounted shared disk.

4. Now use the Red Hat clustering software to set up a new service (the Directory Server) on the shared disk. The controlling script for the service is `/etc/rc.d/init.d/dsa`. Assign a name to the clustered service, for example M-Vault.
5. The cluster should now be operational with one node. Enable the Directory Server service. You should be able to connect to the Directory Server using M-Vault Console.

Caution: Do not attempt to start and stop the Directory Server using M-Vault Console, as this will confuse the clustering software.

6. Exit from M-Vault Console, and disable the Directory Server cluster service (i.e. M-Vault).
7. Now enable clustering on each node. Finally start the Directory Server service (i.e. M-Vault) using the cluster manager software.

9.3.2 On Windows systems

On Windows systems, each node in the Windows cluster must be a member of the same Windows networking Domain. The cluster has a name, and a shared IP address.

1. Install the clustering service on each node in the cluster-to-be:
 - a. Open **Start** → **Settings** → **Control Panel** → **Add/Remove Programs** → **Add/Remove Windows Components** . Select the **Clustering Service** box.
 - b. Follow the instructions, specifying the cluster name and the shared cluster IP address when prompted.
 - c. If this is the first node in the cluster-to-be, reboot the machine and start the cluster service. Repeat steps 1 and 2 for the other nodes.
2. Install the M-Vault software on each node. Ensure the other nodes are disabled, and that the node being installed has the shared disk mounted. Halt each node before installing the next node. If prompted, you should overwrite the installation files on the shared disk.

On the node with the shared disk mounted:

- a. Use M-Vault Console to create a new Directory Server on the shared disk. Make sure the presentation address uses the cluster's shared network address, not the node's local network address. Do not start the Directory Server at this point.
- b. Open **Start** → **Programs** → **Administrator Tools** → **Cluster Manager** .
- c. Right-click on **Resources**, select **New** and **Resource**.
- d. Choose a name for this resource, e.g. `Isode Virtual Server`. Select **IP Address** from the pull down menu. Click **Next**.
- e. Make all of the nodes possible owners. Click **Next**.
- f. Add the shared disk as a dependency. Click **Next**.
- g. Enter the cluster's shared IP address and netmask. Choose a name for this resource, e.g. `M-Vault`. Click **Next**.
- h. Make the service available to all nodes. Click **Next**.
- i. Add the shared resource (e.g. `Isode Virtual Server`) as a dependency. Click **Next**.
- j. Enter `isode.x500dsa` as the service name and in the other field type in the Directory Server's configuration path in the form `-D configuration path` . Click **Next**.
- k. Click **Next** again (you don't need to add any shared registry keys.)
1. Right-click on the shared service (e.g. **Isode Virtual Server**) and bring it online.

The Directory Server should now be operational, and you should be able to connect to it from any machine, and from M-Vault Console.

Test manually failing over the service by right-clicking on the shared drive within the **Cluster Manager**, selecting **Advanced** and the **Do not restart** option. Click **OK**, right-click on the shared disk again, and select **Initiate failure**.

Chapter 10 HTTP And OCSP Services

M-Vault incorporates a Web server that can be used to serve the following:

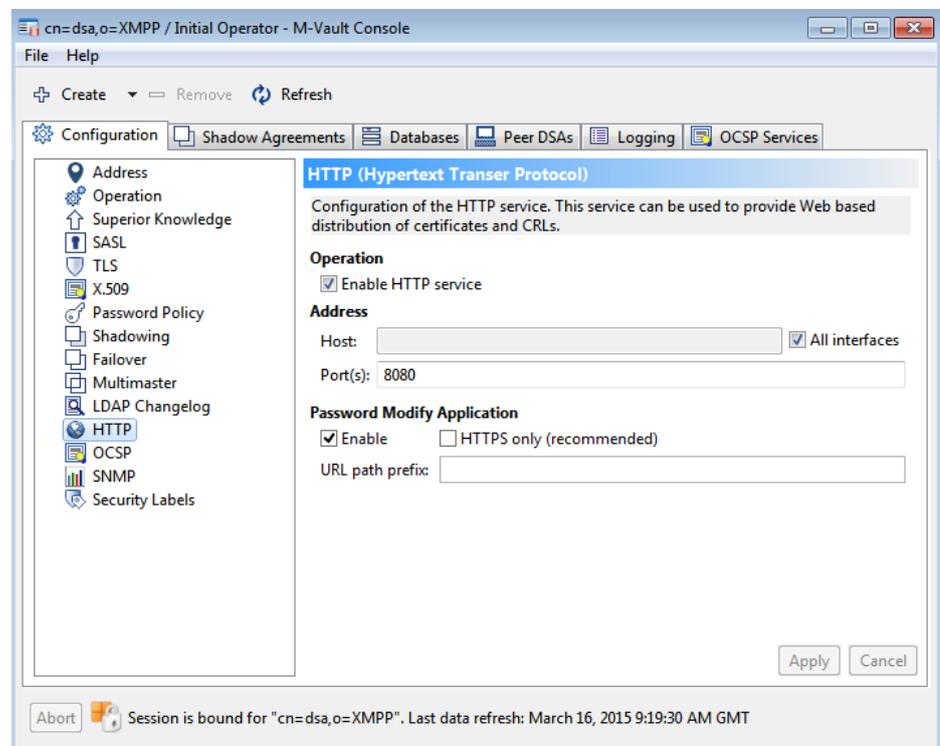
- PKI information - HTTP serving of PKI and CA related directory entry attributes (e.g. **certificateRevocationList**).
- Web applications - currently a Web application (and underlying API) providing an account password modification user interface.
- OCSP service - provision of OCSP (Online Certificate Status Protocol) services on the basis of stored CRLs.

This chapter describes how to configure these services using M-Vault Console and Sodium.

10.1 Configuring HTTP Services

This section covers configuration of the HTTP listener, particularly for the serving of Web applications (e.g. the password modify API and Web application) and PKI information as stored in entry attributes (for example user cross certificate pairs and CRLs).

To enable the HTTP service first connect to the directory server using M-Vault Console, then navigate to the **HTTP** page of the **Configuration** group. Set the **Service Enabled** checkbox and then specify the listen address by supplying a hostname (or select **All Interfaces** to listen on all network interfaces) and a set of port numbers for provision of Web services over HTTP or HTTPS.



Note that:

- A restart of the Directory Server will be required before the HTTP service is enabled or if the set of listen ports has been changed.

- The HTTPS service will only function correctly if a suitable TLS identity has been configured (see [Section 3.10.1, “Generating a certificate request”](#)).

10.1.1 Configuring the Password Modify Web Application

To enable the password modify Web application, first enable the HTTP responder (see [Section 10.1, “Configuring HTTP Services”](#)), and then check the **Enabled** button in the **Password Modify Application** section of the **HTTP** page of M-Vault Console. A directory server restart is required before the application is served. Note also that a valid SASL configuration must be present before the password change Web application can function correctly, as the user ID requested in the user interface is the SASL ID.

Note: In this release password changes made using the Web application are not replicated in multimaster. Such changes *are* replicated in failover and shadowing (i.e. X.500 DISP). Support for multimaster will be added in an R19.0 update release.

Two options are provided that control access to the Web application:

- **HTTPS only** Whether the application should be served over HTTPS only (i.e. not over HTTP). It is highly recommended that this option is enabled, as otherwise passwords will be transmitted over the network in the clear.
- **URL path prefix** Specify a URL path prefix that is used to select the Web application.

If a URL path prefix of `password` is specified and HTTPS enabled then a URL like the one below would direct the browser to the Web application:

```
https://www.example.com:8443/password
```

The Web application is compatible with Internet Explorer 11, Microsoft Edge, and recent versions of Chrome and Firefox.

10.1.2 Publishing PKI Information Over HTTP

PKI information served by the Web server is configured on a per-entry basis, through use of the **isodeHTTPDirective** attribute. Note that the **isodeHTTPDirective** attribute is an optional member of the **isodeHTTPResource** object class and the object class of any entry providing an HTTP served resource needs to be updated to include this value first. Values of **isodeHTTPDirective** consist of a set of fields that control how information is served. Some of these fields are common and some are specific to the resource type. The sections below describe how each served type is configured.

10.1.2.1 Revocation List

The directive consists of a set of fields separated by a `$` character (the string representation of a **CaseIgnoreList**). Each field consists of a key and value pair of the form `key=value`. The fields are as follows:

- `type` - The attribute type. For revocation lists this can be **certificateRevocationList** or **authorityRevocationList**.
- `resource` - The path of the resource, as expected in the right side of the HTTP URL.
- `filename` - The value of the filename field to be in the returned HTTP header.

An example of a valid value is below:

```
type=certificateRevocationList$resource=crl/ca.crl$filename=ca.crl
```

The HTTP header returned with requests for revocation lists contain a number of fields and values that are specific to the list being returned. An example HTTP response header is:

```
HTTP/1.1 200 OK
Content-Type: application/pkix-crl
Content-Length: 538
Content-disposition: attachment; filename="ca.crl"
Etag: "ddfa3bd3d0da544b079b581e5505a37b047371a1"
Last-Modified: Thu, 24 Oct 2013 08:34:05 GMT
Expires: Fri, 25 Oct 2013 02:33:05 GMT
Cache-Control: max-age=3600
Server: M-Vault/16.1
Date: Fri, 29 Nov 2013 12:22:29 GMT
Accept-Ranges: none
Connection: close
```

Fields specific to revocation lists are:

- `Content-Type`: Value always `application/pkix-crl`.
- `Etag`: SHA-1 hash of the CRL.
- `Last-Modified`: The time the CRL was updated. Derived from the value of `thisUpdate` in the revocation list.
- `Expires`: The time at which the CRL is due for renewal. Derived from the value of `nextUpdate` in the revocation list.
- `Cache-Control: max-age=` The number of seconds until the next time CRL is due to be updated.

10.1.2.2 Cross Certificate Pair

The directive consists of a set of fields separated by a `$` character (the string representation of a **CaseIgnoreList**). Each field consists of a key and value pair of the form `key=value`. The fields are as follows:

- `type` - The attribute type. This must take the value **crossCertificatePair**.
- `resource` - The path of the resource, as expected in the right side of the HTTP URL.
- `filename` - The value of the filename field to be in the returned HTTP header.
- `scope` - Optional field which can be used to limit the set of certificates included in the output bundle to those issued by this CA (`issued_by`) or those issued to this CA (`issued_to`).

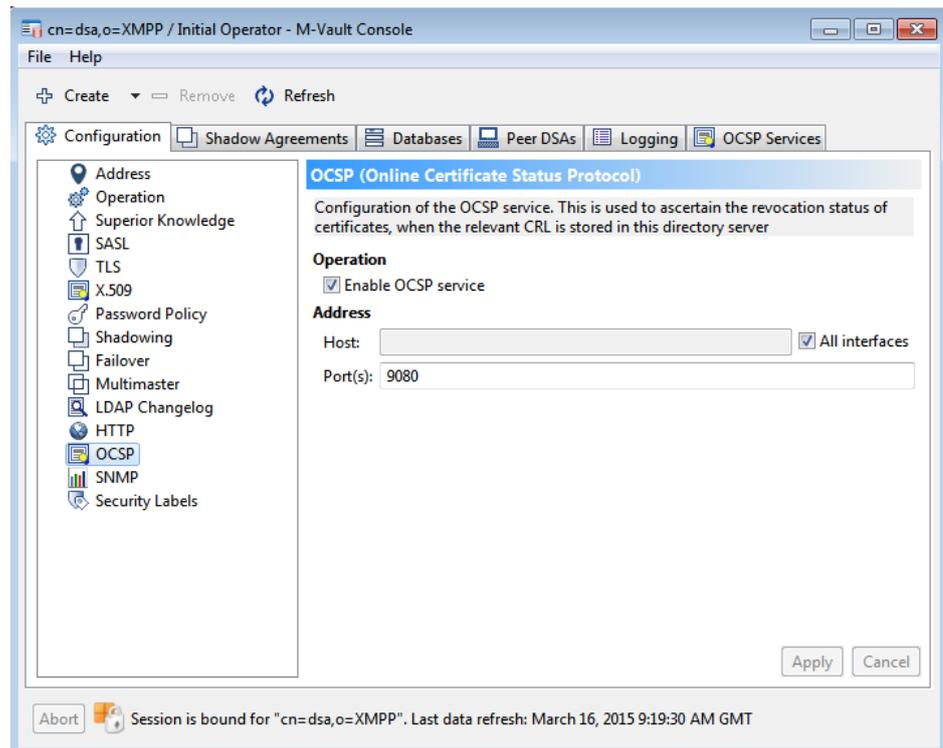
Examples (omitting the `type=crossCertificatePair` field for the sake of brevity):

```
resource=issuedBy.p7c$scope=issued_by$filename=issuedBy.p7c
resource=issuedTo.p7c$scope=issued_to$filename=issuedTo.p7c
resource=all.p7c$filename=issuedTo.p7c
```

10.2 Configuring OCSP

To enable the OCSP service first connect to the Directory Server using M-Vault Console, then navigate to **OCSP** page of the **Configuration** group. Set the **Service Enabled** checkbox and then specify the listen address by supplying a hostname (or select **All Interfaces** to

listen on all network interfaces) and a set of port numbers (which must not overlap with any ports used to serve HTTP). Note that the Directory Server will not start serving OCSPP until it has been restarted.



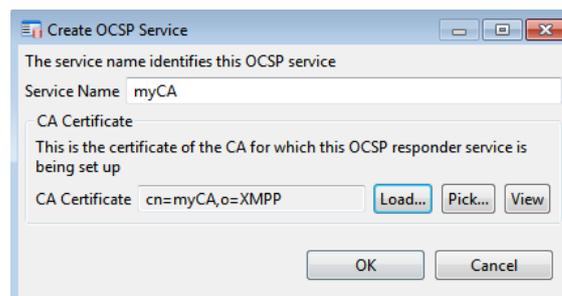
Once the OCSP listener has been enabled one or more logical services need to be set up, where each logical service corresponds to a single CA.

10.2.1 Configuring a Logical OCSP Service

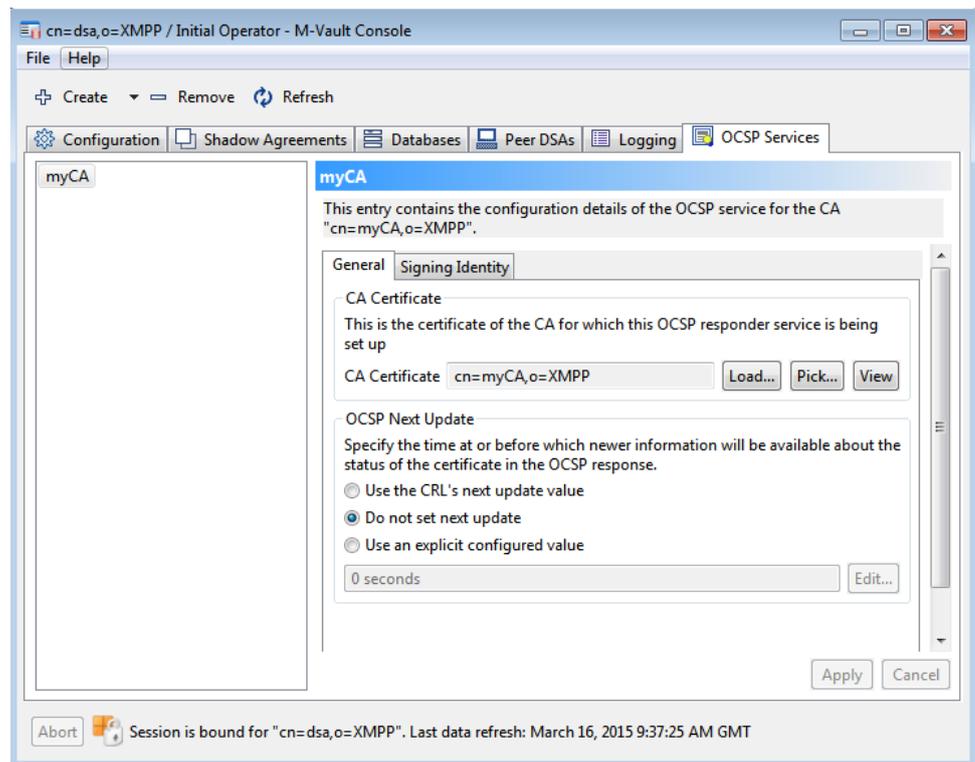
A logical OCSP service corresponds to the OCSP provision for a single given CA, and multiple logical OCSP services can be configured for provision by M-Vault. To support OCSP for a CA the M-Vault instance must contain the CA or CRL DP (CRL Distribution Point) directory entry that contains the relevant CRL. To configure a service on that basis navigate to the **OCSP Services** tab of the DSA's properties window.

In order to setup an OCSP service for a CA, select **Create** → **OCSP Service** from the toolbar. A dialog will appear to provide the following details.

- **Service Name** An informal string value used to identify the OCSP service.
- **CA Certificate** The CA's certificate. This is used to derive hashes of the CA's name and public key, as these are required in the OCSP protocol. Note that the Distinguished Name (DN) of the CA entry is derived from this certificate and this is where the OCSP implementation will look for the relevant CRL.

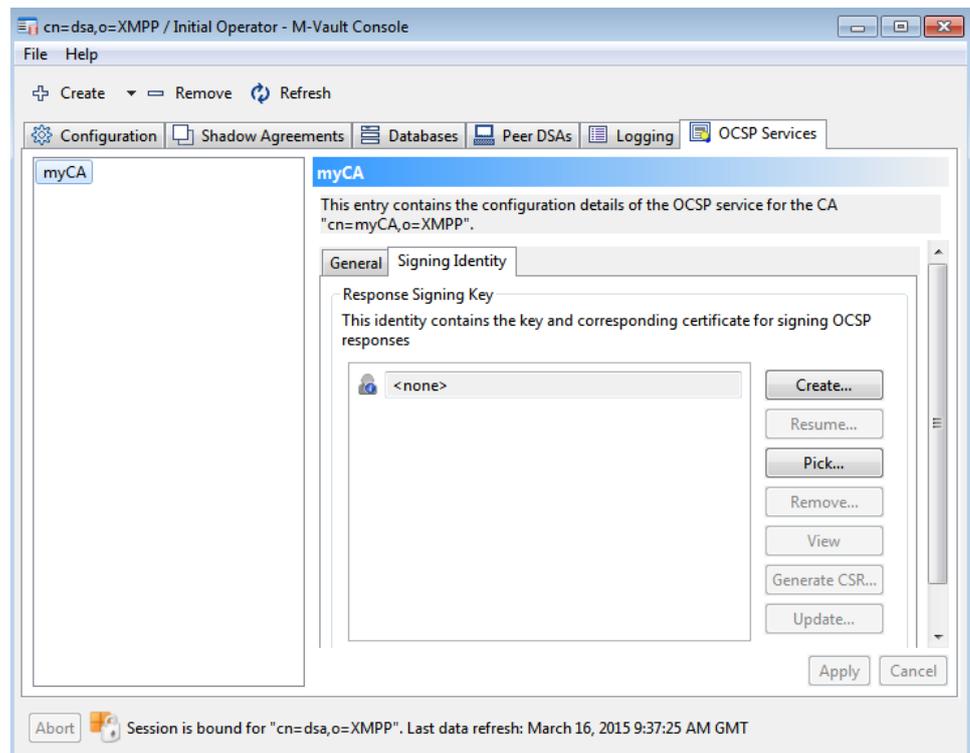


The following figure illustrates the MVC window after an OCSP service has been created as a first step:



Once a service is created, the next step is to create a **Response Signing Key** which is the identity used to sign OCSF responses. This can be the CA's identity or, more likely, an identity certified by the CA for the specific purpose of signing OCSF responses. An OCSF signing identity will most likely be conferred the `id-pkix-ocsp-nocheck` identity, as this tells the OCSF client not to perform full path checking of the response signature. This identity should be provided in a PKCS#12 file and associated passphrase as held on the *server*. The PKCS#12 file should contain the signing key and, at a minimum, the certificate of the signing identity.

The UI on selecting the **Signing Identity** tab will guide you with the process of generating a signing Identity for the OCSF service if one does not exist. The steps for generating an Identity are similar to the ones as described in [Section 3.10.1, "Generating a certificate request"](#). Alternately, you can pick an existing PKCS#12 file if a signing identity exists for the CA.



Existing logical OCSP services can later be removed or modified in order to change the CA certificate or response signing identity by selecting the service on the **OCSP Services** tab.

Chapter 11 Monitoring the Directory

The Directory Service can be monitored in several ways. Logs can be inspected, and certain status information and statistics are kept by the system which can be displayed.

11.1 Logging

This section begins with the use of M-Vault Console to configure logging. This is followed by an overview of the general structure of the Isode logging subsystem.

Note: Since in most cases you will be using M-Vault Console to view and update logging configuration, it is not necessary to be familiar with the details of the logging implementation, but it may be useful to have an understanding of some of the concepts involved.

11.1.1 How logging works

This section contains information to help you understand the content and configuration of log files in more detail.

11.1.1.1 Record types

All Isode applications generate two types of log records during normal execution: audit records and event records.

- Audit records are used to record “auditable events” – Directory Server startup and shutdown, for example. Audit records do not have a severity level associated with them, and have a well-defined format, so that they can be easily parsed.
- Event records are used to record errors, normal program operation, or to provide debugging information. They are associated with a particular severity level, and contain free-form text with substituted data items. The free-form text is contained in a separate dynamically-loaded library (on Windows) or a message catalog (on UNIX), which makes it possible to replace the standard set of English messages with equivalent text in other languages simply by substituting a suitable message file.

No output mechanism is directly associated with log records. When an event or audit record is generated by an application, then whether or not it is logged, where it is logged to, and what the output of the log looks like, depends on what output streams have been configured.

11.1.1.2 Output streams

An output stream is a description of how a particular set of event and audit records should be recorded or displayed. Multiple output streams may be configured for an application, and whenever an event or audit record is generated, the logging subsystem checks to see which, if any, of the available output streams is eligible to process it.

As well as defining which records are eligible to be logged, the configuration of an output stream also determines the format of the messages that are produced by the stream.

This means that a single event or audit record may be processed by one or more separate streams (or by no stream at all), and that, in the case of multiple streams, the messages output by the streams may be of differing formats, containing more or less detail. For example, it would be possible to configure one output stream to generate a brief message

about all “warning” level events, and another to generate a detailed message about a specific “warning” event which is of particular interest.

Five stream types are currently available:

- the `file` type, where the records are output to a file
- the `mpp` type, which sends logging over the network to Isode’s server watch daemon, which enables further processing and/or consolidation.
- the `system` type, where the records are passed to the system event log (syslog on UNIX-type systems and the **Application Event Log** on Windows)
- the `tty` type, which is identical to file type, except that the records are written to either `stdout` or `stderr`

It is possible to access log files remotely. If M-Switch is co-located, the Queue Manager provides remote access to authenticated clients. If M-Switch is not being used, Isode’s eventd server can be used instead. See [Section 11.1.5, “Remote monitoring of log files”](#) for more details.

11.1.1.3 Format of messages in output streams

When a given audit or event is generated, then for each output stream that is configured to process records of that type, the settings for the output stream determine the format of the message that is output. In the case of `file` and `tty` streams, the stream may be configured to contain any combination (including none) of the following fields:

- **date and time:** the format of date and time is configurable on a per-stream basis.
- **program name:** the name of the program generating the message. Any “isode” prefix will have been removed, and the program name will be truncated to 8 characters.
- **process id**
- **thread id:** this field may be useful to distinguish separate threads in the same process.
- **username:** the username of the process which generated the record. This field is only meaningful on Unix systems. If the username cannot be established, then a numeric UID is logged.
- **severity:** audit records have no associated severity, but event records always have a severity, which, if displayed, is represented using one of the following single letters, as follows:

I – Info	N – Notice	S – Success	D – Detail
W – Warning	E – Error	F – Fatal	C – Critical
L – AuthOK	A – Authfail	X – Debug	P – PDU

- **facility code:** the name of the facility which generated the message. Audit records are not associated with a particular facility.
- **message identifier:** an identifier representing the event. Audit records do not have a message identifier.
- **text:** the formatted text describing this event. Audit records do not have a text field.
- **supplementary audit record parameters**

For certain types of audit records, extra information may be associated with the record, and if the stream is suitably configured, this will be included as a sequence of “key:value” pairs on the end of the message.

As an example, consider that a Directory Server generates both an audit record and an event record when it is shut down. Assuming audit and event streams have been created to capture such records, and that the streams are configured to display all possible fields, then the resultant message from the audit output stream will look like this:

```
2019-08-08 15:22:56 x500dsa 12068 (root ) Stopped
```

while the corresponding event record will look like this:

```
2019-08-08 15:22:54 x500dsa 12068 (root ) N-DSA-ServiceState
state:Stopping name:"cn=DSA1,c=xx"
dir:"/var/isode/master-dsa-db"
2019-08-08 15:22:56 x500dsa 12068 (root ) N-DSA-ServiceState
state:Stopped name:"cn=DSA1,c=xx"
dir:"/var/isode/master-dsa-db"
```

In both cases, the date, program name (`x500dsa`), process and thread id (`00464.00150`) and username (`root`) are included.

The event record contains severity (N, for “Notice”), facility (DSA) and identifier (ServiceState) of the event, as well as a number of event specific fields, e.g. name (the DN of the DSA that stopped).

The audit record above is identified by the fixed string `Stopped`. This type of audit record has no associated supplementary parameters. For audit records which contain supplementary information, this will (assuming the stream is configured to display them) be shown in the log as a sequence of `key:value` pairs, for example:

```
2019-08-08 13:47:33 x500dsa 12068 (root ) Search-res-out
id:16 assoc:21 hits:1
```

In this case, the `Search-res-out` operation (search result sent out to client) is logged with three supplementary parameters, `assoc` (the network connection ID), `id` (the operation invocation ID) and `hits` (the number of matches found).

11.1.1.4 Logging configuration

Information about output stream configuration is stored as XML data. All Isode applications will load the XML contained in the file `logtailor.xml`, if it exists, at startup. The search path for `logtailor.xml` is first (`ETCDIR`), and then (`SHAREDIR`). The filename and location can be overridden if required by defining the environment variable `LOGTAILOR` to be an alternate filename or filepath.

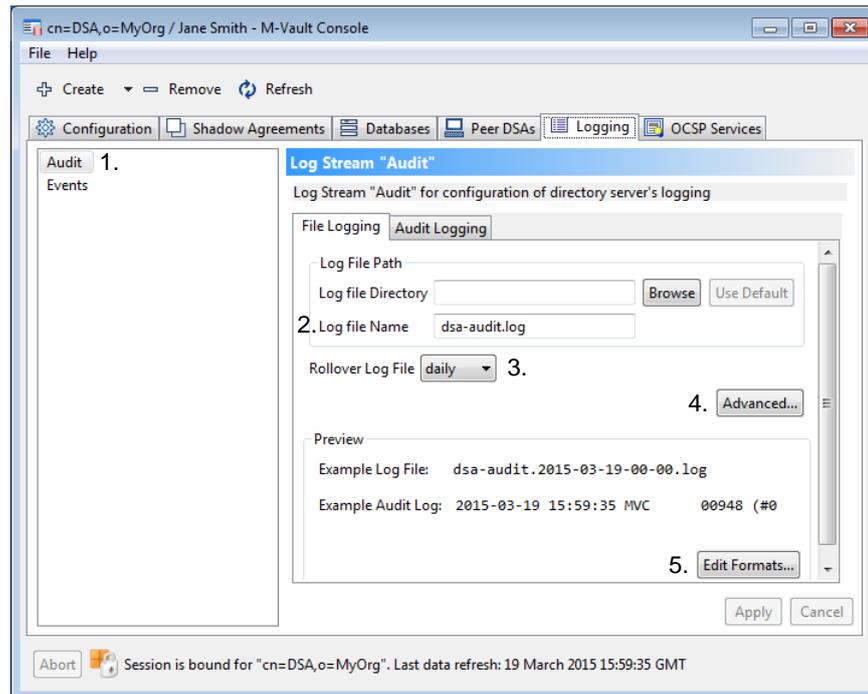
An application may then load a private stream configuration. For Isode DUAs (such as M-Vault Console, Tcldish, and Sodium), this is contained in the `dualogging.xml` file, located in either (`ETCDIR`) or (`SHAREDIR`). In the case of M-Vault Directory Servers, the corresponding `dsalogging.xml` file contains configuration used for the period when the Directory Server is being started, but in normal operation, the logging configuration is maintained as part of an entry inside the Directory itself.

While it is possible to edit the XML files using a normal text editor, the managing of logging streams is typically performed using a GUI, which is run either as a standalone tool (in the case of individual XML files, see [Section 11.1.3, “Using the standalone logconfig tool”](#)), or within M-Vault Console (to manage Directory Server logging configuration, see [Section 11.1.2, “Changing Directory Server logging using M-Vault Console”](#)).

11.1.2 Changing Directory Server logging using M-Vault Console

You can use M-Vault Console to configure logging for your Directory Server.

Connect to the Directory for which you want to configure logging and click the **Logging** tab.

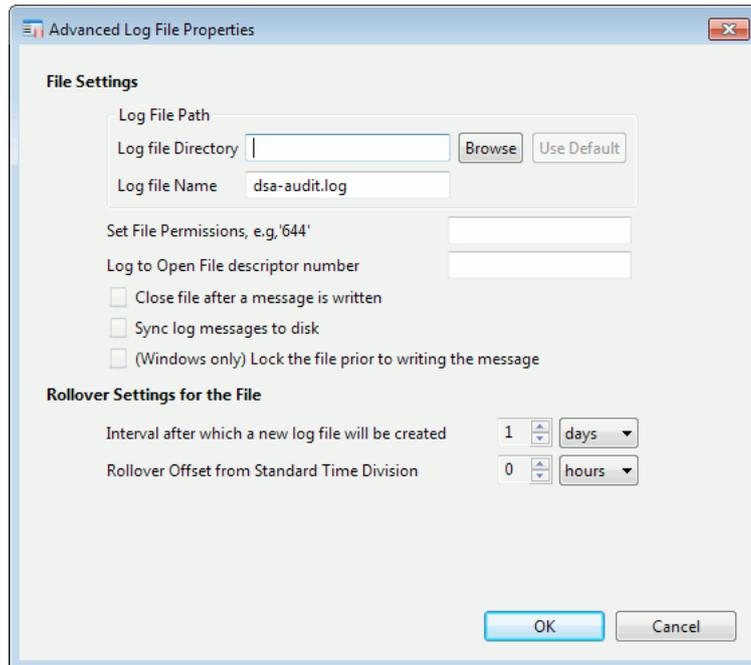
Figure 11.1. Configuring logging in M-Vault Console

1. Two different record types (see [Section 11.1.4, “What is written to the log files?”](#)) are shown on the left (**Audit** and **Events**):
 - If you select **Audit** (as shown above), the information on the two pages on the right relates to the audit record type and the second page on the right is called **Audit Logging**.
 - If you select **Events**, the information relates to the event record type and the second page is called **Event Logging**.
2. Either **Browse** to find a suitable directory in which to store the log files, or click **Use Default**.

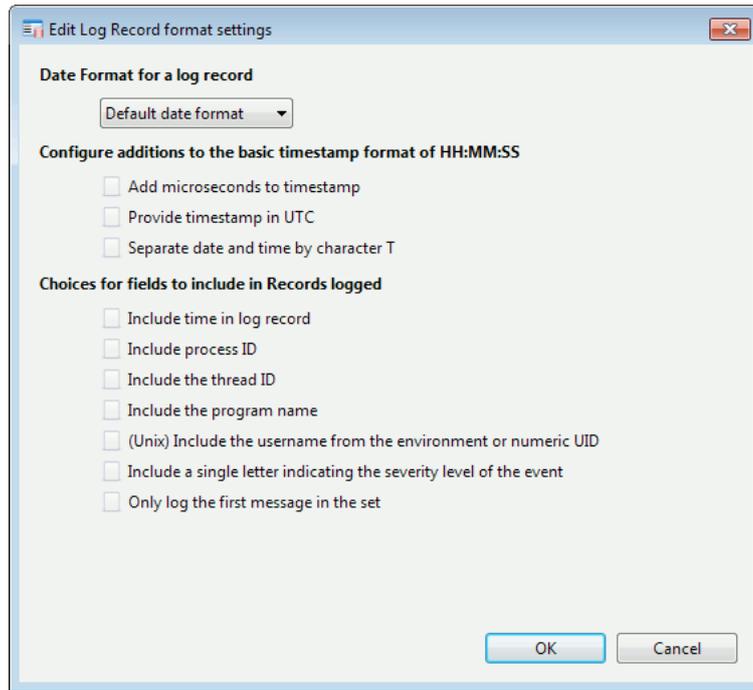
The **Log file Name** is shown by default. For the **Audit** log it is *dsa-audit.log* and for the **Event** log it is *dsa-event.log*.

3. You can choose to create a new log file at regular intervals. The default is for a new file to be created daily, but you can change this to hourly or weekly if you prefer. The name of the log file contains the date and time at which it was created; for example, *dsa-event.2010-06-13-00-00.log*

Click **Advanced** to specify more details.



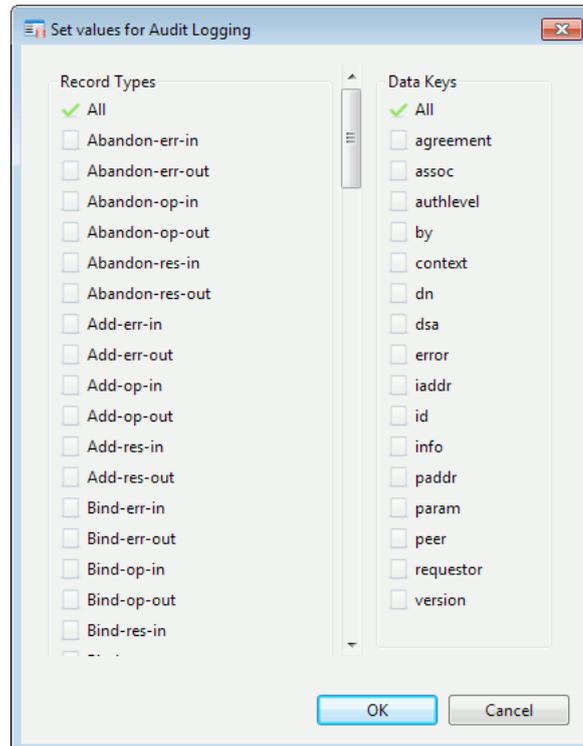
- **Log File Path** information is as on the basic page.
 - **Set File Permissions** You can set read, write and execute file permissions.
 - **Log to Open File descriptor number** Enter the integer that identifies the file.
 - **Close file after a message is written** The file is opened to write the message and closed again immediately afterwards. This helps to ensure security of the data but there is a significant performance overhead.
 - **Sync log messages to disk** asks the operating system to ensure that messages are written to disk. The helps to ensure security of the data but there is a significant performance overhead.
 - **(Windows only) Lock the file prior to writing the message** ensures that if multiple processes are logging to the same file, the messages are not mixed.
 - **Rollover Settings for the File** sets a rollover interval for the file and enables you to specify an offset from the default start point of the specified period. For example, the default start time for a daily roll-over is midnight, and the default start point for a weekly roll-over is 00:00 hours on a Sunday.
4. Click **Edit Formats** to change the content of the log file. Examples are shown of the current format in the **Preview** area – you may need to enlarge the window to see them.



- Select your preferred date format from the options available. The default is YYYY-MM-DD HH:MM:SS.
- Select any additions you want to make to the timestamp.
- Select any fields you want to be included or excluded from the records. This option is: a green tick specifies that a field will be included, a red cross specifies that it will not. Leave the option blank if you do not want to specify.

11.1.2.1 Audit logging

Choose whether to record no audit information, all audit information or specific audit information. If you chose **Specific Audits**, click **Edit** to select the ones to include.

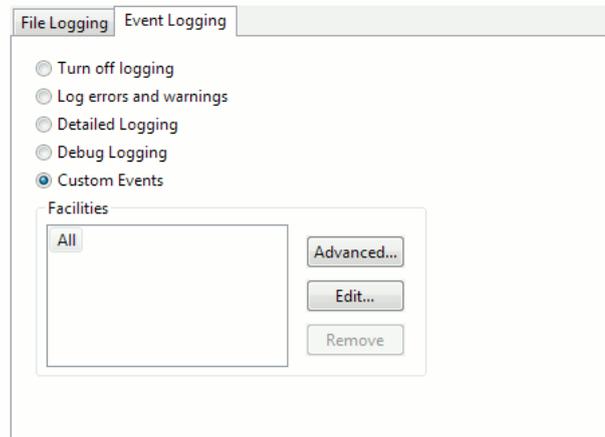


The example above shows that all **Record Types** and all **Data Keys** will be included.

- Click an item to include it.
- Click a included item to exclude it.

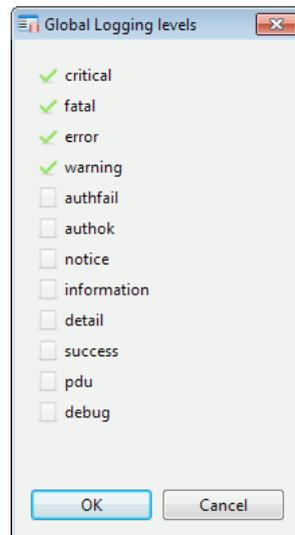
11.1.2.2 Event logging

First choose the level of event logging you want to include in the log files.



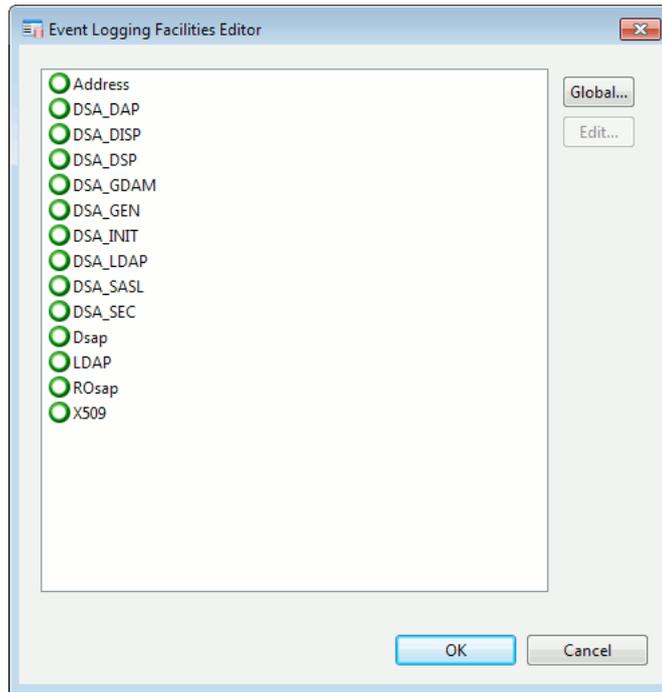
If you chose **Custom Events**, click **Edit...** to specify them.

Note: To see what is included and excluded at each level, select the level and click **Edit...** The example below shows what is included when **Log errors and warnings** is selected.

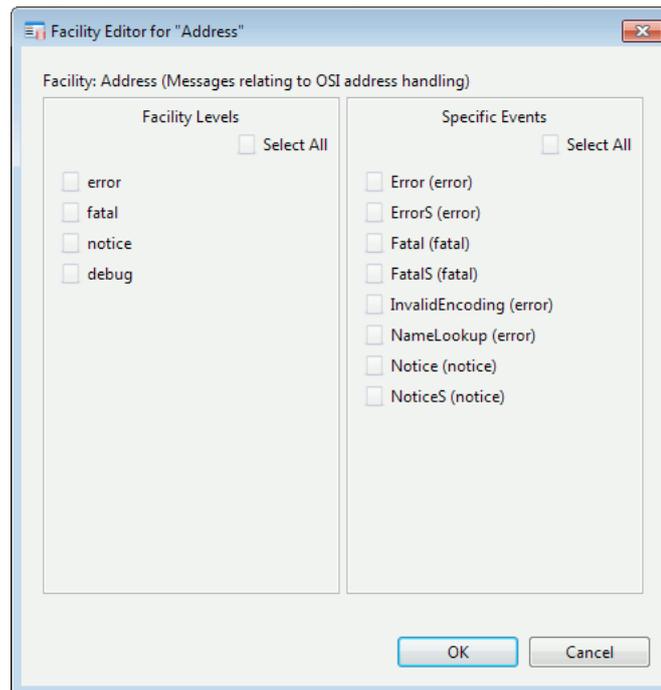


Click **Advanced...** to specify in more detail exactly what you want to log.

Note: A tool tip is displayed if you hover your mouse over an entry giving details of the type of event referenced by that entry. For example, **Address** displays **Messages relating to OSI address handling**.



- **Global...** takes you to the **Global Logging Levels** window (above).
- To specify more details about a particular event type, select it and click **Edit...**



11.1.2.3 Creating a new logging stream

You can create a new logging stream for information of a particular type or from a specific tool or program. To create the new log stream:

Click **Create** on the tool bar and select **Log Stream** from the options displayed. Follow the instructions given in [Section 11.1.3, “Using the standalone logconfig tool”](#), except that you will not need to specify the **Application Type**.

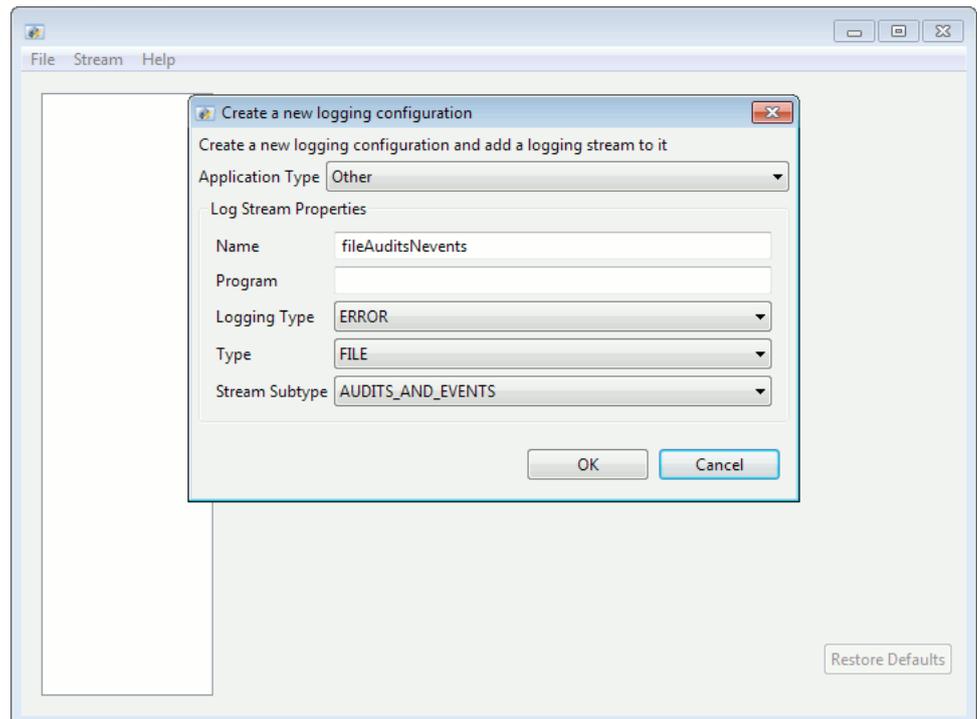
11.1.3 Using the standalone logconfig tool

This section describes the use of the standalone logconfig tool, and as an example shows how you can create new log streams in the *logtailor.xml* file. This file, if it exists, is loaded by all Isode applications, although in most cases the application will go on to define and use application-specific streams. This section assumes that no previous version of *logtailor.xml* exists in (*ETCDIR*).

On UNIX systems, run */opt/isode/sbin/logconfig*

On Windows, a shortcut to the **Log Configuration Tool** will have been set up in the **Isode** folder on your **Start** menu.

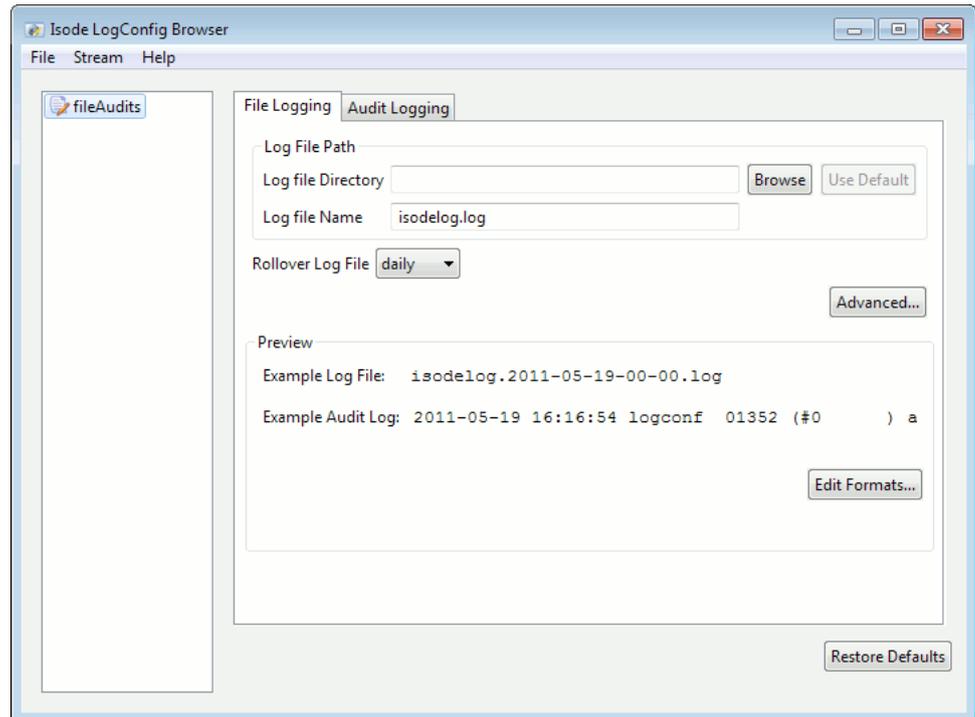
If no logs have been configured, the tool opens displaying a window ready to create a new logging configuration.



1. Select **M-Vault** from the list in **Application Type** (assuming you are creating a configuration for the Directory Server).
2. The default **Name** will change to a new name if you change the **Type** or **Stream Subtype** (final options).
3. If you want to associate this particular logging stream with a program or utility, type its name in **Program**. For example, if you want to create a logging stream for the **dbulk** utility, type **dbulk** here. If you do not specify anything here, this stream will be used for all programs and utilities.
4. Select the **Logging Type**: **ALL**, **DETAIL**, **ERROR**, **NONE** or **WARNING**.
5. Select the **Type** of output (see [Section 11.1.1.2, “Output streams”](#) for an explanation of the options available).
6. Select the **Stream Subtype**. This option is only available if **FILE** or **TTY** was selected in **Type**. You can choose from **AUDITS**, **EVENTS** or **AUDITS_AND_EVENTS**.

Once the stream has been created, it should be set to output to a suitable log file, and will be used by any application which uses *logtailor.xml*. The stream may be configured using the various available tabs.

Since *logtailor.xml* is potentially used by multiple Isode programs, you may wish to create several program-specific streams. To create a new stream, select **Stream** → **Add** from the menu.



All of the other configuration options available in the standalone logconfig tool are identical to those within M-Vault Console, and are described in [Section 11.1.2, “Changing Directory Server logging using M-Vault Console”](#).

11.1.4 What is written to the log files?

The default configuration for an M-Vault Directory Server provides two file output streams:

- The **Events** stream captures all event records with severity of Notice (N), Warning (W), Error (E), Fatal (F), Critical (C), or AuthFail (A). These are output to *dsa-event.log* in (*LOGDIR*).

If you are reporting a potential bug to bug-report@isode.com, then it may be useful to configure the **Events** stream (or to create another output stream), so that while reproducing the problem, all levels of event records are logged. The resulting output should then be included in your report.

Note: Operational Directory Servers should not be run with full logging, as this can significantly impact performance (and use up large amounts of disk space).

- The **Audit** stream captures all audit records, with the exception of those relating to “internal” events, and outputs them to the *dsa-audit.log* in (*LOGDIR*).

The following sections describe logging behaviour when these default settings are in effect. However, since the streams are fully configurable, and streams may be added or removed, it may be that the filenames and file contents will be different on a given system.

11.1.4.1 Events stream

Note: The file *dsa-event.log*, in (*LOGDIR*), is roughly analogous to the *dsap.log* file which was used in pre-11.0 releases of M-Vault.

A new record is appended to it whenever the Directory Server generates an event with a severity level of N, W, E, F, C or A (see [Section 11.1.1.3, “Format of messages in output streams”](#)). Problems that prevent the Directory server from operating correctly have a severity level of E, F or C. Possible problems that may be worthy of investigation have N, W, or A severity codes.

For an example of what the contents of *dsa-event.log* look like, see [Section 11.1.1.3, “Format of messages in output streams”](#).

11.1.4.2 Audit stream

Note: The file *dsa-audit.log*, in (*LOGDIR*), is roughly analogous to the *dsa.log* file which was used in pre-11.0 releases of M-Vault.

A new record is appended to this file whenever an auditable event, such as an incoming connection, or a Directory Server shutdown, is generated by the Directory Server.

Each audit record may include supplementary information which is shown as a sequence of *key:value* pairs. The types of audit records that may be logged, with their corresponding supplementary parameters, are described in the following sections. Bear in mind that it is possible to configure a stream so that audit message parameters are omitted from the log file; the examples below assume that all parameters are being logged.

11.1.4.2.1 Process start and termination

When the Directory Server has read its configuration files and initialized the GDAMs, a `Started(dsa, version, info)` message is logged. For example:

```
2019-08-07 13:57:50 x500dsa 12068 (root ) Started
dsa:cn=DSA,c=xx version:R18.0.0.0
info:"Copyright (c), Isode Limited, London, England."
```

When the Directory Server shuts down, it logs a `Stopped` message (which has no associated parameters):

```
2011-03-15 15:39:40 x500dsa 13120 (root ) Stopped
```

11.1.4.2.2 Association management

The start of an incoming association to the Directory Server is logged with a `Bind-op-in`, e.g.:

```
2019-08-08 16:01:32 x500dsa 01819 (root) Bind-op-in
id:0 assoc:21 context:DAP type:Anon
addr:Internet=127.0.0.1
```

The possible values of *context* are:

- DAP
- DSP
- LDAP
- DISP

When the association is terminated by the client `Unbind-op-in(assoc)` is logged. The *dn* parameter may be empty, in which case it will not be logged:

```
2011-03-16 12:44:18 x500dsa 13120 (root ) Unbind-op-in assoc:48
```

Outbound associations may also be made, for example when the server is chaining client operations or when replicating (DISP or multimaster). The example below shows an outbound shadowing (DISP) bind request:

```
2019-08-08 16:30:45 x500dsa 01819 (root) Bind-op-out
assoc:22 context:DISP
addr:"URI+0000+URL+itot://shadow.isode.net:19999"
```

11.1.4.23 Incoming and outbound operations

Audit records are logged for the following operations **Read, Compare, Abandon, List, Search, Add, Remove, Modify, ModifyDN** and **PasswdModify**. Common parameters for each of these records are (*assoc, id, dn, param, requestor, sig*). Search also supports a filter parameter. For example:

```
2019-08-08 16:15:07 x500dsa 01819 (dsm ) Search-op-in
id:1 assoc:21 dn:<empty> scope:Subtree
filter:sn=smith user:* oper:* vals:1
```

Each operation audit record comprises three components in the record name:

- Operation, e.g. Search
- PDU type, which is one of *op* (invocation), *res* (result) or *err* (error).
- Direction, which is one of *in* (inbound PDU) or *out* (outbound PDU).

11.1.5 Remote monitoring of log files

Access to the log files can be provided to clients running the Isode Event Viewer application. This can monitor specific files, or all log files, and is also able to provide details on specific error messages. For more details on the Event Viewer application, see the *M-Switch Administration Guide*.

The Event Viewer application connects to, and authenticates with, an instance of the Isode eventd server.

To configure the authentication parameters, edit the (*ETCDIR*)/*isotailor* file (a sample file is provided) and set the value for the *eventd_auth* key to the desired user ID and password.

To enable the server on Windows, start it using the Isode Service Manager as described in [Section F.1, "Linux services"](#).

To enable the server on other platforms, edit (*ETCDIR*)/*dsa.rc* and set the *USE_EVENTD* variable to "yes".

Chapter 12 Synchronising Directories (using Sodium Sync)

This chapter explains how to use Sodium Sync for synchronizing data between directories, LDIF files, CSV files and SQL databases.

12.1 Overview

Sodium Sync provides a mechanism to copy a set of data from a source Directory Server to a target Directory Server, and to ensure that the target remains up to date by performing regular updates to take account of any subsequent changes in the source Directory Server.

Synchronization occurs in one direction only: whilst changes, additions and deletions made to data held on the source Directory Server will be copied to the target, any local changes made to data in the target Directory Server will not be copied back to the source, and will normally be lost when the next synchronization operation takes place.

In particular Sodium Sync is designed to be able to handle synchronization from non-Isode DSAs (for example Active Directory) to Isode's M-Vault. Sodium Sync has a number of features to make it easier to deal with translation between directories which are not completely compatible with one another.

When configuring the synchronization operation, Sodium requires that you specify:

- the base of a subtree in the source Directory from which entries will be copied
- the location of the entry in the target Directory that will form the base of the copied subtree; any existing entries under this base entry on the target will be deleted

Sodium Sync will automatically rename entries if the source and target base DNs are different, and it is possible to synchronize between two separate subtrees on the same Directory.

In more advanced use, Sodium Sync also allows the specification of:

- constraints which limit the area beneath the subtree which will be copied
- a search filter which can be used to determine which entries from the subtree are included in the copy operation
- rules which determine which attributes within an entry are copied
- rules which determine how particular attribute values are copied or translated between different systems
- rules which may result in new attributes and values being added to the data on the target Directory
- rules which modify the RDN of entries as they are moved
- how to handle orphan entries: by reparenting them (flattening the tree), or by replacing the missing parents

In addition, it is possible to perform synchronizations that operate partially or completely on LDIF or CSV files or SQL databases instead of directories. For example:

- compare source and target trees on directories and generate a change-LDIF (an LDIF containing change records) instead of applying the changes to the target

- apply a previously generated change-LDIF to a target Directory
- compare a subtree from LDIF, CSV or SQL with a subtree from a Directory to generate changes (i.e. synchronize from LDIF)
- compare two LDIF subtrees (from the same or different LDIF files) to generate changes.

In more recent releases, the Sync has been extended with features to handle additional variations on the basic synchronization process. For example:

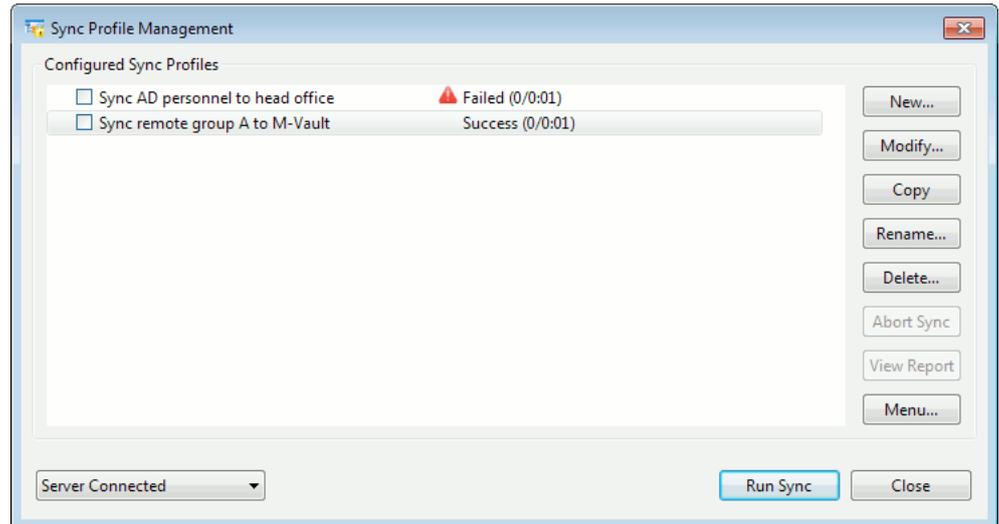
- A ‘cached’ sync mode which does not require access to the target DSA in order to generate changes. This is suitable for ‘air gap’ synchronization, or sync-over-Email, or when the remote server is on a slow network connection.
- Queues and hook support, to handle situations where external scripts or executables must transmit, receive or otherwise process the input or output of the sync. This also integrates with the M-Switch FTBE (File Transfer by E-mail) functionality to enable synchronization over E-mail.
- Correlated ‘merge’ syncs ([Section 12.8, “Correlated syncs”](#) and [Section 12.4, “Setting up a merge-sync”](#)), where two sources with different information structures are correlated by exact or inexact matches on a certain key piece of information that they have in common (for example an ID number, or a user’s full name). When those correlations are approved, selected pieces of data may be synchronized from one side to the other, merging them into the existing data.
- Replication Workflow with Checks ([Section 12.7, “Sync groups and replication workflow”](#)), where a number of syncs run in sequence, and the sequence can be aborted if any sync fails, or if any configured check fails.

The following sections provide more detailed information on how Sodium Sync is configured. There is also a discussion of an example set of mapping rules which can be used to synchronize between Active Directory and M-Vault.

12.2 Setting up a simple sync from Active Directory

In this example we will configure a simple synchronization between Active Directory and M-Vault, using the default mapping rules.

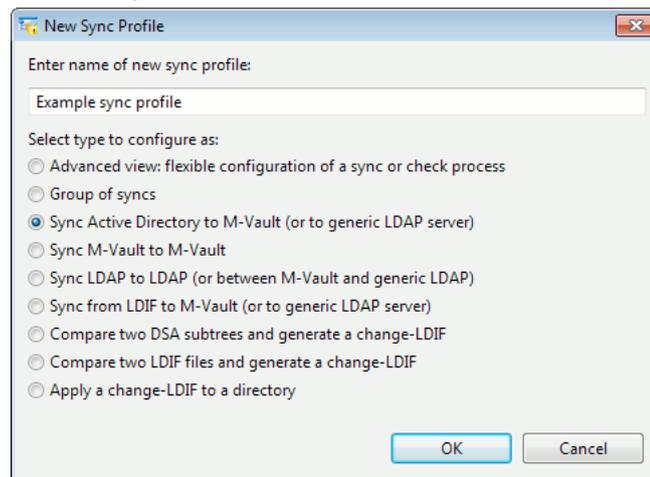
Sodium Sync is configured and controlled from the **Sync Profile Manager**. This is started from within a Sodium session by selecting **Session** → **Sync** → **Manager** from the menus. The **Sync Profile Management** window shows all configured sync profiles and the next synchronization time for scheduled profiles.

Figure 12.1. Sync Profile Manager dialog

To set up a simple synchronization from Active Directory to M-Vault:

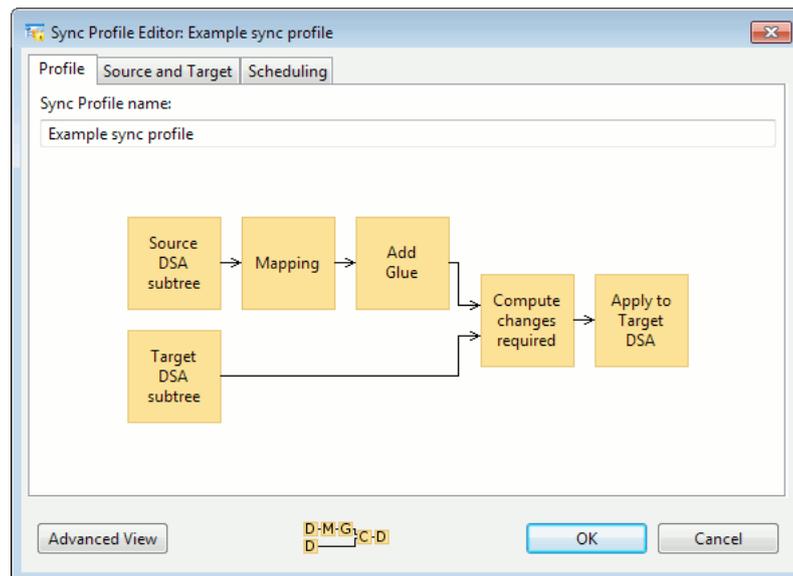
1. Click **New...**

The **New Sync Profile** box is shown.



2. Give the profile a name.
3. Select **Sync Active Directory to M-Vault (or to generic LDAP Server)**.
4. Click **OK**.

The **Sync Profile Editor** is displayed, in simple view.



The flow diagram shown in the editor illustrates the flow of data during the synchronization process: in this case, source DSA subtree entries are read, then mapped using the Active Directory mapping rules, glue entries are added if required, and then this is compared to the target DSA subtree to find what changes need to be made, which are finally applied to the target DSA.

5. Click the **Source and Target** tab.

Enter the source and target bind-profile information and starting-point DNs. For example, the starting-point DN for Active Directory might be **cn=Users,dc=acme,dc=com**, and the starting-point DN on M-Vault might be **cn=AD-Users,c=US**, which is a container entry you should create separately.

Finally , and then on **OK** to save the profile.

6. Click the **Scheduling** tab to enter scheduling details, if required.
7. Click **OK**.

Note: Unattended synchronization will only happen if the bind profiles have associated passwords. This means you must encrypt your bind profiles (by clicking **Encrypt...** in the **Bind Profile Manager** window) and enter passwords for the bind profiles if this has not already been done.

Caution: Any existing entries under the target starting-point DN will be deleted by the synchronization operation. These entries will be replaced with the subtree synchronized from the source Directory. This is the reason why a container entry is normally used, rather than synchronizing directly to a top-level DN like **c=US**.

12.2.1 Modifying a profile

To modify a profile, go to the **Sync Profile Management** window, select the profile you want to change and click **Modify...**

12.2.2 Running a sync

- To run a sync manually, go to the **Sync Profile Management** window and click **Run Sync**.

Note: Any errors are shown on a **Log** page that will be available from the main window.

- Scheduled syncs will only run if the Sync Server is running (see [Section 12.10, “Configuring Sodium Sync Server”](#)). You may close the **Sync Profile Management** window, unbind from any DSA, close Sodium and log off your session without affecting the operation of scheduled syncs.

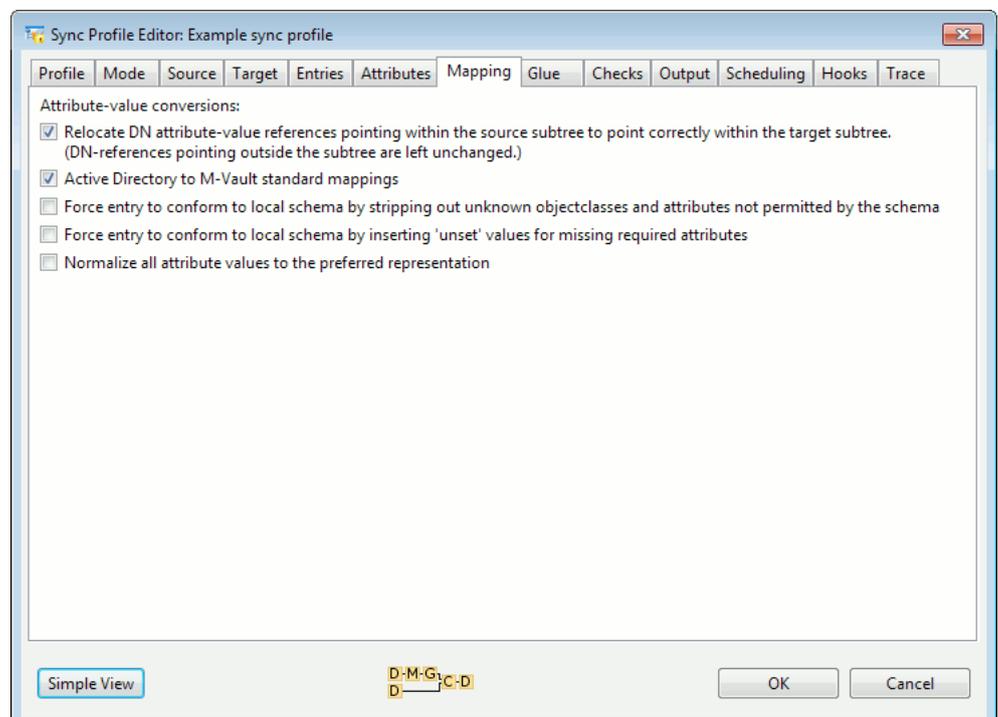
The status information shown in the **Sync Profile Management** window is updated when a sync occurs, and a message is added to the DUA event logs to record the event.

If errors occur, then the first few messages may be displayed by double-clicking on the sync name or by selecting **Show Status** either from the right-click menu or by clicking **Menu...** and selecting from there. The full list of errors will be written to the DUA event log stream, which by default means that they will be found in the DUA event log files.

12.3 More advanced use of Sodium Sync

The advanced view in the **Sync Profile Editor** exposes the full current functionality of Sodium Sync, with some future planned functionality shown greyed-out.

Figure 12.2. Advanced view of the Sync Profile Editor



A few parts need explanation:

- The **Source** page has an option for **Optimised data pre-fetch**. When this option is selected, all the data from the source directory is fetched in one pass, using as few

directory operations as possible. It is temporarily cached to disk, and then the main part of the sync proceeds using the cached data as source. This is useful when the source directory is remote with a large round-trip-time which makes requests slow. This requires that the source directory support page results or large searches, so it will not work with all directories.

If possible, the data is fetched using a single LDAP search. However in the case of chop-points in the **Entries** tab, the search must descend one level at a time down to the level of the chop-points in order to exclude them. In addition if **Also apply filter to parent entries** is enabled, then much of the optimisation is lost as it is necessary to search level by level. However using an entry selection filter without **Also apply filter to parent entries** *will* work efficiently with this optimisation; it is possible to fetch an entire subtree with an entry selection filter using a single LDAP paged search.

The page-size specification below **Optimised data pre-fetch** allows page-sizes to be tuned. The first number is the page-size used when requesting just DN's from the remote directory, and the second is the page-size used when requesting whole entries from the remote directory. Larger numbers may make things slightly faster, but only if the remote directory supports those larger page-sizes.

- The **Entries** page allows you to control which entries are synchronized. An arbitrary LDAP filter may be used to select the entries to include. LDAP filters may be used with all source types, even CSV, SQL or LDIF files.

The LDAP filter may be applied in two ways, either globally, like an LDAP search, where child entries are included whether or not the parent matches (which may result in gaps in the hierarchy), or top-down where the filter must match all parents as well as the target child entry for that child to be included in the sync. For example, if you want to include all **person** entries, but they are found in a subtree reached via **OU** entries, then with **Also apply filter to parent entries** selected, you would need a filter of “((objectclass=organizationalUnit)(objectclass=person))”, not simply “(objectclass=person)”. If **Also apply filter to parent entries** is not selected, “(objectclass=person)” would work, but all the OU entries would be missing and would have to be replaced with glue (see the **Glue** tab) for the sync to be successful.

A maximum depth to sync in the subtree may be specified, and “chop-points” may be placed at particular DN's to exclude whole subtrees, chopping either before or after the configured DN.

- The **Attributes** page allows you to control which attributes are synchronized. The filtering rules are selected by objectclass using the same keyclass scoring rules as are used for Sodium templates. For any given keyclass, a list of attributes and objectclasses may be included or excluded. For any given entry, the rule with the highest priority whose keyclass matches the entry's objectclasses will be used. The default is to pass through all attributes.

For example, we may wish to exclude the **userPassword** attribute from the synchronization when copying **person** entries. To do this:

1. Click **Add a rule**
2. Click **Change** next to the objectclass
3. Enter **person**, then click **OK**.
4. Click **Change** next to **Allow all attributes** and select **Delete these attributes**.
5. Click **Change** next to **(none)**.
6. Select **userPassword**, then click **OK**.

By default both entry and attribute filtering are applied only to the source subtree before comparison to the (unfiltered) target subtree. This is good when you want the target to be a precise copy of the filtered version of the source subtree. However, by using filters on the target subtree as well, you can choose which entries (or even attributes) in the target subtree are affected by the sync, therefore allowing merging or selective copying

of data. In this case the entry and/or attribute filters should be applied to both source and target subtrees. For example you could have two syncs with the same target subtree, one filtering on “(mail=*company1.com)”, and the other on “(mail=*company2.com)”, and one sync would leave untouched the entries belonging to the other. Merges like this require some care in setting up. See [Section 12.4, “Setting up a merge-sync”](#) for details on setting up an attribute-based merge.

- The **Mapping** page allows selection of various preset mapping rule-sets. These are defined in the *mapping-rulesets.xml* file shipped with Sodium, and may be reconfigured if necessary (see [Section 12.11.1, “Configuring mapping rule-sets”](#)).
- The **Glue** page can be used to handle the situation where the mapping or filtering has left a child without parents. This may happen especially as a result of the force-conformity options on the **Mapping** page (which may be implied by other rule-sets as well) when the source DSA contains parent entries with completely unknown schema.
 - **Add glue entries for missing parents** creates synthesized entries, which have an objectclass of **untypedObject** and appear in Sodium’s main window as **Container** entries, in the place of the missing parents.
 - **Reparent orphan entries to nearest ancestor** reparents the stranded children up to the nearest known ancestor. This can be used to intentionally flatten trees.
- The **Checks** page allows checks to be configured as part of the sync (see [Section 12.6, “Checking syncs”](#)).
- The **Trace** page contains options that may aid in debugging problem configurations.
 - **Generate a debugging trace to the file:** saves a trace of the entire sync operation to the specified file. With all the options enabled, detailed information on the processing of entries through all the parts of the flow-graph is saved
 - **For output over DAP or LDAP, save all failed changes to an LDIF file:** saves failed changes with comments that show the resulting error to the specified file.
 - **Archive and date-stamp old trace and failed-changes files** enables you to keep of files instead of overwriting them.

12.3.1 Limits

- Sodium Sync uses a streaming model, so there are no limits on the total size of the subtree which may be synchronized.
- If the Directory supports paged results, then the fan-out at any level of the tree is limited only by available memory (required to sort the list of DNs). If the Directory does not support paged results, then the fan-out is limited by the administrative size limit configured in the DSA. Sodium Sync will warn if an administrative limit has been hit.
- The size of the entries that can be synchronized is limited only by available memory.
- For **Optimised data pre-fetch**, there must be enough disk space to store the temporary copy of the source sub-tree.

12.4 Setting up a merge-sync

A merge-sync is one in which individual attributes are synchronized between directories rather than whole entries. This is a special case which requires some care to set up, and so is described below.

As an example, you may have two source directories on different systems, each a ‘master’ for different attributes. One will be used to synchronize the entries and the bulk of their attributes to the target, and then the other will be used as a merge-sync to fill in just a few

additional attributes. The merge-sync relies on the entries already being present on the target server from the first sync.

The first sync is configured as a standard basic sync, with the following addition:

- On the **Attribute** tab, a filter is set up that excludes the 'merge' attributes that will come from the second Directory. This needs to be set to apply to both source and target. This filter stops the first sync from trying to delete the attributes merged in by the second sync.

The second sync (the merge-sync) is configured as follows:

- On the **Entries** tab, the entry filter uses an LDAP search to match just those entries which have attributes that need synchronizing. This usually means leaving **Also apply filter to parent entries** unselected.
- On the **Attribute** tab, the attribute filter is set up to include only the naming attribute (e.g. **cn**), **objectclass**, and the 'merge' attributes that need merging in.
- Both entry and attribute filters are set to apply to both source and target.
- On the **Output** tab, the **Safeguard merge-sync: block entry additions or deletions** is selected.

The result of the filters on the merge-sync is that the comparison processing node only sees the differences between the specified 'merge' attributes on the source and target, and so only generates changes which correct those particular values on the target. The result is that only the 'merge' attributes are synchronized from source to target.

However, this works only when the entries exist on both source and target. If they are out of sync due to a delay between setting up a new entry on one master compared to the other, then the merge-sync might think that it should try to add or delete an entry. This is not what is desired, so the **Safeguard merge-sync** option is used to block add-entry and delete-entry operations, generating a warning instead. This makes a merge-sync safe even when there may be temporary discrepancies between the master servers.

With the syncs configured as described, both can be scheduled to run regularly, and the target will be maintained as a stable merge of the configured data from the two source directories.

12.5 Synchronizing to Active Directory

Active Directory (AD) can be a challenging sync target for a number of reasons and various techniques may be used in Sodium Sync to handle the problems that come up. This section documents the problems and work-arounds available when synchronizing from a typical LDAP or X.500 Directory to AD.

The following issues need to be considered:

- The source data has to be manipulated either to strip out attributes that AD will not accept or to convert their values to attribute types that AD will accept. The conversions required depend on the source data to be synchronized, and may be handled with entry and attribute filters. For complex translations, the conversion may require a custom ruleset to be created in the mapping rulesets file (see [Section 12.11.1, "Configuring mapping rule-sets"](#)).
- AD adds a number of internal attributes to entries, but does not treat them as operational attributes. For this reason, the Sync must be manually configured to filter these

pseudo-operational attributes out of the incoming data. To do this, on the **Attributes** page:

1. Click **Change** alongside **Apply filters to source, before processing** and change it to **Apply filters to source and target, before processing**.
2. Click **Use base AD list** (the final option on the page).

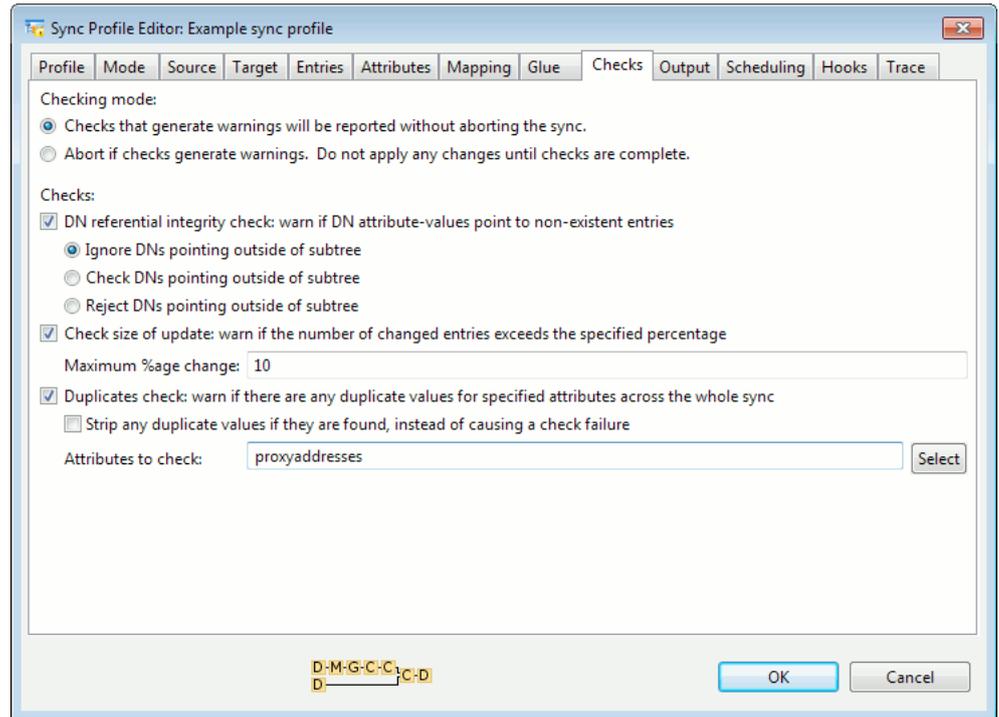
It may be necessary to add attributes to this base list if AD sends back additional pseudo-operational attributes.

- AD adds automatic objectclasses in some cases, for example adding **user** to **inetOrgPerson** entries. The sync would normally see this as a difference that needs correcting, so it is necessary to also filter out these special objectclasses to avoid problems. This is achieved by adding a specific rule to the **Attributes** page, for example: “Match on objectclass **person**; Allow all attributes; Delete these objectclasses: **user**.”
- AD uses an old pre-LDAP representation of O/R addresses, referred to as F.401. This may be converted using `to_syntax="ad_oraddr" ' and `from_syntax="ad_oraddr" in the mapping rules. AD insists that all DN values refer to an existing entry. This causes severe problems for loading up a set of entries that have DN references within the set. Any forward references cause the entry addition to be rejected. This could be solved with a two-pass ‘apply’ (writing entries on the first pass, and DN values on the second), but this has not yet been implemented. For now, DN references to entries existing outside of the synchronized set of entries are fine, but anything else may give problems.`
- Password synchronization to AD requires use of a special operation on AD that is not currently supported.

12.6 Checking syncs

Various checks may be configured as part of a sync. The checks may be used simply to give warnings, or may be used to stop the sync from applying changes until all the checks have passed. It is possible to run a sync simply for the checks and make it throw away its results, either by running a **Source Only** sync (set on the **Mode** page), or by selecting **Discard changes** on the **Output** page.

Most of the checks are configured on the **Checks** page:

Figure 12.3. Configuring checks

The following checks are available:

- **DN referential integrity:** checks that all DNs point to entries that actually exist within the set of data being synchronized. It is also possible to check DNs that point outside the subtree if required.
- **Check size of update:** If used with the “Abort if checks fail” setting, then all changes will be held until it is certain that the size of the update does not exceed the specified percentage. This can be used to protect downstream servers from accidental or catastrophic large-scale changes on upstream servers.
- **Duplicates check:** This checks for duplicate values across the whole of the data set being synchronized. This can be important for some systems, like Exchange, which do not tolerate duplicate values in certain attributes. Duplicate values can cause a check failure, or alternatively can be dropped from the sync with a warning.
- **Mapping rule based checks:** Checks may be configured in the mapping rulesets. These are typically regex-based checks on attribute values, or checks on the numbers of attribute values. However, using scripting it is possible to do much more complex forms of checking, for example entry-wide or even sync-wide checks. See [Section 12.11.1, “Configuring mapping rule-sets”](#) for more details.

12.7

Sync groups and replication workflow

Using Sync Groups, a number of syncs may be grouped together into a sequence. When the group of syncs runs, the syncs within it are executed in sequence until a ‘stop’ condition is reached. The group can be configured to stop on error, or failure, or not to stop at all.

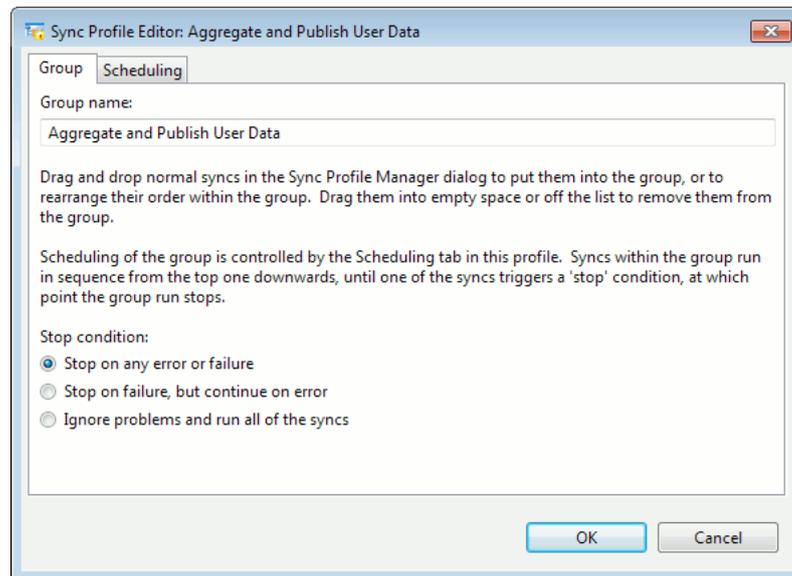
Sync Groups enable setting up a Replication Workflow. A typical replication workflow might consist of several syncs which copy in data from various different sources onto a staging server, followed by a number of checking syncs to verify that all the data on the staging server is sound (e.g. DN referential integrity, valid data values, no duplicates, and

so on). The final sync in the sequence copies the data from the staging server onto the downstream server and makes it 'live'. This final sync will only be run once all the previous syncs and checks have passed, so this ensures that the live downstream server always shows valid, fully checked data.

To create a group of syncs:

1. Click **New...** in the **Sync Profile Management** window.
2. Select **Group of Syncs** and click **OK**.

Figure 12.4. The Sync Profile Editor



The **Scheduling** page configures scheduling for the whole group. Individual members cannot be scheduled independently.

Syncs may be added to the group using drag and drop, or by clicking **New...** with the group profile selected in the **Sync Profile Management** window.

12.8 Correlated syncs

A correlated sync is designed for cases where there is no simple DN-based correspondence between the entries in two directories or databases. In general the DN of the corresponding entry on one database cannot be algorithmically derived from the DN on the other, and must be searched for by examining data within the records or entries on both sides.

Correlation in Sodium Sync is based on extracting key values from both source and target databases, and then comparing those key values and matching them up into pairs, forming a list of correlated pairs and a list of "rogues" (unmatched entries). Key value matches may be an exact match, which is preferable, but could also be an inexact match based on the smallest edit distance (a modified Levenshtein Distance).

Example 12.1. Example 1

An LDAP Directory contains user details and login names, organized by DN. An SQL Directory contains user addresses, contact details and social security numbers, indexed by payroll number. The correlation key could be formed from the user's name, normalized to

the form “<surname>, <firstname> <initials>” and lowercased. Where there are missing initials or typing errors, inexact matching would help match things up.

Key value generation is scripted, which means that the values can be normalized or flattened to improve matching between two databases maintained to different conventions.

Example 12.2. Example 2

The same department may be known by different codes in different parts of an organization, and in this case the values would have to be mapped to one single set of codes for the purpose of matching. Let’s say that we are correlating with a key formatted as “<department> <surname> <firstname>” and lowercased. Fred Bloggs works in Acme Technical Support. This department is referred to as “SUPP” on the payroll SQL database, but as “ATS” in the login LDAP database. We choose to convert the SQL code “SUPP” to the LDAP code “ATS” whilst forming the correlation key from the SQL database, so this means that the final correlation key value would be “ats bloggs fred” on both sides, allowing the records to be matched up successfully.

A correlated sync proceeds through a number of phases:

- The correlation profile is written and undergoes testing. Once the profile is working, no further changes to it should be required, except perhaps for tweaking the key generation to improve the numbers of exact matches in a complex correlation scenario. See [Section 12.11.4, “Correlation profile”](#) for details on creating a correlation profile.
- The correlation pass runs. This generates keys from the two databases, then indexes them and extracts exact matches, then inexact matches, then rogues. If there are correlations that have previously been approved, then these are taken into account, and they are checked to see that they are still valid.
- The administrator enters the approval GUI (by clicking **View Report**) and checks the correlations that have been generated. If correlations have previously been approved, then the administrator would only need to attend to the things that have changed. The correlation and approval phases might be run weekly to daily, depending on how quickly newly-added entries should propagate through the system.

Normally, exact correlations will be approved in bulk, inexact correlations will be checked by eye for errors before approval, and rogues may be set to be ignored or may be matched up by hand. Typically, the aim is for all matches to be exact matches. To achieve this may require adjustments to the key generation and normalization in the script code (first step), or changes to be submitted to the upstream database maintainers if the data held in them is incorrect.

- The correlated sync now runs, using the approved list of correlations as a basis. Attributes are synchronized between the correlated entries. This might be run daily, hourly, or perhaps more often, depending on how quickly data within entries should propagate through the system.

12.8.1 A simple worked example

We have an LDAP Directory containing information on toaster spare parts. We have a CSV file containing a dump from the warehouse database system containing information on the bin numbers for these parts. We wish to load the new bin numbers into the main LDAP spare part Directory. The CSV file and the LDAP Directory both contain the toaster part numbers, so we can use the part number as the key for correlation.

The warehouse system is indexed by inventory number. Here is small part of the CSV dump:

```
INUM,BIN,PART,DESC
000178432,CP-43-D,483-4732,PT-2000 baseplt hold clip
000178433,CK-82-A,744-2583,PT-2000 depr knob orange
```

Here are the equivalent entries in the LDAP Directory, cut-down:

```
dn: cn=483-4132,cn=PT-2000,ou=TOAST,ou=KWG,o=ACME
objectclass: part
objectclass: top
cn: 483-4132
description: Acme Premium Toaster baseplate holding clip
...

dn: cn=744-2583,cn=PT-2000,ou=TOAST,ou=KWG,o=ACME
objectclass: part
objectclass: top
cn: 744-2583
description: Acme Premium Toaster depression knob orange
...
```

From the CSV file a correlation key is generated for each record:

```
000178432: 483-4732
000178433: 744-2583
```

From the LDAP Directory, a correlation key is generated for each entry:

```
cn=483-4132,cn=PT-2000,ou=TOAST,ou=KWG,o=ACME: 483-4132
cn=744-2583,cn=PT-2000,ou=TOAST,ou=KWG,o=ACME: 744-2583
```

Now we run the correlation to try to match things up. The exact match pass runs through and finds the following correlation:

```
000178433 = cn=744-2583,cn=PT-2000,ou=TOAST,ou=KWG,o=ACME
```

It continues onto the inexact match pass which finds the following correlation:

```
000178432 = cn=483-4132,cn=PT-2000,ou=TOAST,ou=KWG,o=ACME
```

It appears that there was a data entry error with the part number on one of the systems, which is why one of them came up as an inexact match. We go to the correlation approval GUI, and check and approve both the exact and inexact correlations. This list of mappings is now saved.

When the correlation sync runs, it will go through the approved correlations in pairs, looking up the record in the CSV file using the inventory number (e.g. 000178432), and the entry in the LDAP Directory using the DN (e.g. **cn=483-4132,cn=PT-2000...**). The loaded data goes through normal sync mapping, filtering and difference-comparison to synchronize the bin number across from the CSV record into the LDAP Directory as intended.

Now that we have established a correlation between inventory number and LDAP DN for those spare parts, we don't need to correlate them again. Next time we get a CSV dump to synchronize, the correlation will only need to check to see if parts have been deleted on either side, and correlate any new parts that have been added since the last correlation.

12.8.2 Setting up a correlated sync

Once the correlation profile has been created (see [Section 12.11.4, "Correlation profile"](#)), a sync profile may be created, using the **Advanced View**.

1. On the **Mode** page, select **Correlated**.

2. On the **Correlation** page:
 - a. Select the correlation **Profile** that you have already defined.
 - b. In **Parameter**, find the file containing the correlations (or type the name of the file that will contain them)

The correlations are written in a text form, so .txt would be a suitable extension.

3. Still on the **Correlation** page, the next two fields enable you to fine-tune the processing of inexact correlations.
 - Setting a limit on the number of inexact entries to process lets you control runaway correlations.

The time to run the inexact correlations requires $O(N.M)$ time (i.e. twice as many on each side takes four times as long to process)

 - Inexact matches work on the basis of finding the pairs with the smallest edit distance between their keys. The Inexact maximum distance field lets you control the maximum difference (expressed as a count of character insertions and deletions) that should be considered for an inexact match.
4. **Source** and **Target** pages are completed as normal.
5. On the **Attributes** page:

Note: Since normally a correlated sync is used to merge a few attributes from the source into the target, this needs to be set up as a merge sync.

- Change the attribute selection filters to **Apply filters to source and target, after processing**.
 - Add a rule corresponding to the objectclass you want to sync. Change **Allow all attributes** to **Allow only these attributes**, and set it to pass through just the attributes that you wish to merge.
6. You may also need to filter out entries that you don't want to see, either using another rule on the **Attributes** page, or by using a filter on the **Entry** tab.
 7. On the **Output** page, select **Safeguard merge-sync**.
 8. Check the other pages for settings that may be required.

The sync should now be ready to run.

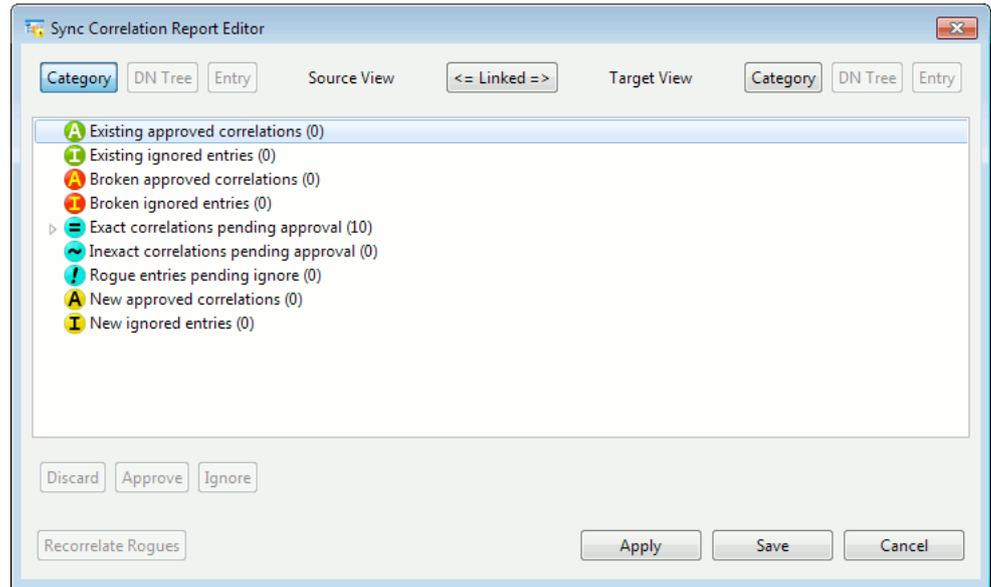
Note: On the **Scheduling** page you will see two sets of parameters because the correlation pass is scheduled independently from the sync pass.

9. Return to the **Sync Management Profile** window.

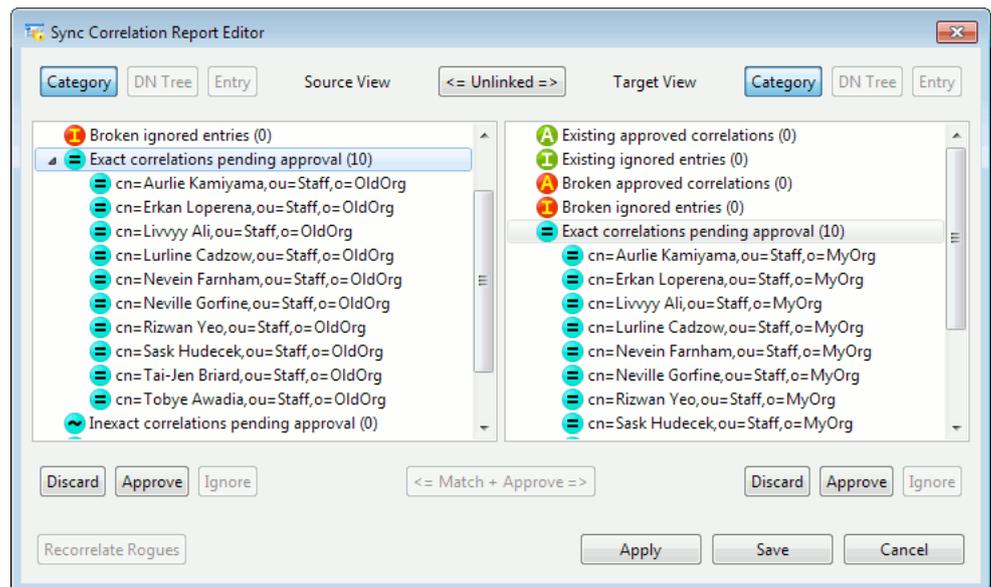
A **Correlate** button now appears alongside the **Run Sync** button. Click it to run the first correlation.

12.9 Approving correlations

After running a correlation pass, a green star is displayed alongside the profile name. Click **View Report** to enter the correlation report editor.

Figure 12.5. Correlation report editor: linked mode

This editor works in two modes: Linked (above) and Unlinked (below):

Figure 12.6. Correlation report editor: unlinked mode

The linked mode shows source and target DNs bound together, whereas the unlinked mode allows source and target DNs to be selected independently. The unlinked view has independent **Discard**, **Approve** and **Ignore** buttons for source and target, and a **Match + Approve** button which is used for manually matching up entries from the two sides.

In general, it is possible to do **Discard**, **Approve** and **Ignore** operations either on single items or on items in bulk. To operate in bulk, then a category or group item in the display should be selected instead of one of the individual items before clicking on the button for the required operation.

The normal workflow in this editor is to start from the top and work downwards:

- **Existing approved correlations** and **Existing ignored correlations** do not normally need to be reviewed, but if necessary you can navigate into these and **Discard** individual items, which sends the DNs into the **Rogues pending ignore** category.

- If there are items in **Broken approved correlations** or **Broken ignored entries**, then you need to see what the problem is. The choice is to **Discard** the correlation/ignore item, which sends the DNs into the **Rogues pending ignore** category, or **Approve** the item once again, which forces it to be included (although it will probably cause a failure on the next Sync run).
- **Exact correlations pending approval** can normally be approved in bulk without review. Select the category line, then press **Approve** and click **OK**.
- **Inexact correlations pending approval** will probably need reviewing. If you spot a bad correlation, you can send the DN to the **Rogues pending ignore** section by clicking **Discard**.

Alternatively you could switch to the **Unlinked** view and try and match it up manually with another DN. Once all the problems have been identified and eliminated, you can approve the rest in bulk by selecting the category line and clicking **Approve**.

- **Rogues pending ignore** will contain all the odd DNs found by the correlation pass, and also any other odd DNs that have been moved there by clicking **Discard** in the previous steps.

If you have moved a number of DNs here, it may be worth clicking **Recorrelate Rogues** to see if there are any new exact or inexact matches to be found amongst the rogues. This runs a mini correlation pass just for the rogue DNs. This is often helpful if entries have been renamed on one of the sources.

For the rogue DNs that remain, your options are: to try and match them up by hand using the **Unlinked** view and the **Match + Approve** button, or to mark them as entries to ignore by clicking the **Ignore** button.

- **New approved correlations** and **New ignored entries** will contain all the correlations that you have set to be approved and entries that you have set to be ignored in this session. You can check through them and **Discard** any that are incorrect, if you wish.

Once you have reviewed and approved the correlations, click **Apply** to make the new correlation list live.

To save your work to complete the process later, click **Save**. Re-enter the editor later using the **View Report** button. If you want to discard all your changes, click **Cancel**.

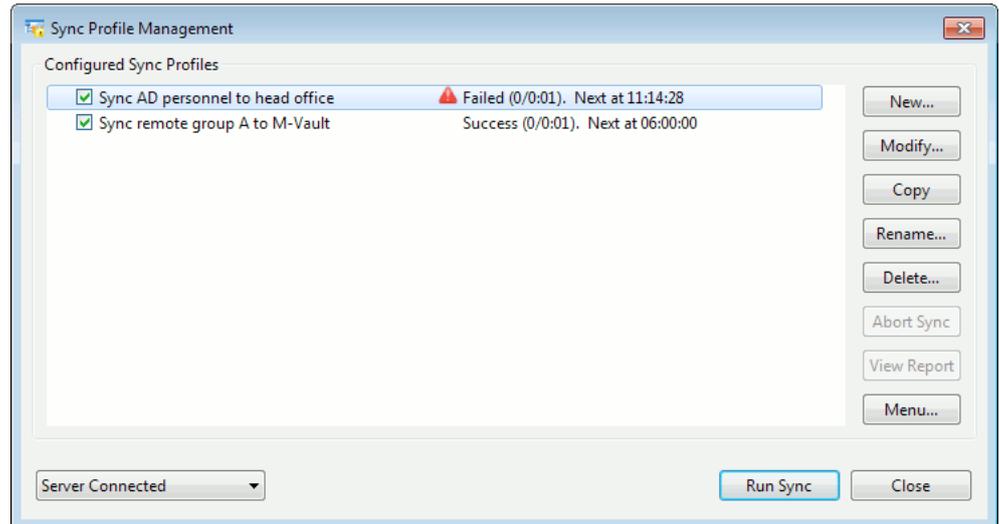
As soon as they have been applied, the changes will be picked up by the next correlated sync to run. A correlated sync is the same as a normal sync, except that it only works with the pairs of entries that have previously been approved. Mapping and attribute filtering all work just the same as for any other sync.

12.10

Configuring Sodium Sync Server

To run syncs automatically at regular scheduled intervals, you must start the Sync Server. This is a background process that runs continuously, even over system reboots, and which runs the scheduled syncs without needing to have the Sodium GUI application running.

When the Sync Server is not running, scheduled syncs are displayed in the **Sync Profile Management** window with an error triangle to warn that the server is not running. When the server is running, the status of the last sync and the time of the next sync are shown. If a sync is in progress, then a progress indication is also displayed.

Figure 12.7. Sync Profile Management window showing scheduled syncs

You may enable or disable the scheduling of a sync by selecting or clearing the checkboxes. A complete update for cached syncs may be forced using the **Force Complete Update** option (from the right-click menu or from the menu displayed when you click **Menu...**). The way that the Sync Server is managed depends on the operating system, but on both Windows and UNIX the server status control that appears in the bottom-left corner of the **Sync Profile Management** window is used to control it.

On Windows, the Sync Server currently runs as a Windows Service.

- “Home” editions of Windows 2000 and Windows XP are not supported as they do not allow Services to run as a user.
- There may be only one Sync Server running on a given machine, and the user who controls it and sets it up must be an administrative user.
- On Vista and Windows 7, you must start Sodium with Administrator privileges to successfully install and start the Sync Server service from Sodium. To do this, right-click on the shortcut to start Sodium and select **Run as administrator**.

Select **Install and Start Server...** from the server status control to install and start the service as the currently logged-in user. A box is displayed asking for the user’s password and the port number to use: the password may either be provided here or directly in the Windows Services control panel (**Control Panel** → **Administrative Tools** → **Services** → **Isode Sodium Sync** then **Properties** → **Log On**), leaving the password field blank in Sodium.

Note: Sodium does not need this password and does not store it permanently, but if it is provided, Sodium passes it on to Windows Service Management as a convenience to the user.

The first time the Sodium Sync service is installed, there may be a log-on failure when starting it because the user has not been given the `Logon as a Service` right. This may be solved by going to the **Services** control panel, and re-entering the password as described above. After that point, it should start without problem, either from Sodium or from the Windows Services control panel directly.

Once the Windows Service is installed, it may be started and stopped quickly using the server status control. To remove the Windows Service, select either **Stop and Uninstall server** or **Uninstall server**.

On UNIX, the Sync Server runs as a detached user background process, and is restarted using cron. Due to this, each user may have their own Sync Server running, although they would have to each select different port numbers in this case.

Start the Sync Server using **Start Server** from the server status control. Accepting the defaults installs a user crontab entry, and starts the server. Stop the server using the **Stop Server** selection. This removes the crontab entry to prevent the server from starting up again in a few minutes time. The installed crontab entry can be checked with `crontab -l`.

The Sync Server process runs detached from the terminal and the user may close Sodium and log out without affecting it.

12.11 Configuration files

There are two files which contain configuration of profiles used by the Sync. These are used for mapping rule-sets, correlation profiles, SQL profiles and CSV profiles.

They are searched for first in `(ETCDIR)/sodium` and then in `(SHAREDIR)/sodium`. If you modify a file found under `(SHAREDIR)/sodium`, save it back under `(ETCDIR)/sodium`, or otherwise your changes will be lost if you upgrade to a new version of Sodium. The two files are `mapping-rulesets.xml` and `config-profiles.xml`, documented below.

12.11.1 Configuring mapping rule-sets

The mapping rule-sets may be configured by someone proficient in Perl-compatible regular expressions and XML. The shipped mapping rule-sets are stored in `(SHAREDIR)/sodium/mapping-rulesets.xml`.

12.11.1.1 Mapping rule-set file syntax reference

The top level element is `<mapping>` which contains an optional `<script>` tag followed by a number of `<ruleset>` child elements.

The optional `<script>` tag is used if a JSR-223 scripting language (see <http://java.sun.com/javase/6/docs/technotes/guides/scripting/>) is to be used within the mapping file. Note that scripting requires Java 6 or later. This tag is used to define functions and initialise global variables for use in script fragments later on. All of the script code within one mapping file runs in the same global-variable context, independent from any other scripting in the application. The tag has an optional `lang` parameter which specifies the scripting language to use, defaulting to JavaScript (which is shipped with Java 6). The contents of the `<script>` tag is the scripting code to run. You may use the XML construct `<![CDATA[...]]>` to enclose the code, to avoid having to quote XML special characters. If the `<script>` tag is omitted, then the default scripting language (JavaScript) is used for any script fragments found later on.

The `<ruleset>` element has parameters `name` (a unique internal reference-name for the ruleset) and `label` (the visible name of the rule-set to show on the **Mapping** page). This contains a number of `<rule>` child elements.

The `<rule>` element has the parameter `keyclass` (optional, containing zero, one or more space-separated key objectclasses to test on). Out of all the rules, the one that best matches the target entry is used, with scoring rules the same as for the form templates. A rule with no keyclasses will act as the default if nothing else matches. The rule contains a number of action elements listed in the order that they should be executed. Action elements are:

<delete>, <copy>, <move>, <map>, <add>, <add_opt_oc>, <set_missing>, <build_postal_address>, <conform>, <normalize>, <nop>, <check>, <script>.

The action element parameters follow a pattern and are as follows:

- *attr* selects the destination attribute type.
- *value* gives a value to write if that is required.
- *from* selects one or more attribute types which are the source of the data, as a space-separated list.
- *match* includes a Java regex (similar to a Perl regex) which makes the operation conditional on a match of the value to that regex.
- *notmatch* includes a Java regex which makes the operation conditional on the value not matching that regex.
- *s_match* includes a script fragment that checks whether a value should be included in the operation. The global 'val' in the scripting context is set to the string value to check, and the return value of the script expression should be a boolean, with 'true' indicating that the value should be included.
- *subst* includes a Perl-like substitution command, used to make modifications to the values.
- *s_subst* includes a script fragment to convert a value. The global 'val' in the scripting context is set to the string value to convert, and the return value should be a string of the replacement value, or null to leave the same.
- *rdn* indicates that the value or values involved in this action should add to (rdn="add") or replace (rdn="set") the current list of RDNs for the entry.
- *check* may be used to disable syntax checks when it is known that a value is in a foreign syntax (i.e. incorrect according to the local schema) that needs to be handled unchecked. The default is *check="yes"* which does the checks; *check="no"* may be used to disable syntax checks.
- *from_syntax* and *to_syntax* may be used to convert to/from an external syntax when values are passed through the action. The only supported syntax currently is "*ad_oraddr*" for conversion to/from Active Directory format O/R addresses. If this is combined with substitution (*subst* or *s_subst*) then substitution is performed after the *from_syntax* conversion, and before the *to_syntax* conversion. If *to_syntax* is used, then the default for *check* changes to *check="no"*, although this may be overridden.

The regular expression handling is done with Java's regex package, which is very similar to Perl's regex handling. For the syntax of <pattern>, see the Java documentation for `java.util.regex.Pattern`: <http://java.sun.com/j2se/1.5.0/docs/api/java/util/regex/Pattern.html> [<http://java.sun.com/j2se/1.5.0/docs/api/java/util/regex/Pattern.html>]

For *match* and *notmatch*, case-insensitive matching can be turned on using `(?i)` in the regex, and other flags may be similarly controlled (see the Java regex documentation). Just as in Perl, `^` and/or `$` must be used if you want your match to be anchored to the start and/or end of the value, otherwise the pattern matches anywhere within the string.

The substitution command for *subst* takes the form:

```
"s/<pattern>/<replacement>/<flags>"
```

The <replacement> text uses \$1, \$2, etc. to insert matched sub-sequences from the <pattern>. The <flags> supported are zero or more of the set: *i* for case-insensitive, *s* for single-line (`Pattern.DOTALL`), *m* for multi-line (`Pattern.MULTILINE`), and *g* for global

(i.e. replace-all). Some character other than / may be used for the command separator, as in Perl, but Perl's brace-pair forms are not handled.

Note: In general it is permissible to create duplicate values for any given attribute using the actions below, and those duplicates will always be reduced to a single value.

The <delete> action has an *attr* parameter to select the attribute to delete. If a *match*, *notmatch* or *s_match* parameter is specified, only values matching/not-matching are deleted. For example, to delete all non-Acme email addresses:

```
<delete attr="mail" notmatch="acme\.com" />
```

The <copy> action has *attr* and *from* parameters. Values from all the *from* attributes are copied to the *attr* attribute. If “*match*”, “*notmatch*” or “*s_match*” is specified, then only values matching/not-matching are copied. If “*subst*” or “*s_subst*” is specified, then the values are modified with the given substitution whilst being copied. If “*rdn*” is specified, then the copied values will replace or add to the RDN. For example, to add all the other telephone numbers to those already present in one attribute (perhaps for the benefit of a limited client):

```
<copy attr="telephoneNumber" from="mobile homePhone" />
```

Or to copy the display name to the cn and make it the entry's RDN:

```
<copy attr="cn" from="displayname" rdn="set" />
```

The <move> action has *attr* and *from* parameters. Values from all the *from* attributes are moved to the *attr* attribute (i.e. they are deleted from the source attributes). If *match*, *notmatch* or *s_match* is specified, then only values matching/not-matching are moved. If *subst* or *s_subst* is specified, then the values are modified with the given substitution. If *rdn* is specified, then the moved values will replace or add to the RDN. For example, to transfer a web-address from one schema to another:

```
<move attr="labeledURI" from="wWWHomePage" />
```

The <map> action has *attr* and *subst* parameters. Values in *attr* are modified in place. If *match*, *notmatch* or *s_match* is specified, then only values matching/not-matching are modified. If *rdn* is specified, then the mapped values will replace or add to the RDN. For example, to internationalize a UK phone number:

```
<map attr="telephoneNumber" match="^ *0[1-9]"
  subst="s/^ *0/+44 /" />
```

The <add> action has *attr* and *value* parameters. A new attribute-value is added to the existing values of the given attribute. If *from* and *match/s_match* (or *notmatch*) are also supplied, then the action is only performed if one of the values in the *from* list matches (or does not match in the case of *notmatch*), which makes the <add> action conditional. If *rdn* is specified, then the new values will replace or add to the RDN. For example, to add a particular organizational unit conditional on a match on the E-mail address:

```
<add attr="ou" value="Sales" from="mail"
  match="sales\.acme\.com" />
```

The `<add_opt_oc>` action has a *value* parameter which specifies an objectclass name. If the named objectclass is required according to the local schema, then it is added to the entry along with any other objectclasses that it depends on. ‘Required’ means that there are attributes handled by the named objectclass which are not currently handled by any other objectclass.

The `<set_missing>` has *attr* and *value* parameters. If the attribute has no values, then the given value is added. For example, to set the preferred language for anyone who has not already set it:

```
<set_missing attr="preferredLanguage" value="English"/>
```

The `<build_postal_address>` action has *attr* and *from* parameters. The values of the *from* attributes are used in order to build an address to put in the postal-address syntax attribute *attr* (i.e. up to 6 lines of up to 30 characters, separated by “\$”). For example:

```
<build_postal_address attr="postalAddress"
  from="street postOfficeBox 1 st postalcode c"/>
```

The `<conform>` action imposes conformance of the entry to the loaded schema. The optional *strip* parameter if present should have a “true” or “false” value, and if true causes all unknown objectclasses and all attributes not belonging to the objectclasses in the entry to be stripped out of the entry. The optional *insert* parameter if present should have a “true” or “false” value, and if true causes “unset” values to be inserted (if possible) for all attributes required according to the schema for the objectclasses. This ensures that the entry is valid to be sent to M-Vault, assuming that the Directory Server is running with the same schema as Sodium. If, as a result of the strip operation, the distinguished value (for the RDN) is deleted, then the entire entry is considered unviable and is skipped. For example:

```
<conform strip="true" insert="false"/>
```

The `<normalize>` action normalizes attribute values to the default printable representation if there are different representations. By default all attributes in the entry are normalized by this action. The optional *attr* parameter if present limits the operation to just the specified attribute. This action converts some alternative syntax representations into the preferred syntax, e.g. for OR address, which means that strings that represent the same value but appear different in printable form do not cause the sync to generate unnecessary changes. For example:

```
<normalize attr="objectclass"/>
```

The `<nop>` action does nothing (no-operation), but can be used with the *rdn* parameter for its side-effects. With *rdn="set"* and no other parameters, it clears the RDN list. With *rdn="set"* or *rdn="add"*, an *attr* parameter, and optionally a *match*, *notmatch* or *s_match* parameter, values from that attribute are used to replace or add-to the existing RDN values. For example, to set the RDN to the single cn value which has a format of "name.name" in lowercase:

```
<nop rdn="set" attr="cn" match="^[a-z]+\.[a-z]+$"/>
```

(For example, if there were two CNs: **cn=Fred Smith** and **cn=fred.smith**, this action would match on only the second one and would make it the RDN of the entry; if more than one cn matches, an error would result.)

The `<check>` action checks that values match or do not match the given regexes or conditions. If a check fails, then this causes a check failure to be flagged, which may cause either an abort or a warning according to the top-level checking mode. The *attr* parameter

gives the attribute to check. The optional *match* or *s_match* parameter gives a test that the values must match to pass the check. The optional *notmatch* parameter gives a regex that the values must not match to pass the check. The optional *count* parameter checks the number of values present; the parameter is a list of comma-separated integers or ranges, for example "1-" or "0,1" or "0-1" or "1-5" etc. Remember to use ^ and \$ anchors in regexes if you want to check the whole value. For example, to check for valid phone numbers and for at least one certificate:

```
<check attr="telephonenumber" match="^[0-9 ]*$"/>
<check attr="usercertificate" count="1-"/>
```

The `<script>` tag encloses some scripting code which can do arbitrary checks and modifications on the data in the entry. The optional *checks* parameter should be set to "true" if the script may call `entry.fail()`, e.g. due to doing validity checks. The optional *rename* parameter should be set to "true" if the script does any modification of RDNs or DN. If either of these flags is unset when it should have been set, then attempting a `fail()` or `rename()` operation will result in a fatal abort of the sync.

Within the scripting context, the global variable `entry` is set to a `SiEntry` value which represents the entry to modify. See [Section 12.11.6, "Scripting interface to Directory entries"](#) for details of the `SiEntry` interface. Note that JavaScript allows providing interface callbacks with a simple inline function definition, for example:

```
<script checks="true"><![CDATA[
  entry.foreach("phone fax", function(attr, enc, val) {
    if (enc || !/^[0-9 ]*$/.test(val)) {
      entry.fail("Invalid " + attr + " value: " + val);
    }
  });
]]></script>
```

12.11.1.2 Annotated example of mapping rule-sets

This example *mapping-rulesets.xml* file illustrates a set of mapping rules designed to handle conversion of attributes from Active Directory to Isode M-Vault. Active Directory uses its own custom schema, and our task is to convert just those attributes that are of interest, and to leave the rest. This example considers only **person** entries. Similar rules could be set up for other types of entry.

The particular attributes we handle in this example are:

- The **proxyaddresses** attribute which is created by Exchange, which may contain E-mail addresses, O/R addresses and various other types, according to a prefix on the value. We must split these different types into different attributes, removing the prefix.
- The `wwwHomePage` attribute, which may contain a web-page link.
- Various 'other-' attributes which are used in Active Directory to store a second phone number, pager number, fax number, etc, which we add to the set of values for the base attribute.
- The **postalAddress** attribute, which has to be filled in from a number of other individual Active Directory attributes.
- The **street** attribute, which on Active Directory may contain newlines which can cause problems in other environments and must be translated. The **objectClass** attribute, which has to be manipulated to include the correct structural and auxiliary objectclasses for the attributes we wish to pass through.

Here is the file:

```
<mapping>
  <ruleset name="ad-mappings"
    label="Active Directory to M-Vault standard mappings">
```

Define a ruleset internally known as “ad-mappings” with the given label which will appear on the **Mapping** page in the **Sync Profile Editor** with that label for users to turn on or off.

```
<rule keyclass="person">
```

Specify a rule that applies when the **person** objectclass is present, so long as there is not a more specific objectclass match later on.

```
<add attr="objectclass" value="inetOrgPerson"/>
```

Always add the **inetOrgPerson** objectclass (and all superclasses), enabling use of various extra attributes that we may need. If there is already that objectclass present, no change is made (duplicates are always deleted).

```
<move attr="mail" from="proxyaddresses"
  match="( ?i)^smtp:" subst="s/^smtp://i"/>
<move attr="mhsORAddresses" from="proxyaddresses"
  match="( ?i)^x[45]00:" from_syntax="ad_oraddr"/>
```

Move addresses from an Active Directory-specific **proxyaddresses** attribute into standard **mail** and **mhsORAddresses** attributes. The operations use *match* to move only certain matching values. For example the first move only operates on values that start with *smtp:*, matched case-insensitively: *(?i)* means case-insensitive, *^* matches the start of the string, and *smtp:* matches itself. It is also necessary to remove the *smtp:* prefix as the values are transferred, so this is done with a Perl-style **subst** command: in this example *s/* means substitute, *^smtp:* matches the initial string, */* introduces the replacement string (an empty string), and */i* ends the command, specifying a case-insensitive match.

The second example is much the same, except that *x[45]00* specifies either the string “x400” or “x500”, and the syntax is converted from AD format to LDAP format using a built-in rule instead of a substitution. For example, “smtp:xxx@yyy.com” will match the first rule, and get stored as a mail attribute with value “xxx@yyy.com”; however, “x400:c=us;o=acme;cn=joe” will match the second rule and get stored as an **mhsORAddresses** attribute with a value of “/cn=joe/o=acme/c=us/”.

```
<add attr="objectclass" value="mhsUser"
  from="mhsORAddresses" match="."/>
```

This is an example of a conditional **<add>** operator. The **mhsUser** objectclass is added only if there is a value for the **mhsORAddresses** attribute present. The match is for “.” which means “any character”. **<add_opt_oc>** could also be used here.

```
<move from="wWWHomePage" attr="labeledURI"
  match=" ^http://"/>
<move from="wWWHomePage" attr="labeledURI"
  match=" ^https://"/>
<move from="wWWHomePage" attr="labeledURI"
  subst="s|^(.+) |http://$1|"/>
```

Here we move values from **wWWHomePage** to the standard **labeledURI** attribute. Ones which already have the “http:” or “https:” prefix are passed unchanged, and the others have “http:” prefixed: *s|* means substitute, using *|* as the delimiter for the command, *^* matches the start of the string, *(. +)* matches one or more characters and remembers them, *|*

introduces the replacement string, “http://” is inserted directly, \$1 inserts the string remembered previously, and | completes the command.

```
<move from="othertelephone" attr="telephonenumber" />
<move from="otherhomephone" attr="homephone" />
<move from="othermobile" attr="mobile" />
<move from="otherpager" attr="pager" />
<move from="otherfacsimiletelephonenumber"
  attr="facsimiletelephonenumber" />
```

Here values are moved from Active Directory **other-** attributes to similar standard base ones.

```
<build_postal_address attr="postalAddress"
  from="street postOfficeBox 1 st postalcode c" />
```

Here the special **postalAddress** attribute is filled from a set of other attributes.

```
<map attr="street" subst="s/ *\r?\n */, /gs" />
```

This handles newline characters in the street address. Active Directory permits newlines in this attribute, but this is not always well-supported elsewhere. This mapping operation replaces each newline and its surrounding white space with comma-space: *s/* means substitute, *** (a space followed by an asterisk) matches 0 or more spaces, *\r?* optionally matches a carriage-return, *\n* matches a linefeed, *** (a space followed by an asterisk) matches 0 or more spaces, */* introduces the replacement text, *,* (comma-space) is the replacement text, and */gs* terminates the command, giving flags *g* for global (i.e. replace-all) and *s* (treat as a single line, i.e. allow matching on newline characters).

```
<conform strip="true" insert="true" />
```

Now all substitutions and mapping have been done, strip out unknown objectclasses and any attributes not belonging to the remaining objectclasses, according to the local schema.

```
</rule>
<rule>
  <conform strip="true" insert="true" />
</rule>
```

Also create a fall-back rule to be used for non-person entries, which just strips the entry back to what is understood by the local schema. Note that if none of the objectclasses are known to the local schema, the attribute holding the DN value may be deleted, in which case the entry is dropped from the sync due to being unviable in the local schema.

```
</ruleset>
</mapping>
```

Close the ruleset and the top-level mapping element.

12.11.2 Using CSV files as input or output

There is no GUI for configuring CSV file formats, and this must be done by setting up an XML profile in the (*ETCDIR*)/*sodium/config-profiles.xml* file. There is an example file in (*SHAREDIR*)/*sodium*. Each mapping of CSV columns to attribute names must be given its own profile. A CSV profile must also include scripting to generate a DN on import, and optionally to convert values on export or import, if necessary.

```

<csvprofile label="Personal Info">
  <format header="true" charset="utf-8"/>
  <columns> cn, sn, givenName, telephoneNumber,
    homePhone, uid, mail </columns>
  <header_check> CN, SN, GIVEN, PHONE,
    HOMEPHONE, UID, MAIL </header_check>
  <script lang="JavaScript">
    <export name="mapin" call="mapin()"/>
    <export name="mapout" call="mapout()"/>
    <![CDATA[
      function mapin() {
        var cn_value = entry.getValue("cn");
        entry.setDNArr(["cn", cn_value]);
        entry.setOC("inetorgperson");
      }
      function mapout() {}
    ]]>
  </script>
</csvprofile>

```

The above example demonstrates the format of a simple CSV profile. This profile describes a CSV file that has a header line and is stored as UTF-8 characters on disk. The columns are associated with the attribute names as listed in `<columns>`. A sanity check is made on import that the CSV file has the expected header field names, as listed in `<header_check>`. The script tag sets up mapping which in this case fills in the DN and objectclasses after the attributes have been imported.

Here is an example CSV file for the above profile containing two entries (the lines have been wrapped to fit):

```

CN,SN,GIVEN,PHONE,HOMEPHONE,UID,MAIL
Fred Bloggs,Bloggs,Fred,01234 567 890,07987 654 321,
  f.bloggs,f.bloggs@acme.com
John Smith,Smith,John,01432 543 654,01531 642 753,
  j.smith,j.smith@acme.com

```

Here is the entry data that will be output from the script:

```

dn: cn=Fred Bloggs
objectclass: top
objectclass: person
objectclass: organizationalPerson
objectclass: inetOrgPerson
cn: Fred Bloggs
sn: Bloggs
givenName: Fred
telephoneNumber: 01234 567 890
homePhone: 07987 654 321
uid: f.bloggs
mail: f.bloggs@acme.com

dn: cn=John Smith
objectclass: top
objectclass: person
objectclass: organizationalPerson
objectclass: inetOrgPerson
cn: John Smith
sn: Smith
givenName: John
telephoneNumber: 01432 543 654
homePhone: 01531 642 753

```

```
uid: j.smith
mail: j.smith@acme.com
```

12.11.2.1 CSV profile syntax reference

The `<csvprofile>` tag has a *label* parameter which gives the name that will appear in the configuration GUI. It contains an optional `<format>` tag, a `<columns>` tag, an optional `<header_check>` tag, and a `<script>` tag.

The `<format>` tag has optional parameter *header* which specifies whether the CSV files have a header line. It is not possible in general to detect whether the first line of a CSV file is a header line or not, so this must be specified in the profile. The *header* parameter's value must be "true" or "false". The default is "false" if unspecified. The `<format>` tag has an optional parameter *charset* which specifies the character set used by the CSV file. Example values include "us-ascii", "utf-8", "iso-8859-1", "windows-1252" and "MacRoman". If unspecified, then the Java default character set is used, which will usually be one that is typical for the OS environment.

The `<columns>` tag contains a list of attribute types which correspond to the columns in the CSV file. The attributes type names should appear as a space- or comma-separated list. On reading, the columns are read into attributes with these names. On writing, values in these attributes will be written into the CSV columns. Multiple values give a warning, and missing values are written as an empty string. If a header line is enabled, then the attribute names are used in the header, unless `<header_check>` is also specified.

The `<header_check>` tag can be used to make sure that the headers on an incoming CSV file exactly match the specified list of names. This may be useful to make sure that the correct CSV file has been provided, and that nothing has changed in the upstream database. The names should appear as a space- or comma-separated list, and they are compared case-insensitively. If `<header_check>` is included, then these names are used on the header line when the CSV is written, instead of the attribute names.

The `<script>` tag (see [Section 12.11.5, "Script tag with exports"](#)) must export two calls: `mapin` and `mapout`:

- `mapin` is called when reading an entry from the CSV file. It should extract a DN from the values that have been loaded, and set that DN on the entry. It may also change the objectclasses and do any other required mapping or fixup of incoming values. The global variable `entry` is an `SiEntry` (see [Section 12.11.6, "Scripting interface to Directory entries"](#)) which initially contains the values read from the CSV row in the attributes specified in `<columns>`, plus `untypedobject` and `extensibleobject` objectclasses and a fallback DN. Note that the base-DN of a scan is the root DN, so the DNs of entries read from a CSV file should normally have the root as their parent, or be in a hierarchy starting at the root.
- `mapout` is called when writing an entry to the CSV file. There is no specific action required on writing, but the script may map or transform values ready for writing if necessary. The global variable `entry` is a `SiEntry` containing the data which will be output.

12.11.3 Using an SQL database as source or target

For maximum flexibility, the interface between Sodium Sync and SQL databases relies on glue code written in a scripting language. This allows the interface to be adapted to any database interface which has Java bindings. The example code works with JDBC (which gives access to all the major SQL databases), but the script could equally well interface with any other database classes.

The supplied demonstration code interfaces with SQLite over JDBC. SQLite is shipped with Sodium, so testing with this demo profile does not require any other packages to be installed. The glue code for other SQL databases is likely to be very similar to this code.

To interface with another JDBC-supported database, you will need the JDBC driver for that database as a JAR file, the JDBC driver class-name and the driver URL prefix. This JAR file should be installed somewhere where Java will find it. This may be the *lib/ext* folder within the JRE installation directory, or else the common Java extension folder which is *%SystemRoot%\Sun\Java\lib\ext* on Windows and */usr/java/packages/lib/ext* on Linux. The class-name of the driver should appear in the `jdbc` parameter of the `<script>` tag, and the driver URL prefix should then be used in the JDBC `getConnection()` call in the `connect()` function.

SQL profiles are stored in the *sodium/config-profiles.xml* file, which is found within (*ETCDIR*) or (*SHAREDIR*). See the example file provided under (*SHAREDIR*). Here is the outer form of a SQL profile:

```
<sqlprofile label="Example SQL profile">
  <script lang="JavaScript" jdbc="org.sqlite.JDBC">
    <export name="connect" call="connect()" />
    ...other exports...
    <![CDATA[
      ...script code...
    ]]>
  </script>
</sqlprofile>
```

For the `<script>` tag specification, see [Section 12.11.5, “Script tag with exports”](#). The *label* parameter on the `<sqlprofile>` tag gives the name which will be presented in the configuration GUI for the user to select.

Not all of the possible exports need to be implemented and exported. For example, if you do not want to support a “delete everything” operation, then do not export `remove_all`, and if a sync is run which tries to do that operation, it will simply report an error and fail the sync.

Note: To write the glue code, you must select some key or keys from the SQL database that can be translated to/from a DN. It should be possible to uniquely identify and access a SQL record using this DN as a key. DNs should be assigned based on the root DN as parent, i.e. with all the entries placed immediately under root, or in a hierarchy starting at root.

Entries are manipulated via a global variable `entry` which points to a `SiEntry` instance initialised for the call (see [Section 12.11.6, “Scripting interface to Directory entries”](#)). SQL fields should be mapped to entry attributes, and it is usually convenient to add objectclasses so that the entry can be processed like any other Directory entry. The glue code must take care of this mapping in both directions.

Here are the exports making up the SQL interface:

`connect`

This routine should set up the connection to the SQL database.

`connect_write`

If this routine is provided, then it will be used instead of ‘connect’ when a read-write connection is required to the SQL database. If it is not provided, then ‘connect’ will be used in both read-only and read-write cases. This allows a low-privileged or anonymous connection to be made for ‘connect’, using a higher-privileged connection only when necessary.

`close`

This routine should close the connection to the SQL database.

`search`

This routine should start a DN-ordered search of the whole dataset that this profile is designed to access.

next

Get the next DN from the search. The global variable 'entry' is a blank SiEntry whose DN should be set. The routine should return 'true' if the DN has been set, or 'false' if there are no more entries. The DN will be used as a key to fetch the record using 'read'.

read

Read a single entry using its DN. The global variable 'entry' is a SiEntry which contains the DN to read. The entry should be filled in with the data read from the SQL database. The routine should return 'true' if the read was successful, else 'false'.

remove

Delete a single record from the SQL database using its DN. The global variable 'entry' contains the DN of the record to delete. The routine should return 'true' if the delete completed successfully, else 'false'.

remove_all

Delete all the records in this dataset. This may be called if the SQL database is the target of a 'recreate' or 'cached' sync. The routine should return 'true' if the delete completed successfully, else 'false'.

add

Add a single record to the SQL database. The global variable 'entry' contains the entry to add. The routine should return 'true' if the record was added successfully, else 'false'.

update

Apply a modification to a record in the SQL database. The global variable `entry` contains the DN and the attributes to update. Attributes to change or update will have a value, and all other attributes will be missing. Attributes to delete will have an empty-string value. The driving code for this interface assumes that all attributes will be single-valued. Warnings are given if that is not the case. This routine should return 'true' if the update completed successfully, else 'false'.

As noted above, you may implement any subset of these calls:

- If you only need read access, then 'connect', 'close', 'search', 'next' and 'read' would be sufficient.
- 'remove_all' is only necessary for recreate/cached syncs.
- 'update' is only necessary when incremental changes will be made to a SQL database.

12.11.4 Correlation profile

Correlation profiles are stored in the `sodium/config-profiles.xml` file, which is searched for in (`ETCDIR`) and (`SHAREDIR`). Here is an example of a simple correlation profile:

```
<correlprofile label="File correlation" type="file">
  <script lang="JavaScript">
    <export name="get_source_key" call="getkey()"/>
    <export name="get_target_key" call="getkey()"/>
    <![CDATA[
      function getkey() {
        return entry.getValue("cn").toLowerCase();
      }
    ]]>
  </script>
</correlprofile>
```

This correlation is a file-based correlation, which means that the correlations are stored as lists in normal files. It uses the same function `getkey()` for both source and target key generation, which generates a key from the lower-cased CN.

The `<correlprofile>` tag has a `label` parameter which gives the name which is used in the configuration GUI, and an optional `type` parameter which specifies where the

correlations are stored. At the moment, the only valid value for *type* is *"file"*, although in a future version, it may be possible to store correlations in the Directory. The `<correlprofile>` tag contains a `<script>` tag which should provide two exports: *"get_source_key"* and *"get_target_key"*. See [Section 12.11.5, "Script tag with exports"](#) for details of the format of the script tag.

The two calls are used to generate a correlation key value from an entry. One is used for the 'source' Directory or database, and the other for the 'target'. Two different key-generation calls are provided because it may be necessary to do different normalization or mapping, or even to generate the key out of different attributes, depending on whether the entry comes from the source or target. However, in some cases the key generation will be the same, in which case the same routine can be put in the `call` expression, as was done in the example above.

Both of the exports pass in the entry via the `entry` global variable, as a `SiEntry` (see [Section 12.11.6, "Scripting interface to Directory entries"](#)), and should return the key as a string.

12.11.5 Script tag with exports

This tag acts as a module of JSR-223 scripting code (see <http://java.sun.com/javase/6/docs/technotes/guides/scripting/>), and allows exports from that scripting code to be defined. The exports are the only call interface between Sodium and the code within that module. All the code within this tag is loaded into its own independent global variable context. It is used in CSV, SQL and correlation profiles.

Here is a short example tag:

```
<script lang="JavaScript" jdbc="org.sqlite.JDBC">
  <export name="get_source_key" call="getKey()" />
  <export name="get_target_key" call="getKey()" />
  <![CDATA[
    function getKey() {
      return entry.getValue("cn").toLowerCase();
    }
  ]]>
</script>
```

This defines a JavaScript module which loads the SQLite JDBC driver, and which has two exports: *get_source_key* and *get_target_key*. In this case, the code that is run in both cases is the same: the function call `getKey()`. The function definition itself is defined in the CDATA block. In this case, it reads the global variable `entry`, and returns a value to the caller. Typically input parameters will be passed in global variables, and output results could be returned in global variables or function return values. Any exception thrown within the script will be passed back to the caller and reported as an error.

The `<script>` tag has optional parameter *lang* which specifies the scripting language (defaulting to JavaScript), and optional parameter *jdbc* which specifies the Java class to load to enable a particular JDBC driver. The `<script>` tag contains one or more `<export>` tags followed by the script code, which is usually enclosed in a CDATA tag.

The `<export>` tag has two parameters: *name* is the name of the exported symbol, and *call* is a scripting language expression which will be evaluated when that export is being called. This expression will normally be a call to a function defined in the main body of the `<script>` tag.

12.11.6 Scripting interface to Directory entries

The `SiEntry` class provides an interface between a JSR-223 scripting language (see <http://java.sun.com/javase/6/docs/technotes/guides/scripting/>) and the entry being handled. It is used in mapping rulesets, CSV profiles, SQL profiles and correlation profiles. It provides

facilities to read and modify the entry, convert values, and report failures and warnings. In the reference below, the specification of each call is given first, followed by examples written in JavaScript.

Note that when using the JavaScript implementation shipped with Java, you may come across unexpected problems when calling methods on strings. This is because there are two types of string in this implementation: Java String and JavaScript String. Strings passed from Sodium Sync into JavaScript may be Java strings which won't work with JavaScript methods (e.g. replace with a regex). The solution is to use the JavaScript expression `String(val)` to convert to a JavaScript string first.

12.11.6.1 Reading values

```
Object getValue(String name);

var value = entry.getValue("sn");
```

Get the value of a single-valued attribute. An exception is thrown if the value is missing or if there is more than one value, or if the value is BER and `;binary` was not specified as a suffix to the attribute name. The return value is normally a String, or otherwise it is a byte[] if there is no string value (e.g. JPEG) or if BER was requested with a `;binary` suffix on the attribute name.

```
Object getValue(String name, Object def);

var value = entry.getValue("phone", null);
str = "City is: " + entry.getValue("l", "(unspecified)");
```

Get the value of a single-valued attribute, or return the provided default value “def” if the attribute is missing. Apart from the special handling of missing values, this call works the same as the single-argument `getValue()` call.

```
Object[] getValues(String names);

var arr = entry.getValues("homephone mobile pager");
```

Get the values of a multi-valued attribute or list of attributes as an array. Throws an exception if a value to be returned is BER and `;binary` was not specified in the attribute name. The `names` argument is a space-separated list of attribute names to read. The objects in the returned array are normally Strings, but may include byte[] values if the attribute value does not have a string value, or if BER values were requested.

12.11.6.2 Modifying values

```
void add(String name, Object value);
void add(String name, Object... values);
void set(String name, Object value);
void set(String name, Object... values);
void remove(String name, Object value);
void remove(String name, Object... values);

entry.add("-objectclass", "user");
entry.set("mobile", "07890 567 890", "07890 123 456");
entry.remove("ou", "test");
```

Set, add or remove the given value or values to/from the given attribute. In the case of `set`, previous values are deleted, whereas for `add` new values are added to the old ones. Duplicate values are always eliminated. A prefix of “-” on the attribute name disables syntax checking of the values, which is done even for `remove`. A suffix of `;binary` on the attribute name

causes values to be treated as BER. Each value may be a String, or a byte[] for types that contain raw data (e.g. JPEG). If ;binary is specified, then the value must be a byte[].

```
void remove(String name);
```

Remove all values from the given attribute, which is the same as doing `set` with no values.

```
void setOC(String oc_list);
void addOC(String oc_list);
boolean isOCReqd(String oc);

entry.setOC("inetorgperson mboxUser");
if (entry.isOCReqd("mhsuser")) entry.addOC("mhsuser");
```

Add the given space-separated list of objectclasses to the entry, and also add all the objectclasses that they depend on according to the local schema. In the case of `set`, the existing objectclasses are cleared first. If the objectclass is unknown to the schema, then it is added anyway, but no dependent objectclasses will be added in that case. For example, **inetOrgPerson** would also add **organizationalPerson**, **person** and **top**.

The `isOCReqd()` call tests whether there are attributes present in the entry that require the given objectclass to be present, but which are not already handled by any of the current set of objectclasses.

12.11.6.3 DN and RDN manipulation

DNs can be handled either as strings (for a brief description of the format, see [Section C.2.10, “DN”](#)), or as an array of strings containing a DN unescaped and decomposed into parts. The advantage of using the decomposed form is that you do not have to worry about escaping mechanisms. For example, the DN of “cn=fred+sn=bloggs,o=at+t,c=us” would be represented as the array [“cn”, “fred”, “+”, “sn”, “bloggs”, “”, “”, “o”, “at+t”, “”, “”, “c”, “us”]. Note that the “at+t” value is unescaped in the array, but must be escaped in the string form.

```
string getDNStr();
void setDNStr(String dn);

entry.setDNStr("cn=test," + entry.getDNStr());
```

Get and set the entry’s DN using strings.

```
string[] getDNArr();
void setDNArr(String[] arr);

var arr = entry.getDNArr();
entry.setDNArr(
    ["cn", value, ",", "ou", my_ou, ",", "c", "us"]);
```

Get and set the entry’s DN using DNs in the array format, containing attribute names and values in pairs, separated by “,” and “+” strings.

```
void setRDN(String name, String value);
void addRDN(String name, String value);

entry.setRDN("cn", "fred");
entry.addRDN("sn", "bloggs");
```

Set the RDN, or add values to the existing RDN. These modify the RDN of the entry, but leave the parent DN unchanged.

12.11.6.4 Iterators

```
interface SiForeachAttrCB {
    void run(String attr);
}
void foreachAttr(SiForeachAttrCB cb);

entry.foreachAttr(function(attr) {
    java.lang.System.out.println(attr);
});
```

Iterate through all the attributes present in the entry, calling the given callback handler for each attribute name. The set of values are cached before the iteration, which means that it is possible to make changes to the entry from the callback without upsetting the iterator.

```
interface SiForeachCB {
    void run(String attr, String enc, Object val);
}
void foreach(SiForeachCB cb);
void foreach(String names, SiForeachCB cb);

entry.foreach("pager mobile", function(attr, enc, val) {
    entry.add("telephonenumber", val);
});
```

Iterate through all values of all attributes (first call) or all values of the named attributes (second call), calling the given callback handler for each value. The list of attribute names is a space-separated string. The set of values is cached before the iteration, which means that it is possible to modify the entry from the callback without upsetting the iteration. Callback argument `attr` is the attribute name, `enc` is the encoding of the value: `null` for `String`, `"data"` for `byte[]` (e.g. JPEG), or `"ber"` for `BER byte[]` (e.g. certificate), and `val` is the value: either `String` or `byte[]`.

```
interface SiMapper {
    Object run(String attr, String enc, Object val);
}
void map(SiMapper cb);
void map(String names, SiMapper cb);

entry.map("pager mobile", function(attr, enc, val) {
    // Make sure it is a JavaScript string
    val = String(val);
    // Discard non-07 numbers
    if (!/^07/.test(val)) return null;
    // Add international code
    return val.replace(/^0/, "+44 ");
}
```

Iterate through all the values of all the attributes in the entry (first call) or all the values of the named attributes (second call), calling the callback handler for each value, allowing it the opportunity to delete the value (return `null`), modify it (return a new value), or leave it unchanged (return the original value). The arguments to the callback routine are the same as for the `foreach()` method.

12.11.6.5 External conversions and conformance

```
boolean testSyntax(String name, Object value);
boolean testSyntax(String name, Object... values);

if (!entry.testSyntax("telephoneNumber", value))
    entry.fail("Invalid telephone number: " + value);
```

Test that one or more values have the correct syntax for the given attribute type. Does not make any change to the entry, and does not record checking errors against the entry in case the test fails. Returns ‘true’ if all values have valid syntax, and ‘false’ otherwise.

```
String convFrom(String syntax, String value);
String convTo(String syntax, String value);

entry.map("mhsoraddresses", function(attr,env,val) {
    return entry.convFrom("ad_oraddr", val);
});
```

Convert an attribute value to or from a known external syntax. The only external syntax supported is `ad_oraddr` for Active Directory O/R Addresses. The return value is ‘null’ if the conversion is not possible.

```
void normalize(String names);

entry.normalize("mhsoraddresses");
```

Normalize all the values in the given attributes (a space-separated list) to the preferred printable representation, if there is one.

```
void conform(boolean strip, boolean insert);

entry.conform(true, true);
```

Force the entry to conform to the local schema. If `strip` is true, then all unknown objectclasses and all attributes not belonging to an objectclass are stripped from the entry. If `insert` is true, then ‘unset’ values are inserted for all attributes which are missing but required by the objectclasses present.

```
void loadResultSet(ResultSet rs);
void loadResultSet(ResultSet rs, String... attr_names);

entry.loadResultSet(rs, ["cn", null, "sn", "title"]);
```

Load all the attribute values from the given JDBC ResultSet into the entry. The objectclasses are set to **untypedobject** and **extensibleobject**. All the columns in the result set are loaded up as strings using their SQL column names as unchecked attribute names (first form), or using the attribute names provided (second form). A SQL NULL value causes the corresponding attribute to be omitted. In the second form, a null attribute name indicates that the corresponding column should be skipped.

12.11.6.6 Reported warnings and failures

```
void warn(String msg);
void fail(String msg);

entry.warn("This is a warning");
```

```
if (!/^07/.test(entry.getValue("mobile")))
    entry.fail("Check failed: bad mobile number");
```

Report a warning (first call) or a check failure (second call). A check failure may cause a complete failure of the sync or may be treated as warning, depending on the overall checking mode set in the sync profile.

12.12 Fixing broken sync states

The basic sync operations are stateless, and may be interrupted and restarted without any difficulty. However, certain sync configurations do maintain their own state locally or modify external state, and so may be left in a condition that requires operator intervention in certain cases of unexpected error, of server shutdown or of user abort whilst a sync was in progress. Normally the Sodium Sync will attempt to correct the situations by ‘rolling back’ to a previous safe state, but there are cases where this is not possible, especially when changes are being applied directly to a DSA. In these cases, it is necessary that the operator examine the situation and the contents of the files left behind to decide what is the best course of action.

In each case below, the simplest option is described first, and the following options only need to be considered if the implications of the first option are unacceptable.

12.12.1 Cached sync

If a cached sync is interrupted and cannot recover, there may be a tree file with the extension *.ldif.NEW* still present in the cache folder. This is the partial new tree file created by the sync. There are three options to fix things up:

- Delete the *.ldif.NEW* file, which will cause any changes already generated and committed by the previous sync to be duplicated on the next one, probably causing very many warnings when these duplicate changes come to be applied.
- Force the cached sync to do a complete update on the next sync, for example by deleting all the *TREE*.ldif* files or by setting up a complete update in the GUI.
- Merge the *.ldif.NEW* with the remainder of the previous tree LDIF file (spliced in at the point in the DN order where the incomplete file left off) to create a new tree file with the *.ldif* extension. This avoids duplicating changes as far as possible, although there may be a few entries missing from the end of the incomplete tree file.

Explanation: The Cached Sync works by storing a ‘tree’ LDIF file of the complete contents of the target subtree immediately after each sync (assuming that all the changes generated by that sync have been applied correctly on the target.) Several previous tree files are kept, but only the last one is used to generate changes during the following sync. If the sync is interrupted, a partial tree file may be left behind. In this situation only a part of the source and target subtrees have been scanned to generate changes, and changes have not yet been generated for the part of the subtree not yet reached in the partial tree LDIF file. However, looking at the DN reached in the partial file, and searching the previous tree file for that same DN, the partial tree file can be reconstructed with the correct contents of the remainder of the target subtree which can be copied from the previous tree file. This is valid because this data is unchanged since the last sync as the scan had not reached that point.

12.12.2 Queue out

If a previous sync was interrupted, there may be a part-written change file with the extension *.ldif.NEW* in the queue directory. There are two options to fix things up:

- Delete the file and discard any changes that might be in it. Note that if this file contains changes for a remote DSA, doing so will cause those changes to be missing from the remote DSA until the next complete update is made.
- Rename it with a plain *.ldif* extension and update the serial number in the *output_next_serial.txt* file so that the changes will be processed by whatever reads the queue. Note that it may be necessary to edit the file and clean up the last entry in the LDIF which may be incomplete.

Explanation: The queue output works by creating a 'new' LDIF file to write to, renaming it to the correct extension only when it is complete. If the sync was interrupted, the last few changes written to the LDIF may be cut short due to output buffering, so this needs to be checked and corrected before the file can be sent on.

12.12.3 Queue in

If the previous sync was interrupted, there may be a part-processed input change-file with the extension *.ldif.PROCESSING* in the queue directory. There are three options to fix things up:

- Rename the file back to the plain *.ldif* extension. The file will be processed again from the start at the next sync. This might cause a large number of changes to be repeated, probably giving many warnings.
- Delete the file, discarding all the changes in the later part of the file that have not been processed. The *input_next_serial.txt* file will have to be updated with the next expected serial number.
- Look to see what changes have actually been applied to the final destination, and edit the file to remove those changes, before renaming the file back to a plain *.ldif* extension so that it will be processed again.

Explanation: The queue input works by processing the input LDIF from the beginning to the end. If the sync was interrupted and a file is left in a part-processed state, then only the first part of the file will have been processed, and the rest will be unprocessed. The choice is whether to repeat the first part, or discard the second part, or attempt to remove the initial part that has already been processed.

Chapter 13 Managing Certificate Authorities (using Sodium CA)

This chapter describes the Sodium CA application, and explains how to use it to help configure and manage a PKI (Public Key Infrastructure) for Isode products.

13.1 Introduction

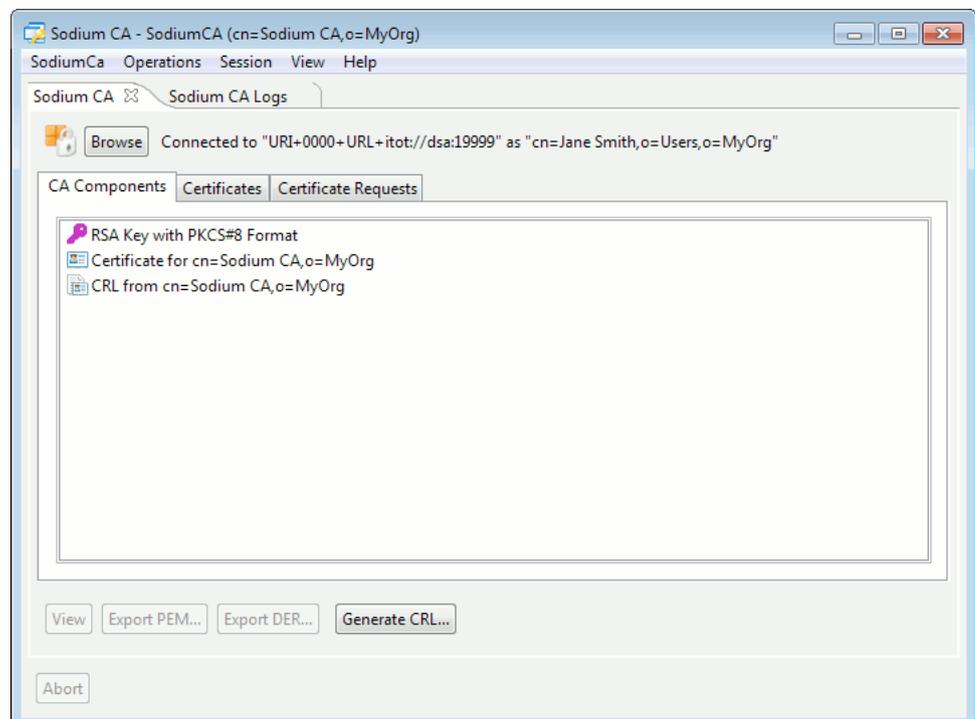
Sodium CA is a GUI utility that allows you to create and administer one or more Certificate Authorities (CAs). The primary purpose of Sodium CA is to make it easy to configure and manage X.509 PKI for Isode servers and clients without having to rely on a third-party CA. It is not necessary to use Sodium CA in order to be able to use X.509 functionality in Isode products, but in many cases it may prove to be the simplest or most cost-effective means of doing so.

Each CA that is managed using Sodium CA maintains a private database (CADB) which includes the following information:

- the CA's own certificate and private key
- a list of certificates issued by the CA and revoked certificates
- an up to date Certificate Revocation List (CRL).

This information is shown in Sodium CA's **CA Components** tab.

Figure 13.1. Sodium CA's CA Components



The CADB also contains a copy of all certificates that have been issued, and provides a simple interface that allows you to locate and examine any certificate (see [Figure 13.7, "Sodium CA's certificate view"](#)).

Sodium CA shares many capabilities with Sodium, and is designed to allow you to associate a CA with a specific Directory Server. Once this has been done, the CA can publish data directly into the Directory, where client applications typically expect to find it.

Having an associated Directory also means that Sodium CA is able to browse inside the Directory and issue certificates and identities corresponding to entries inside the Directory without requiring that users submit CSRs.

You can use Sodium CA to create an arbitrary number of separate CAs. For any CA that Sodium CA creates, you can:

- Generate a Certificate Signing Request (CSR) for the CA itself
- View information about Certificates, CSRs, etc., that the CA knows about
- Issue certificates in response to a CSR
- Revoke a previously issued certificate
- Renew or rekey previously issued certificates
- Generate and publish Certificate Revocation Lists (CRLs)
- Generate and publish cross-certificates for other CAs.

The following sections will describe these functions in more detail.

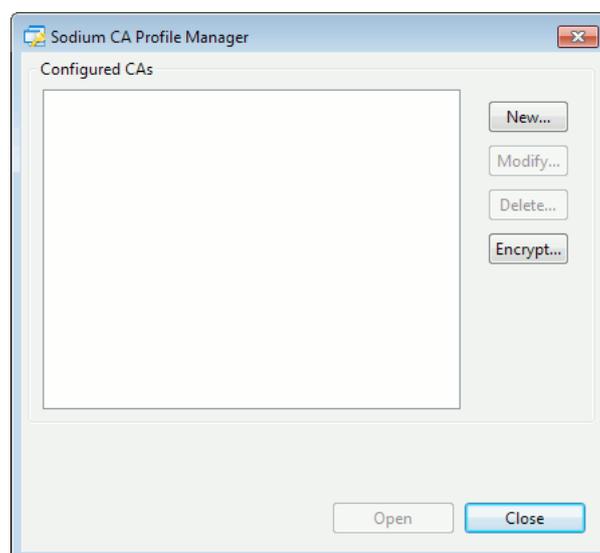
13.2 Creating a CA

Sodium CA allows you to configure and manage multiple CAs, each of which is described in a separate CA “profile” (rather like Sodium’s list of bind profiles).

When Sodium CA is started, it will display a dialog box listing the profiles corresponding to CAs that have previously been configured, or will prompt for the password to be used to decrypt the file containing CA profiles.

The first time you start Sodium CA, an empty list will be displayed, and the **Encrypt** button is enabled.

Figure 13.2. Empty list of CA profiles



Isode recommends that you encrypt the CA profiles file, which will mean that Sodium CA will be able to save sensitive information (such as passwords) in the file. You can encrypt the profile at any stage, but until you do, passwords will not be stored in it, and Sodium CA will prompt you for passwords and passphrases every time they are required.

To create an operational CA, Sodium CA will:

- create a CA database on the local file system
- generate a private key for the CA
- create a self-signed certificate for the CA, or generate a CSR to be signed by another CA
- (optionally) create an entry in a Directory where the CA is configured to publish PKI information.

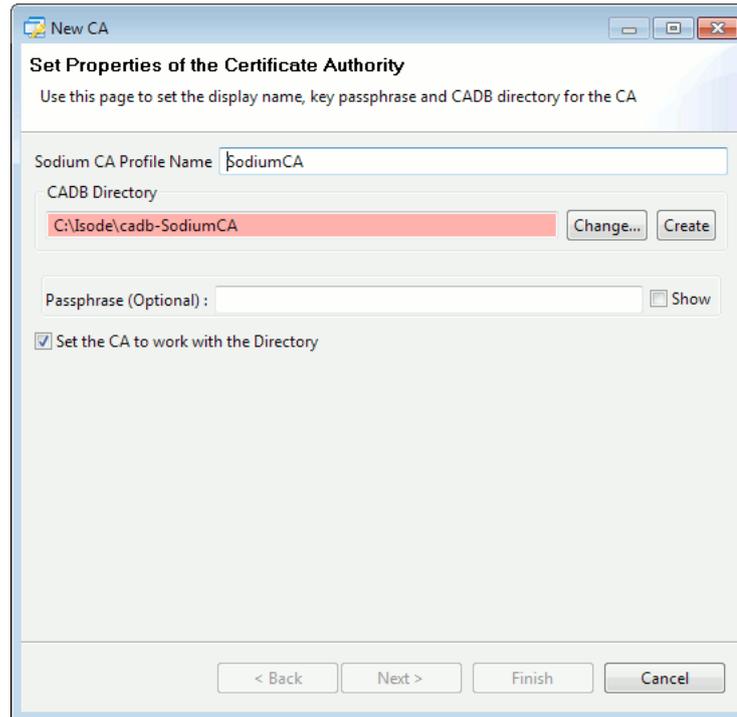
A CA has a distinguished name (DN) which will appear as the “issuer” field of any certificate which it issues. Isode recommends that any CA you create be associated with a Directory, in which case the DN that you choose for your CA will reflect the structure of the Directory being used.

For example, if your Directory is structured with a single top-level entry of **c=US**, then your CA’s DN might be **cn=Sodium CA, c=US**. The wizard used to create a new CA allows you to browse a Directory tree and select a suitable location for the CA’s entry. If you do not wish to associate the CA with a Directory, then you will be able to use any legal DN for the name of your CA.

So when creating a new CA, you need to choose:

- a “display name” to identify this CA in the list of those managed by Sodium CA
- where on the file system the CA’s database (CADB) should be located
- whether the CA is to be associated with a Directory
- a suitable DN for the CA (which may be depend on Directory structure)
- various options relating to the CA’s own private key and certificate content
- whether the CA is a root CA (i.e. it has a self-signed certificate).

To invoke the wizard to create a new CA, either use **SodiumCa > Create** menu option, or click on the **New** button in the **Profile Manager** window. A series of pages will prompt for the required information, and will provide suitable default values where appropriate.

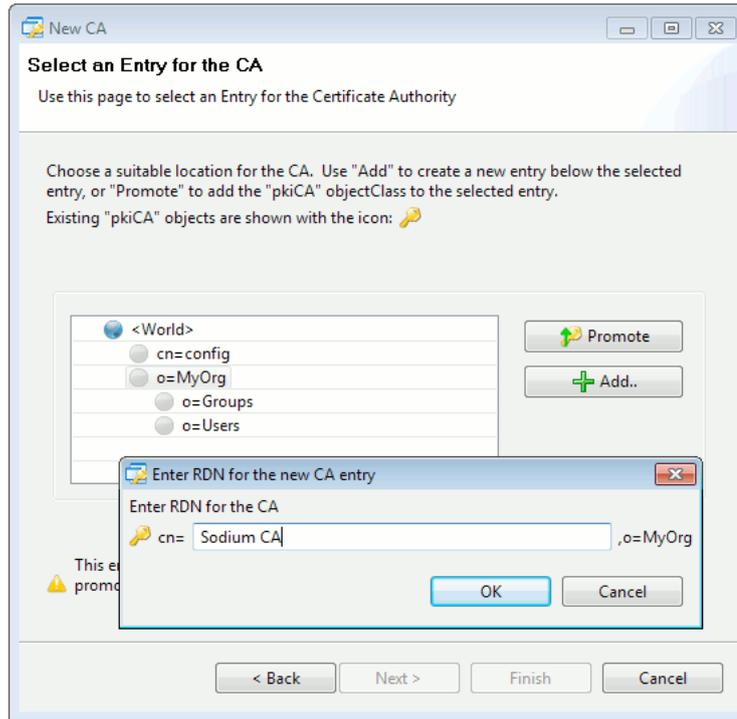
Figure 13.3. Creating a new CA

The CA's private key is used whenever the CA issues certificates or CRLs, and must be considered as a particularly sensitive piece of information. The CA's key is stored in the CADB, and so Isode recommends that the CADB be located on a system which is suitably secure (i.e. such that unauthorized users have no access to it).

For extra security, the CA's private key may be encrypted. A passphrase is used in this case to encrypt the CA's key, and will be required whenever the CA is operated. If you opt to encrypt the CA's private key, then the passphrase will be stored in the CA profile, provided the profile itself has been encrypted. In this way, you can use a single passphrase (the Sodium CA profile passphrase) to protect a set of separate CAs.

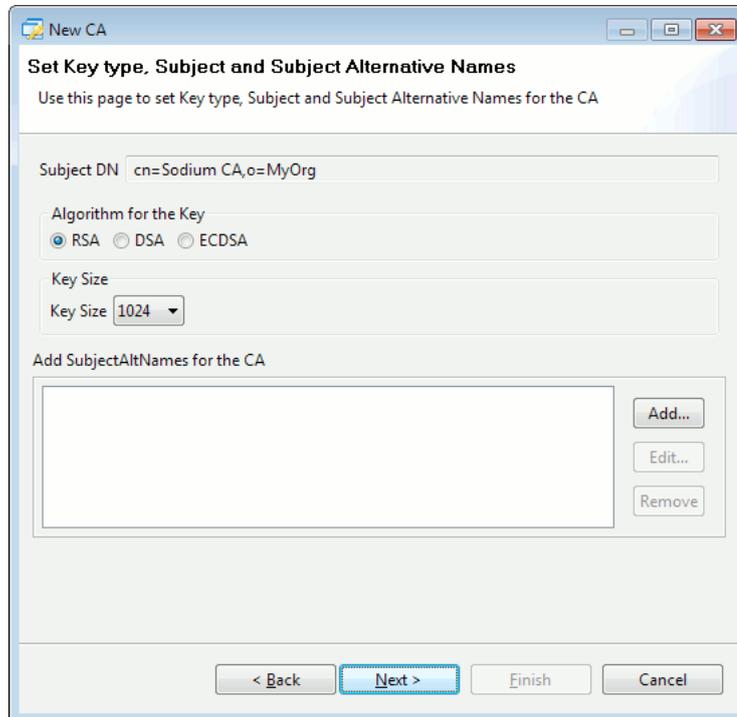
By default, Sodium CA assumes that any CA you create will be associated with a Directory. If you leave this option checked, then you will be prompted to enter suitable bind information for the Directory (again, any passphrase you specify will be saved in the Sodium CA profile file, provided it has been encrypted). Please ensure that the chosen bind DN has access to create and modify the contents of CA entry and is able to write certificates (attribute type **userCertificate**) to other entries. You will then be able to browse the Directory to find a suitable location for the CA itself. If an entry in the DSA already exists corresponding to your CA, you may choose to "promote" the entry into the CA's entry; alternatively you may create a new entry in the Directory beneath an existing entry.

Figure 13.4. Choosing to add a new entry for a CA

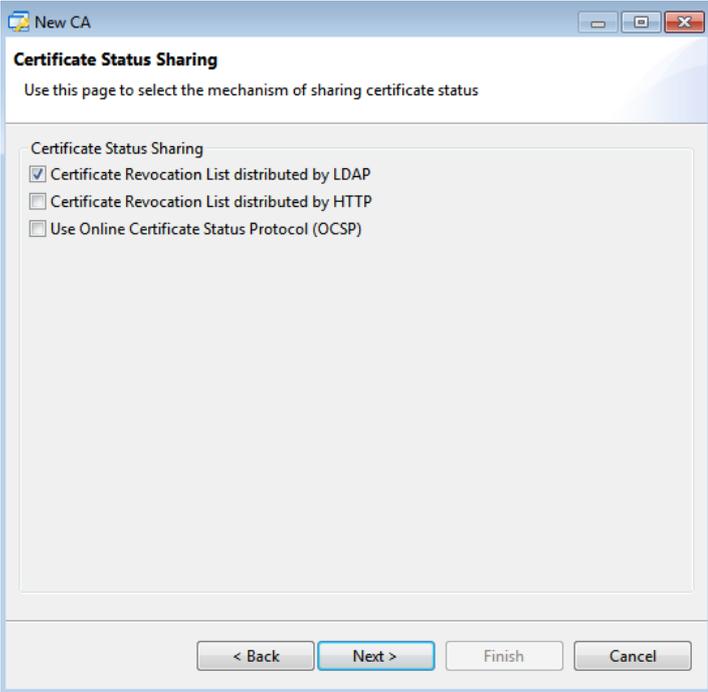


If the CA is not associated with a Directory, then you will be prompted to enter a DN of your own choice.

Various options relating to key generation may be specified, as well as specific certificate extensions that should appear in the CA’s own certificate. Refer to RFC 5280 for details on these values.

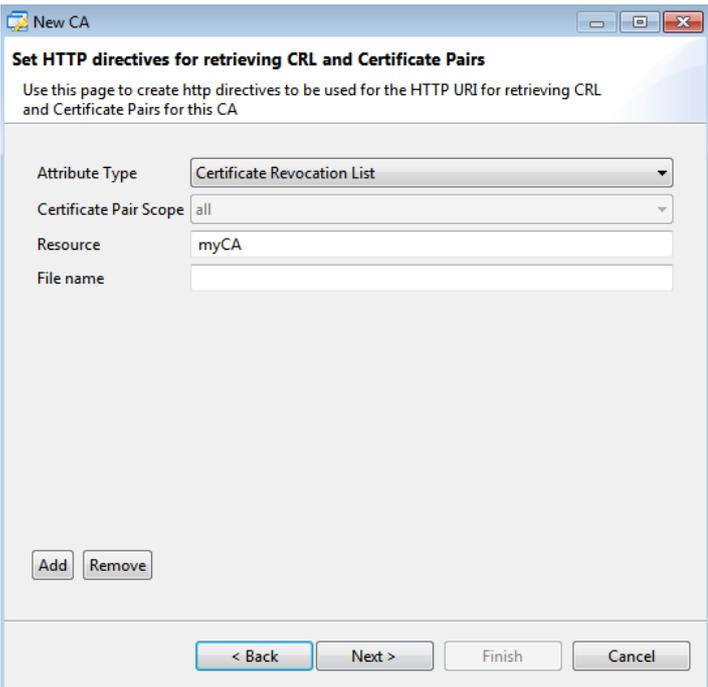


On clicking Next button, you can choose to select protocol(s) for sharing certificate status. The options provided are shown in the figure below. Selecting one or more options will have an impact on the CRL Distribution Point and Authority Key Identifier extensions that the CA will use in the extensions of the issued certificates (Refer RFC 5280).



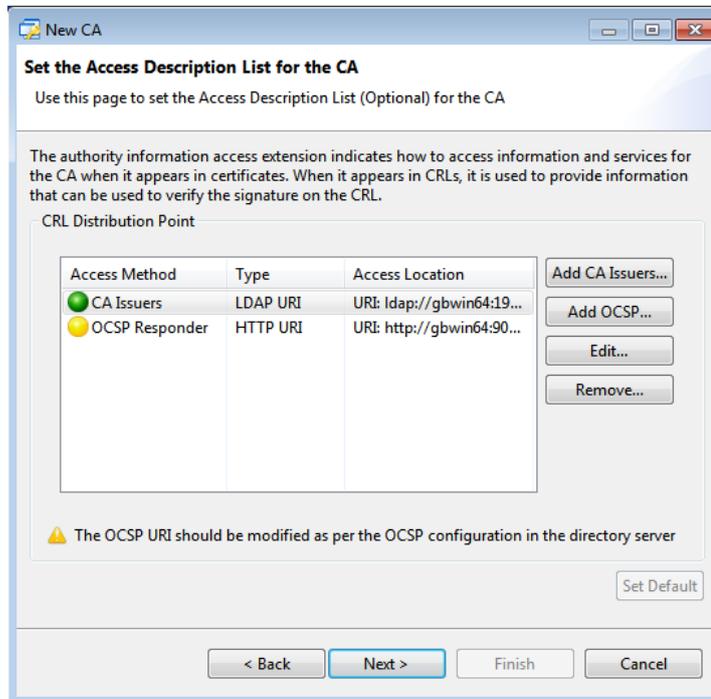
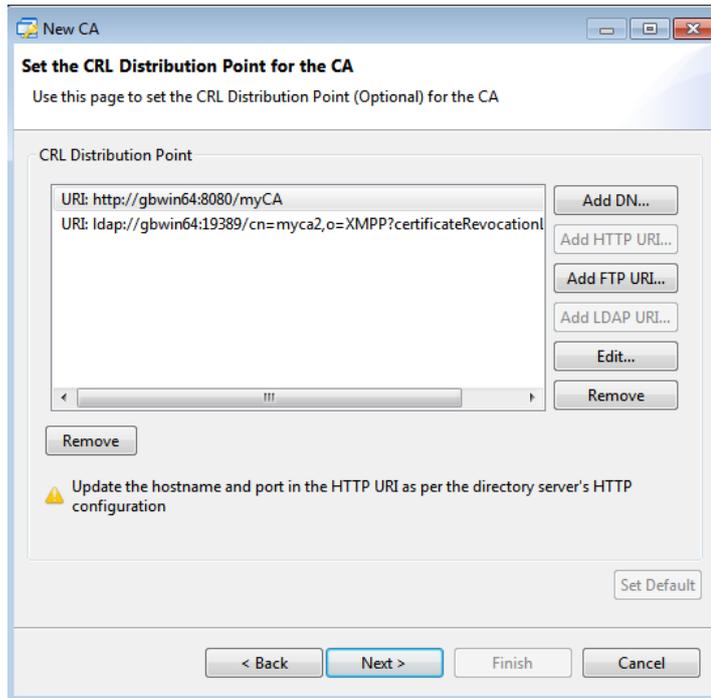
The screenshot shows a window titled "New CA" with a sub-header "Certificate Status Sharing". Below the sub-header is the instruction: "Use this page to select the mechanism of sharing certificate status". The main area contains a section titled "Certificate Status Sharing" with three checkboxes: "Certificate Revocation List distributed by LDAP" (checked), "Certificate Revocation List distributed by HTTP" (unchecked), and "Use Online Certificate Status Protocol (OCSP)" (unchecked). At the bottom of the window are four buttons: "< Back", "Next >", "Finish", and "Cancel".

If HTTP was chosen as one of the options to share CRLs, the next page will let you configure HTTP directives provided the CA was configured to work with an M-Vault Directory Server.

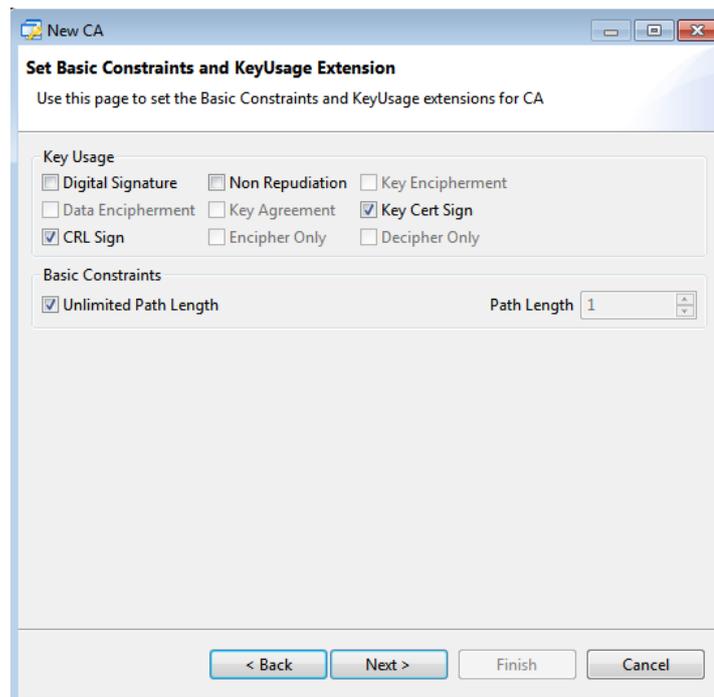


The screenshot shows a window titled "New CA" with a sub-header "Set HTTP directives for retrieving CRL and Certificate Pairs". Below the sub-header is the instruction: "Use this page to create http directives to be used for the HTTP URI for retrieving CRL and Certificate Pairs for this CA". The main area contains four fields: "Attribute Type" (dropdown menu with "Certificate Revocation List" selected), "Certificate Pair Scope" (dropdown menu with "all" selected), "Resource" (text input field with "myCA" entered), and "File name" (empty text input field). Below these fields are two buttons: "Add" and "Remove". At the bottom of the window are four buttons: "< Back", "Next >", "Finish", and "Cancel".

The next two pages will let you configure CRL Distribution point and Authority Information Access extensions. Default values will be offered based on the Directory Server details and options chosen for certificate status sharing.



On pressing Next, the wizard page lets you set the Key Usage and Basics Constraints extension (RFC 5280) for the CA.

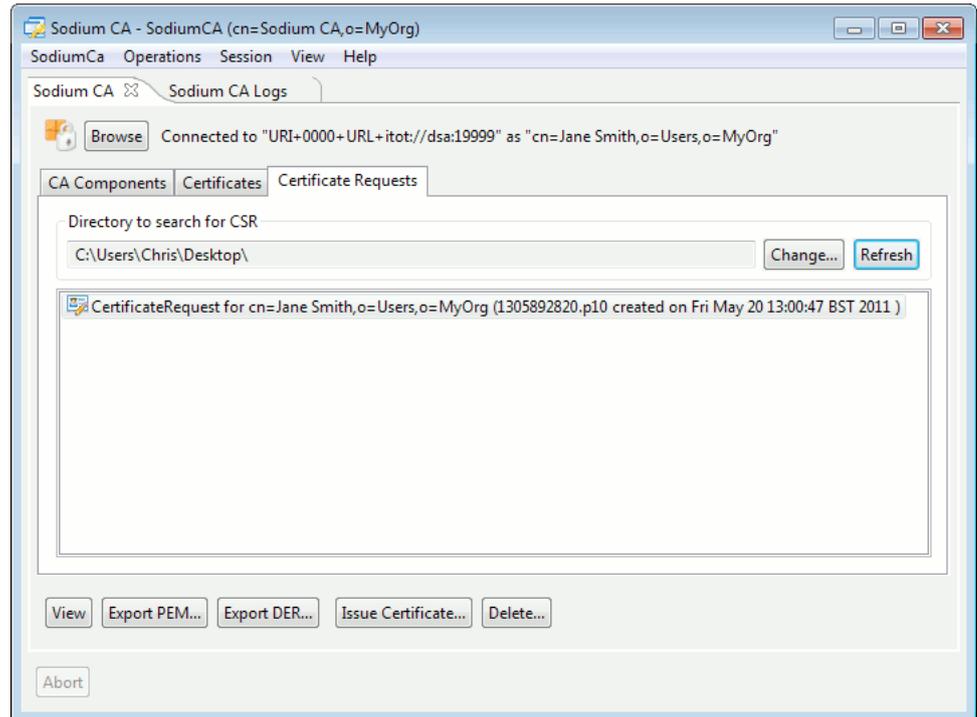


Once the extensions have been configured, the wizard will offer the choice of whether it is a root CA or a subordinate CA. If the CA is to be a root CA, then you should create a *self-signed* certificate. In this case, a CA certificate will be created and installed in the CA database. Additionally, if you have established an association with a Directory server, the Directory entry will be updated so that it contains the CA's certificate and (initially empty) CRL.

If the new CA is to be a subordinate CA, then you should generate a Certificate Signing Request (CSR) for the CA itself. The CSR will be stored in the CADB, where it can be viewed or exported (from the **CA Components** page). Once a corresponding certificate has been obtained from a suitable CA, you can "import" the certificate into the CADB, and the CA will then be operational.

13.3 Issuing certificates

In most cases, certificates are issued in response to a Certificate Signing Request (CSR). Use the **Certificate Requests** page to view pending CSRs. This window allows you to browse all CSRs in a given Directory.

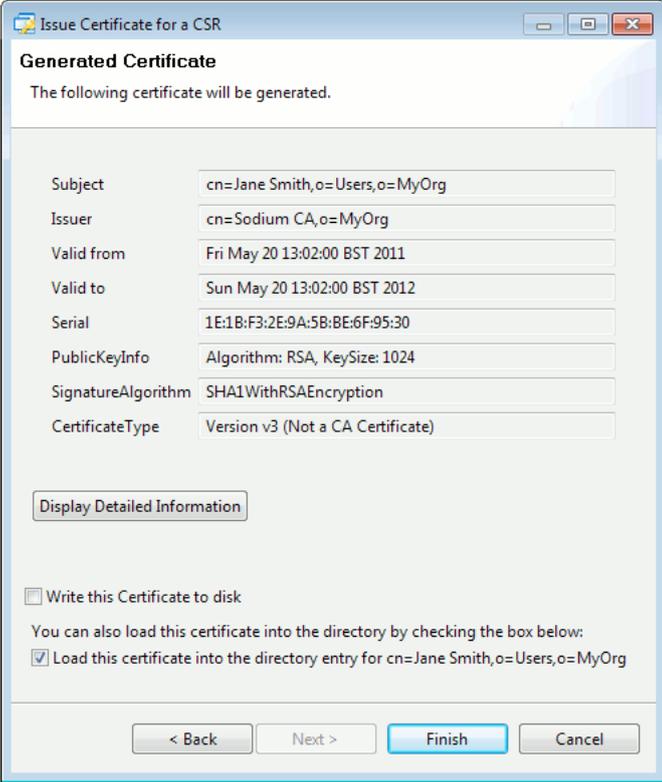
Figure 13.5. Displaying certificate signing requests

Sodium CA will display any files in the selected Directory which have an extension of either *.p10*, *.pem*, *.csr* or *.req* provided they contain a valid CSR. You can view, copy, or delete any CSR using this page. If appropriate, you can issue a certificate in response to a CSR, by clicking **Issue Certificate**.

A CSR is a *request* for a certificate, and contains information which the requestor would like to have appearing in the certificate. However, when issuing a certificate, you may wish to disallow or modify certain information supplied in the request. Specifically, Sodium CA allows you to override the following information which would otherwise be copied from the CSR into the certificate:

- the subject DN for the certificate
- any subjectAltNames
- supported extensions.

Once you have confirmed the information required, Sodium CA displays information about the certificate that will be issued.

Figure 13.6. Issuing a certificate

Issue Certificate for a CSR

Generated Certificate
The following certificate will be generated.

Subject	cn=Jane Smith,o=Users,o=MyOrg
Issuer	cn=Sodium CA,o=MyOrg
Valid from	Fri May 20 13:02:00 BST 2011
Valid to	Sun May 20 13:02:00 BST 2012
Serial	1E:1B:F3:2E:9A:5B:BE:6F:95:30
PublicKeyInfo	Algorithm: RSA, KeySize: 1024
SignatureAlgorithm	SHA1WithRSAEncryption
CertificateType	Version v3 (Not a CA Certificate)

Write this Certificate to disk

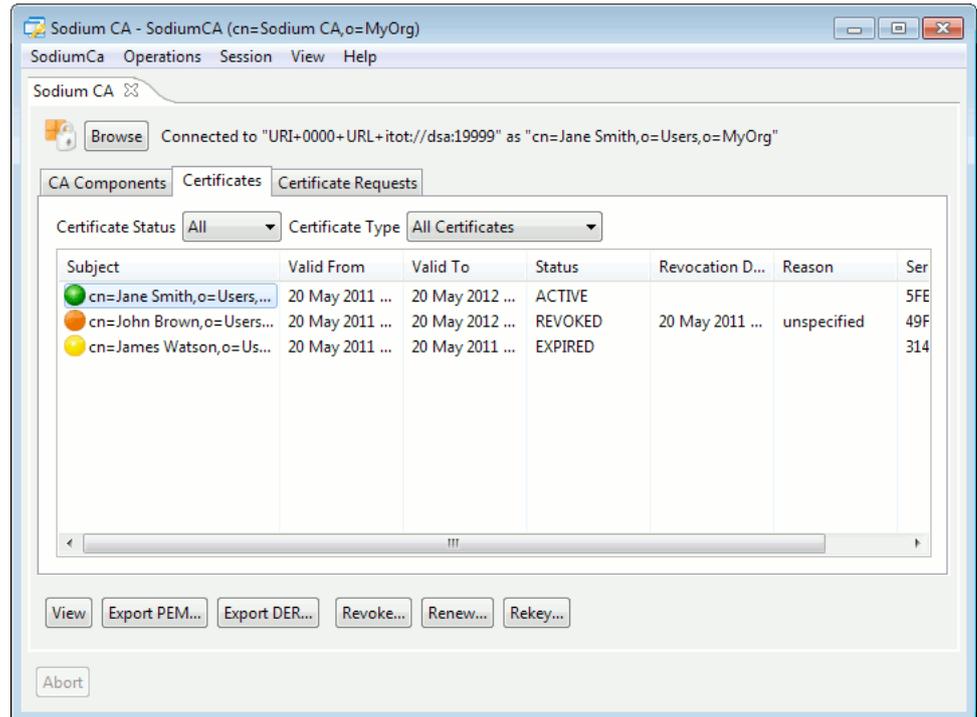
You can also load this certificate into the directory by checking the box below:

Load this certificate into the directory entry for cn=Jane Smith,o=Users,o=MyOrg

The certificate, once issued, will be added to the CADB and can optionally be written to an external file. Additionally, if the CA is associated with a Directory, and the subject DN of the certificate matches an entry in the Directory, you also have the option of updating the Directory entry so that it contains a copy of the certificate. Once a certificate has been issued, it is not possible to change any of the fields inside it. However, if you do realise that you have used the wrong values, then you may be able to use the **Renew Certificate** option (see below).

13.4 Managing certificates

The **Certificates** page in Sodium CA allows you to view all the certificates that have been issued by the CA.

Figure 13.7. Sodium CA's certificate view

- A summary line is shown for each certificate in the CADB, including a coloured icon to indicate whether the certificate is active (green), expired (yellow) or revoked (orange). The icon also contains a “+” sign for any certificate which is itself a CA certificate.
- You can customize the view to show only certain types of certificates (e.g. just the active ones), and you can examine a specific certificate in detail by using the **View** button.

13.5 Revoking a certificate

A certificate can be revoked using the **Revoke** option. The consequences of revoking a certificate are:

- the certificate is marked as revoked in the CADB
- a new CRL is generated, which will include the revoked certificate.

Revoking a certificate has no effect on users unless certificate validation makes use of the newly generated CRL. It is therefore important to publish the updated CRL as soon as it is created, to minimize the risk of a revoked certificate being used.

If the CA is associated with a Directory, then Sodium CA will automatically update the CA's Directory entry to contain the newly generated CRL. Otherwise, you should use the **CA Components** page to export a copy of the CRL and publish it in accordance with whatever local policy is appropriate

13.6 Renewing a certificate

A previously issued certificate may be “renewed”, which means that a new certificate is issued, based on information in an existing certificate. It may be useful to do this when, for example, a previously issued certificate has, or is about to, expire. Alternatively, you might do this if you made a mistake and forgot to include a certain **subjectAltName** in the original certificate.

The new certificate will contain the same public key as the original certificate. To issue a replacement certificate that uses a different key, use the “Rekey” option (see below).

The **Renew Certificate** option will invoke a set of pages similar to those displayed for **Issue Certificate** (in some ways it is equivalent to responding to the initial CSR again). However, since a “renewed” certificate is replacing an existing one, you have the option to revoke the old certificate. Should you choose to revoke the old certificate, then a CRL will be produced and should be published. (See [Section 13.5, “Revoking a certificate”](#)).

13.7 Rekeying a certificate

When a certificate is issued in response to a CSR (or “renewed”), the public key in the certificate will be taken from the CSR. In this case, the corresponding private key is known only to the originator of the CSR (and not to Sodium CA).

The **Rekey** option will generate a new public/private key pair, and use the new public key to issue a new certificate which is in all other respects (apart from validity dates) a copy of one that has previously been issued.

Rekeying a certificate may be appropriate in cases where, for example, a user has lost his original private key (or forgotten the passphrase which protects it): rather than requiring that the user create a new key pair and CSR which contains all the same options as the original, it may be easier to “rekey” his certificate.

Note: Since the rekey operation means that Sodium CA will itself have access to the user’s private key, it may not be an appropriate mechanism in all environments (depending on what security policy is defined). Additionally, once a certificate has been rekeyed, the user will not be able to use it until he has been told what the private key is.

When you invoke the option to “rekey” a certificate, a series of screens will take you through the process of choosing key parameters and validity dates for the new certificate, as well as providing you with the option of revoking the original certificate (in which case a CRL will be generated, and you should ensure that it is published).

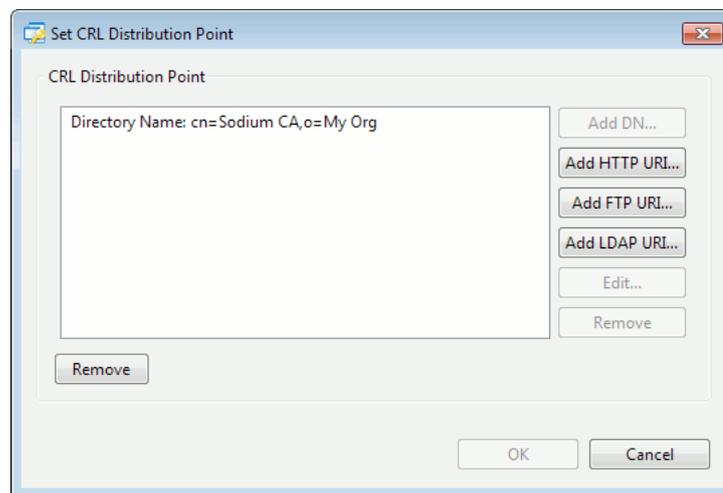
The final step in rekeying a certificate is the generation of a passphrase-protected PKCS#12 file containing the certificate and private key (as well as copy of the CA’s own certificate). The PKCS#12 file can be supplied to the user (e.g. via email), who will also require the passphrase in order to make use of it (the passphrase should be communicated by some mechanism other than the email message containing the PKCS#12 file).

Once in receipt of the PKCS#12 file and passphrase, the user will be able to make use of the new certificate. It is probably appropriate for the user to change the PKCS#12 passphrase, which can be done using Sodium (see [Section 3.10, “Managing identities”](#)).

13.8 Updating the CRL Distribution Point

The CRL Distribution Point appears in the Issuing Distribution Point and the CRL Distribution Point extension of the CRL (Certificate Revocation List) and the issued Certificates respectively. These extensions identify how CRL information is obtained. The CRL Distribution Point for the CA can be updated by selecting **Operations** → **CA** → **Update CRL Distribution Point** menu option.

Figure 13.8. Sodium CA’s CRL Distribution Point Editor



One or more distribution point names which describe a different mechanism to obtain the same CRL can be added to the distribution point.

The **Add DN...** button is used to specify the name of a directory entry that contains CRL information in either the **certificateRevocationList** or **authorityRevocationList** attribute. This value will be used only if the application has a locally configured directory server.

The **Add LDAP URI...** button should be used to point to the relevant attribute type (**certificateRevocationList** or **authorityRevocationList**) in a specified directory entry on a given LDAP host and port that holds the CRL.

The **Add HTTP URI...** and **Add FTP URI...** buttons can be used to specify HTTP and FTP URIs for fetching the CRL.

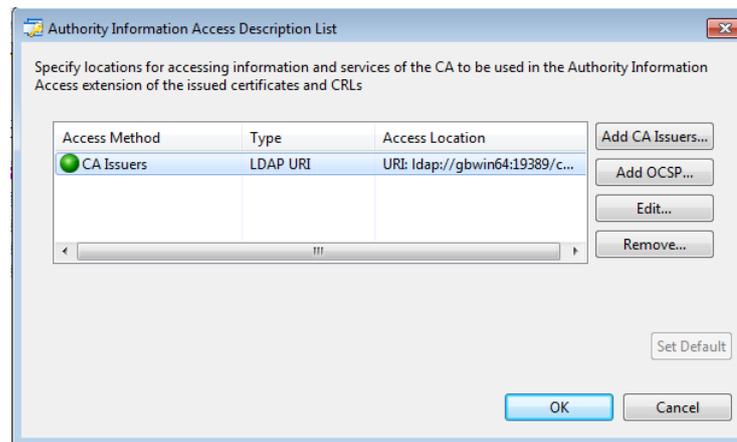
The distribution point names can be subsequently edited and removed using the **Edit...** and **Remove** buttons. The **Remove** button provided at the bottom should be used for removing the distribution point of the CA.

Once the distribution point has been updated, subsequently issued certificates and CRLs will contain the updated distribution point in the relevant extension. Note that the CRL Distribution Point and Issuing Distribution Point extension in the CRL and Certificates is optional and will only be added if the option to include the extension has been selected while generating them.

13.9 Updating the Access Description List

The Access Description List appears in the Authority Information Access extension of the CRL (Certificate Revocation List) and the issued Certificates. This extensions indicates how to access information and services for the issuer of the certificate or CRL in which the extension appears. The Access Description List for the CA can be updated by selecting **Operations** → **CA** → **Update Access Description List** menu option.

Figure 13.9. Sodium CA's Access Description List Editor



One or more locations to obtain the information and services of the CA can be added to the access description list.

The **Add CA Issuers...** button is used to specify means to access additional information that lists certificates that were issued to the CA to aid in certificate path generation.

The **Add OCSP...** button is to specify the location of the OCSP responder and is used when revocation information for the certificate containing the Authority Information Access extension is available using the Online Certificate Status Protocol (OCSP).

The authority information access description list contents can be subsequently edited and removed using the **Edit...** and **Remove** buttons.

Once the access description list has been updated, subsequently issued certificates and CRLs will contain the updated list in the relevant extension. Note that the Authority information Access extension in the CRL and Certificates is optional and will only be added if the option to include the extension has been selected while generating them.

13.10 Directory operations

Associating a CA with a Directory provides a number of benefits:

- When Sodium CA connects to the Directory, it checks PKI information inside the CA's own entry, and automatically updates any stale information (such as CRLs).

Additionally, Sodium CA will look for any new information in the Directory which may have been written by another CA. Specifically, it will attempt to complete any outstanding

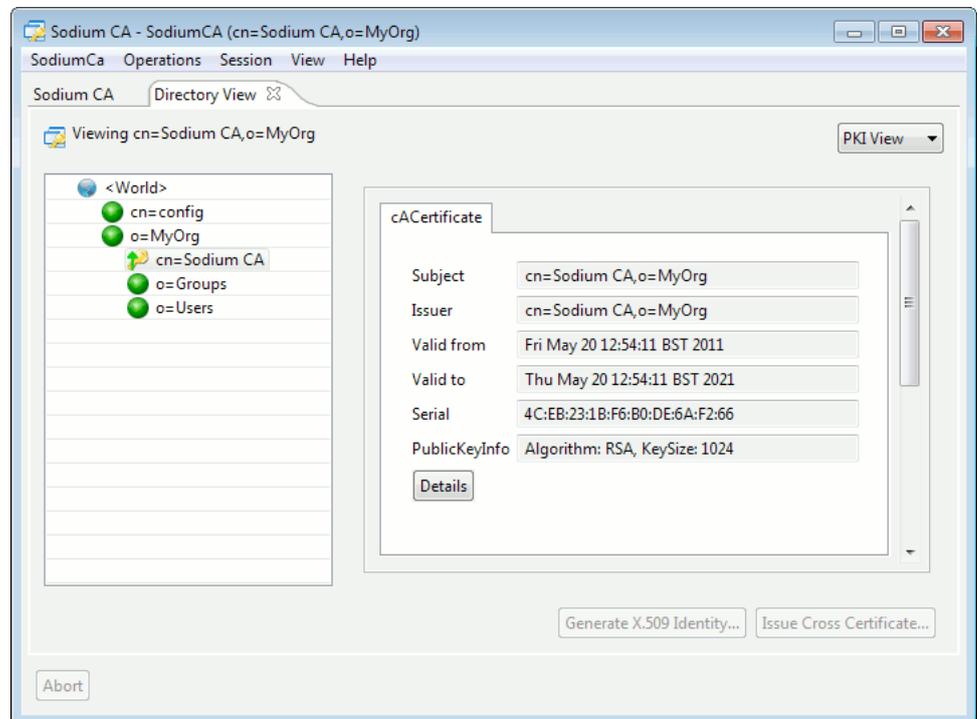
cross-certificate pairs (see below), and, for a CA which is a subordinate CA, Sodium CA will check to see whether the CA's own certificate has been published by another CA.

- When a new certificate is issued, then Sodium CA will look in the Directory for an entry matching the subject DN of the certificate. If there is a matching entry, Sodium CA will provide the option of updating the entry so that it contains the new certificate.
- Whenever a certificate is revoked, the new CRL is automatically published to the CA's entry.
- A "Directory browse" view is enabled, which gives a PKI-oriented view of information inside the Directory, and allows the creation of new X.509 identities and cross-certificates for existing entries inside the Directory.

13.10.1 Browsing the Directory

To browse the Directory with which a CA has been associated, use the **View** → **New Directory Browse View** menu option.

Figure 13.10. Browsing the Directory inside Sodium CA



Browsing a Directory using Sodium CA works in a similar way to Sodium, allowing you to navigate the Directory tree to locate and examine individual entries. However, the view differs from Sodium in that it displays only information that is likely to be useful for a CA administrator; specifically, when you select an entry in the tree, the entry viewer will display a single page containing PKI related attributes.

In the case of an entry corresponding to a CA, the attributes shown include:

- any user or CA certificates
- any Certificate Revocation Lists or Authority Revocation Lists
- any cross-certificates.

For a non CA entry, the viewer will show only certificate attributes.

13.10.2 Creating an X.509 identity

Sodium CA uses PKCS#12 files to store identities. An identity contains a certificate and a private key, and is protected by a passphrase.

Typically, it is not appropriate for anyone other than the certificate owner to have access to the private key, and this is the model followed when using Sodium to create an identity: when a user uses the **New X.509 Identity** option in Sodium (see [Section 3.10, “Managing identities”](#)), a private key and CSR are generated, but only the CSR is sent to the CA. Once the certificate has been issued, Sodium combines the certificate with the private key to build the PKCS#12 file which may then be used as an identity.

Sodium CA also provides the means to create an identity in a single step, obviating the need for a client to generate a CSR and “request” an identity. In this case, Sodium CA takes responsibility for generating a key pair, and then issues a certificate and builds a PKCS#12 file which can be given to the user.

An advantage of having Sodium CA be able to create identities is that it can simplify the process so far as the user is concerned — rather than having to “request an identity” and to specify various certificate options, he is just given a file and told “use this”.

A disadvantage of this mechanism is that the private key and PKCS#12 file do not remain in the exclusive possession of the user. When Sodium CA is used to generate an identity, then it would be possible for the CA administrator to keep hold of a copy of the identity and pretend to be the user by “borrowing” the identity. Such “borrowing” would not be possible when the private key is unavailable to the CA. The same issue applies in the case of “rekeying” a certificate (see above). However, depending on what security policy is in effect, you may decide that it makes sense to use Sodium CA to generate identities.

Sodium CA can generate an X.509 identity corresponding to any entry in the Directory (with the exception of CA entries). Use the **Generate X.509 Identity** button to initiate the process; a series of pages are displayed which allow you to specify the properties of the certificate to be created.

Once the identity has been created, the issued certificate will be added to the CADB, and the identity will be written to disk in a passphrase-protected PKCS#12 file. You also have the option of updating the user’s Directory entry so that it contains the newly issued certificate.

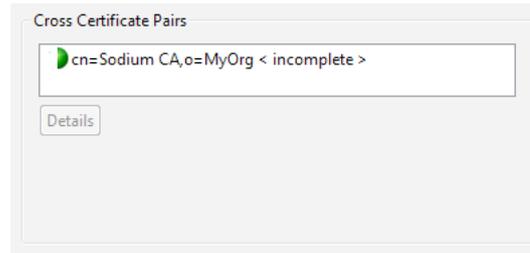
13.10.3 Issuing cross-certificates

In a PKI where more than one CA is being used, it may be convenient to indicate the level of trust between two CAs. A cross-certificate is a certificate that is issued by one CA to another.

Typically (but not necessarily), when CA1 trusts CA2, then CA2 will also trust CA1. In this situation, there will be two cross-certificates: one issued by CA1 to CA2, and one issued by CA2 to CA1. Because cross-certificates are often paired in this way, they are stored in the Directory inside “cross-certificate pair” values.

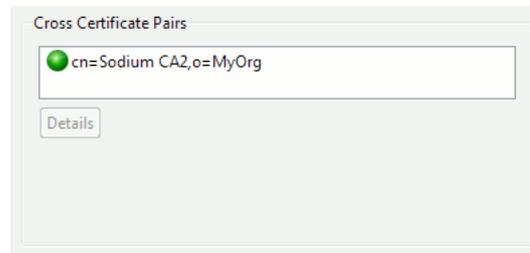
When browsing the Directory, if you select an entry which represents a CA, Sodium CA will enable the “Issue Cross Certificate” option if it is possible to issue a cross-certificate for the CA in question.

When CA1 issues a cross-certificate to CA2, the new certificate will be added to CA1’s database, and it will also be stored in CA1’s Directory entry as one half of a “cross-certificate pair” attribute.

Figure 13.11. A partially completed cross-certificate pair

The picture above shows what has happened after the CA has issued a cross-certificate for another "Sodium CA", but there was no corresponding cross-certificate that had been issued by "Sodium CA".

Whenever Sodium CA is used to manage the other CA, it will check in the Directory entry for that CA to see if a reciprocal certificate has been issued. If it finds one, then it will copy it and complete the cross-certificate pair.

Figure 13.12. Completed cross-certificate pair

Chapter 14 OAuth2 Capabilities

OAuth2 is a Web based framework that provides secure authorization services for Web based applications. The system is based on a set of HTTP and token exchanges between the Web service, Web browser and OAuth2 server. M-Vault includes an OAuth2 server component that provides authentication and authorization services to Isode Web-based applications, a current example being Red/Black.

Part of the OAuth2 configuration is intended to be managed by the Cobalt user and account provisioning application and the Cobalt manual should be consulted for this purpose. This chapter provides a lower level and more detailed view of the M-Vault OAuth2 service configuration, including instructions for how to view and manage the configuration using Sodium.

14.1 Overview

In order to authenticate a user to an Isode application (the OAuth2 client) the application first requests a session token from the OAuth2 service. At this point the OAuth2 service presents a login screen to the user (through their browser). Credentials are checked against the credentials stored in the user's entry. Once successfully authenticated the OAuth2 server returns session and refresh tokens to the Isode application (OAuth2 client) and redirects the user back to that service. Client tokens have a lifetime (duration) and must be refreshed periodically by the client service. If the session token expires and cannot be refreshed due to refresh token expiry then the user must re-authenticate. Note also that before the OAuth2 server will accept requests from an OAuth2 client trust has to be configured, where trust is established by way of a shared secret. Once a user has been authenticated the Web application can then also request what application specific permissions have been granted to the user.

Since version R19.0, it is also possible to leverage Single Sign-On mechanism instead of using a login form (Windows only, see [Section 14.5, "Single Sign-On \(SSO\)"](#)).

The OAuth2 service is part of M-Vault and is activated when it is configured and enabled. The configuration stored in and read from M-Vault includes:

- Server network configuration (listen address, TLS configuration).
- OAuth2 client configuration. An OAuth2 client is a Web service or application. The client configuration consists of the credentials that establish trust between the client and the OAuth2 server (a client identifier and a shared secret or TLS based authentication) and the address of the OAuth2 client (in the form of a URI). Other configuration includes per-client customization of Web pages (specifically a login page) that the OAuth2 server presents to users.
- OAuth2 client specific permissions. Clients can configure permissions relevant to the Web service they provide. These are configured in the directory.
- Per-user permissions. Web service permissions are in the user's entry in the directory.

In the following sections we detail the steps required to get the OAuth2 service up and running.

14.2 OAuth2 Server Configuration

This configuration object holds the operational parameters of the OAuth server.

14.2.1 OAuth Startup

On M-Vault's startup, the web API will start up if the OAuth Server configuration is present and **oauthEnable** is set to `true`.

14.2.2 Location in the Directory

The OAuth2 server configuration is stored in **cn=oauth, cn=config** and is managed by Sodium and MVC.

14.2.3 Configuring with Sodium

The creation of the server configuration can be done by using Sodium "Add below..." feature on **cn=config** entry and choosing "OAuth2 Config". Keep in mind that this subtree requires specific access rights.

The fields (with corresponding attribute types) shown in Sodium for the server configuration entry are:

- **Server Address (oauthServerAddress)** The address the OAuth2 service will listen on. Optional, default value: 0.0.0.0, i.e. all network interfaces.
- **Authorize Port (oauthServerPort)** Specifies the port the OAuth2 service will listen on for browser requests. Optional, default value: 19443.
- **Token Port (oauthTokenPort)** Specifies the port the OAuth2 service will listen on for OAuth2 Client requests. Optional, default value: 19543.
- **Token Duration (oauthTokenDuration)** Specifies how long the access token are valid for, in seconds. Optional, default value: 3600.
- **Refresh Token Duration (oauthRefreshTokenDuration)** Specifies how long the refresh token is valid for, in seconds. Optional, default value: 7200.
- **Use TLS Strong Client Authentication (oauthUseClientStrongAuth)** Specifies if the OAuth2 Server uses TLS to authenticate OAuth Clients. Trust anchors must be available in M-Vault for this to work. If operational, the OAuth2 Client's **oauthClientSecret** attribute is ignored. Optional, default value: `false`.
- **Enable OAuth2 (oauthEnable)** Turn on or off OAuth2 feature. Note that it will dynamically turn on/off the OAuth server without the need to restart M-Vault. Optional, default value: `true`.

14.2.4 Example LDIF

An alternative way to create the OAuth2 server configuration directory entry is to LDIF load a configuration. This is advanced usage and you should not use this if you're not sure of what you are doing. Again, Sodium is the preferred method to configure the OAuth2 service.

Example LDIF:

```
version: 1
```

```
dn: cn=oauth,cn=config
objectClass: oauthConfig
cn: oauth
oauthServerAddress: 127.0.0.1
oauthServerPort: 19443
oauthTokenPort: 19543
oauthTokenDuration: 3600
oauthRefreshDuration: 7200
oauthEnable: true
```

14.2.5 TLS configuration

TLS is mandatory in order to protect the issued tokens. The OAuth service won't start otherwise. You need a suitable TLS identity configured in the DSA (see [Section 3.10.1](#), "Generating a certificate request").

14.3 OAuth2 Service Configuration

The OAuth2 service configuration holds aspects that are domain dependant.

In general OAuth2 service configuration should be performed using the Isode Cobalt product, and configuration of the OAuth2 service is described in the Cobalt manual. The following sections go through the configuration schema in detail, and show how it can be view and modified directly using Sodium. Again, please note that Cobalt is the preferred method of configuring the OAuth2 service.

14.3.1 Configuring With Sodium

The fields (with corresponding attribute types) shown in Sodium for the service configuration entry are:

- **User Base (oauthUserBase)** The base of the subtree where in the directory the OAuth service should look for user entries being authenticated. The user must have a **userPassword** attribute, and an attribute with a unique value accross users to identify it. Optional, default value: **cn=Users, c=xx**.
- **User Attribute (oauthUserAttribute)** This field specifies which attribute to use as an identifier for the user. It is optional and its default value is **mail**.

Such service configuration can be created by right clicking on the domain entry, selecting "Add below..." and choosing "OAuth2 Service".

14.4 OAuth2 Client Registration

The client is an Isode application that uses the OAuth2 service as its means of authentication and authorization. In order to permit clients to access the service they must be configured (or pre-registered) in the the directory. The per-client configuration consists of a single entry in the directory.

This registration is usually taken care of by Cobalt.

14.4.1 OAuth2 Client Object

Available configuration fields (as shown in Sodium):

- **Client ID (`oauthClientID`)** A unique identifier for the OAuth Client.
- **Client Secret (`oauthClientSecret`)** The password sent by the OAuth2 client to authenticate with the OAuth2 server. In the case of strong client authentication using TLS, this value is ignored.
- **Redirect URI (`oauthRedirectURI`)** The URL the client listens as part of the OAuth2 flow. This attribute must be a hostname.
- **Server Type (`oauthServerType`)** The type of product this OAuth Client is. This is a free form string used by Cobalt to associate a particular application type with an appropriate set of configuration options (particularly with respect to the permissions provided by that application).
- **Service Domain (`oauthServiceDomain`)** The domain where to find the OAuth2 Service Configuration (see [Section 14.3, “OAuth2 Service Configuration”](#)) to use.
- **Token Duration (`oauthTokenDuration`)** The lifetime of the access token. This overrides the value set in the OAuth2 server configuration entry, so each OAuth2 client instances can be configured with specific session lengths. The OAuth2 client will maintains sessions for as long as the access token is valid.
- **Refresh Token Duration (`oauthRefreshTokenDuration`)** A refresh token is used by the client to refresh an expired access token. This value controls the lifetime of refresh tokens supplied to this client. This overrides the global value set in the server configuration entry so that each client can be configured with its own session length. The OAuth Client uses this to get a new access token without having the user to enter its credentials again.
- **Use SSO (`oauthUseSSO`)** Windows only (see [Section 14.5, “Single Sign-On \(SSO\)”](#)). This field specifies that by default, OAuth Client will use SSO to authenticate the user. It is optional and its default value is `false`.
- **SSO Domain (`oauthSSODomain`)** Windows only (see [Section 14.5, “Single Sign-On \(SSO\)”](#)). This field specifies which domain to use when using SSO. `oauthUseSSO` must be set to `true` for this attribute to be used. It is optional and its default value is `mail`.

14.4.2 Example LDIF

An alternative way to create the OAuth2 service configuration directory entry is to LDIF load a configuration. This is advanced usage and you should not use this if you're not sure of what you are doing. Again, Cobalt is the preferred method to configure the OAuth2 service.

Example LDIF containing an OAuth2 client registration entry:

```
version: 1

dn: oauthClientId=isode-dummy-1,cn=OAuthConfig,c=xx
objectClass: oauthClient
oauthClientId: isode-dummy-1
oauthClientSecret: password
oauthRedirectUri: https://localhost:19444/callback
oauthServerType: dummy_example
oauthTokenDuration: 3600
oauthRefreshTokenDuration: 7200
```

Edit the values to suit your needs then use Sodium to load the file. Please refer to Sodium's manual for the instructions. Refer to Cobalt's manual for valid `oauthServerType` values.

14.5 Single Sign-On (SSO)

Since version 19.0, SSO is supported on Windows only. This allows users to authenticate without the need of a login form, streamlining the usage of application that uses OAuth2.

14.5.1 Overview

On a Windows workstation which is part of a Windows Domain managed by an ActiveDirectory, a user using an Isode web application supporting OAuth2 (e.g. Red/Black) should be able to be authenticated without the need for entering a password.

The browser trying to access an Isode application (e.g. Red/Black) which has been configured to use SSO will be challenged by the OAuth2 server to get a Kerberos token from the ActiveDirectory service managing the domain. Once the negotiation is finished, the OAuth2 process will resume normally.

To enable SSO, you need:

- A Windows network using ActiveDirectory Domain Service (AD DS). This manual will not cover this part.
- M-Vault fully configured for OAuth2 and SSO activated (see above)
- Properly configured browser to allow them to speak with AD (see below)

14.5.1.1 Mozilla Firefox Configuration

- Type `about:config` in the address bar
- Search for `trusted-uris`
- Put in both `network.automatic-ntlm-auth.trusted-uris` and `network.negotiate-auth.trusted-uris`, `https://your_network_domain` (e.g. `https://kdc.local`)

14.5.1.2 Internet Explorer Configuration

- Open "Control Panel" -> "Network and Internet" -> "Internet Option"
- In the dialog, go to "Security" tab
- Select "Local Intranet" and click "Sites" button
- Add `https://your_network_domain` (e.g. `https://kdc.local`)

14.5.1.3 Google Chrome Configuration

Follow the same instructions as for Internet Explorer.

Chapter 15 SPIF Editor

This chapter describes the SPIF Editor application and explains how to use it to create, edit and view a SPIF (Security Policy Information File) and various utility functions.

The term SPIF refers to Security Policy Information File. A Security Policy is represented as an SDN.801c SPIF in the Open XML SPIF format. A SPIF is structured data which defines for a given policy ID the valid classifications and security categories. It also can define strings to be associated with labels, which are used for mark-up of data for human reading. It can define equivalent policies, which enables labels defined by a different authority to be associated with labels defined in this SPIF. It also defines how the 'Access Control Decision Function' (ACDF) is to be applied.

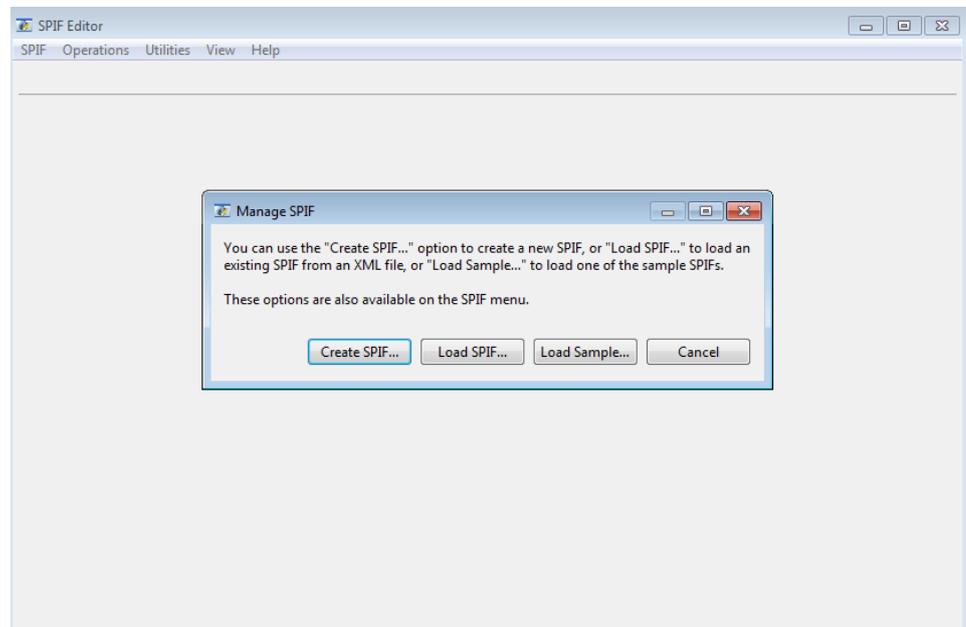
15.1 SPIF Editor Overview

SPIF Editor is a GUI that allows you to create, edit and view SPIFs. The primary purpose of the editor is to make it easy to create and manage security labeling for Isode servers and clients. It is not necessary to use the SPIF Editor in order to use security labeling in Isode products, but in many cases it may prove to be the simplest means of doing so.

15.1.1 Getting started

On launching the SPIF editor, a dialog will appear that provides options to create a new SPIF, load an existing SPIF XML file or load one of the samples provided as part of its installation.

Figure 15.1. SPIF Editor Launch Screen

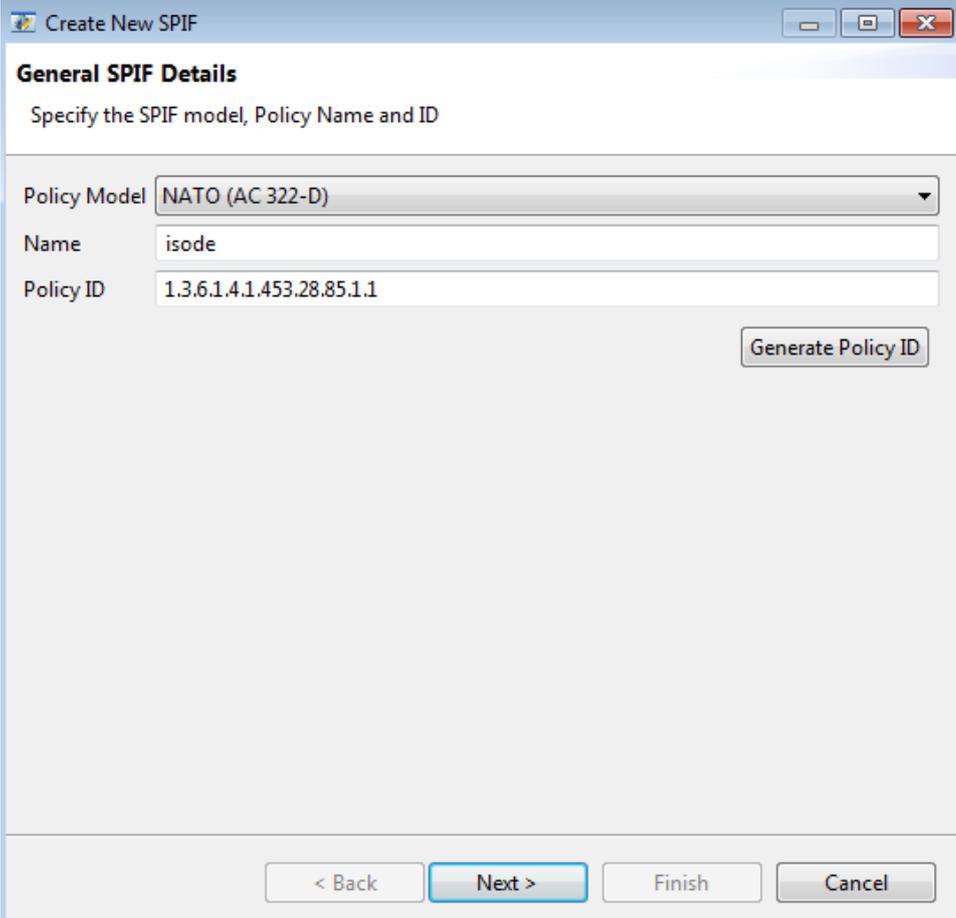


15.2 Creating New SPIF

A new SPIF can be created by choosing the "**Create SPIF...**" button on the launch dialog. The option is also available on the **SPIF** → **Create...** menu. The wizard for creating a new SPIF is shown in the figure below.

Provide the policy model, name and ID for the SPIF on the first page.

Figure 15.2. Create SPIF



The screenshot shows a window titled "Create New SPIF" with a standard Windows-style title bar (minimize, maximize, close buttons). The window content is titled "General SPIF Details" and includes the instruction "Specify the SPIF model, Policy Name and ID".

The form contains three input fields:

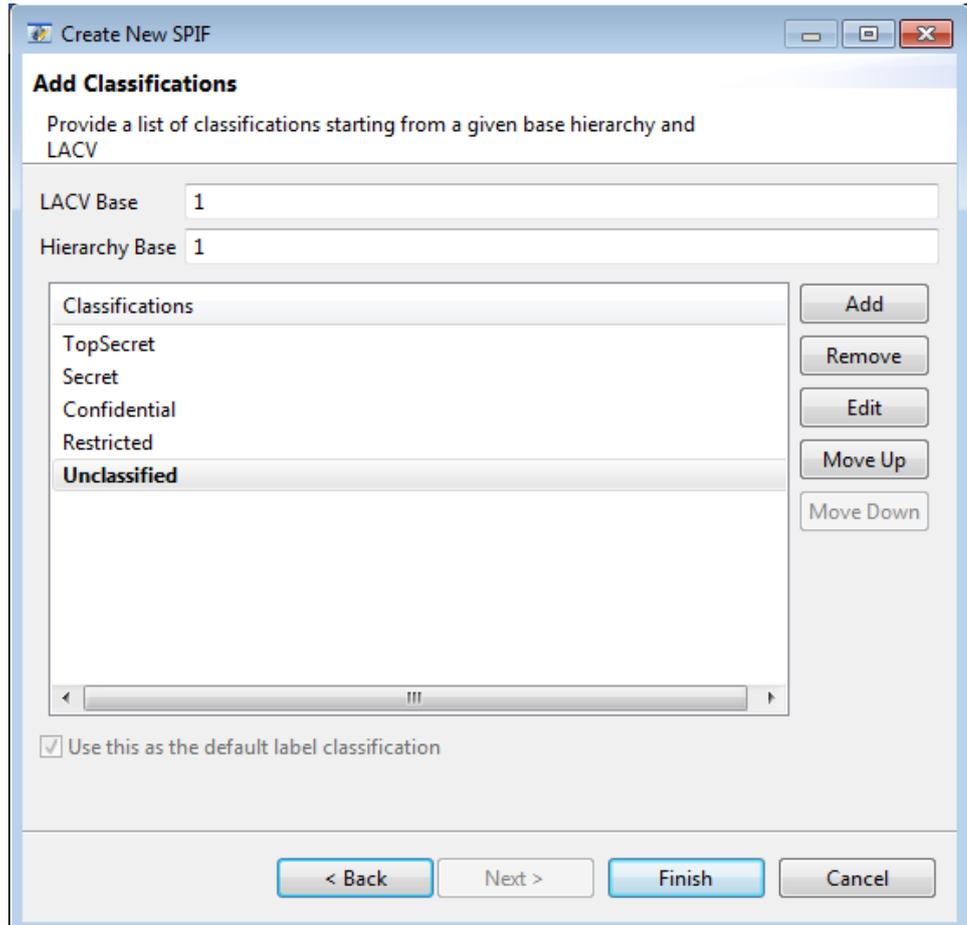
- Policy Model:** A dropdown menu with "NATO (AC 322-D)" selected.
- Name:** A text box containing "isode".
- Policy ID:** A text box containing "1.3.6.1.4.1.453.28.85.1.1".

Below the Policy ID field is a button labeled "Generate Policy ID".

At the bottom of the dialog, there are four navigation buttons: "< Back", "Next >" (highlighted with a blue border), "Finish", and "Cancel".

On pressing **Next** button, list of standard classifications will be offered as a default.

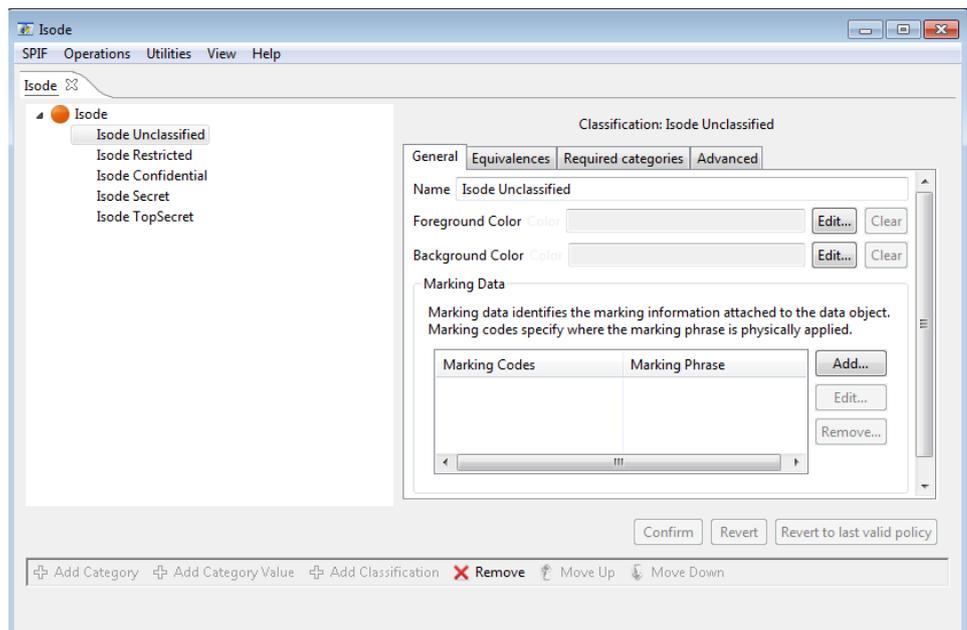
Figure 15.3. Create SPIF Classifications



The list can be modified to add or remove classifications. The name of the classifications can be modified using **Edit...** button. The LACV value stands for the classification value whereas the hierarchy governs the ordering of the classifications in the SPIF.

On pressing **Finish**, the SPIF will appear on the SPIF editor as shown below.

Figure 15.4. Created SPIF



15.3 Managing Existing SPIF

An existing SPIF XML file can be loaded in a SPIF editor using the **SPIF** → **Load...** menu.

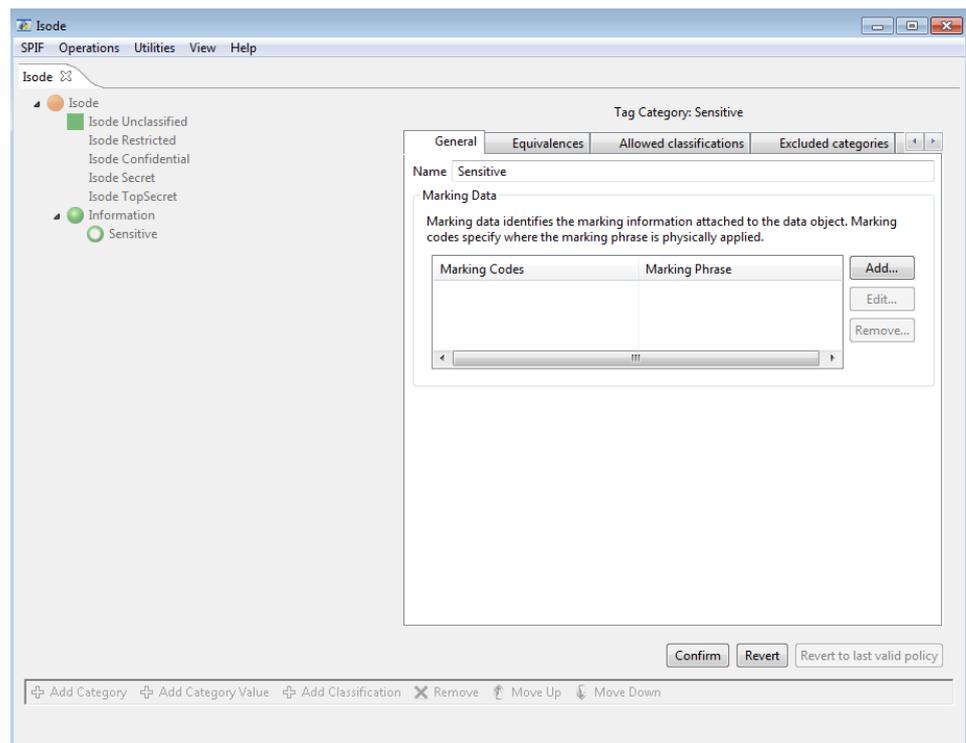
Once a SPIF has been created or loaded from an XML file, it can be viewed or edited using the SPIF editor. The left hand side presents the classifications and categories of the SPIF in a tree format. The classifications are listed on top of the tree followed by categories.

Classifications are displayed using their background color icon and categories are displayed using green circle icons. Note that categories are optional and may not exist in most SPIFs.

On selecting a classification or a category, the right hand side pane will display the details of selected classification or category.

When the selected classification or category is edited, the **Confirm** and **Revert** buttons will get enabled to let you apply the current set of changes or cancel them. Note that the **Confirm** button will not get enabled until the current set of changes made are complete and valid.

Figure 15.5. Category Edit



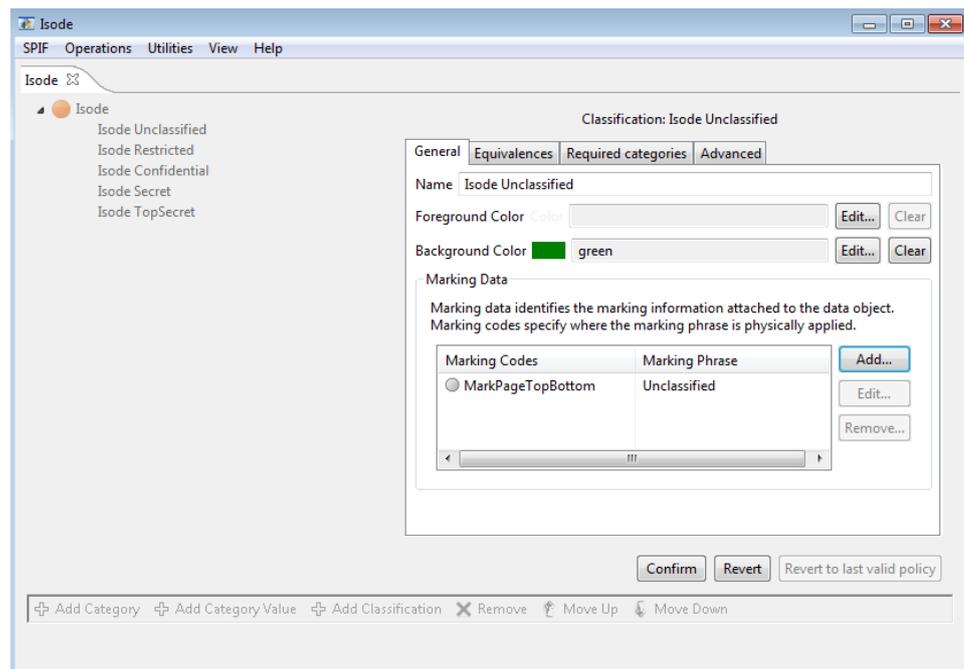
The editor allows you to modify only one classification or category in one operation. For complex policies, a change in more than one classification or category may be required to create a valid policy. If an edit makes the policy invalid **Revert to last valid policy** will get enabled to allow you to revert to last valid state to undo the changes that lead to the invalid policy.

15.4 SPIF Classifications

Select a classification on the left hand side side in order to edit it. After making the required changes, click the **Confirm** button.

The following figure displays the SPIF after adding a marking code and changing the background color of the classification.

Figure 15.6. Classification Edit

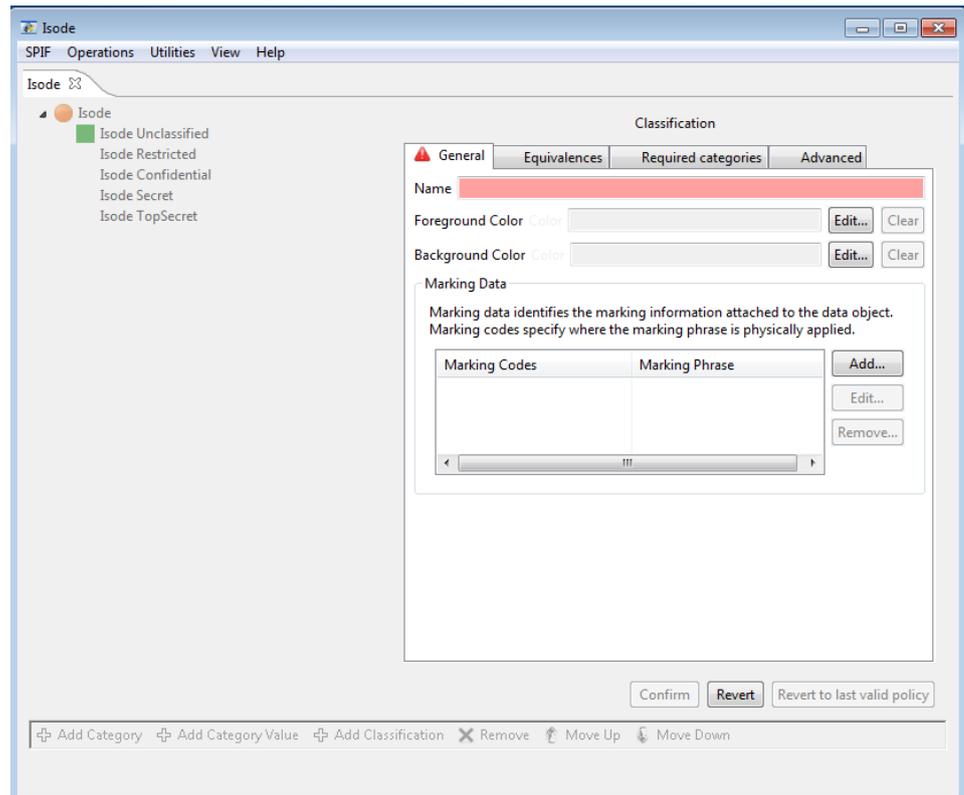


The **General** tab lists the most commonly used attributes. Rest of the tabs define advanced parameters that are required for complex SPIFs.

15.4.1 Adding Classifications

Select the topmost tree item for the policy and click **Add Classification** button to add a new classification. Provide the details of the new classification to be added on the right hand side pane. The tabs that require mandatory parameters for completing classification creation will display a red icon on the top.

Figure 15.7. Add Classification



Once the details of the new classification have been provided, press the **Confirm** button to create the new classification. By default, the editor will fill in default values and in simple cases you will only need to provide the classification name. Colors and markings can be added to suit the requirements.

15.4.2 Removing Classifications

Select a classification and click **Remove** button and confirm to remove the selected classification from the policy. Errors will be reported if removal of a classification invalidates the security policy.

15.5 SPIF Categories

15.5.1 Adding Category

In order to add a new category group, select the topmost tree item for the policy and click **Add Category** button. A wizard to add a new category will be displayed.

Figure 15.8. Adding Category

Create Category Wizard

Category Details
Provide the details of the new category to be added

General | Advanced

Category Name: Information

First Category Name: sesitive

First Category value: 0

Category Type and Encoding

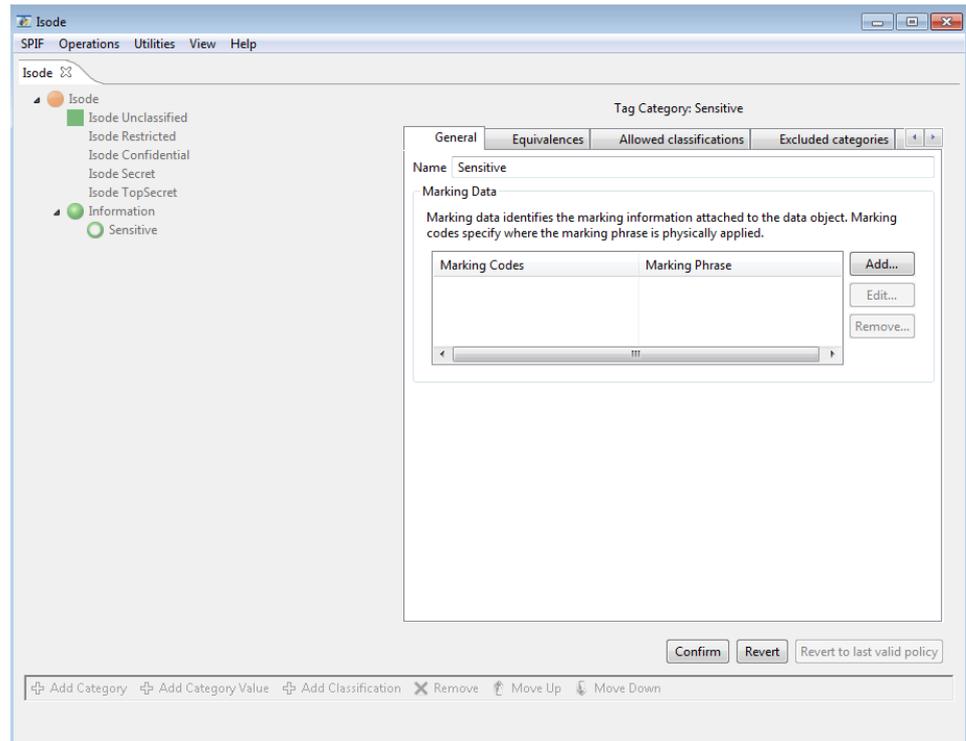
Category Type: Informative | Encoding Type: Enumerated

Finish | Cancel

The wizard page will ask you to provide the details of the new category group and first value in the group. For simple case, category name and type will have to be provided. See the tooltips on the widgets for more information on the parameters.

On pressing the **Finish** button the editor will display the category to be added on the editor. You can make further changes to the values for this category. Press **Confirm** button to add the category to the SPIF.

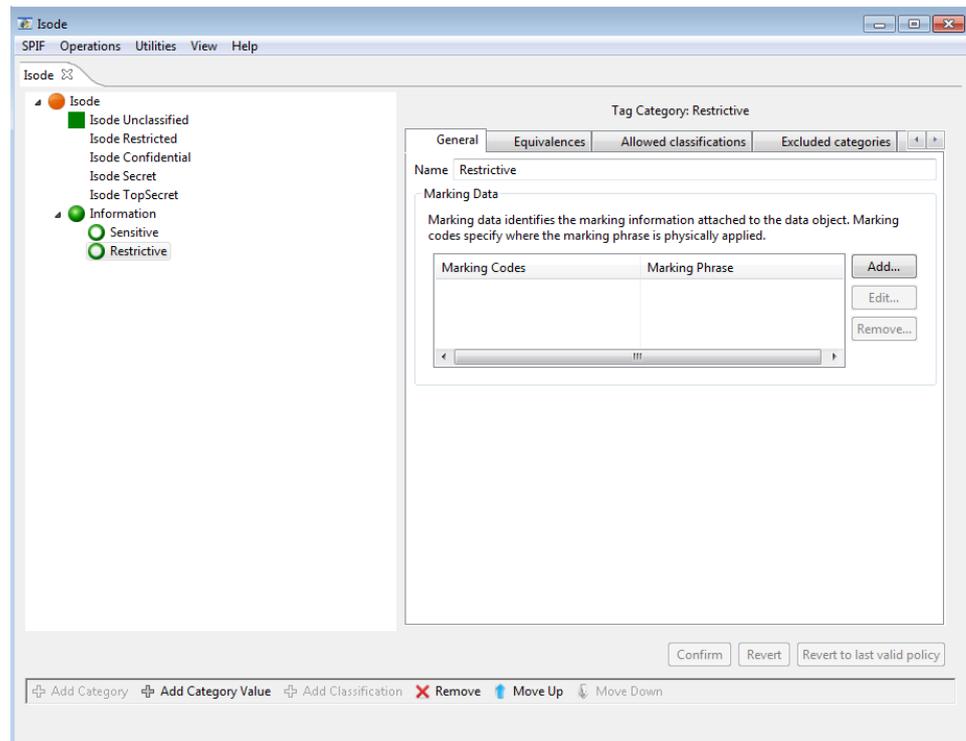
Figure 15.9. New Category



15.5.2 Adding Category Value

To add a new category value to an existing category group, select the category group and click the **Add Category Value** button. The right hand side pane will change to a mode for adding a new category. Provide the name of the category value and other details if required and press the **Confirm** button. The new value will be added to the SPiF as shown below.

Figure 15.10. Adding Category Value



15.5.3 Removing Category

Select the category value and click **Remove** button to confirm removal of the category value. Similarly, select a category group and click **Remove** button to remove the selected category group and all its values from the policy. Errors will be reported if removal of a category invalidates the security policy.

15.5.4 Moving Categories

Select a category group or value and click on **Move Up** or **Move Down** button to move it up or down the hierarchy. Press **Confirm** to apply the changes to the SPIF.

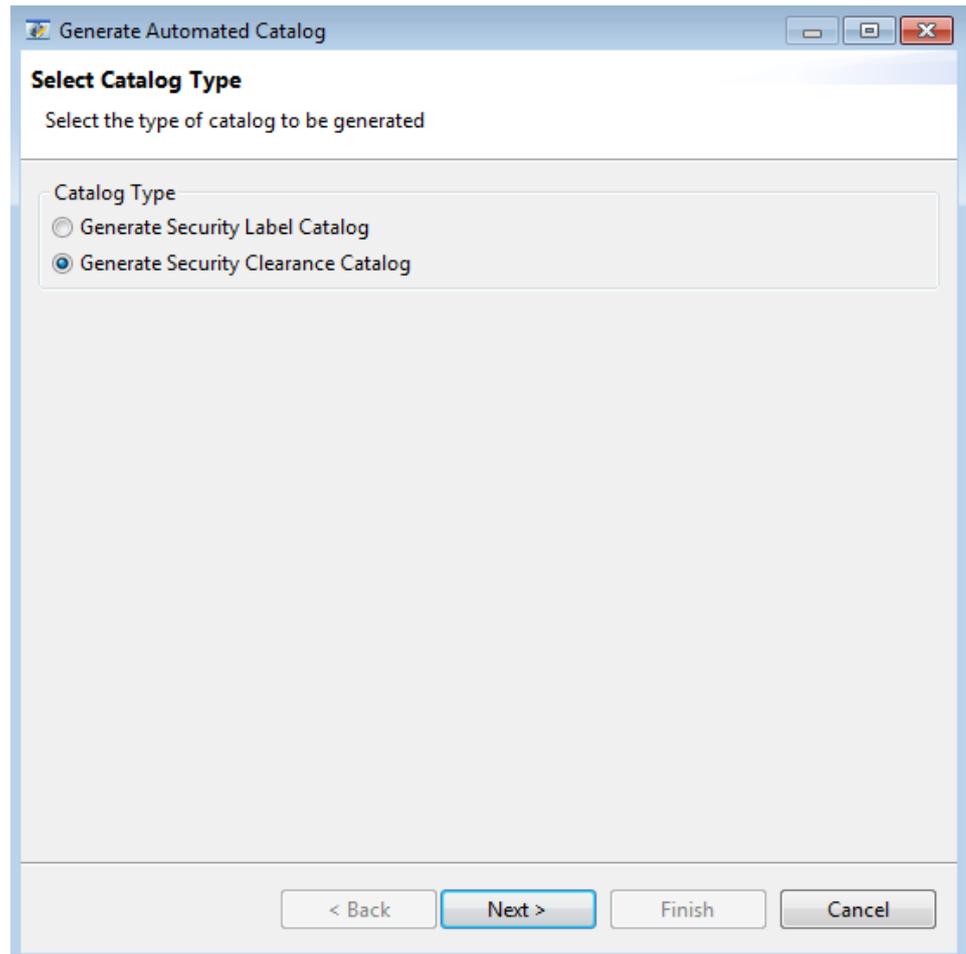
15.6 SPIF Utilities

The SPIF editor provides commonly used functions that are available via the **Utilities** menu.

15.6.1 Generate Catalog

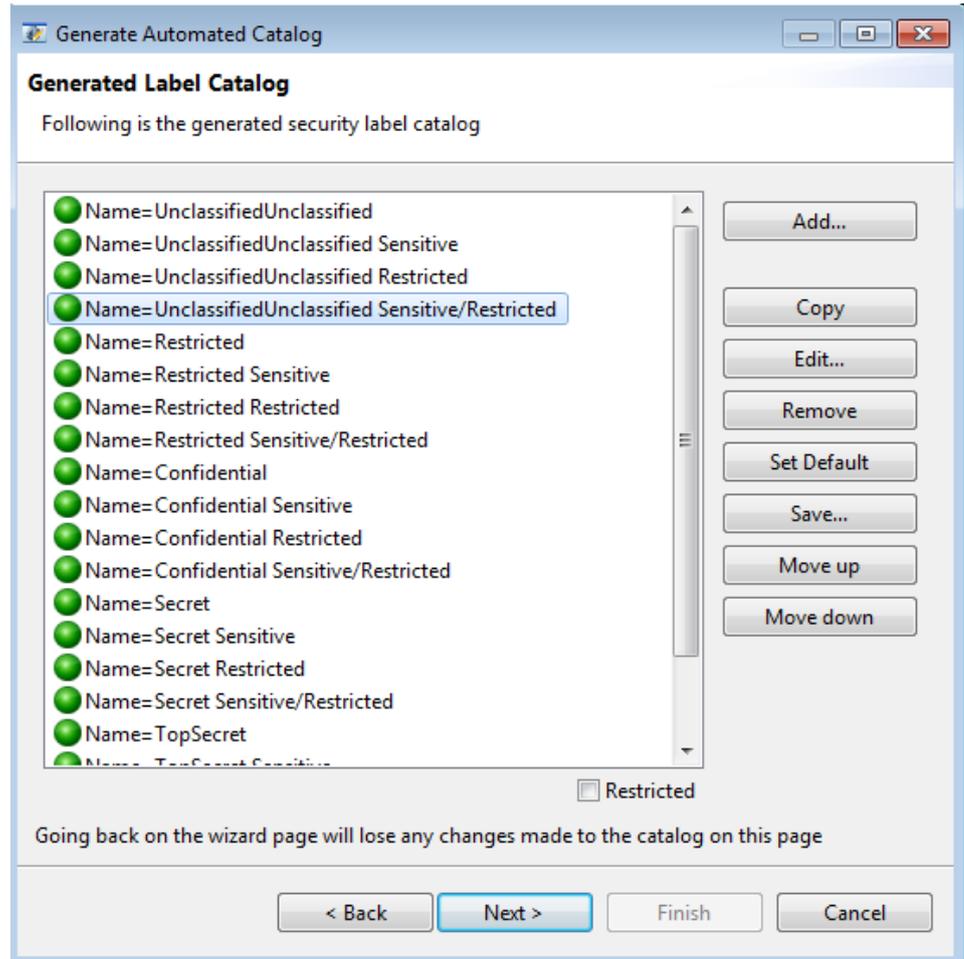
Security label and clearance catalogs are collections of security labels and clearances. Click the **Utilities** → **Generate Catalog...** menu to launch a wizard to auto generate label and clearance catalogs.

First select the type of catalog to be generated.

Figure 15.11. Select Catalog Type

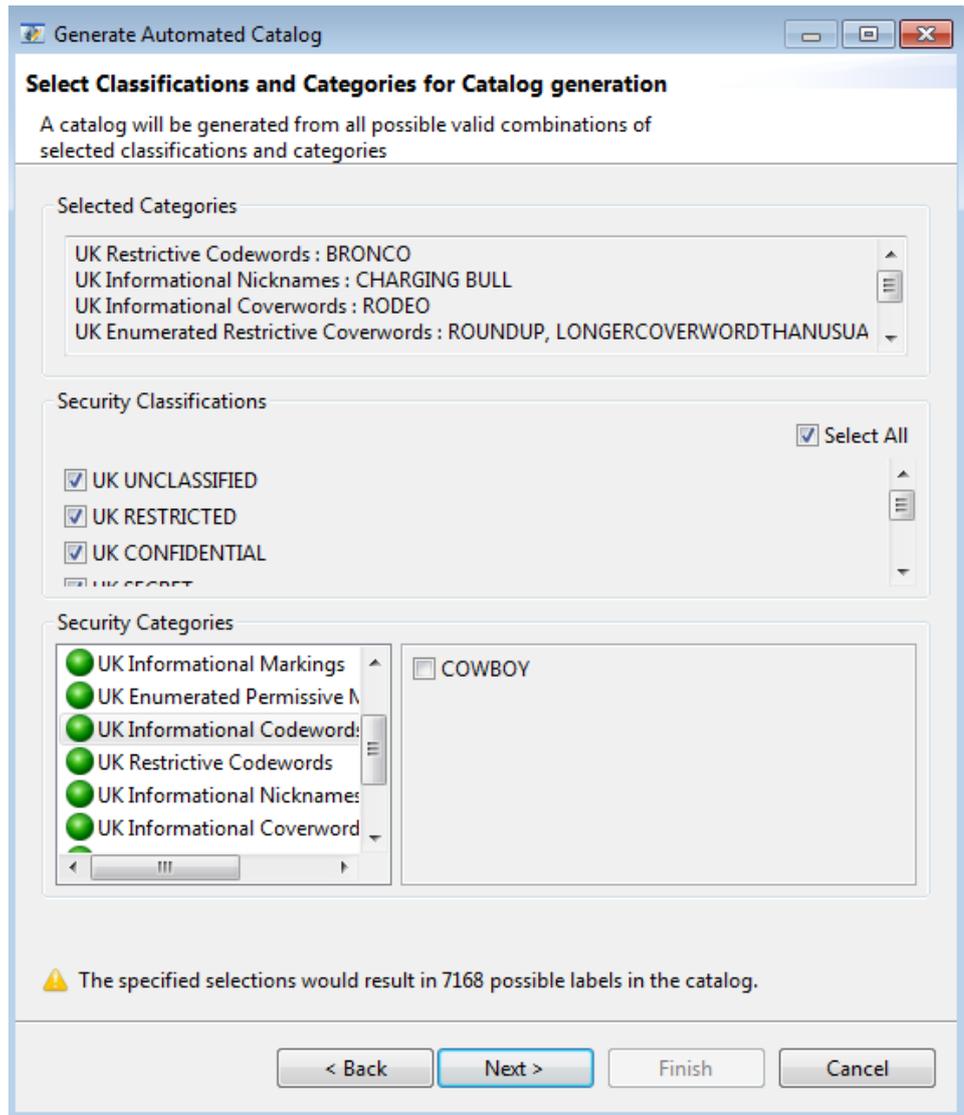
If the policy is simple with few classifications and categories (optional), the wizard will generate a catalog with all possible combinations of classifications and categories. The generated catalog will appear as shown in the figure below that displays a sample auto generated label catalog.

Figure 15.12. Label Catalog



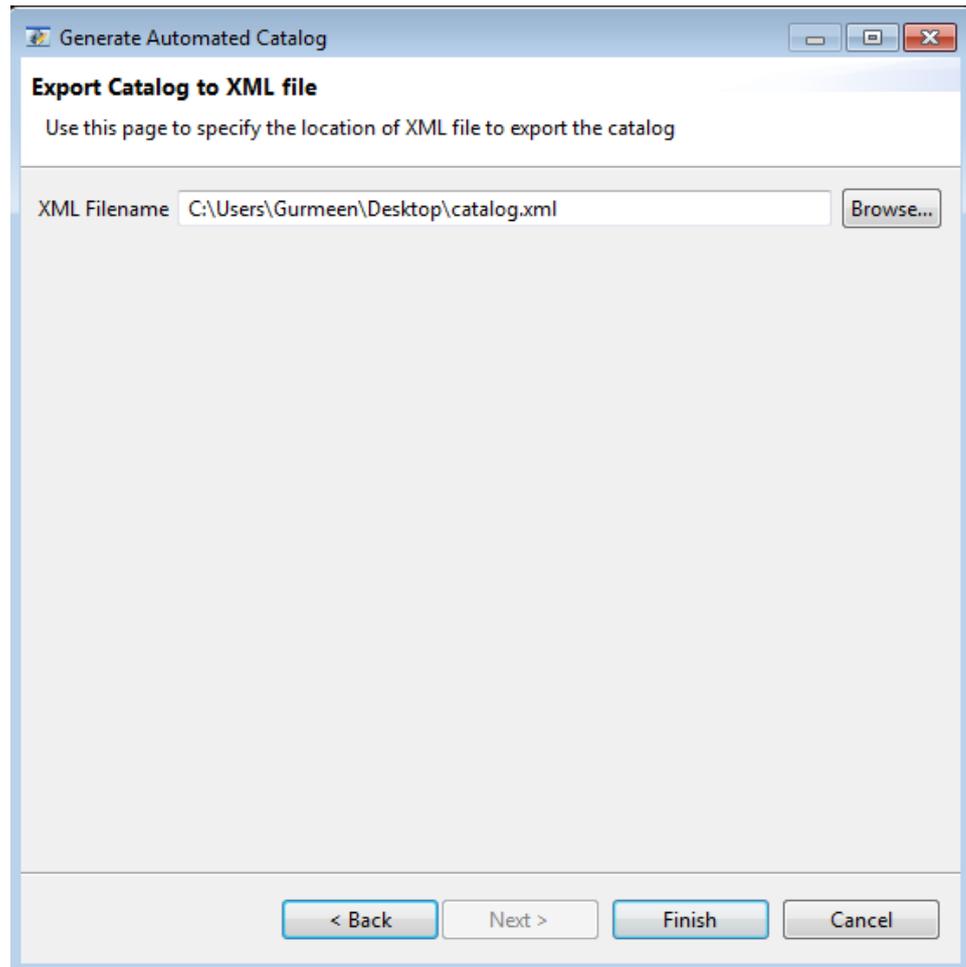
However, for a policy which has large number of classifications and categories, the wizard will present a page to choose a set of classifications and categories to be included in the catalog.

Figure 15.13. Selected Classifications and Categories



The wizard will attempt to generate a catalog from all possible combinations of selected classifications and categories. A warning will be displayed on the bottom of wizard page if the number of possible combinations is high (in terms of hundreds) and an error will be displayed if the number of combinations is very high (in terms of thousands).

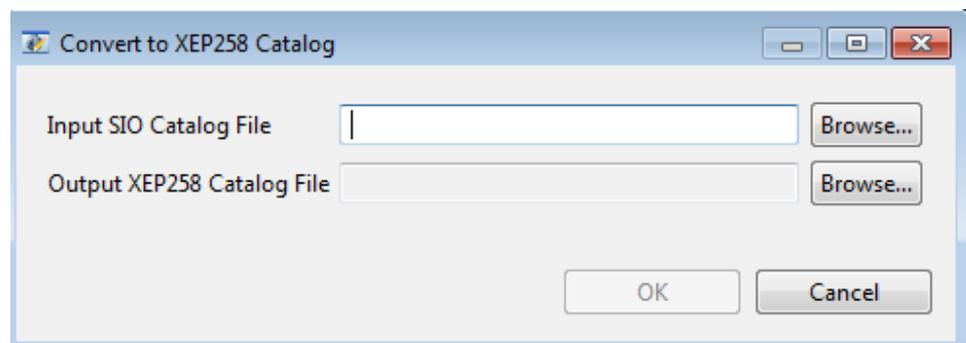
Once a label or clearance catalog has been generated, it can be edited if required on the wizard page that displays the generated list as shown in figure [Figure 15.12, “Label Catalog”](#). On pressing the **Next** button, the page will prompt you to select an XML file location to save the catalog.

Figure 15.14. Export Catalog

Press **Finish** to complete the catalog generation.

15.6.2 Converting Label Catalog to XEP-258 Format

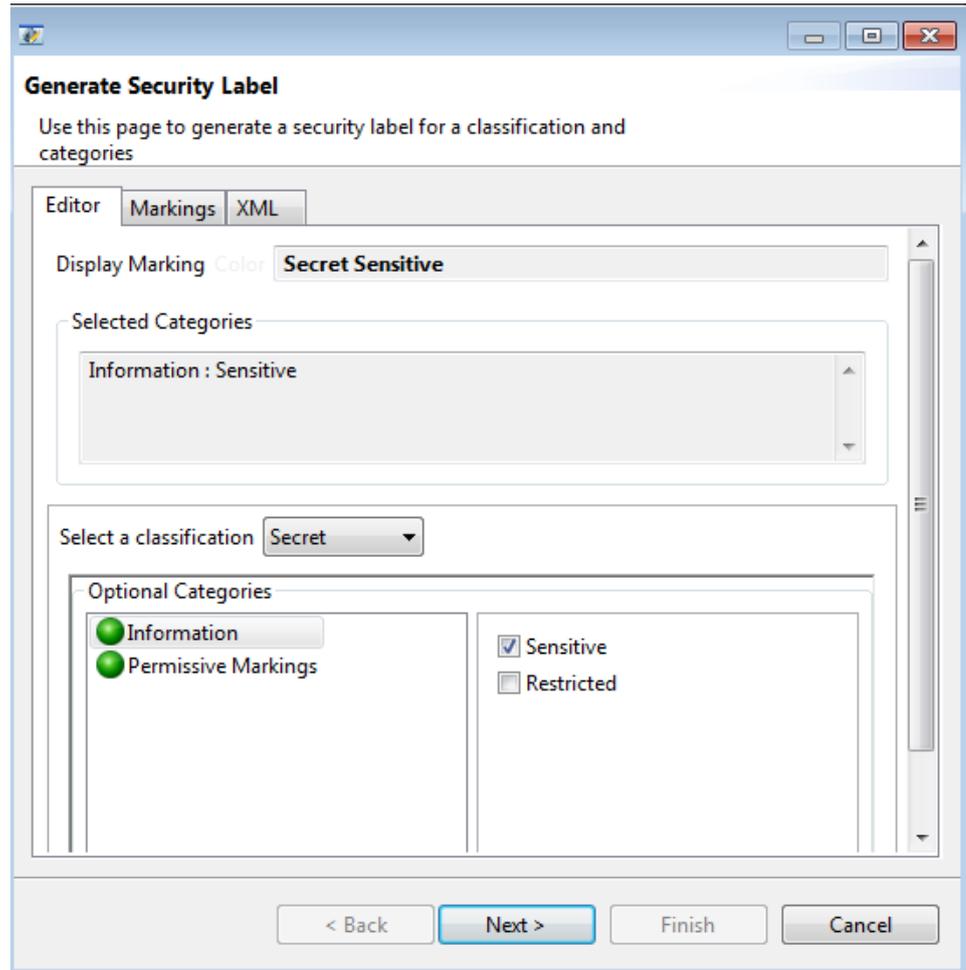
The **Utilities** → **Convert to XEP258 Catalog...** can be used to convert a label catalog to a format that conforms to the XEP 258 format of the XMPP standards.

Figure 15.15. Convert to XEP258 Catalog

15.6.3 Generate Label

For simple policies, select a classification and one or more categories to generate a label.

Figure 15.16. Generate Label for Simple Policy



Label generation for complex policies is described below.

Figure 15.17. Generate Label for Complex Policy

Generate Security Label

Use this page to generate a security label for a classification and categories

Editor | Markings | XML

Display Marking

Selected Categories

UK Informational Descriptors : APPOINTMENTS

Select a classification

Mandatory Categories

Select One or More

UK Informational Descriptors

APPOINTMENTS

BUDGET

COMMERCIAL

CONTRACTS

CONTROL

Optional Categories

UK No Foreign Transmission

UK Enumerated Permissive Nation

UK Informational Descriptors

UK Informational Markings

UK Enumerated Permissive Marki

UK Informational Codewords

< Back | Next > | Finish | Cancel

First select a classification from the drop-down list. Selecting a classification may or may not require inclusion of certain categories. The required categories if any will be displayed as a list in the **Required Categories** pane. The **Optional Categories** pane lists all the categories in the configured policy from which the user can select certain categories to be added to the label. The categories which are disallowed based on the selection of a certain category or the classification will be disabled automatically on the editor. The obsolete categories will be allowed for editing based on whether **Edit obsolete elements** is selected or not.

Selection rules of a category group determine whether it allows selection of single or multiple categories in the group. For single category selection, the categories are displayed using radio buttons and for multiple category selection they are displayed as check-boxes.

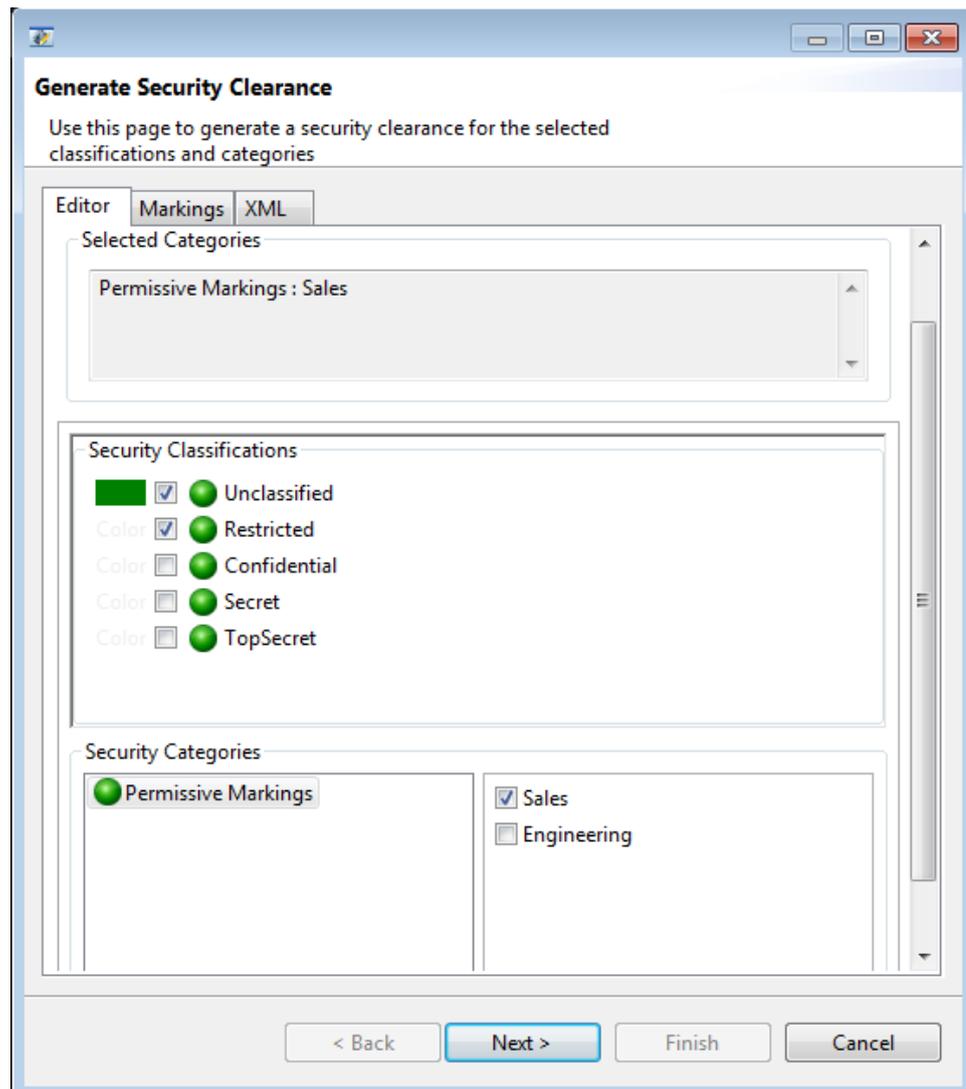
The markings get updated on the **Markings** tab when a valid combination of categories has been selected.

Note: Rules for label editing are based on SDN.801c and are configured in the security policy.

15.6.4 Generate Clearance

The **Utilities** → **Generate Clearance...** can be used to generate a security clearance using the selected policy. The following wizard will be presented for clearance generation.

Figure 15.18. Generate Clearance

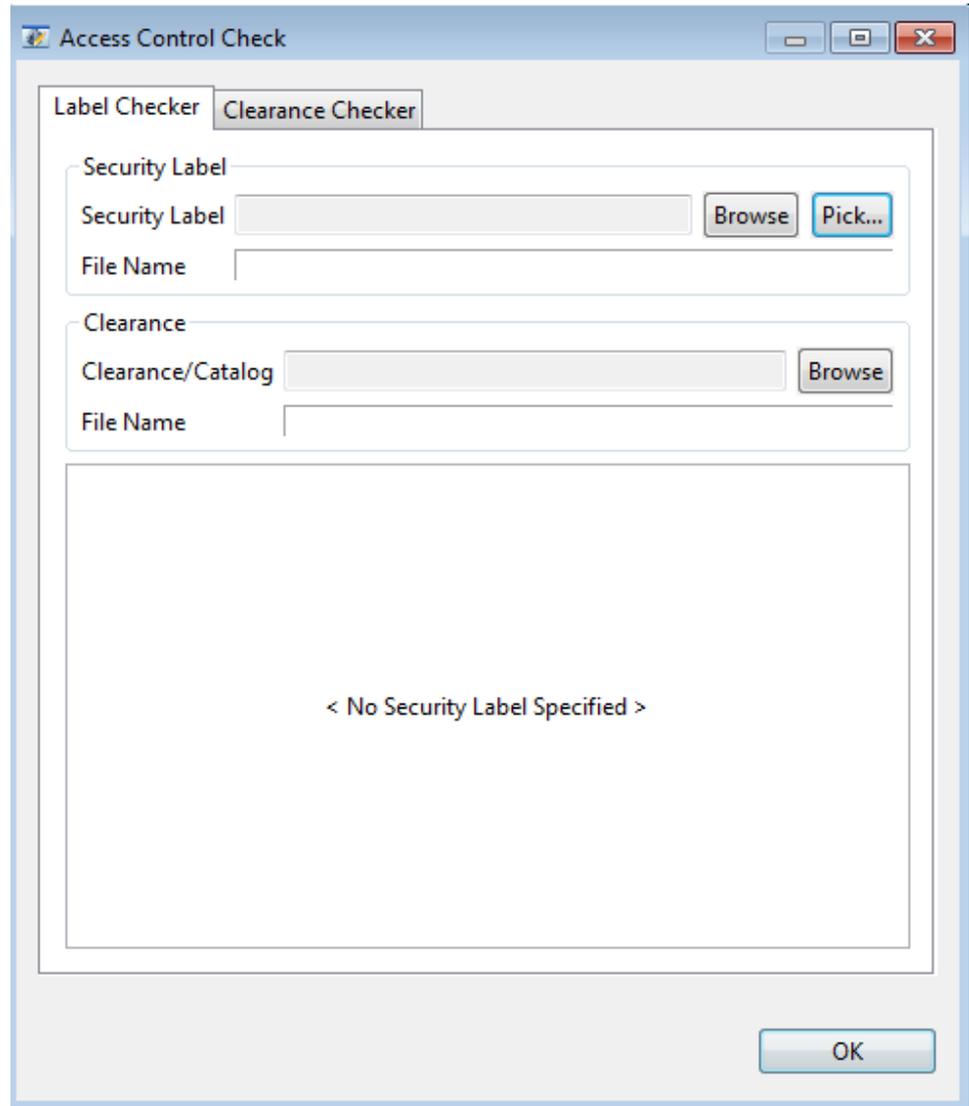


One or more classifications can be selected from the **Security Classifications** pane to be added to the clearance. The **Security Categories** pane lists all the categories in the configured policy from which the user can select certain categories to be added to the clearance. The markings get updated on the **Markings** tab as and when the clearance is edited.

15.6.5 Access Control Checks

SPIF editor can be used to verify a security label against a security clearance and vice versa. To check access controls, select **Utilities** → **Access control checks....** menu. A dialog for performing these checks will be presented.

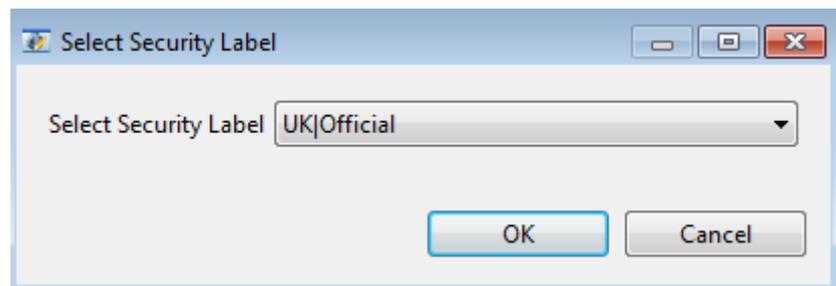
Figure 15.19. Access Control Checks



Select **Label Checker** to check access control of a label against a clearance or a catalog of clearances. Select **Clearance Checker** to check access control of a clearance against a label or a catalog of labels.

The label can be selected from an XML file by browsing the file system using the **Browse** button or picked up from a label catalog using the **Pick...** button. The **Pick...** button will offer the labels in the catalog in the form of a dropdown list as shown below.

Figure 15.20. Pick Label from Catalog



The label can be checked against a clearance or a clearance catalog that can be selected from the file system using the **Browse** button.

Once a label and clearance/catalog has been selected, the result of access control checks will be displayed in the bottom pane.

Figure 15.21. Access Control Check of Label with Clearance Catalog

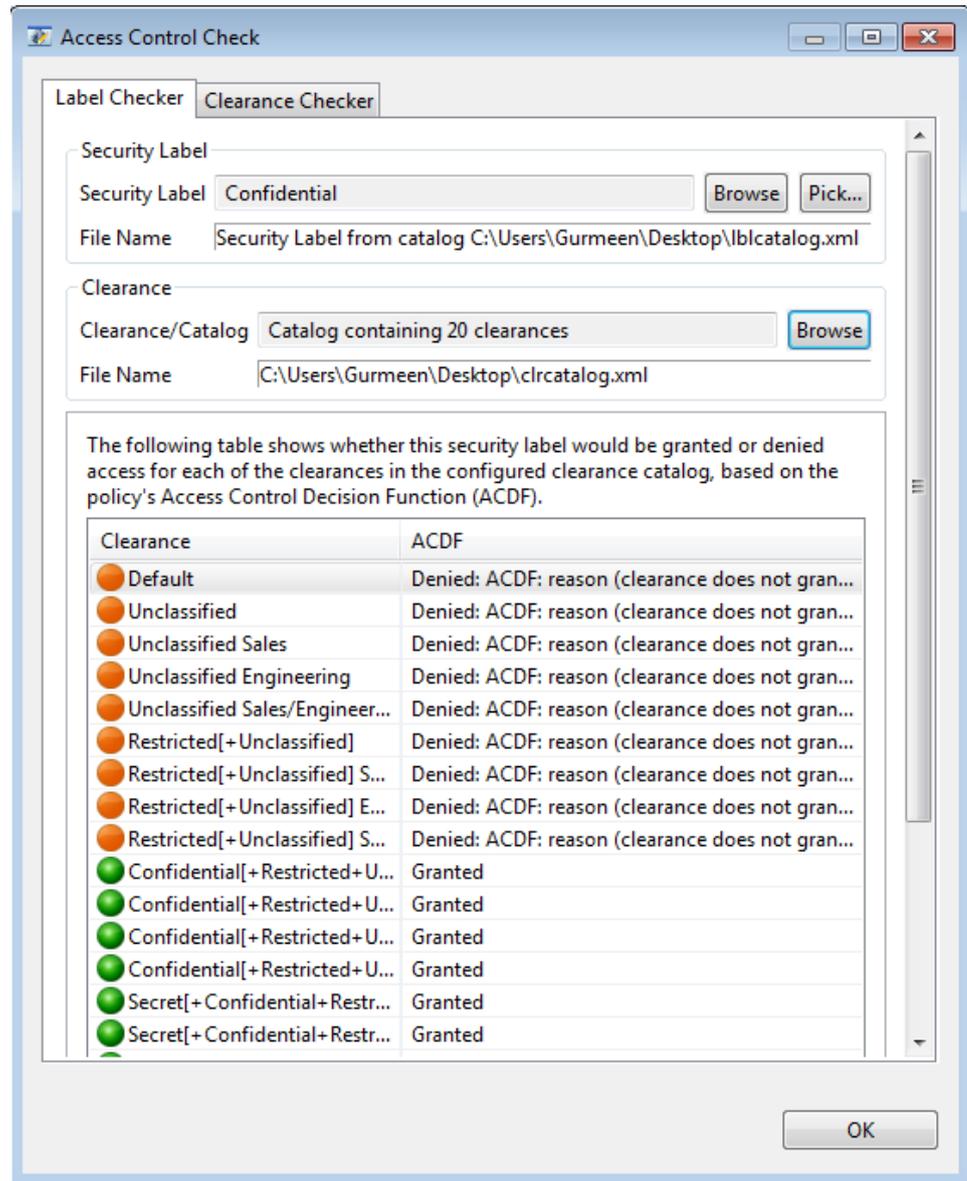


Figure 15.22. Access Control Check of Label - Access Granted

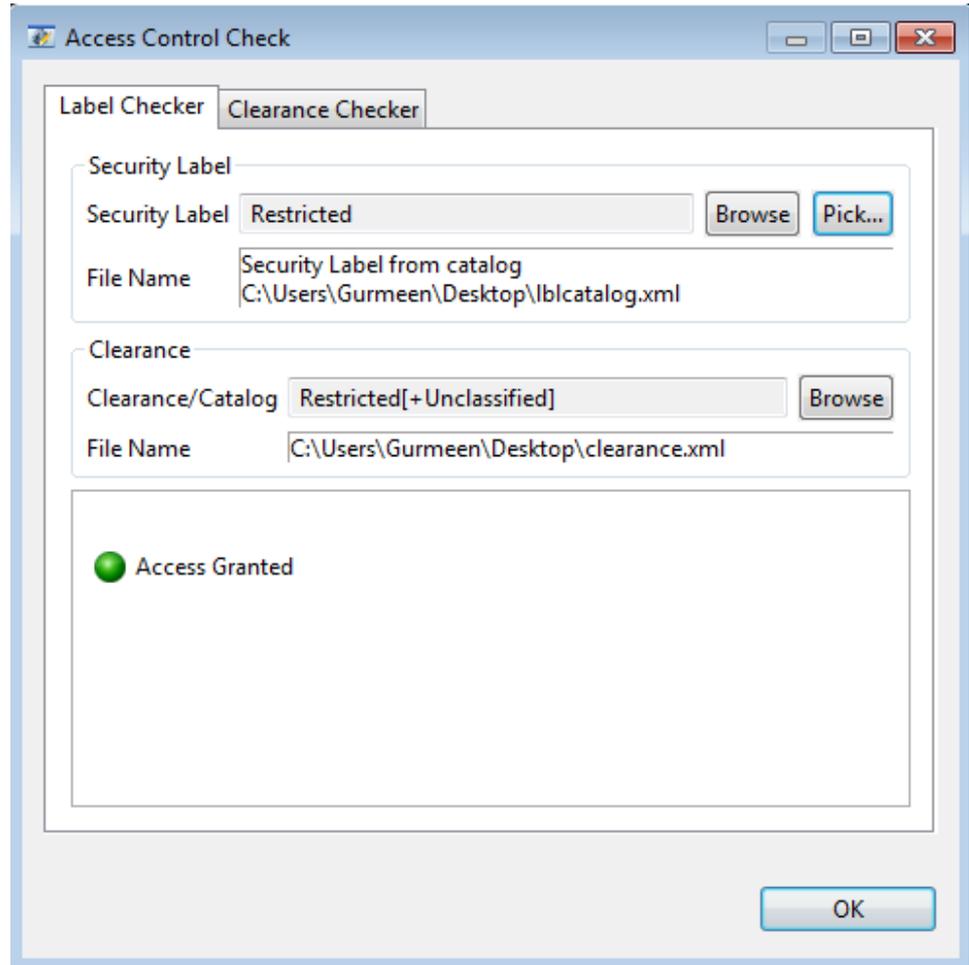
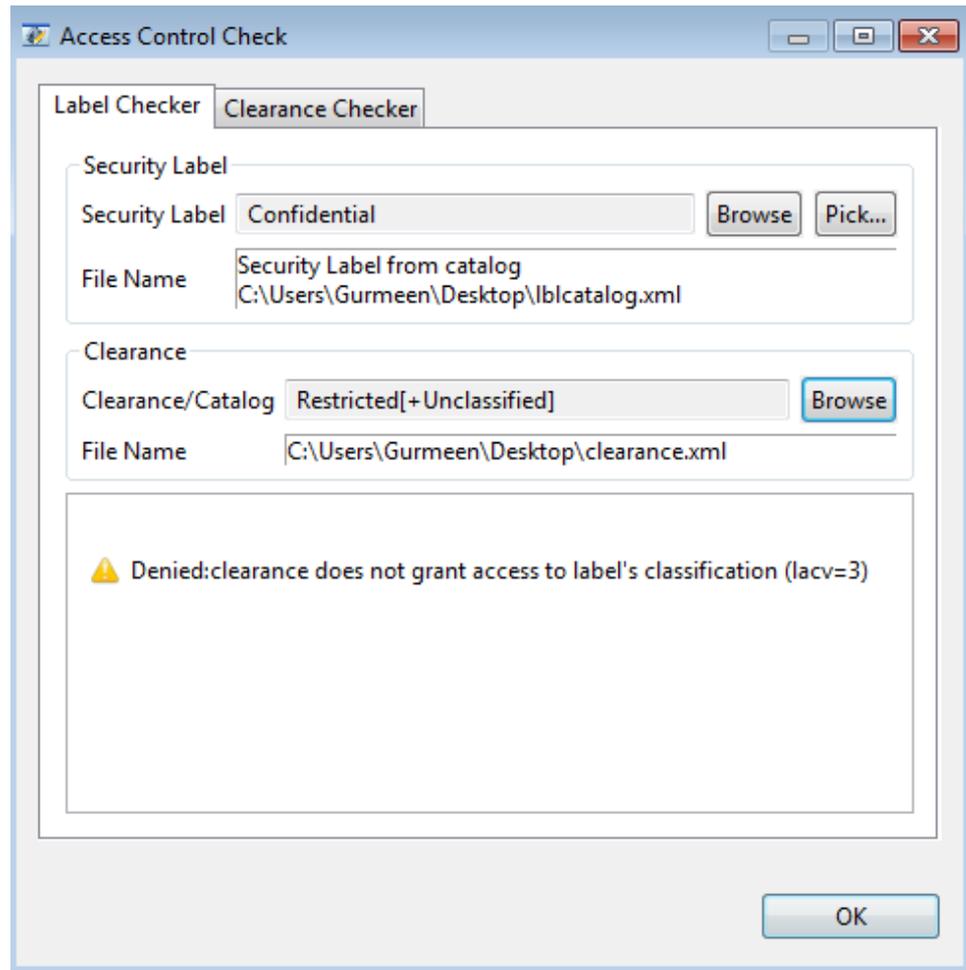


Figure 15.23. Access Control Check of Label - Access Denied

Follow the above steps in a similar way to check a clearance against a label or a catalog of labels by selecting the **Clearance Checker** tab.

Appendix A Introduction to Directories

This appendix provides a background to X.500 and LDAP Directories. It provides a context for Isode's implementation of the Directory, and should be read by anyone not familiar with the X.500 standard or who wants to understand how Isode has implemented it.

A.1 Definition of the Directory

The Directory described in this manual is in line with that of the LDAP and X.500 standards. Detailed references to the individual standards are given in [Appendix I, References](#).

Providing an Enterprise Directory using open systems standards enables a wide range of enterprise applications to plug into a common Directory core.

A Directory is a special purpose database. The essential features which make it a Directory are:

- It can be very large and highly distributed, in most cases on an organizational basis, over a number of Directory Servers. This distribution is largely transparent to Directory users. Answers to queries are independent of the location of the user agent making the query and of the Directory Server holding the data.
- It contains information about objects of interest within the enterprise (for example, people or pieces of equipment). Each entry in the Directory represents a single object. There is a fixed structure for well-known object types and for the information associated with those objects. This structure is highly extensible.
- It is hierarchically structured, the entries being arranged in the form of a Directory Information Tree (DIT), as described later. Objects are named within the hierarchy and globally, ensuring that all objects have unique Directory names. This global naming may not be visible within the enterprise (the global prefix may be hidden), but it is important as it facilitates inter-working between enterprises.
- It supports read operations, full facilities to modify and update data, and search operations optimized to follow the hierarchy of information. Read and search operations are more common than modification and are very fast.
- Data can be replicated across a collection of distributed Directory Servers. The temporary inconsistency between data held on one server and that on another is highly manageable.
- It is possible for one Directory Server to pass on a request to another server or to refer the request back to the originator, if required.
- There is security functionality to provide access control to the information in the Directory and to authenticate those requesting that information.

A.2 Directory user information

The Directory contains a database of information on objects:

- People
- Organizations

- Application processes
- Devices, etc.

The information comprises various types of names and addresses for the objects. The information about objects stored in a Directory is known as the Directory Information Base (DIB). An object in the real world is represented by an entry in the DIB (the object entry).

The information can be updated, searched, processed and eventually delivered to the requesting client or user. In order to support these types of operations uniformly, the information in the Directory must be uniquely categorized and identified since the Directory could be a repository for information derived from a large number of sources.

Because this information in the DIB is available to the normal users of the Directory (human or computer application), it is often termed user information. The user information is usually under the control of a Data Manager.

As we shall see later, there is also another type of information in the Directory which is used to control the user information. This is called operational information and is described in [Section A.3, “Operational information”](#). This operational information is not available to the normal user, but only to the Directory Management System, and is usually under the control of a Server Manager.

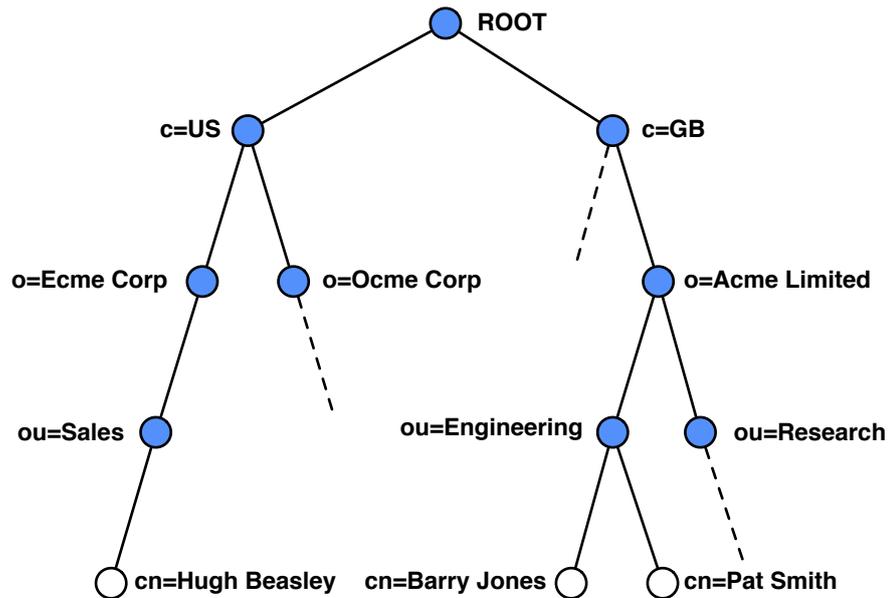
A.2.1 Directory information tree

The standards impose a hierarchical structure on the information. This hierarchy is called the Directory Information Tree (DIT), and is illustrated in [Figure A.1, “Example Directory Information Tree Structure”](#).

The hierarchical structure of a Directory leads to a situation where most of the useful information from the perspective of a user is found at the lower levels, while the top levels hold information which facilitates unique naming and navigation across the hierarchy. This structure enables the information held in an enterprise-wide or even global Directory Service to be partitioned. It allows separate administrative domains a high degree of autonomy to manage their own Directory structures and the information within them, while simultaneously remaining connected to a wider Directory system.

In a global Directory, the top level is organized along geopolitical boundaries and then around organizational or institutional boundaries at subsequent layers. [Figure A.1, “Example Directory Information Tree Structure”](#) illustrates how entries at the top are typically used to represent countries (for example, **c=US** is used to represent the United States of America) and entries within countries are represented by organizations (for example, **o=Acme Limited** is used to represent a hypothetical organization of that name). Intermediate levels are commonly used to represent the intra-organizational structure that facilitates the conceptual understanding and administration of information held lower down (for example, **ou=Research** is used to represent an organizational unit, or department, called Research). The information held at the bottom usually represents a person (for example, Pat Smith), but may also represent a piece of office equipment such as a fax machine, a PC, or application process such as a print server. In this example, these lowest entries are identified by their common name (**cn=Pat Smith**).

Figure A.1. Example Directory Information Tree Structure



Within the hierarchy of the DIT, an entry can be a node entry (●), with other entries below it in the structure, or a leaf entry (○), without. The top of the hierarchy is called the root. The root itself is not an object; it does not have an entry.

Although most entries in the DIT are object entries, some leaf entries are known as alias entries. They point to other object entries (leaf or not) and provide the mechanism for the alternative naming of objects.

There are also special node entries called glue entries. These are not object entries. They are simply placeholder entries in the DIT, showing the structure.

A.2.2 Entries and attributes

An entry is made up of a set of attributes, each concerned with some aspect of the object. The attribute type classifies the aspect, and the attribute values describe that aspect. For example, an entry for an organization could contain attribute types giving values for the postal address, telephone number, fax number, business category, and so on. It would also have to contain an attribute defining the entry as an organization (see [Section A.2.3, “Object classes and the schema”](#)) and an attribute naming the organization (see [Section A.2.4, “Naming objects”](#)).

Within the standards, some attribute types are allowed only single values; others can have several values. Attribute values have a defined syntax, which determines how values are represented, and what range of values and matching rules are applicable to the attribute type. Attribute syntax is described in detail in [Appendix C, *Attribute Syntaxes*](#).

A.2.3 Object classes and the schema

Objects are classified within the DIB. An object class has certain attribute types associated with it. There are two sets of attribute types for the object class: its mandatory set and its optional set. The object class attribute must be present in each entry. The object class is sometimes referred to as the entry type. Thus, in an entry with an object class of organization, the mandatory set of attributes includes the **organizationName**, and the optional set includes **postalAddress** amongst other useful attributes.

The most important type of object class is the structural object class. This imposes the hierarchical structure on the data. For a list of structural object classes, together with their

naming attribute and other mandatory attribute types, plus other optional attributes, see [Section A.3.2, “Distribution of the Directory”](#).

The following rules are contained in the Directory schema:

- Constraints on the hierarchical structure of the DIT, that is, permitted hierarchical relationships between object classes.
- For each object class, a specification of which attributes are mandatory and which are optional.
- The characteristics and syntax of each attribute.

A.2.4 Naming objects

Each entry must also have at least one naming attribute. The collection of these attributes in the entry form the Relative Distinguished Name (RDN).

The first goal of naming is to uniquely identify entries. Once this is achieved, the next major goal when naming entries is to facilitate querying of the Directory. For White Pages applications in particular, a naming structure that supports searching and identification of entries (user friendly naming, as defined in *RFC 1781*) is desirable. Other considerations, such as accurately reflecting the organizational structure of an organization, should be disregarded if this has an adverse effect on normal querying.

An RDN is usually represented by a single naming attribute with a single value. It is commonly represented as an abbreviated form of the attribute type and the value, separated by an equals sign. These are both valid RDNs:

cn=Barry Jones

o=Acme Ltd

Within the Directory, objects are uniquely identified by a Distinguished Name (DN). The Distinguished Name (DN) of the object is constructed by concatenating the Relative Distinguished Names (RDNs) of all the entries in the DIT at and above the entry, up to the root. The Distinguished Name thus shows the position of the entry in the DIT as well as identifying the object.

For example, the DN of the entry in the lower left corner of [Figure A.1, “Example Directory Information Tree Structure”](#) is:

cn=Hugh Beasley, ou=Sales, o=Acme Corporation, c=US

That is, this Hugh Beasley is uniquely identified as being the Hugh Beasley in the Sales Department of Acme Corporation.

A.2.4.1 Choice of names

If you want your Directory to form part of an overall Directory structure, you have to make sure that the names you use are of the correct format for that structure. This means:

- If you are adding an organization to the DIT in a country that has established guidelines, you should follow them. These guidelines might be based on an established registration authority, or may make use of an existing registration mechanism (for example, company name registration).
- If you are adding an organization to the DIT in a country that does *not* have any established guidelines or existing national registration, you should choose a name that is meaningful. This will typically be the full, well-known name of the organization.

The name must uniquely identify the organization and be one that is unlikely to be challenged in an open registration process. For example, if the organization has a name

that is registered in an existing registry (such as company name registration), this name is likely to be appropriate for use in the Directory.

For example, O=UCL would not be a good choice of RDN for University College London, as it might well be challenged by United Carriers Ltd.

If your Directory is self-contained, you do not need to consider the national guidelines as you will not be attempting to integrate into an existing structure. You can start your Directory from the **o=MyOrganization** level.

You should use culturally meaningful names if you expect users to browse the Directory looking for an entry. For example, someone is much likely to recognize that **cn=John Jackson** is the entry required rather than **uid=jaj1234**. For the same reason, although abbreviated names may have an obvious meaning to one person, they can have a completely different meaning to someone else, even within the same organization. In many countries the best choice will be in the form of familiar-first-name and surname. Many organizations use the first and last name, with the option of a middle initial to distinguish between two persons with the same first and last names. Pragmatic choices will have to be made for other cultures.

The common name attribute should not be used to hold other attribute information such as telephone numbers, room numbers, or local codes.

If there are large numbers of entries located at a single level of the DIT, it may be necessary to adopt a more complicated naming scheme to guarantee uniqueness of the RDNs, as there may be two employees called “John Smith” who must have distinct entries. This can be solved using multi-component RDNs: the standard *X.521* recommends that an **organizationalUnitName** attribute can also be used as a naming attribute to differentiate entries, and the **personalTitle** or **userId** attributes could also be used in the RDN. Both of these are needed to identify this person’s entry. The RDN must distinguish the entry from all other entries with the same parent entry. For example combining common name and department distinguishes the two following entries:

cn=John Smith+ou=Sales

cn=John Smith+ou=Finance

The use of multi-component RDNs enables you to avoid artificial naming values such as “John Z Smith” or “John Smith-2”.

For **organization**, **organizationalUnit**, and **person** entries, extra naming information can be stored in the Directory as alternative values of the naming attribute. For example, if an organization is called Multi-National Network, MNN could be stored as an alternative **organizationName** attribute value.

A.2.4.2 Aliases

Aliases are used to point to another entry in the DIT. Aliases should only be used when you need to reference an entry in two separate naming contexts. If a person splits his or her work between two departments within an organization, for example, you may want to have two Directory entries for that person, assuming that your Directory entries were organized by department. One of these would correspond to the person’s “real” entry, and the other would be an alias that references that entry, and would enable that person to be found whichever department you expected to find him or her within.

Aliasing can also be used to reference an entry outside the scope of a single organization. For example, **cn=A N Other, o=Ownworkshop, c=GB** could be an alias to **cn=Andrew Other, ou=Engineering, o=Acme Limited, c=GB**.

A.3 Operational information

Operational information is held in a Directory to govern access to the user information described above. It comprises:

- Rules on how the user information is to be administered.
- Details of the distribution of user information over the network.

The same concepts apply to operational information as to user information, that is, object classes and attribute types are used to classify the information.

Operational information can be found in two places in the DIT:

- As additional attributes (operational attributes) within entries which also carry user information.
- As additional entries, created especially for the purpose of carrying operational information.

The set of rules governing the operational information is called the system schema.

A.3.1 Administration of the Directory

The way a Directory is administered and operated may be different in different areas of the Directory.

A DIT domain is that subset of the DIT held within a particular Directory Management Domain (DMD). A DIT domain is composed of one or more subtrees of the DIT, each of which is termed an Autonomous Administrative Area (AAA). The top of the subtree within one Autonomous Administrative Area is called the Autonomous Administrative Point (AAP).

Each entry can be located in only one Autonomous Administrative Area, and all entries in an AAA are controlled by the same administrative authority. As shown in [Figure A.2, “Administrative areas”](#), an AAA includes its Autonomous Administrative Point and all other entries below it, down to the leaves of the DIT or another AAP.

The Autonomous Administrative Point is used to define:

- access control administration for the area, see [Section A.3.1.2, “Security”](#).
- attributes which are common to all entries within the area, that is, collective attributes.

These definitions are held in subentries beneath the Administrative Point.

A.3.1.1 Administrative points

An administrative point is a location in the Directory tree at the top of an area with its own access controls and collective attributes. There are three types of administrative point: an autonomous administrative point, a specific administrative point and an inner administrative point.

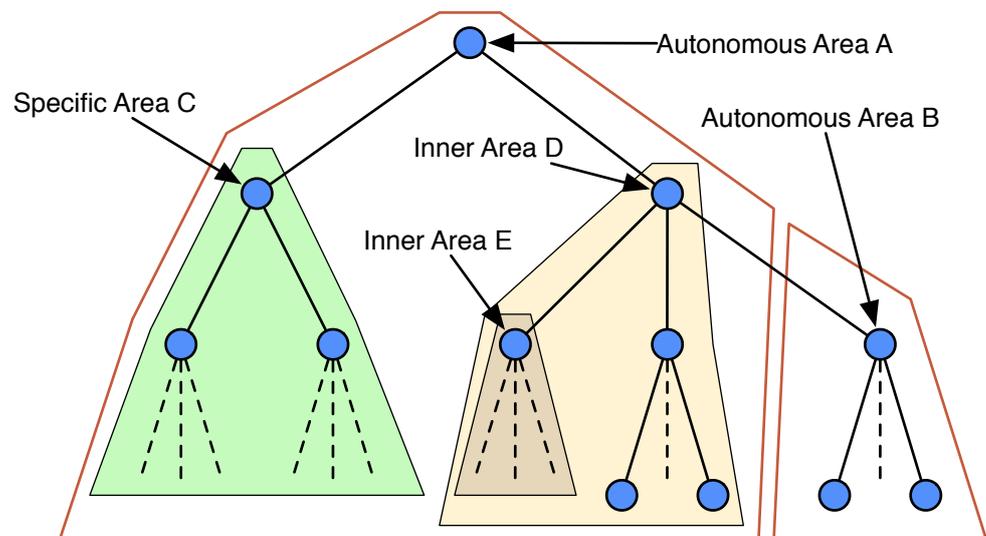
- An autonomous administrative point heads an autonomous administration area and contains the schema, access and collective attributes all the way down to the leaf entries, excluding any subtree with its own autonomous point. There can only be one autonomous administration point for a section of the tree.

- A specific administrative point enables you to define different security access or collective attributes for an area. Specific administrative areas cannot be nested inside other administrative areas but can themselves contain inner administrative areas.
- An inner administration point heads an area within an autonomous area, which can refine the access and collective attribute information. It extends down to lowest entries of the autonomous point that contains it. Inner administrative areas can be nested one inside another.

The difference between the types is shown in [Figure A.2, “Administrative areas”](#):

- Two autonomous administrative areas, with ‘A’ and ‘B’ as the administrative points at their apexes. None of the administrative properties of the area managed by ‘A’ are passed to the area managed by ‘B’.
- A specific administrative area, with ‘C’ as its administrative point. This area will use the schema defined by ‘A’, but may have completely different collective attribute details or access policies. For example, if this subtree reflected a physical location (an overseas office, perhaps) then some of the contact information held in collective attributes (such as the company telephone number) may be different. Alternatively, this subtree may reflect a department within the organization that requires either relaxed or enhanced security clearance, different from that set for the organization as a whole.
- Two inner administrative areas, headed by ‘D’ and ‘E’. These areas take the schema, collective attribute and security information from the containing administrative area and add to it. For example, some people may be given administrative roles within an inner area enabling them to modify entries; these same people will retain the access levels granted at the higher level for the rest of the Directory entries.

Figure A.2. Administrative areas



A.3.1.2 Security

There are three aspects of security provided by the Directory Service:

- authentication
- access control, and
- confidentiality.

A.3.1.2.1 Authentication

The steps taken by parties in an interaction to verify each others identity is called authentication.

Most applications where parties are connected by communications networks require some form of authentication. Usually, the requester of a service initiates the connection and presents a user identity and password (credentials) to the service provider. Access is only granted if the credentials are those agreed in advance with the service provider.

The purpose of authentication in a Directory is to support Access Control (see below).

Several different types of credentials may be used to establish identity, details of which are given in [Section 5.2.1, “Establishing identity”](#), but may be summarised as follows:

- no authentication, where no credentials are presented.
- simple authentication, where the Distinguished Name (DN) and a password are used. The DN given must correspond to the name of an entry in the Directory.
- SASL authentication, which uses userids instead of DNs, and supports various mechanisms which can avoid passing credentials over the network in the clear.
- strong authentication, where a digitally signed token is used.

Each is useful in certain circumstances.

A.3.1.2.2 Access control

Access control is a security service aimed at preventing unauthorized access to a capability.

Once an identity has been established (authenticated), the access control system determines what data and operations can be accessed by that identity. It is also possible to require that certain operations are digitally signed.

The purpose of access control is to protect entries, attributes and their values against unauthorized disclosure or modification. Access control regulates what type of operation can be performed on an entry and on an attribute or value.

More information on access control is given in [Chapter 6, *Controlling Access*](#).

A.3.2 Distribution of the Directory

A Directory can be distributed widely over many computer systems, linked together in a network. The user information held on these systems can be shared or duplicated in a flexible way, while also ensuring that the required security of access is maintained.

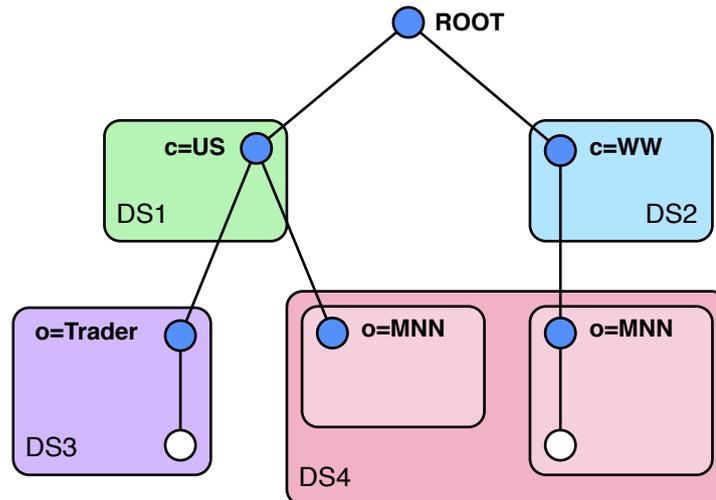
A.3.2.1 Directory Servers

A Directory can be composed of many Directory Servers containing copies of the application used to maintain and provide access to the Directory and defined parts of the DIT. In practice, this usually means that each Directory Server Application (DSA) is held on a separate machine in the network. Directory Server applications provide the mechanisms for access control, data management and system management.

A.3.2.1.1 The naming context

The DIT can be made up of a number of subtrees which do not overlap, each of which is provided by a different Directory Server. Each such subtree of the DIT is called a naming context. A single Directory Server may provide one or more naming contexts, as illustrated in [Figure A.3, “Example of Naming Contexts”](#).

A basic principle is that each entry in the DIT is mastered by a single Directory Server, which is the only Directory Server where that entry may be modified.

Figure A.3. Example of Naming Contexts

The administrator setting up a Directory Server defines its (initial) naming context(s) by specifying:

- The Distinguished Name of the entry in the DIT at the top of the naming context to be mastered, that is, the context prefix.
- If the naming context is not immediately below the root of the DIT, a reference to a DSA holding a naming context which is superior to this one, that is, a superior reference.
- References to any naming contexts below this one in the DIT of which this Directory Server should be aware, that is, subordinate references.

Note: In the M-Vault Server, a Directory Server's naming context may be split into several subordinate naming contexts. This is a relaxation of the standards, which state that for every naming context the superior naming context consists of entries mastered in a different Directory Server.

A naming context is not necessarily an administrative area (see [Section A.3.1, "Administration of the Directory"](#)). An administration may split the area among multiple adjoining naming contexts.

A.3.2.1.2 Subschema

Each Directory Server contains information to regulate the entries held by that Directory Server. This is called its subschema.

Subschema information comprises:

- Constraints on the hierarchical structure of the DIT, that is, permitted hierarchical relationships between object classes.
- For each object class, a specification of which attributes are mandatory and which are optional.
- The characteristics of each attribute in terms of attribute syntax and other operational attributes.

Thus, the subschema defines the shape of the DIT tree held by the Directory Server, what should be held in each entry and in what form it should be held.

A.3.3 Directory User Agent (DUA)

A Directory User Agent (DUA) is a client application used to connect to a Directory Server. The DUA provides the ability to issue operations to the Directory and process the results obtained.

The standards define only the protocol and service requirements of a DUA and do not define the “look-and-feel” of a DUA client. Thus, DUA client applications come in many varieties, though all follow the same basic mechanisms for accessing the Directory and have the same internal model for how information in the Directory is represented in terms of names, object classes, entries and attributes, as previously discussed.

A DUA communicates with the Directory using an access protocol. This can be:

- Lightweight Directory Access Protocol (LDAP),
- X.500 Directory Access Protocol (DAP).

The access protocol implements the Directory Abstract Service which provides the operations to access and update the Directory. Access operations include the ability to read an entry, list all entries subordinate to another entry, and search for entries based on user-specified search filters. Update operations include the ability to add an entry or attributes, remove an entry or attributes, and modify an entry or attributes.

A.3.3.1 Interactions between Directories and DUAs

The Directory Server can respond to requests from a DUA (over DAP or LDAP) or another Directory Server in one of the following ways, as shown in [Figure A.4, “Interactions between Directory Servers and a DUA”](#):

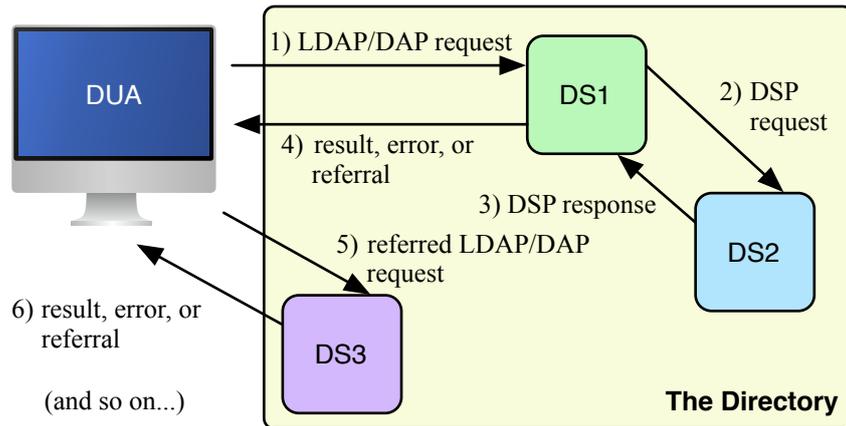
- If the Directory Server holds the information to satisfy the request, then that information is returned to the DUA.
- If the Directory Server does not hold the information, or has only part of the information, it can be configured to obtain the information from another Directory Server. This process is called chaining.
- Alternatively, if the Directory Server does not hold the information, it can be configured to return a referral to the DUA, which can then contact the referred alternative Directory Server for the information.

The configuration for chaining or referral can be set up for each Directory Server and as part of the service controls in the DUA. It is configured separately for requests passed over DAP and over LDAP.

Note: Only operations which read data, not modify it, can be chained over LDAP. The chaining is carried out with no authentication; see [Section A.3.1.2, “Security”](#).

The communication between Directory Servers is via the Directory Server Abstract Service and uses the Directory System Protocol (DSP); this protocol is used for inter-Directory Server communication unless the information is shadowed (see [Section A.3.4, “Shadowing”](#)).

If interaction between Directory Servers is required, each Directory Server needs operational information (knowledge) about the other Directory Server(s).

Figure A.4. Interactions between Directory Servers and a DUA

A.3.4 Shadowing

To minimise access times for Directory information, it may be necessary to ensure that the information is near the users who need it by means of replication. Replication means that several Directory Servers hold a copy of the same information.

Replication of information in the Directory is called shadowing. One Directory Server establishes a shadowing agreement with another, whereby the shadow supplier contracts to provide the shadow consumer with a copy of some part of the DIT. A single Directory Server may act in the role of a shadow supplier, a shadow consumer, or both, depending on the shadowing agreements between it and other Directory Servers.

Thus, many copies of a part of the DIT (a naming context or part of it) may be provided by a number of different Directory Servers. However, only one is considered to be the master copy. When the information changes, it is the master copy which is updated, and then the changes are propagated to the shadow copies, following the rules laid down in the various shadowing agreements.

Depending on the shadowing agreement, a Directory Server in a shadow consumer role may request an update from a shadow supplier Directory Server, and/or the shadow supplier may initiate the update. The update can take place at a defined frequency or on demand, and the actual update can be either incremental or total. An incremental update includes only those modifications since the last update. In both cases, a timestamp uniquely identifies the update transaction.

The protocol used for replication is the Directory Information Shadowing Protocol (DISP).

A.3.5 Directory Server information model

Each Directory Server must maintain an internal model of the information it holds, so that it can represent and manage its portion of the DIT. This internal model is primarily of interest to Directory Server administrators who will configure a Directory Server and populate its database with information.

A Directory Server requires information about how the DIT is organized. This is held in the form of Directory Server operational attributes, which comprise:

- Directory operational attributes, used to operate the Directory Server.
- Directory Server shared attributes, such as knowledge references, used by a Directory Server to contact other Directory Servers.
- Directory Server specific attributes which represent how the DIT is configured for that particular Directory Server.

A.3.5.1 Directory operational attributes

Directory operational attributes apply to the whole Directory and are mainly used to implement the Administrative Model, as described in [Section A.3.1, “Administration of the Directory”](#). They include collective attribute, authentication and access control information and also the Access Point for this Directory Server.

A.3.5.1.1 Access point

An Access Point uniquely identifies a Directory Server. The Access Point consists of the Distinguished Name (DN) of the Directory Server, which defines its location in the DIT, and its Presentation Address (PA), which defines its location on the network.

Note: The Presentation Address is mandatory for Access Points in the M-Vault Server.

A.3.5.2 Directory Server shared attributes

Directory Servers need a way of progressing operations which reference entries outside their naming context(s). Knowledge References are used for this purpose.

- A subordinate reference is the access point of the Directory Server which masters a naming context directly subordinate to the naming context of this Directory Server.
- A cross reference is the access point of the Directory Server mastering a naming context which is not directly subordinate to the one mastered by this Directory Server.
- A supplier reference is held by a shadow consumer Directory Server. It contains the access point of the supplying Directory Server and, optionally, the access point of the master Directory Server, if the updates are not supplied by the master.
- A superior reference is required for all Directory Servers which do not provide a first level naming context. It is the access point of the Directory Server to contact if there is no other suitable reference to progress the operation.

A.3.5.3 Directory Server’s information tree

A Directory Server’s Information Tree (DsaIT) contains all the names in the DIT known to that Directory Server.

If a single Directory Server were providing the whole Directory, all entries would be in its Information Tree. Typically, however, the individual Directory Server’s Information Tree is a very restricted subset of the total DIT.

If a Directory Server does not master any entries that are immediate subordinates of the root, then it needs a reference to the Directory Server that holds a superior naming context (see [Section A.3.2.1, “Directory Servers”](#)). The Directory Server that masters a superior naming context will be able to chain or refer any requests toward a first level Directory Server. A first level Directory Server is one which holds knowledge of all naming contexts that are immediate subordinates of the root.

At each name in the Directory Server’s Information Tree, the Directory Server holds a private internal object, the Directory Server Specific Entry (DSE), which holds the attributes specific to the entry. Directory Server shared and Directory Server specific attributes are kept separate from Directory operational attributes and the user information.

A.4 Functionality of M-Vault

The functionality offered by the M-Vault Server includes:

- Support of the Directory Access Protocol (DAP) as defined in the X.500 (2008) standard.
- Support of the LDAPv3 protocol defined by *RFC 4511*.
- Support of the Directory System Protocol (DSP) as defined in the X.500 (2008) standard, including chaining.
- Schema support of all attribute types and object classes as defined in X.520 (2008) and X.521 (2008) as well as those defined in *RFC 4519* and *RFC 4524*.
- Support of the bind operation using no authentication, simple authentication and strong authentication for DAP, DSP and DISP as defined in the 2008 standards.
- Support for signed DAP, DSP and DISP operations.
- Support of the SASL authentication framework as defined in *RFC 4422* for use with LDAP as defined in *RFC 4513*.
- Support of a subset of X.500 (2008) defined Basic Access Control.
- Replication using X.500 (2008) DISP including support for incremental and full shadow updates, supplier and consumer initiated, scheduled and on-change updates, attribute filtering and chop shadowing.
- Provision of the Directory Abstract Service in a distributed environment as defined in X.518 (2008), including support of chaining and referral modes of operation, superior references, and subordinate references.
- Configuration of the Directory Server using standard knowledge references.
- Indexing and database tools for the management of on-disk databases.
- Audit logging.

Detailed references to the individual standards are given in [Appendix I, References](#).

A.4.1 Isode's interpretation of the standards

Although the M-Vault Server supports the standards as listed above, there are some methods of interpretation which are worth mentioning. These are detailed in the sections below.

A.4.1.1 Storage of and access to entries

The X.500 (2008) specifications require that Directory Servers have some means of accessing the attributes of Directory Server Specific Entries (DSEs), and that overlapping information shadowed by two different sources is kept distinct.

The M-Vault Server architecture contains a two-part solution to the problem of entry storage and access:

- Ordinary (user information) entries are stored on disk in one or more databases.
- Directory Server specific attributes are held in memory as an internal Directory Server knowledge information tree. The in-memory database is termed the Directory Server Information Tree.

The M-Vault Server relies on this Directory Server Information Tree for efficient name resolution, pre-filtering, access control checks, and operation progressing.

The M-Vault Server is a single multi-threaded process on all supported platforms. On Windows systems it runs as a Windows Service. On Unix systems it runs as a system daemon.

A.4.1.2 Operating system support

The M-Vault Server is a single multi-threaded process on all supported platforms. On Windows systems it runs as a Windows Service. On Unix systems it runs as a system daemon.

A.4.1.3 Replication

The M-Vault Server implements both the supplier and consumer sides of the Directory Information Shadowing Protocol (DISP), which allows the replication of a naming context from one Directory Server to another.

The Shadow Update Scheduler module transmits shadow updates of GDAM-held naming contexts at manager-configurable intervals, either when an entry in the context has been modified, at specific times, or when forced by the Directory Service Manager. The shadow update can be either a total update which contains every entry in the naming context, or an incremental update which includes only those entries which have changed since the last update.

The database GDAM includes a change logging mechanism, so that changes to that GDAM can be found and transmitted efficiently.

A.4.1.4 The use of Autonomous Administrative Areas

The X.500 standards state the following restriction:

An Autonomous Administrative Area (AAA) controlled by an authority cannot be immediately subordinate to another AAA controlled by the same authority.

However, this restriction has been relaxed in the M-Vault Server to enable the shadowing of part(s) of a naming context. The parts to be shadowed are defined as AAAs, which means that, in the case of the M-Vault Server, an AAA can be immediately subordinate to another controlled by the same authority. It is assumed that a naming context is an access control and a collective attribute area.

A.4.1.5 The use of Superior Naming Contexts

The X.500 standards state:

For every naming context the superior naming context consists of entries mastered in a different Directory Server.

In the M-Vault Server, a naming context may be split into several subordinate naming contexts to allow for the shadowing of part(s) of the naming context. Thus, the superior naming context may consist of entries mastered in the same Directory Server.

Appendix B Attributes

This appendix provides details of the attributes associated with some of the common object classes, indicating which are mandatory and which are optional.

B.1 Sample attributes of object classes

This section contains extracts of sample entries which illustrate the points made in the text, using the LDIF format.

Attribute type keywords do not have to follow the mixed-case conventions shown in the examples because keywords map, ignoring case, onto the numeric object identifiers used internally. The case of attribute values will be preserved as entered initially, but can usually be matched case-independently. It is not possible to modify the case of an attribute value by DAP without removing the old value and reinstating the modified value.

The remainder of this section gives examples of entries for organizations, persons, and other White Pages objects.

Note: All object classes have a mandatory attribute **objectClass** that specifies what type of object it is. This is in addition to the specified mandatory attributes listed below.

B.1.1 Country

Objects of this class represent a geographic entity.

- There is one mandatory attribute: **countryName**, which gives the name of the country.

The value of this attribute is a printable string exactly two characters long; for example:

```
c: US
```

This attribute is single-valued, and the value is restricted to be an ISO 3166 country code.

- The **friendlyCountry** subclass of country used in *RFC 4524* adds a mandatory attribute, **friendlyCountryName**, which gives the name of the country in a human-readable form. The value of this attribute is a string, for example,

```
friendlyCountryName: The United States of America
```

This attribute is multi-valued. It would not be used as the naming attribute, but is helpful when displaying a country entry.

- Optional attributes include **description** and **searchGuide**.

B.1.2 Organization

Objects of this class represent a top-level organizational entity, such as a corporation, university, government entity, and so on.

- There is one mandatory attribute, **organizationName**, which gives the name of the organization and is multi-valued. The value of this attribute is a string, for example:

```
o: Multi-National Network
```

- There are a number of optional attributes for holding contact and similar information: **description**, **localityName**, **stateOrProvinceName**, **streetAddress**, **physicalDeliveryOfficeName**, **postalAddress**, **postalCode**, **postOfficeBox**, **facsimileTelephoneNumber**, **internationalISDNNumber**, **telephoneNumber**, **teletexTerminalIdentifier**, **telexNumber**, **preferredDeliveryMethod**, **destinationIndicator**, **registeredAddress**, **businessCategory**, **seeAlso**, **searchGuide**, **userPassword**

An example of an organization entry is:

```
dn: o=Multi-National Network
objectClass: organization
objectClass: top
o: Multi-National Network
o: MNN
telephoneNumber: +44 20 8783 2964
businessCategory: Telecommunications
l: Great Britain
l: Canada
l: United States
```

B.1.3 Organizational unit

The **organizationalUnit** object class is used to represent a unit within an organization.

- There is one mandatory attribute, **organizationalUnitName**, which gives the name of the organizational unit. The value of this attribute is a string, for example:

```
ou: Research and Development
```

- Many of the optional attributes are the same as those for **organization**.

An example of an organizational unit entry would be:

```
dn: ou=Marketing,o=Multi-National Network
objectClass: organizationalUnit
objectClass: top
ou: Marketing
businessCategory: Marketing
l: Second Floor
st: Surrey
```

B.1.4 Person

This is a base object class used to represent a person.

- There are two mandatory attributes:
 - **commonName**, which gives a (potentially ambiguous) name for the person. The value of this attribute is a string usually containing the person's first and last names, for example, Steve Kille This attribute is usually multivalued, containing variations on the first, middle, and last names, for example, Stephen E. Kille Steve E. Kille Stephen Kille

- **surname**, which gives the person's last name. In cultures that do not have a distinct surname the common name used in the RDN could also be placed as a value in this attribute.

B.1.4.1 Organizational person and inetOrgPerson

These are auxiliary classes of the **person** object class.

- **organizationalPerson** introduces the following optional attributes:

preferredDeliveryMethod, destinationIndicator, registeredAddress, internationalISDNNumber, x121Address, facsimileTelephoneNumber, teletexTerminalIdentifier, telexNumber, physicalDeliveryOfficeName, postOfficeBox, postalCode, postalAddress, title, organizationalUnitName, streetAddress, stateOrProvinceName, locality

- **inetOrgPerson** adds the following optional attributes to those found in **organizationalPerson**:

audio, businessCategory, carLicense, departmentNumber, displayName, employeeNumber, employeeType, givenName, homePhone, homePostalAddress, initials, jpegPhoto, labeledURI, mail, manager, mobile, organizationName, pager, photo, roomNumber, secretary, uid, userCertificate, x500UniqueIdentifier, preferredLanguage, userSMIMECertificate, userPKCS12

An example person entry would be:

```
dn: cn=Bill Smith,ou=Marketing,o=Multi-National Network
objectClass: inetOrgPerson
objectClass: organizationalPerson
objectClass: person
objectClass: top
cn: Bill Smith
cn: William Smith
mail: B.Smith@research.mnn.com
roomNumber: 123
sn: Smith
telephoneNumber: +44 20 8783 2964
title: Manager
```

B.1.5 Organizational Role

Entries of this class are used to represent a position or role within an organization.

- There is one mandatory attribute, **commonName**, which gives the name of the role. The value of this attribute is a string, for example:

```
cn: postmaster
```

- There are many optional attributes including **roleOccupant** (the distinguished name of the person who fulfils the role), for example:

```
roleOccupant: cn=Bill Smith,ou=Research,o=Multi-National Network
```

The other optional attributes are: **seeAlso, preferredDeliveryMethod, destinationIndicator, registeredAddress, internationalISDNNumber, x121Address, facsimileTelephoneNumber, teletexTerminalIdentifier, telexNumber, telephoneNumber, locality, postOfficeBox, postalCode, postalAddress, description, organizationalUnitName, streetAddress, stateOrProvinceName, physicalDeliveryOfficeName**

For example:

```
dn: cn=Secretary,ou=Marketing,o=Multi-National Network
objectClass: organizationalRole
objectClass: top
cn: Secretary
roleOccupant: sn=Jones,o=Multi-National Network
telephoneNumber: +44 20 8783 2964
```

It is important to remember the distinction between aliases and roles when selecting the attributes of an **organizationalRole** entry. For example if the role was departmental fire safety representative, it would be likely to be filled by a succession of different individuals with different phone numbers. In this case, phone number should not be an attribute of the role entry.

B.1.6 Group of Names

Entries of the object class **groupOfNames** define a set of names.

There are two mandatory attributes:

- **commonName**, which is the name of the group
- **member**, which is multi-valued and identifies members of the group – it must contain at least one value

For example:

```
dn: cn=Managers,ou=Marketing,o=Multi-National Network
objectClass: groupOfNames
objectClass: top
cn: Managers
member: cn=Bill Smith,ou=North,o=Multi-National Network
member: cn=John Bailey,ou=West,o=Multi-National Network
```

B.1.7 Alias

Objects of this class represent an alias to some other entry in the DIT. It is generally used when an entity belongs in more than one subtree of the DIT, and is used to “point” one entry to the other.

There is one mandatory attribute, **aliasedObjectName**, which is a pointer to another object in the Directory. For example:

```
aliasedObjectName: cn=Bill Smith,ou=Marketing,o=Multi-National Network
```

Note: An aliased object must include the attribute, or attributes, used in the Relative Distinguished Name. This attribute should usually also be present in the aliased object.

B.1.8 Domain related object

If an object has some relationship to the Internet Domain Name System (DNS), then this can be represented in the DIT using this auxiliary object class, which would normally be added only to **organization** or **applicationProcess** entries.

This class has one mandatory attribute, **associatedDomain**, which identifies the domain which corresponds to this object. The value is a domain string, for example:

```
associatedDomain: mnn.com
```

B.1.9 LabeledURI object

This auxiliary object class has no mandatory attributes, but it has two optional attributes:

- **labeledURI**, which is used to hold a Uniform Resource Identifier (see *RFC 3986*) together with an associated text label, for example:

```
labeledURI: http://www.mnn.com/ Multi-National Network Limited home page
```

- **labeledURL**, which is used to hold a Uniform Resource Locator together with an associated text label, for example:

```
labeledURL: http://www.mnn.com/ Multi-National Network Limited home page
```

B.2 Extending the schema

If there is a local requirement to hold information that does not map to existing object class or attribute definitions, then it may be necessary to extend the schema used by the Directory Service.

The Directory Server and DAP DUAs read the optional (*ETCDIR*)/*dsaptailor* and (*SHAREDIR*)/*dsaptailor* files on start-up. (An example file is supplied which may be used as the basis of an actual *dsaptailor* file if required.)

Each *dsaptailor* file entry has the form:

```
<key> <value> [<value> ...]
```

where *<key>* is the entry identifier.

All comment lines must be preceded by a #.

The first non-comment line of the *dsaptailor* file must be to configure the Object Identifier (OID) definition tables, giving the files containing the allowed object classes and attributes. For example:

```
oidtable oidtable
```

will direct the Directory Server and DAP DUAs to consult the files *oidtable.gen*, *oidtable.at*, *oidtable.oc*, to obtain the string of OID mappings needed for operating the Directory. If the value is not an absolute path (such as */opt/isode/share/oidtable*), they will attempt to find these files in the (*ETCDIR*) directory, and if not present there, in the (*SHAREDIR*) directory.

There can be multiple values for *oidtable*, so local schema additions can be kept in local files. For example:

```
oidtable    oidtable local
```

will direct the Directory Server and DAP DUAs to consult the files *oidtable.gen*, *oidtable.at*, *oidtable.oc*, *local.gen*, *local.at*, and *local.oc*.

In the absence of a *dsaptailor* file, the Directory Server and DAP DUAs will use a default *oidtable* value of "oidtable mbox atn military".

Note: A fatal error will occur if all three of a set of files is missing (i.e. *oidtable.** or *local.**).

It is strongly recommended that any local schema definitions are made in separate files (e.g. *local.gen*, *local.at* and *local.oc*), as this will simplify upgrades to future versions of M-Vault.

Once a set of additional filenames has been chosen and configured in *dsaptailor*, the actual schema extensions can be entered.

B.2.1 *.gen files

The *.*gen* files contain mappings from OIDs to names used in the *.*at* and *.*oc* files. This can be used to simplify the entries in the other files. The *local.gen* file would usually contain references to the OID arcs allocated to the organization for use within their Directory Service. The format of *.*gen* files is described at the top of *oidtable.gen*.

If for example an organization had been allocated an OID by the IANA, then they might have a *local.gen* file containing:

```
# Start of local definitions
example-org: enterprises.123456
example-org-at: example-org.1
example-org-oc: example-org.2
# End of local definitions
```

B.2.2 *.at files

Any additionally required attributes should be defined in *.*at* files. The format of *.*at* files is described at the top of *oidtable.at*.

Note: Every attribute must be defined with an attribute syntax; for a description of commonly used attribute syntaxes see [Appendix C, Attribute Syntaxes](#).

The organization may for example have a *local.at* file containing:

```
# Start of local definitions
imapQuota: example-org-at.1: Integer: single-value
webQuota: example-org-at.2: Integer: single-value
webPath: example-org-at.3: CaseIgnoreString: \
    single-value
# End of local definitions
```

B.2.3 *.oc files

Any additionally required object classes should be defined in *.*oc* files. The format of *.*oc* files is described at the top of *oidtable.oc*.

These object classes could reference standard attributes, or the locally defined attributes, or a combination. The organization may for example require a *local.oc* file containing:

```
# Start of local definitions
customer: example-org-oc.1: top: : \
    imapQuota, webQuota, webPath : kind=auxiliary
```

In other words, **customer** is a subclass of **top**, and has three optional attributes. It is an auxiliary object class, and could be used together in an entry with **person**, or **inetOrgPerson**.

After restarting the Directory Server and all DAP DUAs, the new schema definitions should be available for use.

B.2.4 Customising DUAs

Further changes may be required in DUAs to allow them to recognise and utilise the custom schema. Often, LDAP-based DUAs will be able to read entries using the custom schema without any modification.

However it can still be useful to customise DUAs so that they display the custom schema in a useful way. To customise Sodium, the ADUA provided with M-Vault, please see [Appendix D, Customising Sodium](#).

Appendix C Attribute Syntaxes

Attribute values have an internal structure described by their syntax. When communicated over LDAP, or displayed in user agents like Sodium and M-Vault Console, the string representations associated with those syntaxes are used. This appendix describes all the currently recognized syntaxes and their LDAP string representations.

The string representations described in this appendix are also used in Sodium when displaying, modifying or adding entries. For most of the string representations, a BNF description is given using the following base descriptions:

```

ALPHA           = (any upper or lower case IA5 character)
DIGIT           = (any digit)
LDIGIT         = (digits 1 to 9)
PrintableCharacter = ALPHA / DIGIT / "'" / "(" / ")" /
                  "+" / "," / "-" / "." / "=" /
                  "/" / ":" / "?" / " "
Octet           = (any octet)
TeletexCharacter = (any TeletexString character, see below)
BMPCharacter    = (any UCS-2 encoded Unicode character)
UniversalCharacter = (any UCS-4 encoded Unicode character)
UTF8Character   = (any UTF-8 encoded Unicode character)

```

C.1 Character sets and matching rules

The Directory Server supports all five of the Directory string character sets: **PrintableString**, **TeletexString**, **BMPString** (Unicode), **UniversalString** (4 octet Unicode) and **UTF8String** (Unicode, but encoded compactly).

C.1.1 PrintableString characters

The printable string characters are the letters, numbers and selected symbols: apostrophe, left and right parenthesis, plus sign, comma, hyphen, period, equals, solidus (forward slash), colon, question mark and space.

C.1.2 TeletexString characters

The following character sets are supported:

Name	Description
ISO-IR-6	Similar to ASCII
ISO-IR-87	Multicode CJK
ISO-IR-102	Similar to ACSCII
ISO-IR-103	Accent modifiers
ISO-IR-106	Control characters (C0)
ISO-IR-107	Control characters (C1)
ISO-IR-126	Greek (ISO-8859-7)
ISO-IR-144	Cyrillic (ISO-8859-5)
ISO-IR-150	Greek (CCITT)
ISO-IR-153	Cyrillic (GOST-19768-74)

Name	Description
ISO-IR-156	Accent modifiers

The initial character set invocations are described in this section, as they form the default.

Name	Code range (hexadecimal)
ISO-IR-106	00-1F
ISO-IR-102	20-7F
ISO-IR-107	80-9F
ISO-IR-103	A0-FF

C.1.3 BMPStrings, UniversalStrings and UTF8Strings

BMPStrings are the same as Unicode (which is a UCS-2 encoding of ISO-10646). Universal strings are 4 byte Unicode (which is a UCS-4 encoding of ISO-10646), but as yet has not extended the character set. UTF8 strings are also Unicode, but encoded more compactly.

C.1.4 Matching rules

An important part of the Directory Server is the ability to search for specified attributes. The most common type of matching is equality: that a presented attribute type and value is the same as a value of that attribute type in an entry. Remember that a large number of the standard attributes match case insensitively; for Directory strings, the match is also independent of the character set; for example “tim” will match “TIM”.

For many string syntaxes an approximate match is also supported; however, the approximate match method used is Directory Server-dependent. A Directory Server can be configured to use either a soundex-based algorithm or a metaphone-based algorithm. Both algorithms work by grouping similar-sounding characters into classes. For soundex, the classes and their corresponding characters are shown in the table below. The first character of a word is always used as the first character of its corresponding soundex code. Adjacent similar characters are ignored. Thus, the word “Robens” has soundex code “R152”. Since the word “Robbins” also has soundex code “R152”, these two words are approximately equal. To match multiple words each of the target words must appear in order in the string to which it is being compared. There may, however, be other items in between the words matched. For example: “Tim Howes” would match “Timothy Alan Howes” since “Tim” matches “Timothy,” “Howes” matches “Howes,” and the matched words are in the proper order.

Table C.1. Soundex character classes

Characters	Soundex class
BFPV	1
SCGJKQXZ	2
DT	3
L	4
MN	5
R	6
all others	ignored

C.2 Common attribute syntaxes

This section describes the syntaxes and string representations used by commonly defined attributes

C.2.1 ASN

No LDAP string representation is defined for this syntax.

C.2.2 Audio

This syntax is used to represent μ -law encoded audio. This format is normally used in files with `.au` extensions.

No human-readable LDAP string representation is defined for this syntax. The octets in the raw file are used as-is in the attribute value.

C.2.3 BitString

```
Value = " " *( "0" / "1" ) "'B"
```

Also note other syntaxes are available which allow named bits to be set in a bit string. For example, see [Section C.3.1, "DSEType"](#).

C.2.4 Boolean

The value represents true or false.

```
Value = "TRUE" / "FALSE"
```

C.2.5 CaseExactString/CaseIgnoreString

```
Value = 1*UTF8Character
```

This syntax represents the **DirectoryString** syntax, which allows **PrintableString**, **TeletexString**, **BMPString**, **UniversalString**, and **UTF8String** values. However the LDAP string representation automatically converts all strings to UTF-8.

The **CaseExactString** variant uses case-sensitive matching, and the **CaseIgnoreString** variant uses case-insensitive matching.

C.2.6 CaseIgnoreList

The **CaseIgnoreList** syntax consists of a sequence of **CaseIgnoreString** values.

```
Value = line *( "$" line )
line = (see CaseIgnoreString syntax)
```

The line values require that any "\$" and "\" characters be escaped i.e. written as `\24` and `\5c` respectively.

Only equality matching is supported for this syntax.

C.2.7 CountryString

```
Value = 2PrintableCharacter
```

This syntax is treated as a **PrintableString**, with case-insensitive matching rules. Note the string must be two letters and one of the country codes defined by ISO 3166. For example:

```
c: GB
```

C.2.8 DeliveryMethod

This defines the priority order when communicating with an object.

```
Value = pdm *( "$" pdm )
pdm   = "any" / "mhs" / "physical" /
       "telex" / "teletex" / "g3fax" /
       "g4fax" / "ia5" / "videotex" /
       "telephone"
```

For example:

```
preferredDeliveryMethod: telephone$videotex
```

C.2.9 DestinationString

This syntax is used to define the addressee as required by the Public Telegram Service.

```
Value = 1*PrintableCharacter
```

Case-insensitive matching rules are used.

C.2.10 DN

The DN syntax represents the distinguished name of an entry which may or may not exist.

```
Value = [ rdn *( "," rdn ) ]
rdn   = ava *( "+" ava )
ava   = attrname "=" attrstring /      ; normal form
       oid "=" "#" attrberhex
```

Two forms of attribute value assertion are defined: the normal form uses an attribute name such as **cn** and a string representation of the attribute value. The string representation used here requires that any “,” “+” or “\” characters be escaped i.e. written as \, , \+, \\\, or \2c, \2b, \5c respectively.

The other form of attribute value assertion uses the OID of the attribute type and the BER encoding (converted to hexadecimal) of the underlying attribute value.

For example:

```
dn: o=Isode,c=GB
dn: cn=Legal Eagle,o=Sue\, Grabbit and Runne,c=GB
dn: 2.5.4.2=#13054c6567616c,dc=example,dc=com
```

C.2.11 FacsimileTelephoneNumber

This syntax represents the telephone number and parameters associated with a fax terminal.

```
Value          = telephone-number *( "$" fax-parameter )
telephone-number = PrintableString
fax-parameter  = "twoDimensional" / "fineResolution" /
                "unlimitedLength" / "b4Length" /
                "a3Width" / "b4Width" /
                "uncompressed"
```

For example:

```
facsimileTelephoneNumber: +44 602 123 4567$twoDimensional
```

C.2.12 GeneralizedTime

The value represents a time and date with a 4-digit year.

```
Value          = YYYY MM DD hh
                [ mm [ second ] ] [ fraction ]
                offset
fraction       = ( "." / "," ) 1*DIGIT
offset        = "Z" / positive / negative
positive      = "+" hh [ mm ]
negative      = "-" hh [ mm ]
YYYY         = 4DIGIT ; year
MM           = 2DIGIT ; month 01 - 12
DD           = 2DIGIT ; day 01 - 31
hh           = 2DIGIT ; hour 00 - 23
mm           = 2DIGIT ; minutes 00 - 59
second       = 2DIGIT ; seconds 00 - 60 (leap second)
```

For example the string 200412161032Z is used to represent 10:32 at UTC, on December 16th, 2004.

C.2.13 IA5String/CaselnoreIA5String

```
IA5Character = (any IA5 character)
Value        = 1*IA5Character
```

This syntax is handled as **PrintableString**, except a wider range of characters are recognized, i.e., any character in IA5 string. The **IA5String** variant uses case-sensitive matching and the **CaselnoreIA5String** variant uses case-insensitive matching. For example:

```
mail: info@isode.com
```

C.2.14 Integer

The value represents a positive, zero, or negative integer.

```
Value = ( "-" LDIGIT *DIGIT ) / number
number = DIGIT / ( LDIGIT 1*DIGIT )
```

For example:

```
adminSizeLimit: 123456
```

C.2.15 JPEG

This syntax is used to represent JPEG images, encoded using either the “JPEG File Interchange Format” (JFIF), or the “Exchangeable Image File Format” (Exif). Both formats are commonly used in files with *.jpg* extensions.

No human-readable LDAP string representation is defined for this syntax. The octets in the raw file are used as-is in the attribute value.

C.2.16 Mailbox

This syntax is used to hold values for non-X.400 and non-Internet email addresses.

```
Value = 1*PrintableCharacter "$" 1*IA5Character
```

For example:

```
otherMailbox: bmail $ info%company
```

C.2.17 NameAndOptionalUID

The value represents a distinguished name together with an optional bit string used for disambiguation.

```
Value      = dn [ "#" bitstring ]
dn         = (see DN syntax)
bitstring  = (see BitString syntax)
```

For example:

```
uniqueMember: cn=John,ou=Staff,o=Company
uniqueMember: cn=John,ou=Staff,o=Company#'1001'B
```

C.2.18 NisBootParameter

```
Value = key "=" server ":" path
key   = (See PrintableString syntax)
server = (See PrintableString syntax)
path  = (See PrintableString syntax)
```

C.2.19 NisNetgroupTriple

```
Value      = "(" hostname "," username "," domainname ")"
hostname   = "" / "-" / p
username   = "" / "-" / p
domainname = "" / "-" / p
p         = (See PrintableString syntax)
```

C.2.20 NisPublicOrSecretKey

```
Value      = "{" keytype "-" keylength "-" algorithm "}" key
keytype    = (See PrintableString syntax)
keylength  = (See PrintableString syntax)
algorithm  = (See PrintableString syntax)
key        = (See PrintableString syntax)
```

C.2.21 NumericString

```
Value = 1*DIGIT
```

The value is a string of digits (0 through 9 only).

C.2.22 ObjectClass

Although essentially an object identifier, a separate syntax is provided as the identifiers have additional semantics when used as an object class.

```
Value = oid / name
oid    = number 1*( "." number )
number = DIGIT / ( LDIGIT 1*DIGIT )
```

For example:

```
objectClass: top
objectClass: person
```

C.2.23 OctetString

```
Value = 1*Octet
```

This syntax represents arbitrary octets, not a text string.

C.2.24 OID

The value in this syntax is an object identifier, i.e., a dotted series of non-negative integers. Any attribute or label defined in the schema files is also a valid value.

```
Value = oid / name
oid    = number 1*( "." number )
number = DIGIT / ( LDIGIT 1*DIGIT )
```

For example:

```
pwdAttribute: userPassword
```

C.2.25 Password/EncryptedPassword

```
Value = 1*Octet
```

Values using the **Password** syntax may be hashed when stored in the GDAMs if the Directory Server has password hashing enabled, see [Section 5.6.3, “Storing passwords in the GDAM”](#). The resulting values then have a string representation:

```
Value           = schemeprefix hashedpassword
schemeprefix    = "{" scheme "}"
scheme          = "crypt" / "md5" / "sha" / "sha2" /
                 "smd5" / "ssha" / "ssha2" / "scram-sha-1"
hashedpassword  = (encoded password octets)
```

The following schemes simply base-64 encode the hashed password values, as per *RFC 2307*:

- md5 – Unsalted MD5
- sha – Unsalted SHA1
- sha2 – Unsalted SHA2
- smd5 – Salted MD5
- ssha – Salted SHA1
- ssha2 – Salted SHA2

The *scram-sha-1* scheme uses the format described in *RFC 5803*. For example:

```
userPassword: secret
userPassword: {CRYPT}50PM1cfj3zSZg
userPassword: {SSHA2}Y3+1weG3ObqWS7eEd4NwxiMhElHfA50yXcJOYTMWESF
Zl3Tv
userPassword: : e1NDUkFNLVNIQS0xfTQwOTY6NHNGTG9UVzc1UFA0MkFtQ1cyN
mw1dz09JGQ3QnplbTlnWVQ5ZTRwamL2aUxIKzN2bvJubz06eG5Talk4V1NBQTZa
TmlCOWdBb1AyTnI5RXBvPQ==
```

Note: The use of `crypt`, `md5`, `sha`, `sha2`, `smd5`, `ssha`, `ssha2` or `scram-sha-1` mechanisms with this syntax changes the behavior of some Directory operations in some possibly unexpected ways.

The **compare**, **bind**, **modify**, and **search** operations behave asymmetrically with values of this syntax: if one of the values is hashed, the other value used in the operation must be the plaintext value. For example, if the stored password is hashed using MD5, the password in the bind operation must be cleartext. If the stored password is not hashed (i.e. using `plain`) then the password in the bind operation may be hashed using any of the other supported hash mechanisms.

This asymmetry is required to prevent an attacker from reusing passwords that have been read from network packets, or otherwise read from the Directory.

The **add** and **modify** operations will hash any plaintext values that the DSA is sent before storing them in the GDAM. Any passwords that the user has already hashed correctly (using the above listed mechanisms) will be stored in the GDAM as is.

C.2.26 Photo

This syntax is used to represent G3 Fax images. No LDAP string representation is defined, and values should be transferred using `binary`.

C.2.27 PostalAddress

The **PostalAddress** syntax consists of a sequence of up to 6 **CaseIgnoreStrings** each of up to 30 characters.

```
Value = addressline *5( "$" addressline )
```

The line values require that any “\$” and “\” characters be escaped i.e. written as \24 and \5c respectively.

Only equality matching is supported for this syntax.

For example:

```
postalAddress: 36 Station Road$Hampton$Middlesex
```

C.2.28 PresentationAddress

```
Value = [[[ psel "/" ] ssel "/" ] tsel "/" ] 1*naddr
psel = [ "" IA5String "" / "'" hexstring "H" ]
ssel = [ "" IA5String "" / "'" hexstring "H" ]
tsel = [ "" IA5String "" / "'" hexstring "H" ]
naddr = (network address string)
```

For example:

```
presentationAddress: "X500"/URI+0000+URL+itot://server.example.com:19999|URI+0000+URL+ldap://server.example.com
```

C.2.29 PrintableString/CaselgnorePrintableString

```
Value = 1*PrintableCharacter
```

The value can be any character listed for **PrintableString**. The **PrintableString** variant uses case-sensitive matching, and the **CaselgnorePrintableString** variant uses case-insensitive matching.

C.2.30 TelephoneNumber

```
Value = 1*PrintableCharacter
```

Values are **PrintableStrings**, but are expected to be formatted as international telephone numbers as per *E.123* i.e.

```
Value      = "+" country national [ ext extension ]
country    = 1*DIGIT
national   = 1*( DIGIT / " " )
ext        = 1*( ALPHA / " " ) ; eg "x" or "ext"
extension  = 1*DIGIT
```

For example:

```
telephoneNumber: +22 607 123 4567
telephoneNumber: +1 302 123 4567 x876
```

Matching is case-insensitive, except that all space and “-” characters are skipped during the comparison.

C.2.31 TelexNumber

```
Value      = number "$" countrycode "$" answerback
number     = PrintableString
countrycode = PrintableString
answerback = PrintableString
```

For example:

```
telexNumber: 12345$G$PHYS
```

C.2.32 UTCTime

The value represents a time and date with a 2-digit year. Use of **GeneralizedTime** is recommended to avoid ambiguities with 2-digit years.

```
Value      = YY MM DD hh mm [ ss ] offset
offset     = "Z" / positive / negative
positive   = "+" hh mm
negative   = "-" hh mm
YY         = 2DIGIT ; two low order digits of the year
MM         = 2DIGIT ; month 01 - 12
DD         = 2DIGIT ; day 01 - 31
hh         = 2DIGIT ; hour 00 - 23
mm         = 2DIGIT ; minutes 00 - 59
ss         = 2DIGIT ; seconds 00 - 59
```

For example the string 980602093221Z is used to represent 09:32:21 at UTC, on June 2nd, 1998.

C.2.33 UUID

This syntax is used to represent universally unique identifiers (UUIDs).

```
Value      = time-low "-" time-mid "-"
              time-high-and-version "-"
              clock-seq-and-reserved
              clock-seq-low "-" node
time-low   = 4hexoctet
time-mid   = 2hexoctet
time-high-and-version = 2hexoctet
clock-seq-and-reserved = hexoctet
clock-seq-low = hexoctet
node       = 6hexoctet
hexoctet   = 2hexdigit
hexdigit   = ( hex digit 0 - 9, A - F, a - f )
```

For example:

```
entryUUID: f81d4fae-7dec-11d0-a765-00a0c91e6bf6
```

C.3 X.500 operational attribute syntaxes

C.3.1 DSEType

This is a named bit string.

```
Value = (" dsebit *( "$" dsebit ) )"
dsebit = "root" / "glue" / "cp" / "entry" /
         "alias" / "subr" / "nssr" / "supr" /
         "xr" / "admPoint" / "subentry" / "shadow" /
         "zombie" / "immSupr" / "rhob" / "sa" /
         "dsSubentry" / "str"
```

For example:

```
dseType: ( entry $ shadow )
```

C.3.2 ProtocolInformation

This holds the bilaterally-agreed profiles (OIDs) associated with a given OSI network address.

```
Value          = networkaddr "#" profiles
profiles       = pi /
                "(" pi *( "$" pi ) ")"
pi             = (see OID syntax)
networkaddr    = (see PresentationAddress syntax)
```

```
protocolInformation: URI+0000+URL+itot://server.example.com:1999
9 # 1.2.3.4.5.6.7
```

C.3.3 AccessPoint93

This is used to hold an X.500 access point.

```
Value          = dn "#" presentationaddr /
                "(" dn "#" presentationaddr "#" protinfo ")"
dn             = (see DN syntax)
presentationaddr = (see PresentationAddress syntax)
protinfo       = (see ProtocolInformation syntax)
```

For example:

```
superiorKnowledge: cn=DSA,o=Company # URI+0000+URL+itot://server
.example.com:19999|URI+0000+URL+ldap://server.example.com
```

C.3.4 MasterAndShadowAccessPoints

```
Value          = masap /
                "(" masap *( "$" masap ) ")"
```

```

masap          = category "#" accesspoint
category       = "master" / "shadow"
accesspoint    = (see AccessPoint93 syntax)

```

For example:

```

specificKnowledge: master # cn=DSA1 # URI+0000+URL+itot://server
.example.com:19999
specificKnowledge: ( master # cn=DSA1 # URI+0000+URL+itot://serv
er.example.com:19999 $ shadow # cn=DSA2 # URI+0000+URL+itot://s
hadow.example.com:19999 )

```

C.3.5 SupplierOrConsumer

This is used to hold shadowing knowledge. Also see [Section C.3.7, “SupplierAndConsumer”](#).

```

Value          = agreement "#" accesspoint
agreement      = bindingid "." bindingversion
bindingid      = 1*DIGIT
bindingversion = 1*DIGIT
accesspoint    = (see AccessPoint93 syntax)

```

For example:

```

consumerKnowledge: 1.1 # cn=DSA2 # URI+0000+URL+itot://server.ex
ample.com:19999

```

C.3.6 SupplierInformation

```

Value = "(" "master" "#" soc ")" /           ; is (master)
      "(" "shadow" "#" soc ")" /           ; isn't, master unknown
      "shadow" "#" soc "#" "(" ap ")"     ; isn't, master known
soc    = (see SupplierOrConsumer syntax)
ap     = (see AccessPoint93 syntax)

```

For example:

```

supplierInformation: shadow # 100.1 # ( cn=DSA1 # URI+0000+URL+i
tot://server.example.com:19999 )

```

C.3.7 SupplierAndConsumer

This is used to hold shadowing knowledge.

```

Value          = supplier "#" consumers
supplier       = (see AccessPoint93 syntax)
consumers     = consumer /
              "(" consumer *( "$" consumer ) ")"
consumer      = (see AccessPoint93 syntax)

```

For example:

```

secondaryShadows: ( (cn=DSA2 # URI+0000+URL+itot://shadow.examp
le.com:19999) $ (cn=DSA3 # URI+0000+URL+itot://shadow2.example.c
om:19999) )

```

C.4 X.400 attribute syntaxes

C.4.1 ORAddress

This is used to hold an X.400 O/R address.

```
Value      = "/" 1*component
component = ctype "=" cvalue "/"
ctype     = (any key from RFC 2156, e.g. "C", "S", etc)
cvalue    = pstring [ "*" tstring ] [ "&" bstring ] [ "%" ustring ]
pstring   = 1*PrintableCharacter
tstring   = (escaped Teletex characters)
bstring   = (BMP characters in UTF-7)
ustring   = (Universal characters in UTF-7)
```

For example:

```
mhsORAddress: /i=P/s=Principle/o=Widget/prmd=Widget Co/admd= /c=
GB/
```

C.4.2 ORName

This is used to hold an X.400 O/R address and an associated DN.

```
Value      = "X400:" [ oraddress ] [ "#X500:" dn ]
oraddress  = (see ORAddress syntax)
dn         = (see DN syntax)
```

For example:

```
mhsDLMembers: X400:/cn=Example/o=None/prmd=Test/admd= /c=US/#X50
0:dc=example,dc=com
mhsDLMembers: X400:/cn=Example/o=None/prmd=Test/admd= /c=US/
mhsDLMembers: X400:#X500:cn=John Doe,dc=example,dc=com
```

Note that previous Isode releases used a different string representation. The above examples would have looked like:

```
mhsDLMembers: dc=example,dc=com $ /cn=Example/O=None/prmd=Test/a
dmd= /c=US
mhsDLMembers: $ /cn=Example/O=None/prmd=Test/admd= /c=US
mhsDLMembers: cn=John Doe,dc=example,dc=com $
```

C.4.3 DLSubmitPermission

```
Value      = "group_member:" dn /
            "individual:" orname /
            "dl_member:" orname /
            "pattern:" orname
dn         = (see DN syntax)
orname    = (see ORName syntax)
```

For example:

```
mhsDLSubmitPermissions: group_member:cn=My Group,c=US
mhsDLSubmitPermissions: dl_member:X400:#X500:cn=John Doe,dc=example,dc=com
mhsDLSubmitPermissions: individual:X400:/cn=Example/o=None/prmd=Test/admd=/c=US/
mhsDLSubmitPermissions: pattern:X400:/O=None/
```

Note that previous Isode releases used a different string representation. The above examples would have looked like:

```
mhsDLSubmitPermissions: GROUP# cn=My Group,c=US
mhsDLSubmitPermissions: MEMBER# cn=John Doe,dc=example,dc=com $
mhsDLSubmitPermissions: INDIVIDUAL# $ /cn=Example/o=None/prmd=Test/admd=/c=US
mhsDLSubmitPermissions: PATTERN# /o=None/
```

C.5 ACP133 syntaxes

The definitions in this section were adapted from version 1.0 of *Combined Communications-Electronic Board (CCEB) Publication 1008: CCEB Guidelines for Implementing the Lightweight Directory Access Protocol (LDAP)*, which describes the implementation of the schema defined in *Allied Communication Publication (ACP) 133* with respect to LDAP clients.

C.5.1 RIParameters

```
Value          = "rI=" ri
                "riType=" ritype
                "minimize=FALSE"
                "sHD=" shd
                "classification=" classification
ri              = 1*PrintableCharacter
ritype         = "0" / ; normal
                "1" / ; off-line
                "2" / ; partTimeTerminal
shd            = 1*PrintableCharacter
classification = "0" / ; unmarked
                "1" / ; unclassified
                "2" / ; restricted
                "3" / ; confidential
                "4" / ; secret
                "5" / ; top secret
```

C.5.2 Remarks

```
Value = [ *p *( "$" *p ) ]
p      = PrintableCharacter
```

C.5.3 ONSupported

```
Value      = namedbits / bitstring
namedbits = "{" [ namedbit * ( "," namedbit ) ] }"
namedbit  = "acpl27-nn" / "acpl27-pn" / "acpl27-tn"
bitstring = (see BitString syntax)
```

C.5.4 MLReceiptPolicy

```
Value      = none / insteadof / inadditionto
none       = "none"
insteadof  = "instead of"
            generalnames *15( "$" generalnames )
generalnames = generalname *( "%" generalname )
generalname = ( "otherName = " othername ) /
              ( "rfc822Name = " ia5string ) /
              ( "dNSName = " ia5string ) /
              ( "x400Address = " oraddress ) /
              ( "directoryName = " name ) /
              ( "ediPartyName = "
                [ "nameAssigner:" dirstring ]
                "partyName:" dirstring ) /
              ( "uniformResourceIdentifier = " ia5string ) /
              ( "iPAddress = " octetstring ) /
              ( "registeredID = " numericoid )
othername  = (BER encoding of type and value pair)
ia5string  = (See IA5String syntax)
oraddress  = (See ORAddress syntax)
name       = (See DN syntax)
dirstring  = (See CaseIgnoreString/CaseExactString syntax)
octetstring = (See OctetString syntax)
```

C.5.5 Addresses

```
Value = [ 1*55p *( "$" 1*55p ) ]
```

C.6 Reading the subschema from a client

The Directory Server now allows client read access to subschema information (as configured in the file-based oidtables). Some clients, such as the latest Active Directory applications, require this ability in order to be able to access M-Vault.

Subschema are published via subschema subentries. The list of known subschema subentries is listed in values of the **subschemaSubentry** attribute of the root DSE (which can be read by DAP and LDAP clients). In addition, the subschema subentry governing a particular entry can be discovered by reading the value of **subschemaSubentry** from the entry in question. Note that LDAP clients must explicitly add **subschemaSubentry** to the entry information selection in order to read the attribute. This is because **subschemaSubentry** is an operational attribute. Also note that LDAP clients can only read subschema subentries by performing a base object search using the filter `objectclass=subschema`.

The subschema attributes currently supported by M-Vault are limited to **attributeTypes** (the set of known attribute types) and **objectClasses** (the set of known object classes).

Others, e.g. `nameForms`, are not currently supported. LDAP subschema publication and retrieval mechanisms are described in *RFC 4512*.

Appendix D Customising Sodium

Sodium's built-in templates can be modified to suit local needs.

When Sodium uses the **Template** or **Full** views, it displays entries that are read from the Directory using one of a number of configured templates. The template used is based on the objectclasses of the entry being read. A built-in **Raw** template is provided for cases when the objectclasses do not match any other templates.

Templates are also used when adding new entries.

You may wish to modify Sodium's templates if:

- you are using the standard schema in a slightly different way (for example, only allowing one telephone number for a person)
- you are using locally-defined custom schema
- you want to set up pre-initialized field values (see [Section D.11, “Add templates”](#)).

Note: Editing the templates in Sodium will not change the schema being used by Sodium and the Directory Server, and you still need to customize the schema files as described in [Section B.2, “Extending the schema”](#).

Templates are stored in the *templates.xml* file, which Sodium will look for in (*ETCDIR*)/*sodium* and then (*SHAREDIR*)/*sodium* when it starts. It can be edited using any plain text editor, or an XML editor. It includes definitions from other template files, including an optional *custom-templates.xml* file. Definitions in the *custom-templates.xml* file will override identically matching templates in the *templates.xml* file.

Each template references a number of forms that are used by Sodium. Each form defines one or more named groups.

Each named group is displayed by Sodium in a separate tab. Each group is defined to have a number of fields.

Each field describes an attribute name, and which editor to use when displaying values. For example, this template describes an **ISP User** which displays two attributes in a tab labelled **Customer**:

```
<editor-templates version="1.2">
  <template rdn="cn" keyclass="ispUser"
    label="ISP User">
    <form ref="isp-user"/>
  </template>
  <form name="isp-user">
    <group label="Customer">
      <editor label="Name" attrtype="cn"
        fields="1"/>
      <editor label="Email" attrtype="mail"
        fields="1"/>
    </group>
  </form>
</editor-templates>
```

D.1 <editor-templates> element

This is the root XML element. It can contain child <template>, <form>, <enumeration> and <include> elements in any order. It has one attribute:

version

the version of this template specification, which is currently "1.2"

D.2 <include> element

This has 2 attributes:

filename

the name of the XML file from which additional definitions are read. This attribute is mandatory.

optional

indicates if the file must be present or not. Values of this XML attribute must be either "yes" or "no", the default is "no".

D.3 <enumeration> element

This defines a set of mappings for an enumeration, which can be referred to from <editor editor="enumeration" ...> elements. It contains a number of child <value> elements defining the mappings. For example:

```
<enumeration name="alert_codes"
  unset="No information" allowinvalid="yes">
  <value internal="AC1" display="Yellow Alert"/>
  <value internal="AC2" display="Red Alert"/>
  <value internal="AC3" display="Battle Stations"/>
</enumeration>
```

There are four attributes:

name

The identifier that can be used to refer to this enumeration from <editor> elements.

unset

This defines what additional text should appear in the case that an attribute has no value.

allowinvalid

This has a value of "yes" or "no" and defines whether illegal values in the field (ones not listed in the enumeration) result in a warning (allowinvalid="yes") or an error (allowinvalid="no"). If not specified, the default is allowinvalid="yes".

`casesensitive`

This has a value of "yes" or "no" and defines whether internal values are matched case sensitively to the attribute values or not. The default is "yes".

D.4 `<value>` element

This defines a single mapping within an `<enumeration>` between an internal stored value and the human-readable text that should be displayed for that value. The attributes are:

`internal`

The internal value to map (as stored on the Directory).

`display`

The human-readable text that should be displayed in its place.

D.5 `<template>` element

This has 4 attributes:

`rdn`

a space-separated list of LDAP attributes that will form the name of the entry

`keyclass`

a space-separated list of objectclasses that are used to match this template (all have to match)

`label`

the label to use when selecting the template when adding an entry

`icon`

the filename of the icon to display next to the template in the **Add** wizard, see [Section D.10.23, "Template icons"](#) for more details.

Only the minimum objectclasses necessary to match the desired entry/entries should be listed in the `keyclass`. Typically this would just be the structural objectclass and any appropriate auxiliary objectclasses.

The `<template>` element contains a number of child `<form>` elements.

D.6 `<form>` element

D.6.1 Within a `<template>` element

When the `<form>` element is a child of a `<template>` element, it defines a reference to the actual `<form>` used. These references can be mandatory or optional. Optional references are useful when defining auxiliary object classes for a template. In this context it has 5 attributes:

<code>ref</code>	the name of the <code><form></code> element being referred to (mandatory)
<code>keyclass</code>	for optional <code><form></code> s, the additional objectclass to use
<code>label</code>	the label to use when selecting the optional parts of the template
<code>icon</code>	the filename of the icon to use when displaying the form in the Add wizard, see Section D.10.23, “Template icons” for more details. If present, the icon will override the icon specified in the <code><template></code> .
<code>use</code>	the situation that this form should be used for, either "add" or "edit" or "add edit" (the default). See Section D.11, “Add templates” for more details.

D.6.2 Within the `<editor-templates>` element

When the `<form>` element is a direct child of the `<editor-templates>` element, it defines the groups used by the form, that is, which attributes to display. In this context, it only has one attribute, and contains one or more child `<group>` elements:

<code>name</code>	the name of the <code><form></code> element being defined.
-------------------	--

D.7 `<group>` element

This has a `label` attribute, which will appear on screen as the tab's label. It can only contain child `<editor>` or `<label>` elements.

Multiple `<group>`s can use the same label. When a template uses forms that use multiple groups with the same label, the contents of the groups are merged together for that template.

Certain optional attributes may be used to create special-purpose groups.

<code>misc</code>	May have a value "yes" or "no", defaulting to "no". If "yes", then this group becomes a 'misc' tab that shows all the remaining user attributes present in the entry that are not yet handled by any other tab, using default editors selected according to syntax type.
<code>addattr</code>	May have a value of "yes" or "no". If "yes" and <code>misc="yes"</code> is also specified, then a button is added to the tab which allows further attributes to be added by name.

D.8 `<label>` element

This simply displays a label covering the entire row. It has one attribute:

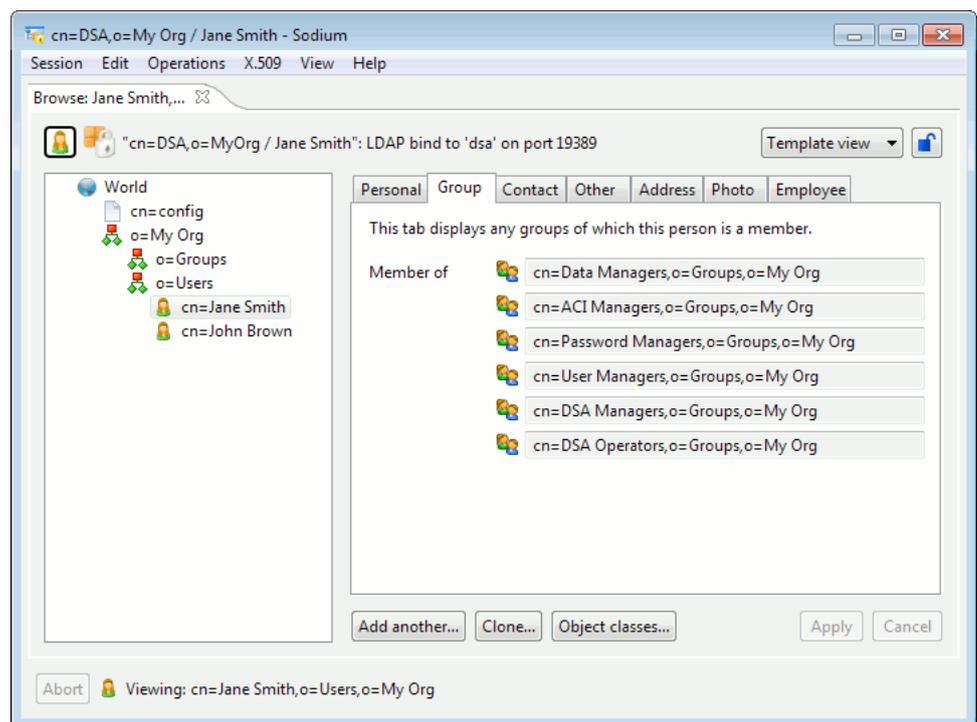
<code>label</code>	the string displayed on screen.
--------------------	---------------------------------

D.9 <memberof> element

This element may be used to display a list containing the names of all the groups that have the selected entry as a member - specifically, Sodium searches for any entries that have an `objectClass` of `groupOfNames`, and displays the DN of any whose `member` attribute contains the DN of the selected entry. It has one attribute:

`label`
the caption to be used on the screen.

Figure D.1. Sodium memberof element



Group names are presented as read-only values using the dn editor (see [Section D.10.8, "dn editor"](#)), which means that you can easily navigate to the group entry by clicking on the icon next to the group name.

D.10 <editor> element

This has several attributes:

`attrtype`
the name of the LDAP attribute type being displayed. This attribute is mandatory.

`label`
the string displayed on screen next to the attribute values. If absent, the `attrtype` is used.

compulsory

indicates if the attribute is mandatory or not, regardless of the schema. Values of this XML attribute must be either "yes" or "no", the default is "no".

readonly

indicates if the attribute may be modified or not, regardless of the Directory Server's access controls. Values of this XML attribute must be either "yes" or "no", the default is "no".

editor

the name of the underlying editor to use. Sodium selects an editor based on the attribute's syntax, so it is generally not recommended to specify this unless it is necessary to override Sodium's default choice, for example to override a "string" editor with a "stringtable" editor.

Some editors support additional XML attributes.

D.10.1 autostring editor

This allows string-valued attributes to be edited. A blank field is automatically inserted when all the other fields contain text. For example:

```
<editor attrtype="telephonenumber"
  editor="autostring" label="Phone" />
```

D.10.2 binary editor

This allows "binary" values to be edited. Instead of displaying them, they are loaded and saved to disk. Values can be deleted using the **Delete** button, and additional values added using the + button. External viewers are supported by specifying one or both of these attributes:

view_cmd

Specify the external command to run when the **View** button is clicked alongside an item. Double or single quotes may be placed around command arguments containing spaces. Some substitutions are made:

- %D is replaced by the DN
- %A by the attribute name
- %V by the filename of the temporary file containing the value, which will have an extension of *.bin* for binary values, or *.ber* for BER-encoded values
- %% gives a single %

view_all_cmd

Display a **View All** button which runs the given external command when clicked. Double or single quotes may be placed around command arguments containing spaces. Some substitutions are made:

- %D is replaced by the DN
- %A by the attribute name
- %L by the list of filenames of the temporary files containing the value, each of which will have an extension of *.bin* for binary values, or *.ber* for BER-encoded values
- %% gives a single %

For example:

```
<editor attrtype="jpegphoto"
  editor="binary" label="Photo"
  view_cmd="/home/user/bin/myviewer %D %A %V"/>
```

D.10.3 boolean editor

This allows boolean values to be edited. A popup menu is used.

The boolean editor also supports this attribute:

fields

controls whether the values "TRUE" and "FALSE" can be used simultaneously or if the attribute is single-valued (fields is "1").

D.10.4 certificate editor

This allows X.509 certificate values to be edited. Certificate values can be saved to or loaded from disk. The editor displays various information from the certificate, including any **subjectAltNames** that are present, and the **Details...** button can be used to display more comprehensive information.

Figure D.2. Sodium certificate editor

The editor will perform various checks on the values inside the certificate, and will highlight fields which contain information that may be questionable. In the example above, the editor warns that the certificate contains a **subjectAltName** value that does not match the information inside the user's entry.

For sessions which are bound using strong authentication, the editor displays a **Verify...** button, which can be used to perform certificate verification of the value.

D.10.5 certificatepair editor

This allows X.509 certificate pairs to be edited. Certificate pair values may be loaded and saved as DER files using **Save...** and **Load...**. To create a new certificate pair, use the **New** button followed by **Issued By...** and **Issued To...** to load two individual certificates.

D.10.6 certificaterevocationlist editor

This allows X.509 certificate revocation list values to be edited. Values can be loaded or saved, and the editor will highlight fields which contain information that may be questionable. The **Details...** button can be used to display more comprehensive information about the contents of a CRL.

Figure D.3. Sodium certificaterevocationlist editor

D.10.7 clearance editor

This allows viewing and editing of both **clearance** and **sioClearance** attributes, as used in the case of Directory Servers which are being used in an environment which implements a Security Policy (see [Section 6.5, “Security labels and clearance”](#)).

Values of type **clearance** will automatically be shown in the clearance editor (unless the template specifies otherwise).

Values of type **sioClearance** have a string syntax (these values hold an XML representation of a security clearance), and so by default will be displayed using a String editor, unless the template specifies otherwise.

For example:

```
<editor attrtype="sioClearance"
  editor="clearance"
  label="SIO Clearance"/>
```

D.10.8 dn editor

This allows DN values to be edited. A **Pick** button is displayed, which opens a dialog to allow a DN to be selected from elsewhere in the Directory. The editor also supports these attributes:

real_dn

controls whether Sodium treats values for this attribute as representing DNs that exist somewhere in the directory. A value of "yes" means that values will be subject to DN verification, and a **Pick** button will be displayed. To specify that a DN value does not represent an actual DN in the directory, use a value of "no". The default value (if **real_dn** is not present) is "yes". Note that this setting only controls the appearance of the Sodium's template view; it has no effect on the results of a bulk referential integrity check (see [Section 3.9, “Checking the referential integrity of attributes”](#)).

verify_dn

controls whether Sodium will check if the value references an existing entry in the Directory. This setting overrides the session-specific DN verification check setting.

The **Session Settings** window (in the **Session** menu, and also in the **Session Management** wizard) controls the number of DN values in an attribute that will be verified.

D.10.9 dlsubmitpermission editor

This allows X.400 DL Submit Permissions to be edited. The values are displayed in a table with columns titled **Permission Type**, **Distinguished Name** and **O/R Address**.

Double-clicking a row or clicking the **Edit** button allows the DN and O/R Address components to be edited. The **Add...** button allows a new value to be added, and the **Remove** button deletes the selected value.

The `dlsubmitpermission` editor also supports this attribute:

`vfill`

indicates if the editor should fill the form (so the scrollbar that appears scrolls the editor, not the form). Values of this XML attribute must be either "yes" or "no", the default is "no".

D.10.10 enumerated editor

This is for fields which can take a number of pre-defined internal values which we wish to map to human-readable text for presentation to the user. If the field is modifiable, the list of human-readable options is displayed as a drop-down list to select from, otherwise the text for the currently-selected option is displayed as a read-only field. The enumerated editor relies on sets of enumerated values defined elsewhere in the template file (see [Section D.10.3, "boolean editor"](#)). The editor supports this attribute:

`enumeration`

indicates the name of the enumeration to use for this field

D.10.11 generalizedtime editor

This provides a read-only decoded view of the a generalized time value, with **Clear** and **Edit...** buttons to allow the internal value to be modified using a dialog box. This operates similarly to the string editor (below) as regards multi-valued attributes:

`minfields`

controls the minimum number of fields that will be displayed for a multi-valued attribute

`maxfields`

controls the maximum number of fields that can be entered for a multi-valued attribute (if the Directory has more values in the attribute, they will always be displayed)

`fields`

the same as setting `minfields` and `maxfields` to the same value.

D.10.12 jpeg editor

This allows JPEG photographs to be edited. The values are displayed as images, and there are buttons to load and save the images to disk. Images can be deleted using the **Remove** button. Use the **View** button or double-click on an image to display it at its original size.

D.10.13 oraddress editor

This allows X.400 O/R Addresses to be edited. The values are displayed as compact text strings in the format described in [Section C.4.3, "DLSubmitPermission"](#). An **Edit** button displays an **O/R Address Editor** window, which allows each component in the address to be edited in a separately labelled text field. The **O/R Address Editor** window supports different O/R Address name forms.

Figure D.4. O/R Address Editor
D.10.14 orname editor

This allows X.400 O/R Names to be edited. The values are displayed in a table with columns titled **Member O/R Address** and **Distinguished Name**. Double-clicking a row or clicking the **Edit** button allows the **O/R Address** and **DN** components to be edited. The **Add...** button allows a new value to be added, and the **Remove** button deletes the selected value. The orname editor also supports this attribute:

`vfill`

indicates if the editor should fill the form (so the scrollbar that appears scrolls the editor, not the form). Values of this XML attribute must be either "yes" or "no", the default is "no".

D.10.15 pkcs7 editor

This editor allows PKCS#7 attribute values to be edited. It displays the certificate from the value, using the same format as the certificate editor (see [Section D.10.4, "certificate editor"](#)).

D.10.16 postaladdress editor

This allows multi-line postal addresses to be edited. An (optional) **F** button will fill the value using values from other attributes. A **T** button will trim extraneous whitespace from the value. The postaladdress editor supports this optional attribute:

`fillwith`

specifies a space-separated list of other attributes which will be used (in the order given) to "fill" the value. This enables the **F** button.

D.10.17 readonlylist editor

This allows multiple string values to be displayed using a popup list.

D.10.18 securitylabel editor

This allows viewing and editing of both **securityLabel** and **sioLabel** attributes, as used in the case of Directory Servers which are being used in an environment which implements a Security Policy (see [Section 6.5, "Security labels and clearance"](#)).

Values of type **securityLabel** will automatically be shown in the securitylabel editor (unless the template specifies otherwise). Values of type **sioLabel** have a string syntax (these

values hold an XML representation of a **securityLabel**), and so by default will be displayed using a String editor, unless the template specifies otherwise. For example:

```
<editor attrtype="sioLabel"
  editor="securitylabel"
  label="SIO Label" />
```

D.10.19 **securitylabelinfo editor**

This allows viewing and editing **securityLabelInfo** attributes, as used in the case of Directory Servers which implement Security Policy (see [Section 6.5, “Security labels and clearance”](#)). This editor provides the ability to browse a catalog of labels, from which new values may be loaded, using the same mechanism as described in [Section D.10.18, “securitylabel editor”](#).

D.10.20 **string editor**

This allows string-valued attributes to be edited. Unlike the autostring editor, new values must be explicitly added using the + button. Deleting all the characters in a field will delete that value.

The string editor also supports these attributes:

`minfields`

controls the minimum number of fields that will be displayed for a multi-valued attribute

`maxfields`

controls the maximum number of fields that can be entered for a multi-valued attribute (if the Directory has more values in the attribute, they will always be displayed)

`fields`

the same as setting `minfields` and `maxfields` to the same value.

D.10.21 **stringtable editor**

This allows very long lists of strings to be edited conveniently in a table-based layout. Double-clicking on a row or pressing the **Edit...** button allows a string to be edited. The **Add...** button allows a new value to be added, and the **Remove** button deletes the selected value.

The editor supports this attribute:

`vfill`

indicates if the editor should fill the form (so the scrollbar that appears scrolls the editor, not the form). Values of this XML attribute must be either "yes" or "no", the default is "no".

D.10.22 **subtreespec editor**

This allows viewing and editing a subtree specification, which is used on subentries to control the area of the DIT that is affected by that subentry.

D.10.23 **Template icons**

Sodium displays an icon next to each entry in the hierarchical tree view, and also next to each named template in the **Add** wizard. A number of icons are pre-installed, and it is possible to use additional icons.

Each icon should be a 16 pixel by 16 pixel PNG file (with optional alpha channel), and should be stored in either `(ETCDIR)/sodium/dit-icons/` or `(SHAREDIR)/sodium/dit-icons/`.

Normally one icon is used for each specific template or form, however it is possible to configure a template to use multiple icons. If the name of the icon in the template contains a %s the lowercased value of the RDN will be substituted into the filename.

For example, if an **organization** template specified an icon of `org_%s.png` the icon `org_acme.png` would be drawn next to **o=ACME** and the icon `org_isode.png` would be drawn next to **o=Isode**.

D.11 ‘Add’ templates

Normally the same template is used for adding a new entry as for editing an existing one. However, it is also possible to specify one or more “add” templates which are used only for adding entries. “Add” templates normally show only a small number of selected fields targeted at a particular application, and the fields may be pre-initialized with values.

The form-references in a template specification apply to both “add” and “edit” operations by default. However, it is possible to mark some or all of the form-references to apply to just one operation or the other with the `"use="` parameter.

- With `<form use="add" ref=... >` the referenced form is only used for adding new entries, and is omitted when the template is used for editing an entry.
- With `<form use="edit" ref=... >` the referenced form is only used for editing existing entries, and is omitted when the template is used for adding an entry.

Normally there is only one template for a particular keyclass match. So by adding `"use="` parameters, this template can be adapted to give a separate “add” template, i.e. a different set of forms to use when adding. If more than one “add” template is required, then this can be configured by inserting additional templates after the first template, using the same keyclass parameter as the first. These additional templates are used only when a new entry is being added, and appear in the **Add below...** or **Add another...** template list. This allows the user to select from a number of templates pre-initialized for different situations, for example: “Accounts Person”, “Sales Person”, etc.

To include initial values, the forms used for “add” templates should use the `"init=..."` parameter in the relevant `<editor>` elements. These initial values are only used when a new entry is added, never for editing.

There are two example “add” templates included in the default templates shipped with Sodium: see **Example pre-initialized form: Person for Accounts** at the end of the list.

Appendix E Advanced Configuration

Once a Directory Server has been set up, you may wish to configure various attributes using a command line scripting interface, such as Teldish. This section describes the various attributes and entries which can be configured.

The Directory Server configuration is held in a subtree starting at **cn=config**. This subtree is only visible to suitably authenticated server managers.

E.1 Core Configuration

The main entry holding the configuration of the Directory Server is at **cn=core, cn=config**. It uses a structural **objectClass** of **isodeDSAConfiguration**. Some auxiliary object classes are also used.

Note: Many of these attributes can be configured using M-Vault Console.

The following attributes are mandatory:

cn

The value must be `core`.

presentationAddress

This holds the server's presentation address.

isodeDSAName

This holds the server's DN. The entry at the given DN is not required to exist.

The remainder of the attributes are optional.

E.1.1 Administrative Limits

The following attributes configure administrative limits applied to **list** and **search** operations:

adminSizeLimit

A single-valued **NumericString** attribute. Its value is the maximum number of entries to return in response to a **list** or **search**. The default value is 200.

adminTimeLimit

A single-valued **NumericString** attribute. This value is the maximum elapsed time, in seconds, within which the results of a **list** or **search** request must be returned. If the time limit is exceeded, some of the results are returned with an error message. The default value is 120.

minSearchLevel

A single-valued **NumericString** of the minimum DN height for a base object of a subtree search (e.g. the root is 0, top-level DNs are 1). If absent, there is no limit on how high in the tree a search may be started.

adminLookthroughLimit

A single-valued **NumericString** attribute. Its value is the maximum number of entries to be considered when determining candidates for a **list** or **search**. It should be greater than the **adminSizeLimit** value, if present. The default value is 5000.

isodeRequireSignedModify

A single-valued **Boolean** attribute. If its value is true, then any modification requests which are not in the form of signed modifications will be rejected.

isodeServiceName

A single-valued **CaseExactString** attribute. If present, its value gives the service name used when decrypting passphrases in the various *pphr* files (if they have been encrypted).

E.1.2 Encrypting *pphr* files

M-Vault uses files to obtain passphrases for PKCS#12 files used for various purposes. These may be encrypted (though the key is itself stored on disk).

The service key can be created using the command line tool (*SBINDIR*)/*spassmgt*:

```
% spassmgt set isode.dsa
```

(A name other than *isode.dsa* may be used, provided the attribute **isodeServiceName** is changed suitably.)

The tool will prompt for a passphrase (16 characters minimum, and must contain at least three out of uppercase, lowercase, numeric digits and punctuation).

On Unix systems, you need to run this command as whatever userid the M-Vault process is using.

Passphrase files can then be encrypted using the (*SBINDIR*)/*spasscrypt* command-line tool:

```
% spasscrypt -e -s isode.dsa -f /var/isode/d3-db/ocsp.pphr
```

which will encrypt the contents of */var/isode/d3-db/ocsp.pphr* using the service key for *isode.dsa*.

(Files may be decrypted using *-d* instead of *-e*.)

E.1.3 X.509 Strong Authentication

The following attributes configure X.509 strong authentication:

dsaStrongAuthCertificate

A multi-valued attribute with **CaseExactString** syntax. This gives the pathnames of additional DER-encoded files with certificates to use as additional certificates during verification. These certificates are not trusted; they have the same status as certificates retrieved from LDAP in that they may be included in certification paths during path discovery. This attribute allows certificates to be available which might not otherwise be found by LDAP lookup, perhaps because their Subject name does not match the entry they are in or because they are not present in the referenced LDAP Directory or because LDAP lookup is not configured.

dsaStrongAuthTrustAnchor

A multi-valued attribute with **CaseExactString** syntax. This gives the pathnames of additional DER-encoded files with certificates to use as additional trust anchors for verification of received strong authentication credentials. (The verifier will always use any self-issued certificates in the PKCS#12 file as trust anchors. This attribute may be used to add further trust anchors.)

dsaStrongAuthCheckCRLs

A single-valued attribute with **Boolean** syntax. If `TRUE`, revocation status will be checked for received strong authentication credentials for the whole certification path constructed.

isodeDAPIncludeCertificationPath

A single-valued attribute with **Boolean** syntax. If `TRUE`, then in DAP bind responses and (if configured) DAP signed responses the full certification path will be sent over protocol (that is, the certificates other than any self-signed certificates from the server's PKCS#12 file).

isodeDSPIncludeCertificationPath

A single-valued attribute with **Boolean** syntax. If `TRUE`, then in DSP bind arguments, responses, and (if configured) signed operations and responses the full certification path will be sent over protocol (that is, the certificates other than any self-signed certificates from the server's PKCS#12 file).

isodeDISPIncludeCertificationPath

A single-valued attribute with **Boolean** syntax. If `TRUE`, then in DISP bind arguments, responses, and (if configured) signed operations and responses the full certification path will be sent over protocol (that is, the certificates other than any self-signed certificates from the server's PKCS#12 file).

isodeDAPStrongTokenExpiry

A single-valued attribute with **NumericString** syntax. This gives the expiry lifetime for DAP bind responses (also signed DAP responses). The default is 900 (15 minutes).

isodeDSPStrongTokenExpiry

A single-valued attribute with **NumericString** syntax. This gives the expiry lifetime for DSP bind responses (also signed DSP responses). The default is 900 (15 minutes).

isodeDISPStrongTokenExpiry

A single-valued attribute with **NumericString** syntax. This gives the expiry lifetime for DISP bind responses (also signed DISP responses). The default is 900 (15 minutes).

dsaStrongAuthCheckLeaf

A single-valued attribute with **Boolean** syntax. If `TRUE`, revocation status will be checked for received strong authentication credentials for the leaf certificate.

dsaStrongAuthOCSPnonce

A single-valued attribute with **Boolean** syntax. If `TRUE`, OCSP requests will be made with the nonce extension.

dsaStrongAuthOCSPuri

A single-valued attribute with **caseexactstring** syntax. This gives a URI (http or https) that will be used for OCSP requests.

dsaStrongAuthOCSPresponder

A single-valued attribute with **caseexactstring** syntax. This gives the filename of a DER-encoded certificate that will be trusted as a signer of OCSP responses.

dsaStrongAuthLookupAvoidOCSPConfigured

A single-valued attribute with **Boolean** syntax. If set to `TRUE` then **dsaStrongAuthOCSPuri** will not be used.

dsaStrongAuthLookupAvoidOCSPURI

A single-valued attribute with **Boolean** syntax. If set to `TRUE` then URIs from certificate extensions will not be used for OCSP.

dsaStrongAuthLookupAvoidCRLConfigured

A single-valued attribute with **Boolean** syntax. If set to `TRUE` then CRLs will not be retrieved from the configured LDAP server.

dsaStrongAuthLookupAvoidCRLURI

A single-valued attribute with **Boolean** syntax. If set to `TRUE` then CRLs will not be retrieved from URIs in CRL or ARL Distribution Point extensions, or from freshestCRL extensions.

dsaStrongAuthLookupAvoidCertConfigured

A single-valued attribute with **Boolean** syntax. If set to `TRUE` then certificates will not be specifically sought from the configured LDAP server. (If we're retrieving CRLs then certificates will be retrieved regardless.)

dsaStrongAuthLookupAvoidCertURI

A single-valued attribute with **Boolean** syntax. If set to `TRUE` then certificates will not be retrieved from URIs in extensions.

dsaStrongAuthLookupAvoidFreshestCRL

A single-valued attribute with **Boolean** syntax. If set to `TRUE` then `FreshestCRL` extensions are ignored.

dsaStrongAuthLookupAvoidOCSPHTTPGET

A single-valued attribute with **Boolean** syntax. If set to `TRUE` then `HTTP GET` is not used for OCSP requests (normally it is used if the encoded request is small); instead, `HTTP POST` is always used.

dsaStrongAuthLDAPhost

A single-valued attribute with **CaseExactString** syntax. This gives the host name or IP address for the LDAP host that the Directory server will use for looking up additional certificates and CRLs for verifying received strong authentication credentials.

dsaStrongAuthLDAPport

A single-valued attribute with **Integer** syntax. This gives the port to use for looking up additional certificates and CRLs for verifying received strong authentication credentials. If not present, the default 389 will be used.

dsaStrongAuthP12file

A single-valued attribute with **CaseExactString** syntax. This gives the pathname relative to the Directory's base directory of a PKCS#12 file that the Directory Server will use for strong authentication. The user certificate in the PKCS#12 file must have a Subject that matches the value of `isodeDSAName`.

dsaStrongAuthPPHRfile

A single-valued attribute with **CaseExactString** syntax. This gives the pathname relative to the Directory's base directory of a file containing the passphrase for the PKCS#12 file.

E.1.4**TLS**

The following attributes configure TLS:

tlsDontTrustIdentities

Single-valued attribute with **Boolean** syntax. If set to `TRUE`, then any self-signed certificates in TLS identities will not be trusted. The default behaviour is that such certificates are trusted.

tlsCheckCRLs

Single-valued attribute with **Boolean** syntax. If `TRUE`, revocation status will be checked for received credentials when performing TLS authentication for the whole certification path constructed.

tlsCheckLeaf

Single-valued attribute with **Boolean** syntax. If `TRUE`, revocation status will be checked for received credentials when performing TLS authentication for the leaf certificate.

tlsOCSPnonce

A single-valued attribute with **Boolean** syntax. If `TRUE`, OCSP requests will be made with the nonce extension.

tlsOCSPuri

A single-valued attribute with **caseexactstring** syntax. This gives a URI (`http` or `https`) that will be used for OCSP requests.

tlsOCSPresponder

A single-valued attribute with **caseexactstring** syntax. This gives the filename of a DER-encoded certificate that will be trusted as a signer of OCSP responses.

tlsLookupAvoidOCSPConfigured

A single-valued attribute with **Boolean** syntax. If set to `TRUE` then **tlsOCSPURI** will not be used.

tlsLookupAvoidOCSPURI

A single-valued attribute with **Boolean** syntax. If set to `TRUE` then URIs from certificate extensions will not be used for OCSP.

tlsLookupAvoidCRLConfigured

A single-valued attribute with **Boolean** syntax. If set to `TRUE` then CRLs will not be retrieved from the configured LDAP server.

tlsLookupAvoidCRLURI

A single-valued attribute with **Boolean** syntax. If set to `TRUE` then CRLs will not be retrieved from URIs in CRL or ARL Distribution Point extensions, or from freshestCRL extensions.

tlsLookupAvoidCertConfigured

A single-valued attribute with **Boolean** syntax. If set to `TRUE` then certificates will not be specifically sought from the configured LDAP server. (If we're retrieving CRLs then certificates will be retrieved regardless.)

tlsLookupAvoidCertURI

A single-valued attribute with **Boolean** syntax. If set to `TRUE` then certificates will not be retrieved from URIs in extensions.

tlsLookupAvoidFreshestCRL

A single-valued attribute with **Boolean** syntax. If set to `TRUE` then FreshestCRL extensions are ignored.

tlsLookupAvoidOCSPHTTPGET

A single-valued attribute with **Boolean** syntax. If set to `TRUE` then HTTP GET is not used for OCSP requests (normally it is used if the encoded request is small); instead, HTTP POST is always used.

tlsLookupAvoidNative

A single-valued attribute with **Boolean** syntax. If set to `TRUE` then native lookup is not used.

tlsLDAPhost

Single-valued attribute with **CaseExactString** syntax. It holds the name of the Directory Server to be used for certificate verification performed during TLS authentication.

tlsLDAPport

Single-valued attribute with **Integer** syntax. It holds the port number of the Directory Server to use for certificate verification performed during TLS authentication. The value is ignored unless **tlsLDAPhost** is set. If not present, the default value of 389 will be used.

tlsCertificate

Multi-valued attribute with **CaseExactString** syntax. This gives the pathnames of additional DER-encoded files with certificates to use as additional certificates for verification during TLS authentication. These certificates are not trusted; they have the same status as certificates retrieved from LDAP in that they may be included in certification paths during path discovery. This attribute allows certificates to be available which might not otherwise be found by LDAP lookup, perhaps because their Subject name does not match the entry they are in, because they are not present in the referenced LDAP Directory or because LDAP lookup is not configured.

tlsTrustAnchor

Multi-valued attribute with **CaseExactString** syntax. It holds the names of trust anchors used by the Directory Server for certificate verification during TLS authentication.

tlsVerifyDepth

Single-valued attribute with **Integer** syntax. It configures the depth of validation of client certificates (if they are validated at all, see **tlsVerifyClient**). For example a value of 2 means that a client may present a certificate that is signed by an intermediate CA, which was issued by one of the CAs in **tlsCaCertificateFile**, provided the client also presents the intermediate CA's certificate.

tlsCaCertificateFile

Single-valued attribute with **CaseExactString** syntax. It holds the pathname of a file containing a series of CA certificates in PEM format. Any presented client certificates will be validated against these certificates if **tlsVerifyClient** is set to optional or require.

tlsVerifyClient

Single-valued attribute with **NamedBitstring** syntax. It permits configuration of client certificate verification. The following names (with bit numbers in parentheses) may be used:

none	(0)
optional	(1)
require	(2)
optionalNoCa	(3)

The meanings are described in [Section 5.7.2.2, "Client authentication"](#).

tlsSessionCacheTimeout

Single-valued attribute with **Integer** syntax. It configures how long TLS session information is held in the session cache against the possibility of session resumption by the same client; this may avoid a full handshake sequence.

- A value greater than 0 is the time-out period in seconds
- A value of 0 implies no caching (i.e. always perform a full handshake)
- A value of -1 implies indefinite caching (not recommended)
- The value of -2 is reserved for future use
- All other negative values are ignored.

If not configured, the default time-out is 86400 seconds (i.e. 24 hours).

tlsSupportFlags

This attribute uses the **NamedBitString** syntax. It permits configuration of certain aspects of the TLS support. The following names (with bit numbers in parentheses) may be used:

rejectSSLv2Hello	(0)
suppressSSLv3	(1)
suppressTLSv1	(2)
disableTLSSRollBackBug	(6)
singleDHUse	(7)
disableBlindRSA	(8)
workaroundMicrosoftSessID	(9)
workaroundNetscapeChallenge	(10)
workaroundNetscapeReuseCipherChange	(11)
workaroundSSLRef2ReuseCertType	(12)
workaroundMicrosoftBigSSLv3Buffer	(13)
workaroundMSISSLv2RSAPadding	(14)
workaroundSSLeay080ClientDH	(15)
workaroundTLSD5	(16)

<code>workaroundTLSBlockPadding</code>	(17)
<code>workaroundDontInsertEmptyFragments</code>	(18)

None of these bits are set by default. The meanings are as follows:

`rejectSSLv2Hello`

If set, the Directory Server never negotiates SSLv2.

`suppressSSLv3`

If set, the Directory Server never negotiates SSLv3.

`suppressTLSv1`

If set, the Directory Server never negotiates TLSv1.

`disableTLSRollBackBug`

Disable version rollback attack detection.

`singleDHUse`

Always create a new key when using temporary/ephemeral DH parameters.

If a bit is not explicitly set, it is assumed to be unset. Thus the default configuration is to support SSLv3 and TLSv1, to accept SSLv2Hello, to reject empty Hello messages and oversized records, and not to ignore a premature EOF at the record level.

Note: There are also a set of workarounds (prefixed with “workaround”), which may help when using particular client software. The OpenSSL documentation suggests they they should not reduce security if set, but otherwise offers almost no information on them.

tlsRandomSeedPath

Single-valued attribute with **CaseExactString** syntax. The value is the name of the file which contains the input for the random number generator. If no filename is configured, the name *seed.dat* is used.

tlsDHParamsPath

Single-valued attribute with **CaseExactString** syntax. The value is the name of the file which contains the generated Diffie-Hellman parameters for DHE/ADH use. If the named file does not exist or does not contain DH parameters, precomputed parameters are used.

If no filename is configured, DHE/ADH may still be used, although only precomputed parameters are used.

tlsKeyInfoPaths

Single-valued attribute with **CaseExactString** syntax. The value is the name of the directory containing PKCS#12 information for the Directory Server. If a passphrase is required for use with the PKCS#12 file, then the passphrase must be placed in a file called *key.pphr*. The passphrase file is read as an ordinary text file. A single trailing linebreak is permitted in the file, and ignored.

tlsConfiguredCipherSuites

Multi-valued attribute with **Integer** syntax. Values are integers corresponding to individual TLS cipher suites; if this is not set, no cipher suites are configured. Values of this attribute should only be set using M-Vault Console in order to ensure that the configuration is valid. See [Section 5.7.3, “Supported TLS cipher suites”](#) for further guidance.

E.1.5 Shadowing

The following attributes configure shadowing (DISP) behaviour:

shadowPrunePeriod

The value of this **NumericString** attribute is the length of time (in secs) to keep changes in *changelog.db* that have been shadowed to consumer Directory Servers.

dsaShadowFailureDelay

The value of this **NumericString** attribute is the minimum delay between retrying unsuccessful shadow updates (in secs).

dsaShadowOnChangeDelay

The value of this **NumericString** attribute is the minimum delay between last change and update in an On Change shadow agreement (in secs).

dsaShadowRetryDelay

The value of this **NumericString** attribute is the minimum delay between successful shadow updates (in secs).

dsaShadowOnChangeHoldOpen

A single-valued **Boolean** attribute. If **TRUE** in an On Change shadow agreement, the connection remains open between consecutive shadow updates.

E.1.6 Chaining

The following attribute configure chaining behaviour:

isodeChainPolicy

A single-valued **CaseIgnoreList** attribute, used to specify the desired ordering of protocols used when chaining. The items in the list can be **ldap** or **dsp**. This example shows a server configured to try DSP first, then LDAP chaining:

```
isodeChainPolicy: dsp$ldap
```

dspIdleTimeOut

A single-valued **Integer** attribute used to time out idle DSP and LDAP chained connections. Idle connections will be closed. The default value is 1800 seconds.

dspGarbageCollectInterval

A single-valued **Integer** attribute used to periodically check DSP and LDAP chained connections. The default value is 15 seconds.

dspBindTimeLimit

A single-valued **Integer** attribute used to time out slow new DSP and LDAP chained connections. The default is 60 seconds.

dspOperationTimeLimit

A single-valued **Integer** attribute used to time out slow operations on DSP and LDAP chained connections. The default is 60 seconds.

E.1.7 Logging

The following attributes configure logging:

dsaLogTailor

A single-valued **CaseExactString** attribute used to hold the XML representation of the server's log configuration.

isodeAuditEnable

A single-valued **Boolean** syntax used to enable audit logging. The default is **TRUE**.

E.1.8 Miscellaneous

The following attributes configure miscellaneous settings:

superiorKnowledge

A single-valued **AccessPoint93** attribute that holds the superior reference of this Directory Server, if it does not master a naming context immediately below the root of the DIT.

authTimestamps

A single-valued **Boolean** attribute that if set to `TRUE` records the time a user last authenticated in a special **authTimestamp** operational attribute on their entry. The default is `FALSE`.

manager

A multi-valued **DN** attribute that if set specifies the DN of the Directory Server “super user”, a user who is not subject to any access controls.

userPassword

This attribute uses the **Password** syntax and it is used as the password for the optional “super user” specified in the **manager** attribute.

E.1.9**SASL**

The following attributes configure SASL:

saslAvailableMechanisms

A single-valued attribute with **CaseIgnorePrintableString** syntax. The value is a space-separated list of enabled SASL mechanisms. If this is not set, no SASL mechanisms are enabled. For the list of supported mechanisms, see [Table 5.1, “SASL mechanisms”](#).

isodeSASLAllowAnonymous

A single-valued attribute with **Boolean** syntax. If the value is `TRUE` then the “ANONYMOUS” SASL mechanism may be used. Normally this would not be used as LDAP directly supports anonymous binds.

isodeSASLAllowPlain

A single-valued attribute with **Boolean** syntax. If the value is `TRUE`, the SASL mechanisms using plaintext (i.e. “PLAIN” and “LOGIN”) are permitted on connections without TLS confidentiality.

isodeSASLMinSSF

A single-valued **Integer** attribute that is used to enforce a minimum level of security layer. Not all SASL mechanisms support security layers. The value approximately indicates the strength of the symmetric key used to encrypt the layer, e.g. 56. The default of 0 means security layers are effectively optional.

isodeSASLMaxSSF

A single-valued **Integer** attribute that is used to enforce a maximum level of security layer. Not all SASL mechanisms support security layers. The value approximately indicates the strength of the symmetric key used to encrypt the layer, e.g. 56. Note that values over 56 will be treated as 56.

Note: Licensing may reduce the effective maximum value.

isodeSASLGenericRule

Selects which rule should be used by the Directory Server to map userids for generic SASL mechanisms into DNs. It is a single-valued **Integer** attribute, and has a default value of 3.

isodeSASLGenericUsers

A single-valued attribute of **DN** syntax that is used in mapping rule 0 for generic SASL mechanisms. It holds the RDN sequence in between the entry and the **dc** portion of the constructed DN. The default is no RDN sequence.

isodeSASLGenericFullMatchAttr

Defines the attribute in user entries that contains the complete userid for generic SASL mechanisms. It is a single-valued **OID**, and if not set defaults to **mail**.

isodeSASLGenericUserMatchAttr

A single-valued attribute of **OID** syntax that defines the attribute used when searching for the user portion of a generic SASL userid. This is used in mapping rule 1 for generic SASL mechanisms. If not set it defaults to **cn**.

isodeSASLGenericDomainMatchAttr

A single-valued attribute of **OID** syntax that defines the attribute used when searching for the domain portion of a generic SASL userid. This is used in mapping rules 1 and 2 for generic SASL mechanisms. If not set it defaults to **cn**.

isodeSASLGenericNamingAttr

A single-valued attribute of **OID** syntax that defines the attribute used when forming an RDN from the user portion of a generic SASL userid. This is used in mapping rules 0 and 1 for generic SASL mechanisms. If not set it defaults to **cn**.

isodeSASLGenericBase

A single-valued attribute of **DN** syntax that defines the base entry for all the generic SASL mapping rules.

isodeSASLGenericDomain

A single-valued attribute of **CaseIgnoreString** syntax that is used for the domain part of SASL userids that have no explicit domain. The generic mapping rules 1 and 2 will also skip a search for domains if this domain is being used.

isodeSASLGSSAPIRule

Selects which rule should be used by the Directory Server to map userids for the GSSAPI SASL mechanism into DNs. It is a single-valued **Integer** attribute, and has a default value of 3.

isodeSASLGSSAPIUsers

A single-valued attribute of **DN** syntax that is used in mapping rule 0 for the GSSAPI SASL mechanism. It holds the RDN sequence in between the entry and the **dc** portion of the constructed DN. The default is no RDN sequence.

isodeSASLGSSAPIFullMatchAttr

Defines the attribute in user entries that contains the complete userid for the GSSAPI SASL mechanism. It is a single-valued **OID**, and if not set defaults to **krbPrincipalName**.

isodeSASLGSSAPIUserMatchAttr

A single-valued attribute of **OID** syntax that defines the attribute used when searching for the user portion of a GSSAPI SASL userid. This is used in mapping rule 1 for the GSSAPI SASL mechanism. If not set it defaults to **cn**.

isodeSASLGSSAPIRealmMatchAttr

A single-valued attribute of **OID** syntax that defines the attribute used when searching for the realm portion of a GSSAPI SASL userid. This is used in mapping rules 1 and 2 for the GSSAPI SASL mechanism. If not set it defaults to **cn**.

isodeSASLGSSAPINamingAttr

A single-valued attribute of **OID** syntax that defines the attribute used when forming an RDN from the user portion of a GSSAPI SASL userid. This is used in mapping rules 0 and 1 for the GSSAPI SASL mechanism. If not set it defaults to **cn**.

isodeSASLGSSAPIBase

A single-valued attribute of **DN** syntax that defines the base entry for all the GSSAPI SASL mapping rules.

isodeSASLGSSAPIRealm

A single-valued attribute of **CaseIgnoreString** syntax that is used for the realm part of GSSAPI SASL userids that have no explicit realm. The GSSAPI mapping rules 1 and 2 will also skip a search for realms if this realm is being used.

E.1.10 Security Labels and Clearance

The following attributes configure security labels and clearance access:

sioClearanceCatalog

A single-valued **CaseExactString** attribute that holds the XML catalog of security clearances.

sioLabelCatalog

A single-valued **CaseExactString** attribute that holds the XML catalog of security labels.

securityLabels

A multi-valued **SecurityLabel** attribute that is used to restrict the clearances that may be used when binding to the Directory Server. For example, all users must have clearance granting access to “SECRET” information.

clearance

A multi-valued **Clearance** attribute that is used to restrict the security labels that can be used on other entries. For example, all objects must have “SECRET” labels.

rbacSecurityPolicy

A single-valued **CaseExactString** attribute that holds an XML representation of a Security Policy Information File “SPIF” as defined by SDN 801c.

E.1.11 Password Policy

If you have a Directory Server configured to use a formal password policy, it will have an additional **objectClass** value of **pwdPolicy** and have the following mandatory attribute:

pwdAttribute

A single-valued **OID** attribute, which contains the attribute type to which the password policy settings are applied. Currently this is ignored, and **userPassword** is assumed.

The following attributes are optional:

pwdSafeModify

A single-valued **Boolean** attribute, which defines if the user must provide (**TRUE**) the previous password in a **Modify** operation. The default is **FALSE**.

pwdCheckQuality

A single-valued attribute with **Integer** syntax which controls how the Directory Server checks user-provided passwords.

- 0 (default) means the server does not check them
- 1 means they are checked but problems encountered during the check are ignored
- 2 means they are checked and problems encountered during the check cause a failure

pwdCheckEntropy

A single-valued attribute with **Integer** syntax. This attribute defines the minimum password entropy. Password entropy is a function of the password length and the size of the symbol space used. e.g. A single-case password 18 character password has an entropy of 84, a similar entropy is achieved with a 13 character password using a mix of uppercase, lowercase, numbers and symbols. **pwdCheckQuality** must be non-zero for the entropy calculation to be used.

pwdMinLength

A single-valued attribute with **Integer** syntax. If **pwdCheckQuality** is non-zero, this attribute forces all passwords to have a certain minimum length.

pwdMinAge

A single-valued attribute with **Integer** syntax, defining the minimum password age in seconds. This prevents passwords from being changed too rapidly.

pwdMaxAge

A single-valued attribute with **Integer** syntax, defining the maximum password age in seconds. This forces the user to change their password over time.

pwdInHistory

A single-valued attribute with **Integer** syntax, defining the number of previous passwords to retain. It is an error to change a password to one in the password history.

pwdExpireWarning

A single-valued attribute with **Integer** syntax, defining the time in seconds before a password is due to expire that a password warning is issued.

pwdGraceAuthNLimit

A single-valued attribute with **Integer** syntax, defining the number of “grace” authentications allowed before an account is locked out.

pwdLockout

A single-valued attribute with **Boolean** syntax. If **TRUE** accounts will be locked out when their passwords expire or have been entered incorrectly too often. The default is **FALSE**.

pwdLockoutDuration

A single-valued attribute with **Integer** syntax, defining the duration in seconds that an account is locked out.

pwdMaxFailure

A single-valued attribute with **Integer** syntax, defining the number of consecutive authentication failures allowed before the account is locked out.

pwdFailureCountInterval

A single-valued attribute with **Integer** syntax, defining the length of time to remember failed authentication attempts.

pwdMustChange

A single-valued attribute with **Boolean** syntax. If **TRUE** then users must change their passwords after they are set/reset by an administrator. The default is **FALSE**.

pwdAllowUserChange

A single-valued attribute with **Boolean** syntax. If **TRUE** (the default) users are allowed to change their own passwords.

E.1.12 Password Hashing

If the Directory Server has an **objectClass** attribute which contains a value of **pwdHashSchemePolicy**, the following additional attribute may be configured:

pwdConfiguredSchemeGenerators

A single-valued attribute with **CaseIgnoreString** syntax. The value must be one of the Root DSE’s **pwdAvailableSchemeGenerators** values, and defines the algorithm used to hash plaintext values being stored in **userPassword** attributes.

E.1.13 Example

The following example shows the core configuration of a Directory Server **cn=DSA, o=Example** supporting TLS, SASL, and X.509 authentication. Note the long value for **presentationAddress** is split over multiple lines.

```
dn: cn=core,cn=config
cn: core
objectClass: isodeCommonAuthInfo
objectClass: isodeDSAConfiguration
objectClass: top
dsaStrongAuthCheckCRLs: TRUE
dsaStrongAuthLDAPhost: localhost
dsaStrongAuthLDAPport: 389
dsaStrongAuthP12file: ssl/rsa.pl2
dsaStrongAuthPPHRfile: ssl/rsa.pl2.pphr
isodeChainPolicy: dsp$ldap
isodeDSAName: cn=DSA,o=Example
isodeDSPAuthModeISend: 0
isodeDSPAuthModesIExpect: 0
isodeLDAPAuthModesIExpect: 2
isodeLDAPAuthModesIExpect: 4
```

```

isodeSASLAllowAnonymous: FALSE
isodeSASLAllowPlain: TRUE
isodeSASLGenericBase: o=Users,o=Example
isodeSASLGenericDomainMatchAttr: commonName
isodeSASLGenericFullMatchAttr: rfc822Mailbox
isodeSASLGenericNamingAttr: commonName
isodeSASLGenericRule: 3
isodeSASLGenericUserMatchAttr: commonName
presentationAddress: URI+0000+URL+itot://x500.example.com|
  URI+0001+URL+ldap://x500.example.com|
  URI+0001+URL+ldaps://x500.example.com
saslAvailableMechanisms: DIGEST-MD5 LOGIN PLAIN SCRAM-SHA-1
tlsCheckCRLs: TRUE
tlsDontTrustIdentities: FALSE
tlsKeyInfoPaths: ssl
tlsLDAPhost: localhost
tlsLDAPport: 389
tlsRandomSeedPath: seed.dat
tlsSessionCacheTimeout: 300
tlsSupportFlags: ( rejectSSLv2Hello $ workaroundAll )
tlsVerifyClient: ( optional )
tlsVerifyDepth: 5

```

E.2 Peer Configuration

Configuration specific to each peer Directory Server used in chaining and shadowing is held in separate entries under **cn=config**. The entries have an **objectClass** of **isodePeerDSA**.

The following attributes are mandatory:

isodeDSAName

A single-valued **DN** attribute that holds the name of the peer Directory Server.

presentationAddress

This holds the peer Directory Server's presentation address.

E.2.1 Shadowing

Configuration of DISP (shadowing) with a peer Directory Server is held in the peer-specific entry. An additional **objectClass** value of **isodePeerAuthInfo** is used.

The following optional attributes can be used:

isodeDISPPasswordIExpect

A single-valued **Password** attribute that holds the password that the peer Directory Server should send when simple DISP authentication is used.

isodeDISPAuthModeIExpect

A single-valued **Integer** attribute describing the form of DISP authentication that is expected from the peer Directory Server. The values are described in [Section 5.2.1, "Establishing identity"](#). If simple authentication is expected for example:

```

isodeDISPAuthModeIExpect: 2
isodeDISPPasswordIExpect: theirpassword

```

isodeDISPPasswordISend

A single-valued **Password** attribute that holds the password that should be sent to the peer Directory Server when simple DISP authentication is used.

isodeDISPAuthModeISend

A single-valued **Integer** attribute describing the form of DISP authentication that this Directory Server will send. The values are described in [Section 5.2.1, “Establishing identity”](#). To use simple authentication for example:

```
isodeDISPAuthModeISend: 2
isodeDISPPasswordISend: mypassword
```

isodeDISPSignSrShaUp

A single-valued **Boolean** attribute that controls if update requests are signed. The default is `FALSE`.

isodeDISPSignShaUp

A single-valued **Boolean** attribute that controls if shadow updates are signed. The default is `FALSE`.

isodeDISPSignCoShaUp

A single-valued **Boolean** attribute that controls if coordinate operations are signed. The default is `FALSE`.

E.2.2 Chaining

Configuration of DSP (chaining) with a peer Directory Server is held in the peer-specific entry. An additional **objectClass** value of **isodePeerAuthInfo** is used.

The following optional attributes can be used:

isodeDSPTrusted

A single-valued **Boolean** attribute that controls whether DSP operations chained through this peer Directory Server are “trusted” or not. Untrusted operations are all degraded by the Directory Server to have no effective authentication level.

isodeDSPDegradeStrong

A single-valued **Boolean** attribute that controls (`TRUE`) whether to degrade strong authenticated operations to an effective simple authentication level. Note that **isodeDSPDegradeSimple** may then further degrade the authentication level. The default is `FALSE`.

isodeDSPDegradeSimple

A single-valued **Boolean** attribute that controls (`TRUE`) whether to degrade simple authenticated operations to an effective anonymous authentication level. The default is `FALSE`.

isodeDSPPasswordIExpect

A single-valued **Password** attribute that holds the password that the peer Directory Server should send when simple DSP authentication is used.

isodeDSPAuthModesIExpect

A single-valued **Integer** attribute describing the form of DSP authentication that is expected from the peer Directory Server. The values are described in [Section 5.2.1, “Establishing identity”](#). If simple authentication is expected for example:

```
isodeDSPAuthModeIExpect: 2
isodeDSPPasswordIExpect: theirpassword
```

isodeDSPPasswordISend

A single-valued **Password** attribute that holds the password that should be sent to the peer Directory Server when simple DSP authentication is used.

isodeDSPAuthModeISend

A single-valued **Integer** attribute describing the form of authentication that this Directory Server will send. The values are described in [Section 5.2.1, “Establishing identity”](#). To use simple authentication for example:

```
isodeDSPAuthModeISend: 2
isodeDSPPasswordISend: mypassword
```

isodeDSPSignRes

A single-valued **Boolean** attribute that controls if DSP results are signed. The default is FALSE.

isodeDSPSignArg

A single-valued **Boolean** attribute that controls if DSP operation arguments are signed. The default is FALSE.

E.2.3 Example

The following entry shows an entry used with DSP and DISP to the Directory Server **cn=DSA,o=MNN.com**. Note escaping is required to use a DN in an RDN, and the long **presentationAddress** is split onto a second line.

```
dn: isodeDSAName=cn\=DSA\,o\=MNN.com,cn=config
objectClass: isodePeerAuthInfo
objectClass: isodePeerDSA
objectClass: top
isodeDISPAuthModeISend: 1
isodeDISPAuthModesIExpect: 1
isodeDSAName: cn=DSA,o=MNN.com
isodeDSPAuthModeISend: 2
isodeDSPAuthModesIExpect: 2
isodeDSPPasswordIExpect: theirpassword
isodeDSPPasswordISend: mypassword
presentationAddress: URI+0000+URL+itot://x500.mnn.com|
  URI+0001+URL+ldap://x500.mnn.com
```

E.3 Shadow Agreements

Shadow agreement configurations are held in entries directly below the peer entry for the Directory Server being shadowed with.

The entries use a structural **objectClass** of **isodeSupplierAgreement** or **isodeConsumerAgreement** as appropriate. These share a superclass of **isodeShadowAgreement**.

All agreements have the following mandatory attributes:

isodeAgreementID

A single-valued **Integer** attribute which holds the unique agreement ID with this peer. The configuration entry is named with this attribute and value.

isodeAgreementVersion

A single-valued **Integer** attribute which holds the version of the agreement.

isodeShadowPrefix

A single-valued **DN** attribute which holds the prefix of the shadowed area.

All agreements have the following optional attributes:

isodeAgreementEnabled

A single-valued **Boolean** attribute which can be used to enable and disable the agreement.

isodeSupplierInitiated

A single-valued **Boolean** attribute which indicates if the agreement is supplier-initiated (**TRUE**) or consumer-initiated.

isodeOnChange

A single-valued **Boolean** attribute which indicates if the agreement is on-change (**TRUE**) or periodic.

isodeBeginTime

A single-valued **GeneralizedTime** attribute which holds the start time of the agreement. The default is to start immediately.

isodeUpdateInterval

A single-valued **Integer** attribute which holds the gap between normal periodic updates.

isodeWindowSize

A single-valued **Integer** attribute which holds the portion of the update interval during which the update will be attempted.

isodeOtherTimes

A single-valued **Boolean** attribute which indicates if a periodic agreement can be updated outside of the normal schedule.

E.3.1 Supplier Agreements

Supplier agreements have the following additional optional attributes:

isodeShadowArea

A single-valued **SubtreeSpecification** attribute which describes which portion of the DIT underneath the **isodeShadowPrefix** is shadowed to the consumer.

isodeAttributeSelection

A multi-valued **AttributeSelection** attribute which configures the attributes to include and exclude from shadowing by **objectClass**. If absent, all attributes are shadowed.

isodeForbidAutoTotal

A single-valued **Boolean** attribute which prevents the supplier from sending an automatic total update. This may be desirable if the update is expected to be very large.

isodeForbidModifyDN

A single-valued **Boolean** attribute which prevents updates from including **ModifyDN** changes. This may be desirable if the update is being consumed (directly or indirectly) by Directory Servers that do not fully support **ModifyDN**.

E.3.2 Consumer Agreements

Consumer agreements have the following additional mandatory attribute:

subtreeReference

A single-valued **DN** attribute which specifies the GDAM that the shadowed data will be stored in.

Consumer agreements have the following additional optional attributes:

isodeSupplierIsMaster

A single-valued **Boolean** attribute which indicates that the supplying Directory Server also masters the data. This is used to optimize the chaining of write operations.

isodePermitIncrReplay

A single-valued **Boolean** attribute which indicates if certain errors in incremental updates are permitted. This can improve robustness at the possible expense of data inconsistency.

E.3.3 Agreement State

The current state of each shadow agreement is held in an entry underneath each agreement configuration entry. Most attributes are read-only.

The entries have an RDN of **cn=state**, and a structural **objectClass** of **isodeSupplierState** or **isodeConsumerState** as appropriate. These share a superclass of **isodeAgreementState**.

All agreement states have the following mandatory attribute:

cn

A single-valued **CaseIgnoreString** attribute which has the value `state`.

All agreement states have the following optional attributes:

isodeLastUpdateTime

A single-valued **GeneralizedTime** attribute which holds the time of the last change made.

isodeLastSuccessTime

A single-valued **GeneralizedTime** attribute which holds the time of the last successful update.

isodeLastErrorTime

A single-valued **GeneralizedTime** attribute which the time of the last unsuccessful update.

isodeShadowError

A single-valued **Integer** attribute containing a code describing the last unsuccessful update.

No error	0
Unknown error	1
Remote DSA is unavailable	2
Remote DSA's credentials are rejected	3
This DSA's credentials were rejected	4
Internal error	5
Total updates are forbidden	6
A bilateral agreement is needed	7
No suitable credentials could be used	8
A strong bind could not be used	9
The agreement ID is invalid	10
The agreement is inactive	11
Invalid information was received	12
An unsupported update strategy was used	13
The remote DSA is missing a previously sent update	14
A full update is required	15
The DSA is unwilling to perform	16
The agreement's timing is "unsuitable"	17
The update has already been received	18

Invalid sequencing was detected	19
Insufficient resources (disk/memory) were available	20

isodeForcedUpdateTime

A single-valued **GeneralizedTime** attribute, which may be modified to force an update at that particular time.

isodeForcedUpdateTotal

A single-valued **Boolean** attribute which alters whether the update forced by setting **isodeForcedUpdateTime** will be a total update or not. The default is to send an incremental update.

E.4 In-memory GDAM

Each in-memory database used for storing mastered or shadowed information is represented by a GDAM entry directly below the **cn=config** entry. The database entries have a structural **objectClass** of **imgdam**.

Each database of this type has the following mandatory attributes:

cn

A single-valued **CaseIgnoreString** attribute which holds the database name.

diskDatabaseDirectory

A single-valued **CaseExactString** attribute which holds the path to the directory holding the database files. For example:

```
diskDatabaseDirectory: C:\\Isode\\my-dsa\\gdam2
```

Each database has the following optional attributes:

description

A multi-valued **CaseIgnoreString** attribute that can be used to hold some textual description of the database contents. For example:

```
description: Shadowed data from Singapore
```

isodeMinFreeDisk

A single-valued **Integer** attribute that holds the minimum amount of disk space (in megabytes) that must be available if directory write operations are to be permitted. The default is 1.

indexBuild

A single-valued **Boolean** attribute that is changed when index building should be started.

isodeMaxSnapshots

A single-valued **Integer** attribute. Its value is the number of most recent snapshots that should be retained for this GDAM. The default value is 3.

isodeMaxLoadThreads

A single-valued **Integer** attribute. Its value is the number of threads, and thus degree of parallelism, that is permitted when loading data from disk to memory at server start up. The default value is the number of CPUs counted on the running system.

isodeMaxEntriesPerFile

A single-valued **Integer** attribute. Its value is the maximum number of entries per data-file in a snapshot. The lower this number the more data files will be required to build a complete snapshot and the greater the level of parallelism permitted when loading data at startup. The default value is 50000.

isodeCheckpointSchedule

A multi-valued **Integer** attribute. Its values control the specific times at which checkpointing is performed and so up-to-date snapshots produced. The integer values provided indicate the number of seconds after midnight (local time) at which a checkpoint should be invoked. The default is for checkpointing to take place one hour after midnight local time, i.e. a value of 3600.

isodeCheckpointInterval

A single-valued **Integer** attribute. Its value controls the interval between checkpoints. No default value. Any value provided here overrides any checkpoint schedule configured in **isodeCheckpointSchedule**.

E.4.1 GDAM Files

Each GDAM is stored in a separate filesystem directory, specified by the configuration entry's **diskDatabaseDirectory** attribute.

The database consists of three sub-directories:

config

This sub-directory contains the index configuration. Specifically this is recorded in the file indexes.ddf.

snapshots

This sub-directory contains snapshots produced after a checkpoint. Snapshots are contained in a child directory named by a hexadecimal encoded 64-bit change sequence number.

changelog

This sub-directory contains a set of files containing the set of changes made to the directory data since the most recent snapshot. Changes are also retained if they are required for shadowing.

E.4.2 Attribute indexes

The index search types described in [Section 4.6.4.1, "Index search types"](#) make use of additional databases. The administrator can configure one or more attributes to be specially indexed for presence, equality, approximate and substring matches.

Attribute indexes are covered in detail in [Section 4.6.1, "Creating a database"](#).

You can create and delete attribute indexes from the **Indexes** sub-page of the **Databases** page in M-Vault Console (see [Section 4.6.1, "Creating a database"](#)), or using `dmish`. This creates and deletes indexes in the background while the Directory Server is running.

Alternatively, you may wish to create indexes while the Directory Server is not running and without using M-Vault Console. For this purpose use the **dsimindex** utility.

E.4.2.1 The dsimindex utility

The command line for this utility is in the form:

```
dsimindex [-x] [-q]
[attribute [:types]...]
```

Where:

- x specifies that the indexes are to be removed instead of created.
 - q only report errors.
 - attribute* is the attribute to index, which if not specified is **cn**.
 - :types* is optional, and specifies the type(s) of indexes to be created. If included, the list consists of a colon plus one or more of the following letters without intervening spaces:
 - p – build a presence index
 - e – build an equality index
 - a – build an approximate index
 - s – build a substring index.
- The default is **es**, i.e. build equality and substring indexes.

The following example makes an approximate and a substring index for the **cn** attribute:

```
$ dsimkindex cn:as
```

The **dsimkindex** utility must be run in the GDAM directory. You cannot use **dsimkindex** while the Directory Server is running. If you wish to create indexes while the Directory Server is running, use either M-Vault Console or dmish.

E.5 Root DSE

The root DSE contains a number of read-only attributes that provide information about the Directory Server.

myAccessPoint

A single-valued **AccessPoint93** attribute that holds the name and presentation address of the Directory Server. This value is derived from the **presentationAddress** attribute on **cn=core, cn=config**.

superiorKnowledge

A single-valued **AccessPoint93** attribute that holds the superior reference of this Directory Server, if it does not master a naming context immediately below the root of the DIT. This value is derived from the **superiorKnowledge** attribute on **cn=core, cn=config**.

supportedLDAPVersion

A multi-valued **Integer** attribute that holds the LDAP versions supported, as per *RFC 4512*.

supportedSASLMechanisms

A multi-valued **IA5String** attribute that holds the names of the currently installed and enabled SASL mechanisms, as per *RFC 4512*. To change the values, modify **saslAvailableMechanisms** on the **cn=core, cn=config** entry.

LDAP clients should read this attribute before selecting a mechanism to use in a SASL bind.

saslInstalledMechanisms

A multi-valued **IA5String** attribute that holds the names of the currently installed (but not necessarily enabled) SASL mechanisms.

namingContexts

A multi-valued **DN** attribute listing all the mastered and shadowed naming contexts held on this Directory Server, as per *RFC 4512*.

subschemaSubentry

A multi-valued **DN** attribute listing all the subschema subentries held on this Directory Server. See [Section C.6, “Reading the subschema from a client”](#).

vendorName

The string “Isode Limited”, in accordance with *RFC 3045*.

dsaVersion

A single-valued **CaseIgnoreString** attribute holding the version of the running Directory Server.

changeLog

A single-valued **DN** attribute holding the DN of the parent of the entries comprising this Directory Server’s LDAP changelog.

strongAuthImplementationVersion

This single-valued **Integer** attribute gives the implementation version, indicating the way to configure strong authentication in the Directory Server. The current value is 1.

strongAuthActive

This single-valued **Boolean** attribute indicates whether strong authentication has been configured (**TRUE**) and is working (principally whether there is a usable PKCS#12 and passphrase file).

strongAuthTrustAnchor

This multi-valued **ASN.1** attribute gives the trust anchors (certificates) being used by the Directory Server for certificate verification.

tlsImplementationVersion

A single-valued **Integer** attribute representing the current Isode implementation supported by the server. The current value is 3.

tlsAvailableCipherSuites

A multi-valued **Integer** attribute showing the set of TLS cipher suites which may be configured on this server.

supportedExtension

This multi-valued **OID** attribute lists the OIDs of supported LDAPv3 protocol extensions, as per *RFC 4512*. Values currently include:

1.3.6.1.4.1.1466.20037

The “Start TLS” extended operation is supported, as defined by *RFC 4513*. This will only be true when TLS has been correctly configured.

1.3.6.1.4.1.4203.1.11.1

The “Password Modify” extended operation is supported, as defined by *RFC 3062*.

1.3.6.1.4.1.4203.1.11.3

The “Who Am I” extended operation is supported, as defined by *RFC 4532*.

supportedControl

This multi-valued **OID** attribute lists the OIDs of supported LDAPv3 controls, as per *RFC 4512*. Values currently include:

1.3.6.1.4.1.4203.1.10.1

The “Subentries” control is supported, as defined by *RFC 3672*.

1.3.6.1.4.1.42.2.27.8.5.1

The “Password Policy” request control is supported.

1.2.840.113556.1.4.473

The “Server-Side Sorting” control is supported, as defined by *RFC 2891*.

1.2.840.113556.1.4.319

The “Simple Paged Results” control is supported, as defined by *RFC 2696*.

2.16.840.1.113730.3.4.2

The “ManageDsaIT” control is supported, as defined by *RFC 3296*.

supportedFeatures

This multi-valued **OID** attribute lists the OIDs of supported LDAPv3 features, as per *RFC 4512*. Values currently include:

1.3.6.1.4.1.4203.1.5.1

All operational attributes can be requested in searches, defined by *RFC 3673*.

1.3.6.1.4.1.4203.1.5.2

Searches can request all attributes from specified object classes, defined by *RFC 4529*.

1.3.6.1.4.1.4203.1.5.3

True/False filters are supported in searches, as defined by *RFC 4526*.

pwdAvailableSchemeGenerators

This multi-valued **CaseIgnoreString** attribute lists the names of the hashing algorithms that can be configured in the **pwdConfiguredSchemeGenerators** attribute. See [Section C.2.25, “Password/EncryptedPassword”](#) for the possible values.

E.5.1 Example

An example of reading some attributes from the root DSE:

```
dn:
myAccessPoint: ( cn=DSA,o=Example Corp #
  URI+0000+URL+itot:/x500.example.com|
  URI+0001+URL+ldap://x500.example.com|
  URI+0001+URL+ldaps://x500.example.com )
namingContexts: o=Example Corp
supportedExtension: 1.3.6.1.4.1.1466.20037
supportedExtension: 1.3.6.1.4.1.4203.1.11.1
supportedExtension: 1.3.6.1.4.1.4203.1.11.3
supportedFeatures: 1.3.6.1.4.1.4203.1.5.1
supportedFeatures: 1.3.6.1.4.1.4203.1.5.2
supportedFeatures: 1.3.6.1.4.1.4203.1.5.3
supportedControl: 1.2.840.113556.1.4.319
supportedControl: 1.2.840.113556.1.4.473
supportedControl: 1.3.6.1.4.1.42.2.27.8.5.1
supportedControl: 1.3.6.1.4.1.4203.1.10.1
supportedControl: 2.16.840.1.113730.3.4.2
supportedSASLMechanisms: DIGEST-MD5
supportedSASLMechanisms: LOGIN
supportedSASLMechanisms: PLAIN
supportedSASLMechanisms: SCRAM-SHA-1
supportedLDAPVersion: 2
supportedLDAPVersion: 3
vendorName: Iside Limited
pwdAvailableSchemeGenerators: SCRAM-SHA-1
pwdAvailableSchemeGenerators: SHA2
pwdAvailableSchemeGenerators: SSHA2
pwdAvailableSchemeGenerators: SHA
pwdAvailableSchemeGenerators: SSHA
pwdAvailableSchemeGenerators: MD5
pwdAvailableSchemeGenerators: SMD5
pwdAvailableSchemeGenerators: CRYPT
```

Appendix F Running as an OS Service

This section describes how to configure and run Operating System services.

F.1 Linux services

M-Vault uses a systemd template unit file for service management, so that a single unit file can be used to manage multiple DSA instances. The unit file relies on DSAs being stored in the `/var/isode` directory. The directory containing the DSA is then used to name the service as follows:

```
isode-dsa@<directory-name>
```

Thus if a DSA is stored in `/var/isode/dsa-db` then the systemd service name would be:

```
isode-dsa@dsa-db
```

The DSA service would be started with:

```
systemctl start isode-dsa@dsa-db
```

F.2 Windows services

Windows supports a class of application known as a service. Applications intended to be run as services are written to conform to the interface rules of the Windows Service Control Manager (which is part of the kernel). Many of the parts of the Windows operating system itself are implemented as services. Services can run in the absence of, and independently of, logged-on users.

Service applications can run either under a specific account, or by default under the `LocalSystem` account (that is, with system privileges).

Installation of an application as a service causes a new entry to be created in the Windows Registry. The entry contains such information as the service's name, dependencies, description, and its location in the file system.

Services can be started either automatically when the system is started, or on demand, via either the **Control Panel** or by any other application which uses the appropriate API for communication with the Service Control Manager.

F.2.1 The Isode Service Configuration tool

Although services can be started and stopped (and to some degree modified) using the Services applet within the **Windows Control Panel**, this does not provide all of the functionality required by Isode services. The Isode Service Configuration tool is therefore provided. This gives the ability to start, stop, add, delete and modify services, and to configure inter-service dependencies.

F.2.1.1 Service dependencies

In some cases, an Isode service will have dependencies on one or more other services. For example, the M-Link xmpp service may make use of an M-Vault Server on the same machine for configuration information, in which case the Directory service will need to be started before the xmpp service. This dependency information is stored as part of the Windows Service configuration, but may be viewed or modified by the Isode Service Configuration tool.

By specifying appropriate service dependencies, you can control in which order services are started.

F.2.1.2 Using the Isode Service Configuration tool

The Isode Service Configuration tool is used to:

- View, add, delete and modify services.
- Configure service dependencies.
- Start and stop services.

To start the Isode Service Configuration tool, run it as the Windows Administrator from within the **Isode** group of the **Start** menu.

F.2.1.3 Installing services

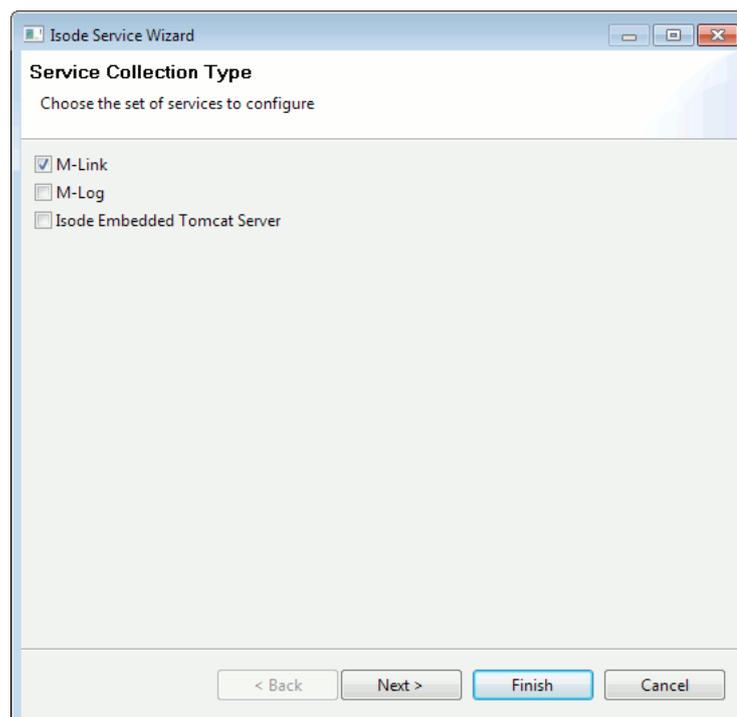
Isode services are typically installed either by the appropriate management tool (for example, M-Vault Console creates a Windows service with the appropriate configuration when you create a local Directory Server), or by the Isode Service Configuration tool itself.

The **Install Isode Services** option on the **Actions** will invoke a wizard that allows you to add Isode services corresponding to the packages you have installed.

The example below installs the M-Link services.

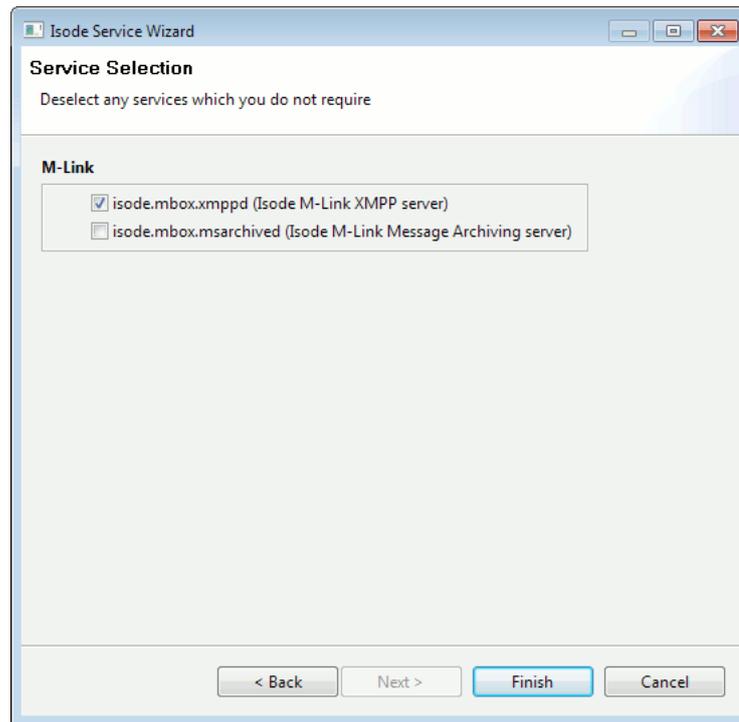
On the first page of the wizard, select **M-Link** to configure.

Figure F.1. Service to install



On the next page of the wizard, deselect the **isode.mbox.msarchived (Isode M-Link Message Archiving server)** service if archiving is not required.

Figure F.2. Services not required

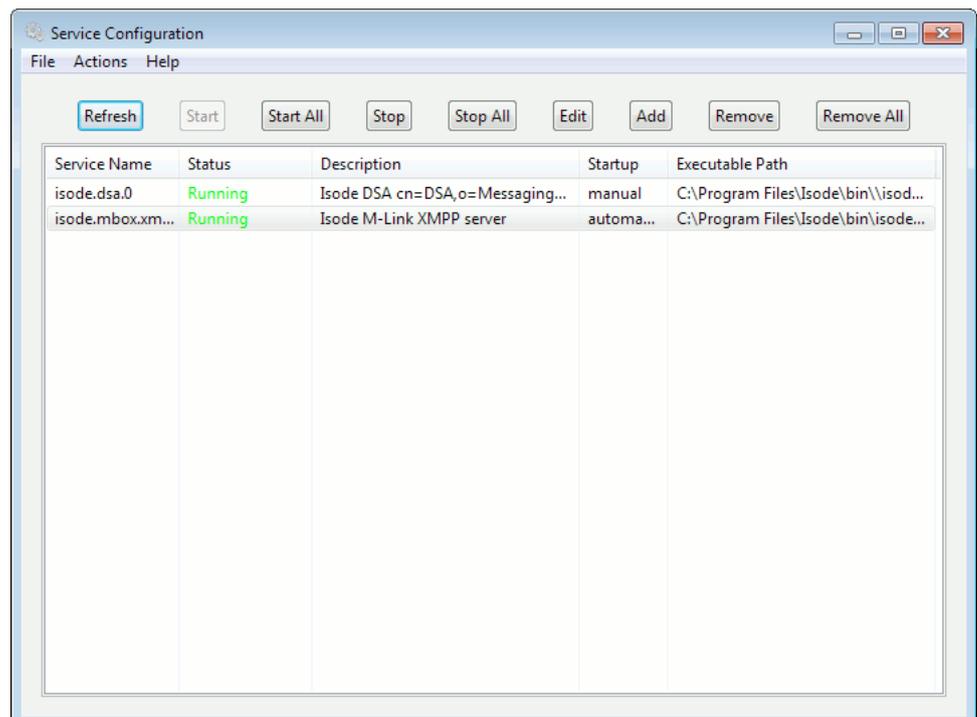


Press **Finish** to create Windows services for M-Link.

F.2.1.4 Viewing services

When Isode Service Configuration is started, a list of the installed Isode services, together with their current status (running or stopped) is displayed, as shown below.

Figure F.3. Isode Service Configuration main window

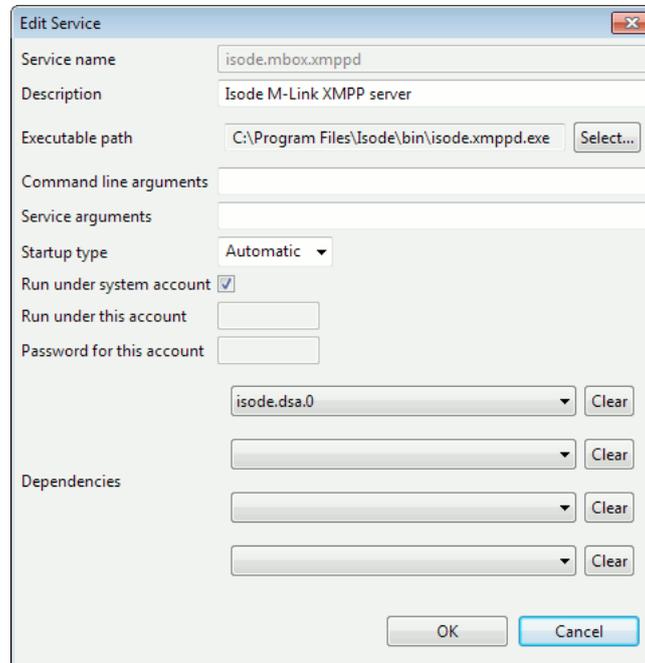


To update the status of the services to reflect the latest status, use the **Refresh** button.

To display a list of *all* of the services installed on your machine, de-select the **Isode Services Only** option on the **File** menu.

You can view the configuration details for any by selecting it and pressing the **Edit** button.

Figure F.4. Edit service



- The **Executable path** contains the executable filename, and is configured by whichever Isode application created the service.
- The **Description** field, which can contain any useful text, although the description of each service should be unique. A default description will normally be provided; for example, in the case of a Directory Server, the description will be the name of the bind profile used by M-Vault Console to manage the server.
- **Command line arguments** and **Service arguments** are configured when the service is created.
- The **Startup Type** for Isode services could be set to **Manual** or **Automatic**, see [the section called “Modifying service details”](#), below.
- **Run under system account** determines whether the service runs under the system account, or another account specified in **Run under this account** and **Password for this account**.
- **Dependencies** shows the (possibly empty) list of other services on which this service depends.

F.2.1.5 Modifying service details

Parameters for a service are typically configured by the Isode application which creates the service, and so in most cases you do not need to change any of the service settings. Typically the only values which you may want to change are:

- The **Description** field: a default description will normally be provided; for example, in the case of a Directory Server, the description will be the name of the bind profile used by M-Vault Console to manage the server. You can change this to another value without affecting the service's operation.
- The **Startup Type**: for example, you may wish to use a value of **Automatic** for services that should always be run when the system starts. In the case of automatic startup, you

may want to specify service dependencies, to ensure that services are started in the correct order.

- **Dependencies** allows you to specify that a given service depends on one or more other services. If you are displaying **Isode Services Only**, then the list of services shown in the combo box will only include Isode services. If you want to make a service dependent on a non-Isode service, then de-select the **Isode Services Only** on the **File** menu before invoking the **Edit Service** dialog.

F.2.1.6 Deleting a service

To delete a service from the list, select it in the main window shown in [Figure F.3, “Isode Service Configuration main window”](#) and click on the **Remove** button. You will be asked to confirm the deletion before it is performed.

It is also possible to disable a service without deleting it by changing the **Startup type** in the window shown in [Figure F.4, “Edit service”](#) to **Disabled**. This may be useful if a service is not required temporarily.

F.2.1.7 Adding a service

It is possible to add a new service, which may be useful if you wish to configure multiple instances of a given server and want to distinguish them with separate service names. However, typically this is unnecessary since Isode applications will add their own services and use unique names; for example, M-Vault Console creates a uniquely named service for each Directory Server that is installed on the local system.

To add a new service, use the **Add** button in the main window shown in [Figure F.3, “Isode Service Configuration main window”](#).

This will display the window shown below.

Figure F.5. Add new service

The contents of this dialog are the same as that for the **Edit Service** dialog shown in [Figure F.5, “Add new service”](#), except in this case you need to provide the **Service name**. The dialog allows you to configure any service, although only service names beginning with the prefix `isode.` will be displayed when the **Isode Services Only** option is selected.

F.2.1.8 Starting and stopping a service

To start a service, simply select the service from the list displayed in the main window (Figure F.3, “Isode Service Configuration main window”) and press **Start**. If the service is started successfully its status will be updated to `running`. Services can be stopped in a similar manner using the **Stop** button.

It is possible to start all of the configured services by pressing the **Start all** button. Service dependencies will be used to determine which order services are started. Similarly, the **Stop all** button stops all of the running configured services .

F.2.1.9 Use of Windows registry

Most of the information shown in the **Edit Service** dialog in Figure F.5, “Add new service” corresponds with that maintained by Windows in its Service Configuration tool, and so it is possible to use that tool to obtain information about a configured Isode service. The exception is the **Service arguments** parameter, which provides a way to provide extra information to an Isode service when it starts up. Any **Service Arguments** are stored in the Windows Registry under an Isode-specific key, located in:

```
HKEY_LOCAL_MACHINE\Isode\Isode\SERVICES
```

Appendix G Tcldish – the Tcldish and Ltcldish DUAs

This chapter describes the DAP and LDAP Directory management DUAs called Tcldish and Ltcldish.

Note: The general term, Tcldish, is used throughout this chapter to refer to both, as they have much in common. Where text applies to only one of the DUAs, the name of the DUA will be explicitly stated.

Tcldish is not intended as a general DUA. Its purpose is to provide data managers and more experienced users with a command line interface to the Directory. Tcldish commands allow you to move around, view and modify parts of the DIT, write and execute scripts and manage Directory Servers.

The interface provided by Tcldish is similar to that provided by a Unix shell or Windows Command Prompt. In Tcldish you can move up and down the DIT in much the same way as you move up and down a file store, and the DIT location for a Directory operation in Tcldish can be specified as a relative or absolute “pathname”, in the same way as file store locations are specified.

G.1 Tcldish and attribute syntax quoting

The management DUAs are based on Tcl (Tool Command Language), and include a complete Tcl interpreter, which has been extended to include commands for accessing the Directory. You can use the Tcl scripting language to customize the DUA environment to suit your own requirements. However, before using or customizing the DUAs, it is advisable to have a reasonable understanding of Tcl and the quoting rules of both Tcl and of the attribute syntaxes being used.

The purpose of all textual quoting is to ensure that the appropriate representation of an underlying value is presented unambiguously to the system. This applies at several levels, and consequently may require successive quoting rules to be applied to ensure the correct underlying value is achieved. For example, when presenting a DN which contains an RDN component containing a UCS-2 string, consideration must be given to quoting the UCS-2 value itself (BMPstring syntax quoting), then the value within the DN (DN syntax quoting), and finally any Tcl quoting required.

This section gives a brief introduction to the subject of quoting for both Tcl and Attribute value syntaxes.

G.1.1 Tcldish quoting

When using a Tcl-based tool such as Tcldish, you should be familiar with the Tcl language quoting rules when including special Tcl characters, such as double quotes `"`, braces `{ }` and back slashes `\`, in Tcldish command arguments. Take care also when values contain white space characters which may require Tcl quoting.

G.1.2 Attribute syntax quoting

Some values of attributes are either not directly expressible (for example, because they contain non-ASCII characters such as “ü”) or would be capable of more than one

interpretation (for example, a comma ‘,’ in a DN might be part of an RDN value or might delimit an RDN).

Syntax quoting is used to provide a resolution for both of these issues. Different attribute syntaxes may use different quoting mechanisms.

TclDish makes a distinction between DNs representing absolute positions in the DIT and DNs representing positions in the DIT relative to the current position, when those values are passed as DN arguments to TclDish commands. Absolute DNs are specified by the use of surrounding angle-brackets “<DN>”.

For more information about quoting used by particular attribute syntaxes, see [Appendix C, Attribute Syntaxes](#).

[Figure G.1, “Flow of quote processing”](#) shows both the flow of quote processing in a TclDish context on the left hand side, and how to derive an appropriately quoted value for use in TclDish on the right hand side.

G.1.3 Examples of the application of quoting

Consider the TclDish representation of the absolute Distinguished Name **cn=Five,o=Widget Ltd,c=GB**.

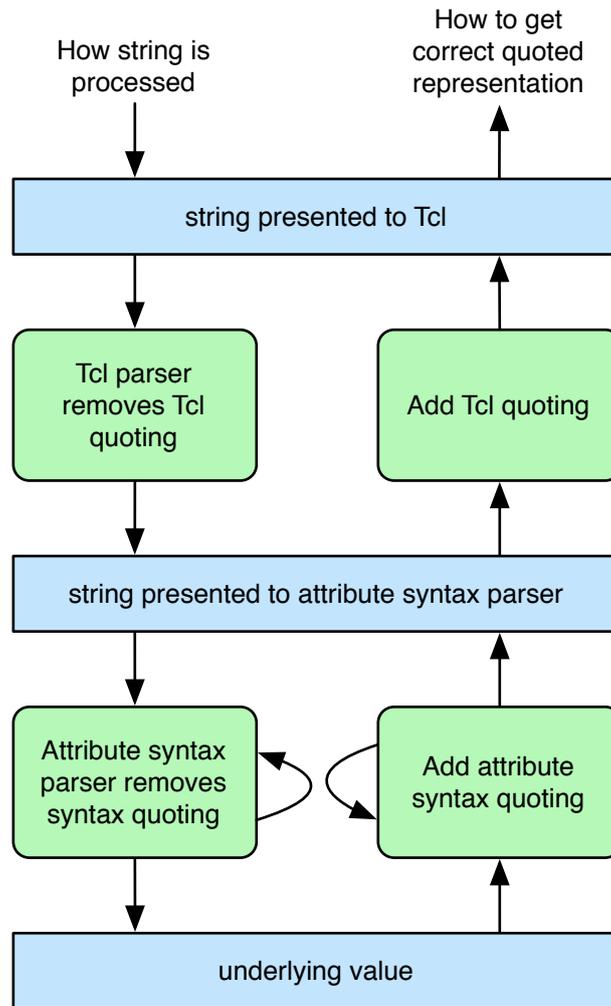
The following string might be passed as an argument to a TclDish command:

```
{<cn=Five,o=Widget Ltd,c=GB>}
```

Here the braces, which are special Tcl characters, prevent the Tcl interpreter from processing the enclosed text, which protects the embedded spaces. At a lower level, the angled brackets tell the Distinguished Name syntax handler that this is an absolute Distinguished Name.

Another representation of the same value, replacing the braces with double quotation marks, is:

```
"<cn=Five,o=Widget Ltd,c=GB>"
```

Figure G.1. Flow of quote processing

The next example shows how to enter a value which contains a special character. Take, for instance, the back slash “\” character, which is special to Tcl, the DN syntax parser and the **CaseIgnoreString** syntax parser. The absolute DN **cn=Section\Five,o=Widget Ltd,c=GB**, where the “\” is part of the **commonName** attribute value, could be passed as an argument to a Tcldish command as follows:

```
{<cn=Section\\\\"Five,o=Widget Ltd,c=GB>}
```

where the special character “\” needs to be escaped twice in a compounded fashion: once for the **CaseIgnoreString** syntax associated with **commonName** attribute, and once for the **DN** syntax.

Another method of presenting the same value using Tcl double quotes is

```
"<cn=Section\\\\"Five,o=Widget Ltd,c=GB>"
```

In Tcldish, double quoting does not offer as much protection as braces, so yet another level of escaping is required. Without any Tcl quoting, the value would appear to be two separate strings, delimited by the space. A specific peer entry (see [Section 7.2, “Connection details for Directory Servers”](#)) describing the remote Directory Server peer **cn=Remote,o=otherOrg,Inc,c=XX** (where the organisation name is “otherOrg,Inc”) below this Directory Server’s **cn=config** entry requires several levels of quoting:

```
{<isodeDSAName=cn\=Remote\,o\=otherOrg\\\,Inc\,c\=XX,cn=config>}
```

This involves two layers of DN syntax quoting, since **isodeDSAName** has a DN-valued syntax. Consequently, the inner DN value's commas, equal signs and escapes need to be escaped in order to avoid confusing parsing of the outer DN. Note also that the inner DN is not surrounded by angle brackets, since it is not of itself a value handled by Tcldish as a DN.

The following examples show how to protect non ASCII characters in Tcldish arguments. (Section C.1, “Character sets and matching rules” describes how to specify different character sets in the Directory). Here an absolute DN with the value **cn=Section\Fünf,o=Widget Ltd,c=GB** is entered as a UCS-2 string. Using Tcldish braces it could be entered as:

```
{<cn={UCS-2}Section\\\F\\00fcnf,o=Widget Ltd,c=GB>}
```

where {UCS-2} is the character set flag. The back slash separator “\” and hex code for the UCS-2 character “00fc” are protected in Tcldish by the braces, but need escaping for the DN syntax handler. Using double quotation marks, the DN would need to be entered as:

```
"<cn={UCS-2}Section\\\F\\00fcnf,o=Widget Ltd,c=GB>"
```

Note: Windows-style pathnames (such as *C:\folder\file*) may also require Tcl quoting, because the \ character is the Tcl escape character. This results in the use of doubled backslash separators; for example *C:\\folder\\file*. Alternatively, the pathname could be specified as *C:/folder/file*.

For more information on programming in Tcl and Tk, see *Practical Programming in Tcl and Tk*: Book and CD-ROM by Brent B. Welch and published by Prentice Hall, ISBN: 0136168302.

G.2 Tailoring Tcldish and Ltcldish

There are three tailoring files which control the operation of Tcldish:

- the system-wide X.500 tailor file (*SHAREDIR*)/*dsaptailor*
- a user's private Directory tailor file, *.duarc* in his home directory
- a user's customizable scripting file, *.tcldishrc* in his home directory

Similarly, Ltcldish is tailored by:

- the system-wide LDAP user agent tailor file (*SHAREDIR*)/*ldaptailor*
- a user's private Directory tailor file *.duarc* in his home directory
- a user's customizable scripting file *.tcldishrc* in his home directory

With both DUAs, some of the values in the system wide tailoring files can be overridden by values in the user's local *.duarc* and *.tcldishrc*. The following sections describe how the management DUAs make use of the tailoring files.

G.2.1 Use of the dsaptailor file by tcldish

Although individual Tcldish DUAs are likely to have their own tailoring file, they should also have access to the system wide (*ETCDIR*)/*dsaptailor* and (*SHAREDIR*)/*dsaptailor* files. On start up, Tcldish references *dsaptailor* for various default values including:

- The local name and Presentation Address of the Directory Server to contact initially. This information is given in the *dsa_address* entry. For example:

```
dsa_address uk "\"X500\"/Internet=dsa.widget.com+19999"
```

This declares that the Directory Server, nicknamed *uk*, is contacted by calling the network address of *dsa.widget.com* at TCP/IP port *19999*. The nickname can be used in tcldish to specify the name of the Directory Server to contact.

The *dsa_address* is also used when Tcldish is required to bind to a Directory Server using strong credentials. In this case, the nickname is the distinguished name of the Directory Server to be bound to, for example:

```
dsa_address "<cn=Strong DSA,o=foo,c=GB>" "\"X500\"/Internet=dsa.widget.com+19999"
```

- The location in the DIT to point to initially. This is given in the *local_DIT* entry, for example:

```
local_DIT "<ou=Research,o=Widget Ltd,c=GB>"
```

If there is more than one *dsa_address* entry in *dsaptailor*, the first one listed is used to supply the address of the default Directory Server to contact.

G.2.2 Use of the ldaptailor file by Ltcldish

Ltcldish uses the (*ETCDIR*)/*ldaptailor* and (*SHAREDIR*)/*ldaptailor* files in much the same way as Tcldish uses the *dsaptailor* file. However, the only values in the *ldaptailor* file, are:

- The nickname and Presentation Address of the Directory Server to contact initially. The latter is given in the *dsa_address* entry in the form of an LDAP URL:

```
ldap://host:port/
```

Note: The trailing slash is required.

To access the Directory Server at Widget Ltd using LDAP, the *dsa_address* entry in the *ldaptailor* file might be:

```
dsa_address ukldap ldap://dsa-host.widget.com:19996/
```

In this example, the nickname of the Directory Server is *ukldap*, and it is contacted by calling the network address of *dsa-host.widget.com* at TCP/IP port *19996*. The nickname can be used in Ltcldish to specify the name of the Directory Server to contact.

- The location in the DIT to point to initially. This is given in the *local_DIT* entry, for example:

```
local_DIT "<ou=Research,o=Widget Ltd,c=GB>"
```

The files which define the LDAP attribute and object class definitions, *ldapv3oid.at* and *ldapv3oid.oc*, will be accessed automatically in (*SHAREDIR*).

G.2.3 User's .duarc file

When a user starts Tcldish or Ltclish, his *.duarc* file, located in his home directory, is read in order to obtain user authentication information and user specified options.

Entries in this file should take the format:

```
option: value
```

Note: Command line arguments passed to Tcldish override values set by any *command specifier* lines, which in turn override values set by service and notype lines.

The following options and values are recognized. Every *option* is case-insensitive:

username: Distinguished Name

Normally the absolute Distinguished Name of the user to be recognized when binding.

Note: The presence of values for *username* and *password* in *.duarc* does not automatically cause **dbind** to use simple authentication.

To bind with simple authentication the *-simple* flag must be included in one of the following:

- The command line to start Tcldish or Ltclish, if you want to bind to a Directory Server on start up.
- The **dbind** command line if you are binding to a Directory Server after Tcldish or Ltclish has been started.
- The command specifier line: *bind: -simple* in the *.duarc* file. This sets the default authentication mode for **dbind** to simple.

password: password

The password to be recognized when binding. For this reason care should be taken to ensure that a users's *.duarc* file is not publicly readable. Alternatively, this option can be omitted, in which case the password may be prompted for. This latter method is more secure as it does not reveal the password on the command line.

service: service control flags

A space separated list of default service control flags and values (see [Section G.4.3, "Service control flags"](#)).

local_DIT: Distinguished Name

The Distinguished Name indicates the point in the DIT from which navigation is to begin. The value given here will override the *local_DIT* value in the *dsaptailor* or *ldaptailor* file, if specified. If no DIT starting point is given in any file, the starting point will be ROOT.

notype: attribute types

This is a space separated list of attributes which will not be displayed by **dshowentry** unless explicitly called for.

nickname: Distinguished Name

This option allows you to assign an alternative name to a Distinguished Name object. For example:

```
sales: <cn=Tom Smith,ou=Sales,o=Widget Ltd,c=GB>
```

assigns the nickname `sales` to **cn=Tom Smith,ou=Sales,o=Widget Ltd,c=GB**.

The nickname, `sales`, can then be used in arguments which require the Distinguished Name. For example:

```
dshowentry sales
```

will show the entry for Tom Smith.

command specifier: command flags

This entry sets the default flags required for a specific type of Tcldish command. There may be several entries of this type, one for each type of command. *command specifier* refers to the type of operation, and can take one of the values listed in [Table G.1, “Mapping of command specifiers to commands”](#). *command flags* is a space separated list of default flags to be used with the Tcldish command(s) corresponding to the *command specifier*. For example, the following entry:

```
showname: -ufn
```

means that **dshowname** and **dpwd**, which are equivalent commands, will always be invoked with `-ufn` set.

Table G.1. Mapping of command specifiers to commands

Command Specifier	Corresponding Tcldish Command(s)
add	dadd
bind	dbind
bulkclean	dbulkclean
bulkload	dbulkload
moveto	dcd
	dmoveto
compare	dcompare
delete	ddelete
	drm
list	dlist
	dls
modify	dmod
	dmodify
modifyrdn	dmodifyrdn
	dmodrdn
showname	dshowname
	dpwd
search	dsearch
showentry	dshow
	dshowentry

Command Specifier	Corresponding Tcldish Command(s)
status	dstatus
unbind	dunbind
	dquit
	quit

An example of a *.duarc* file might be:

```
username: <cn=DSA Manager,cn=DSA,ou=Research,o=Widget Ltd,c=GB>
password: dgt_93vx
bind: -simple
unbind: -noquit
service: -sizelimit 15
notype: userPassword
list: -sizelimit 30
```

This example would have the following effect:

- A default of 15 is set for the service control flag, `-sizelimit`.
- Unless explicitly requested, **userPassword** attributes will not be displayed by **dshowentry**.
- `list` is a command specifier line. The `-sizelimit` parameter for **dlist** and **dls** commands will default to 30.
- `unbind` is a command specifier line. It sets the default action for the **dunbind** command to `-noquit`, i.e. disconnect from the Directory Server, but do not exit Tcldish.
- `bind` is a command specifier line. It sets the default mode of authentication for **dbind** to `simple`. If you then use **dbind** with no authentication flags explicitly set on the command line, it will attempt to bind using simple authentication, and take the values set in the *.duarc* file for username and password to validate this connection.

Caution: It is important to ensure that a user's *.duarc* file is not publicly readable, as it contains the user's password.

G.2.4 The *.tclishrc* tailoring file

A user's *.tclishrc* file should be located in his home directory, and can contain a user defined Tcl script which is loaded when tclish or Ltclish are started. Any valid Tcl commands may appear in this file. For example, you might wish to define a customization for the command line prompt. The following example script would cause the current DIT location to be displayed instead of the default prompt string:

```
set tcl_prompt1 {
  global tcldish_pwd
  if {![directory isbound]} {
    set name "not bound"
  } else {
    if [catch {_text_dn $tcldish_pwd} name] {
      set name "???"
    }
  }
  puts -nonewline "\[$name] TclDish% "
}
```

By default Tcldish simulates the Tcl shell program, `tclsh`, and attempts to invoke an unrecognized command as a system program. This behaviour can be disabled by adding the following command to the *.tclishrc*

```
file: set auto_noexec {}
```

G.3 Running Tcldish and Ltcldish

To start a Tcldish DUA, type:

```
tcldish [-noconnect] [-file filename] [dbind arguments]
```

To start an Ltcldish DUA, type:

```
ltcldish [-noconnect] [-file filename] [dbind arguments]
```

`-noconnect`

starts Tcldish without binding to a Directory Server.

`dbind arguments`

The arguments to the **dbind** command (see [Section G.5.1, “dbind”](#)) can be used on the command line when starting Tcldish.

`-file filename`

directs Tcldish to interpret the Tcl script identified by *filename* on startup. The first Directory related command in the script file must be to bind to the Directory.

The process of connecting, or binding, to the Directory takes place automatically when the DUA is started. If no arguments are given, **dbind** will attempt to connect using the defaults set in the *dsaptailor/ldaptailor* file (for *dsa_address*) and the *.duarc* file (for the bind parameters).

The commands which you can use when Tcldish has been started are described in [Section G.4, “Command overview”](#). To exit Tcldish, type `quit` or use the **dunbind** command (see [Section G.5.2, “dunbind”](#)).

To abandon a Tcldish operation, hold down the **Control** key and press **c**. Note that this is not yet implemented for Ltcldish.

G.4 Command overview

The following sections describe the set of commands which the management DUAs provide. [Section G.5, “Commands for Directory operations”](#) describes the commands for performing common DAP and LDAP Directory operations, such as binding to the required Directory Server, performing searches and modifying entries. [Section G.6, “Other Tcldish commands”](#) describes some additional commands which do not map directly onto DAP or LDAP operations, but which are useful for administration purposes. [Section G.7, “Bulk data utilities”](#) describes the utilities provided for adding or modifying a large number of entries all at once.

As LDAP provides a subset of DAP features, some of the commands and arguments (flags) are not available when you are running Ltcldish. These limitations are listed in [Section G.4.1, “Commands and flags not applicable to Ltcldish”](#).

To avoid a conflict with built-in Tcl commands (for example, `list`), all Tcldish commands are prefixed with the character `d`. In other respects, the commands emulate the equivalent commands in the older dish DUA, except where noted. The following dish commands are not supported by Tcldish: **editentry**, **squid**, **fred**, **dsacontrol**.

You can customize command names and flags by using Tcl to define appropriate aliases in a `.tcldishrc` file. For example:

```
proc dls args { return [ eval dlist $args ] }
```

Most of the Tcldish commands support a number of flags which can be used to tailor the behaviour of the command. To get a list of the flags supported by a command, type the command name and the `-help` flag, for example:

```
dsearch -help
```

The full names of flags are used in the descriptions and examples. However, the shortest unique name is sufficient to select a flag.

G.4.1 Commands and flags not applicable to Ltcldish

The following commands and flags are not currently available for the LDAP DUA, Ltcldish:

- The `-edb read` flag.
- The following service control flags:

```
-[no]preferchain
-[no]chaining
-[no]refer
-[dont]usecopy
-[no]localscope
-managedsait
-low
-medium
-high
```

- The `-strong` and `-protected` flags in **bind**
- The `-modifyRightsRequest` flag in **dshowentry**
- The `dmanager` command described in [Section G.6.4, “dmanager”](#)

G.4.2 How objects can be referenced

With nearly every command it is possible to supply the Distinguished Name (DN) of the object to be referenced. In the syntax used to describe the commands this is represented by:

```
object
```

Names in the Internet DN format and surrounded by angled brackets are taken to be absolute Distinguished Names. Without angled brackets, the name is taken as being relative to the current position in the DIT. The special name `..` is used to mean “one level up” from the current position.

If, for example, the current position is:

```
<ou=Sales,o=Widget Ltd,c=GB>
```

The string `sn=Smith` describes the object, **sn=Smith**, which is relative to the current position:

```
<sn=Smith,ou=Sales,o=Widget Ltd,c=GB>
```

Generally if the Distinguished Name contains spaces, it must be contained within quotes, and these quotes must surround the entire argument containing spaces.

Objects can also be expressed using sequence reference numbers. As results returned by search and list operations may be long, each resulting entry has a reference number printed beside it. This number can then be used as the object in any of the calls to the Directory.

The **dsequence** command, described in [Section G.6.1, “dsequence”](#), is used to name and reset sequences, and to view sequence status details. The `-sequence` flag can be used with the **dsearch** and **dlist** commands to store the results in named sequences.

G.4.3 Service control flags

The Tcldish commands described in [Section G.5, “Commands for Directory operations”](#) map onto standard Directory operations (**Read, Compare, List, Search, AddEntry, RemoveEntry, ModifyEntry**) and have additional flags to control the type of service provided.

The flags recognized are listed below. Those applicable to both Tcldish and Ltldish are:

```
-timelimit n
-notimelimit
-sizelimit n
-nosizelimit
-attributesizelimit n
-noattributesizelimit
-[dont]dereferencealias
```

The following are applicable to Tcldish only:

```
-[no]preferchain
-[no]chaining
-[no]refer
-[dont]usecopy
-[no]localscope
-managedsait
-low
-medium
-high
```

```
-timelimit n
    Set the time limit to n seconds.
```

```
-notimelimit
    No time limit specified. This is the default time limit.
```

```
-sizelimit n
    Set the size limit to n entries. This can be increased up to the administrative limit (typically 200) set in the Directory Server which holds the entries. The attributes which hold the Directory Server administrative limits, adminSizeLimit, adminTimeLimit and adminLookthroughLimit, are described in Section E.1.1, “Administrative Limits”.
```

`-nosizelimit`
No size limit specified. This is the default size limit.

Note: Where there are a large number of entries, the number returned may be restricted by the Directory Server administrative limit mentioned above.

`-attributesizelimit n`
This indicates the largest size of any attribute (type and all its values) that can be included in returned entry information. The value *n* is taken to be the attribute's size in octets.

`-noattributesizelimit`
No size limit is set for returned attributes. This is the default attribute size limit.

`-[dont]dereferencealias`
(Do not) dereference aliases if found in the path of a query.

Note: Tcldish will not notify the user if an alias is de-referenced.

`-[no]preferchain`
Advise the Directory Server (not) to chain the operation if required. However, the Directory Server is allowed to ignore the advice, and return a referral.

`-[no]chaining`
(Prohibit) Allow the use of chaining.

`-[no]refer`
(Do not) automatically follow referrals issued by the Directory Server.

`-[dont]usecopy`
(Prohibit) Allow the use of a shadowed copy of the data.

`-[no]localscope`
(Do not) limit operation to local scope as defined by X.500. The interpretation of this flag is Directory Server dependent.

`-managedsait`
This flag enables an administrator to use Tcldish commands on objects in the Directory Server's own information tree, DSEs for example. This flag can only be used if you are bound as the DSA Manager.

`-low`
Flag the query as low priority. In the M-Vault Server, operations at this priority will give way to other operations more frequently.

`-medium`
Flag the query as medium priority. This is the default.

`-high`
This flag has two effects in the M-Vault Server. Non-local operations will return a referral rather than requiring the Directory Server to open a new association for chaining. Local operations at high priority will give way to others less frequently.

G.4.4 Read flags

The commands **dshowentry**, **dmoveto** and **dsearch** recognize the flags listed below.

`-[no]type attribute-type`
`-[no]all`
`-[no]value`
`-[no]show`
`-[no]key`

- type
flag requests that only the given attributes are read from the Directory Server. `-notype` does not necessarily prevent the attributes being read, but it does inhibit their display.
- all
flag requests that all attributes of an entry are read (default).
- value
flag reads the attribute value (this is the default). `-novalue` causes a read of the attribute types only.
- show
flag is used to make the DUA show the requested attributes (this is the default for read, but not for search).
- key
flag causes the attribute type (key) to be displayed when the value of an attribute is returned (e.g. `cn=` is shown in front of a `cn` value).

G.5 Commands for Directory operations

This section defines all the Tcldish commands for standard Directory operations.

G.5.1 dbind

```
dbind [-call Directory Server nickname or address]
      [-noauthentication]
      [-simple -user Distinguished Name -password password]
      [-protected] [-strong]
```

On startup (see [Section G.3, “Running Tcldish and Ltclish”](#)), Tcldish will bind to the default Directory Server defined in the system-wide (`SHAREDIR`)/`dsaptailor` or (`SHAREDIR`)/`ldaptailor` file by issuing a **dbind** command. Binding to the default Directory Server can be overridden by setting the `-call` flag on the command line:

`-call Directory Server nickname or address`

This flag can be used to connect to a Directory Server other than the default.

Directory Server nickname is the nickname assigned in the `dsaptailor` or `ldaptailor` file to the required Directory Server (see [Section G.2, “Tailoring Tcldish and Ltclish”](#)). Alternatively, you can bind to a Directory Server not defined in the tailor file by specifying *address*, the presentation address of the required Directory Server.

If you do not specify a Directory Server with `-call`, and you are already bound to a Directory Server, Tcldish will attempt to rebind to that Directory Server. If you are not currently bound (at start up or after issuing a **dunbind** command) Tcldish will contact the default Directory Server as set in the `dsaptailor` file.

The following examples show how the **dbind** command might be used:

- In Tcldish, this connects to the DAP Directory Server called `dsa2`. The `dsaptailor` file holds the Presentation Address for this Directory Server. In Ltclish, this connects to an LDAP server called `dsa2`. The `ldaptailor` file holds the URL for this Directory Server.

```
dbind -call dsa2
```

- The following example shows how you can connect to a DAP Directory Server which is not listed in the *dsaptailor* file:

```
dbind -call "Internet=x500.widget.com+19999"
```

- Similarly, the example below shows how you connect to an LDAP Server, ldap.widget.com which is not listed in the *ldaptailor* file:

```
dbind -call ldap://ldap.widget.com:19389/
```

When connecting to the Directory, the user must be authenticated. The flags described below control the level of authentication the DUA attempts to use. [Table G.1, “Mapping of command specifiers to commands”](#) summarises the effect when various combinations of these flags are set. **dbind** binds with no authentication if none of the flags below are set.

If the `-user` and `-password` flags are set, simple authentication is used, even if the `-simple` flag is not set. If none of these flags are used and simple authentication is desired (binding from information in the *.duarc* file, for example) then the `-simple` flag must be set.

`-noauthentication`

This indicates the lowest level of authentication, that is, anonymous. The Directory may respond with the message “inappropriate authentication”, which means a higher level of authentication is required.

`-simple`

This is the next level of authentication. The Directory uses the values given for user and password to authenticate the user. These values can be specified on the command line via the `-user` and `-password` flags, in the *.duarc* file, or, in the case of password, in answer to a prompt. See [Table G.2, “Setting authentication levels using Tcldish”](#) for the different ways of setting simple authentication.

`-user Distinguished Name`

This is the Distinguished Name of the user, and is required together with a password when binding to the Directory using simple authentication. If the `-user` flag is set but the `-simple` flag is not, simple authentication is implied.

`-password password`

This is a textual password. Encrypted passwords are not supported. If simple authentication is explicitly or implicitly requested, but no password is given in the command line or in *.duarc*, you are prompted to enter one.

Caution: Starting Tcldish with the `-password` flag on a multi-user system may reveal your password to other users; if they use the Unix **ps** system command, for instance. However, this is not an issue if the password is specified in the *.duarc* file, or given in answer to the password prompt.

`-protected`

(Tcldish only) This flag tells the DUA that all operations and results must be digitally signed. It can only be used if the `-strong` flag is also specified.

`-strong`

(Tcldish only) This flag tells the DUA to send strong authentication credentials to the Directory. Binding with strong authentication offers the highest level of security. When using strong authentication, the `-user` flag specifies the dn of a certificate in the user’s PKCS#12 file, and the `-password` flag specifies the passphrase necessary to decrypt the PKCS#12 file. See [Section 5.4, “Configuring the Directory for X.509”](#).

Table G.2. Setting authentication levels using Tcldish

Required auth level	-noauthentication	-simple	-user	-password	-strong	Comment
no authentication	not set	not set	not set	not set	not set	Default when no flags are set
no authentication	set	not set	not set	not set	not set	"No authentication" explicitly requested
simple	not set	set	set	set	not set	Password supplied
simple	not set	set	set	not set	not set	Prompted for password
simple	not set	set	not set	not set	not set	Attempts to read user and password from <i>.duarc</i>
simple	not set	not set	set	set	not set	Simple authentication assumed.
simple	not set	not set	set	not set	not set	Simple authentication assumed. Prompted for password.
strong	not set	not set	set	set	set	Certificate required

G.5.2 **dunbind**

```
quit
```

```
dunbind [-[no]quit]
```

This is used to break the connection to a Directory Server, and exit Tcldish.

-noquit

If this flag is specified, Tcldish unbinds from the current Directory Server but does not exit. A **dbind** command must be issued before Tcldish will be able to perform other Directory operations.

G.5.3 **dmoveto**

When you specify an object, the current position in the DIT is not changed. To change the current position you should use the command

```
dmove[to] [-[no]pwd] [-[no]check] object
```

-pwd

This flag tells **dmoveto** to print the current position in the DIT.

-nocheck

Normally, **dmoveto** invokes a read to check that the named entry exists. This flag inhibits such checking.

The current position can also be changed with the `-move` flag to **dshowentry** and **dlist** commands (see [Section G.5.4, “dshowentry”](#) and [Section G.5.6, “dlist”](#), respectively).

G.5.4 **dshowentry**

```
dshow[entry] [object] [-[no]name] [-[no]move]
[-modify[RightsRequest]]
[any of the read flags]
```

dshowentry will display some or all of the attributes of the specified entry. The read flags are used to specify which attributes to show.

`-name`

tells **dshowentry** to show the Distinguished Name of the requested entry as well as its attributes.

`-move`

is used to change the current position in the DIT to the object specified as well as showing the requested attributes.

`-modify`

is used to request modify rights. If modify rights are requested and the Directory Server permits returning them, they will be shown for the attributes the user has requested.

Modify rights of **none**, **add**, **remove** and **rename** are listed along with the requested entry attributes.

The modify rights for the entry as a whole are identified by:

```
modifyRights:entry
```

and the rights associated with a particular attribute in that entry are identified by:

```
modifyRights:attribute
```

where *attribute* is the name of the attribute.

G.5.5 dcompare

```
dcompare [object] [-[no]print] -attribute type=value
```

The **dcompare** command checks to see whether or not the entry holds a particular attribute value. If no object is specified the current object is checked.

`-attribute type=value`

This is the attribute type and value to check for.

`-[no]print`

If `-print` is specified, one of the strings `TRUE` or `FALSE` is printed. If `-noprint` is specified, 1 is returned instead of `TRUE`, and 0 is returned instead of `FALSE`. `-print` is the default setting.

An example of how this command might be used is:

```
dcompare -print -attribute sn=smith
```

G.5.6 dlist

```
dlist [object] [-[no]move] [-subentries]
[-sequence sequence name]
```

This command displays the Relative Distinguished Names of the subordinate entries below the current position in the DIT, or below *object*, if given. The number of subordinates returned can be limited by using the `-sizelimit` service control flag.

`-move`

is used to change the current position in the DIT to the object specified.

`-subentries`

is used to list subentries.

`-sequence sequence name`
stores the resulting entries in the sequence indicated by *sequence name*. For example:

```
dlist "c=US" -sequence US
```

lists the entries below the current position in the DIT, which have `c=US`, and stores these in a new sequence called `US`.

If the sequence already exists, the results will be added to it, otherwise a new sequence will be created. Note that using this option does not change the current default sequence. The **dsequence** command must be used to do this (see [Section G.6.1, “dsequence”](#)).

Reference numbers in this new sequence will start at 1. When the sequence listed is not the current sequence, the reference numbers appear in the form:

```
sequence name.reference number
```

For example:

```
Tcldish% dlist -sizelimit 4 -sequence GB
GB.1 organizationalUnitName=new
GB.2 commonName=Alan Jones
GB.3 commonName=Bill Smith
GB.4 commonName=Colin Green
(sizelimit exceeded)
```

If a sequence is made the current sequence using the **dsequence** command, only the reference numbers are displayed.

If this flag is not specified, the results are stored in the current default sequence.

The example below shows entries returned in response to a **dlist** request:

```
Tcldish% dlist -sizelimit 20
1 aMDMName=Telemail
2 organizationName=Advanced Decision Systems
3 organizationName=Anterior Technology
4 organizationName=Apple Computer, Inc.
5 organizationName=ATT
6 organizationName=Auburn University
7 organizationName=Bellcore
8 organizationName=Bucknell University
9 organizationName=Carnegie Mellon University
10 organizationName=Case Western Reserve University
11 organizationName=Citibank
12 organizationName=Clarkson University
13 organizationName=Columbia University
14 organizationName=Continuous Electron Beam Accelerator Facility
15 organizationName=Control Data
16 organizationName=Corporation for National Research Initiatives
17 organizationName=Cray Research Inc.
18 organizationName=CREN+stateOrProvinceName=District of Columbia
19 organizationName=Dana Farber Cancer Institute
20 organizationName=Defense Communications Agency (Limit problem)
Tcldish%
```

Note: The RDN identified by sequence number 18 is multi-valued.

G.5.7 dsearch

```
dsearch [[-object] object] [-subentries]
[-baseobject] [-singlelevel] [-subtree]
[[-filter] filter] [-matchedvaluesonly] [-[no]relative]
[-[no]searchaliases] [-sequence sequence name]
[-[no]partial]
[any of the read flags]
```

This command searches the DIT, starting at the object specified, for entries that match the given filter. When a matching entry is found, only its Distinguished Name relative to its position in the DIT is printed unless a `-show` read flag option is specified, in which case the attributes are displayed as well.

If no flags are given to a search command, then only one argument is allowed. This is taken to be the filter and not the base object as in all other commands.

The flags `-filter` and `-object` are supplied to allow the user to specify if both a filter and base object are required. (Note that only one need be flagged.) The following are all valid examples of search commands:

```
dsearch
dsearch filter
dsearch -filter filter
dsearch -filter filter -object object
dsearch filter -object object
dsearch -filter filter object
```

whereas the following are not valid:

```
dsearch object
dsearch filter object
```

The protocol standard provides a default filter which “includes all”. To request this default filter enter:

```
dsearch &
```

The format of filters differs for DAP and LDAP. In general, the examples given in this section are in DAP format. [Section G.5.7.1, “LDAP filters”](#) gives details on specifying filters for both protocols.

The first three flags are mutually exclusive.

`-baseobject`
searches only the current object.

`-singlelevel`
specifies all immediate children of the specified object should be searched. This is the default setting.

`-subtree`
searches the current position and all the levels below recursively.

For example:

```
dsearch -object "<o=Widget Ltd,c=GB>" -subtree \
-filter "sn=Beasley" -type telephoneNumber -show
```

will search the DIT subtree below **o=Widget Ltd,c=GB** for an entry with a surname exactly matching Beasley, and then display the **telephoneNumber** attribute.

`-matchedvaluesonly`

is used to limit the attribute values returned by show to those which match the values specified in the filter. Therefore, where the values specified in a filter are completely satisfied by part of a multi-valued attribute, only the attribute values which match those in the filter will be displayed.

For example, an entry contains the following two values for the **cn** attribute: “James Beasley” and “Jim Beasley”. A search without the `-matchedvaluesonly` flag:

```
dsearch -filter "cn=Jim*" -type cn -show
```

will display both values, “James Beasley” and “Jim Beasley”. A search which includes the `-matchedvaluesonly` flag:

```
dsearch -filter "cn=Jim*" -type cn -show -matchedvaluesonly
```

will return only the value “Jim Beasley”.

`-norelative`

is used to tell **dshowname** to print the full Distinguished Name of the results. Otherwise, the name relative to the current position is printed.

`-[no]searchaliases`

indicates whether aliases encountered in the search should be de-referenced and thus searched. This is different from the `-[dont]dereferencealias` service control flag, which defines what happens to aliases found when locating the base object of the search. `-[no]searchaliases` controls aliases encountered in the descendents of the base search object. The default is `-dereferencealias` and `-nosearchaliases`.

`-sequence sequence name`

stores the resulting entries in the sequence indicated by the sequence name specified. If the sequence already exists, the results will be added to it, otherwise a new sequence will be created. Note, this option does not change the current default sequence. The **dsequence** command must be used to do this (see [Section G.6.1, “dsequence”](#)).

Reference numbers in this new sequence will start at 1. As this is not the current sequence, reference numbers will be shown in the form:

```
sequence name.reference number
```

(See [Section G.2, “Tailoring Tcldish and Ltldish”](#) for an example of the reference number format.) If a sequence is made the current sequence using the **dsequence** command, only the reference numbers are displayed.

If this flag is not specified, the results are stored in the current default sequence.

The following example:

```
dsearch -object "<o=Widget Ltd,c=GB>" -subtree
-filter "sn=B*" -sequence B
```

searches the DIT subtree below **o=Widget Ltd,c=GB** for entries in which surname begins with B, and store the results in a new sequence called B.

`-[no]partial`

If the search is not able to do the entire search, for example because one of the Directory Servers to search could not be contacted, it will return partial results. If `-partial` is

set, `-show` is needed to see the full set of partial results, otherwise only a summary is printed. Unlike referral errors, the DUA does not follow these partial references. If `-nopartial` is set, no partial results will be returned.

G.5.7.1 LDAP filters

The filters used in LDAP and DAP are slightly different. This section explains some of these differences. In LDAP filters, the filter as a whole and each expression within the filter must be enclosed in brackets. In addition, operators precede the expression to which they refer. To use the first two filter examples above with `Ltclldish`, they would need to be in the form:

```
"(sn=Beasley)"
"(cn=Jim*)"
```

Full details on LDAP filters are given in *RFC 4515*.

G.5.7.2 Common filters

The following operators can be used to link expressions in both DAP and LDAP filters:

```
&
  logical AND
|
  logical OR
```

Parentheses () enforce the Boolean ordering of the expressions, otherwise the evaluation is left to right. The Boolean negation operator is `!`.

When searching for an exact match `=`, the `*` character is used as a wildcard, and a substring match takes place. If only `*` is specified, a presence match search is performed, and any entry containing that attribute will be matched. For example, `Tim*` will match any attribute value starting with the string “Tim” and `*Howes` any attribute value ending in the string “Howes”.

When substrings in a filter contain non-ASCII characters, each substring must be prefixed with the character set flag. For example:

```
-filter {cn={UCS-2}initial*{UCS-2}any*{UCS-2}final}
```

The `{UCS-2}` flags are needed if each of the substrings, `initial`, `any` and `final` contain non printable UCS-2 characters. If only the `initial` substring contains non printable UCS-2 characters, the filter can be expressed as:

```
-filter {cn={UCS-2}initial*any*final}
```

Note: A Directory Server can resolve searches of the form `string*` or `*string` significantly faster than searches of the `*string*` form.

As well as an exact equality match, `<=` or `>=` can be used, or an approximate match can be specified, by specifying `~=`.

If the filter expression is incomplete and does not specify the match, `Tclldish` will by default use an approximate match filter on the `cn` attribute. `Ltclldish` requires the filter is a complete LDAP filter string and will not use a default filter.

Here is a more complex example of a filter that searches for anyone who matches the following criteria:

- is a member of staff or a student
- has a drink attribute
- common name approximately matches “paul”
- surname is not “Jones”.

The DAP format of this filter is:

```
"cn~=paul & (userClass=staff|userClass=student) &
(!surname=Jones) & drink=*"
```

The LDAP format is:

```
"(&(cn~=paul)(|(userclass=staff)(userclass=student))
(!(surname=Jones))(drink=*))"
```

[Appendix C, Attribute Syntaxes](#) contains a list of all the attributes that can be used to search for an entry, together with a description of the type of matching allowed. Ensure that search filters have adequate quoting (see [Section G.1, “Tel and attribute syntax quoting”](#)).

G.5.8 dadd

```
dadd [object]
[-draft draft [-noedit]] [-template draft]
[-newdraft] [-objectclass objectclass]
```

dadd is used to add entries to the DIT. This invokes an editor on the draft entry. If there is not a draft entry specified, a draft entry of the specified object class is created. The draft is by default the *.dishdraft* in your home directory, but this name can be altered by using the `-draft` flag.

When editing has finished, an add operation is sent to the Directory if any changes have been made to the draft (the `-noedit` flag following a `-draft` stops the editor being invoked).

`-newdraft`
causes the current draft to be overwritten with a new template of the appropriate object class.

`-template`
is used to specify a template draft file that should be used during editing. The template file is not modified, it is copied to the draft file.

If the add completes successfully, the draft entry is renamed with a suffix of *.old*. If the add fails, the draft is left unchanged. If you issue a subsequent add, you will be asked if you wish to use the existing draft or create a new one.

Note: With the M-Vault Server, until the **prescriptiveACI** attribute of the access control subentry is modified, only the DSA Manager is permitted to add entries to the Directory Server.

G.5.9 ddelete

```
ddelete [object]
```

The specified entry is removed from the DIT.

Note: Only leaf entries (that is, entries, without children, at the bottom of the DIT) can be deleted using this command. The `dbulclean` utility, described in [Section G.7, “Bulk data utilities”](#), can be used to delete multiple leaf entries, or an entire subtree.

G.5.10 **dmodify**

```
dmodify [object]
[-draft draft [-noedit]] [-newdraft]
[-add attribute] [-remove attribute]
```

The **dmodify** command is used to modify existing entries in the DIT, and has two modes of operation. The first is used to modify specific attributes and uses the `-add` and `-remove` flags to add and remove single attributes and values. Many of these can be strung together in the same command, for example:

```
dmodify -add "description=new attribute" -remove "drink=Chocolate"
```

The second method is useful for altering many attributes at the same time. If the following is typed:

```
dmodify
```

this will get the current DIT entry and place a copy of it in the `.dishdraft` file. If the `-draft` option is specified, then the given file is used. After editing, any changes to the entry are sent to the Directory (as with adding an entry, the `-noedit` flag following a `-draft` stops the editor being invoked).

The draft file is handled in the same way as with **dadd**, except that when a new draft is created, the current values of the entry's attributes are read from the Directory.

G.5.11 **dmodifyrdn**

```
dmodifyrdn [object] -name attribute type=attribute value
[-[no]delete]
```

This is used to modify the Relative Distinguished Name (RDN) of an entry, as this cannot be changed using the modify operation.

Note: The M-Vault Server only allows this operation on leaf nodes.

```
-name attribute type=attribute value
```

The new RDN of the selected object. For example:

```
dmoveto "<cn=Bill Smith,ou=Research,o=Widget Ltd,c=GB>"
dmodifyrdn -name "cn=Phil Jones"
```

```
-nodelete
```

This flag is used to prevent the old RDN being removed as an attribute of the entry.

G.6 Other Tcldish commands

Tcldish also provides commands which do not map directly onto DAP/LDAP operations but are useful in scripts or for management purposes.

G.6.1 dsequence

```
dsequence [[-sequence] sequence name
[-all] [-reset] [-status]]
```

The results of a search or list operation are stored in a sequence, and each entry is given a reference number. Tcldish starts with a sequence called default as the current sequence. Results can be directed to sequences other than the current sequence by using the `-sequence` option in the **dlist** or **dsearch** commands. This option does not change the current sequence.

The **dsequence** command is used to change the current default sequence, and to view, name and reset sequences.

- Using the command without any options:

```
dsequence
```

returns the current sequence. The **dstatus** command also returns this information.

- To give the current sequence a different name use the following format:

```
dsequence [-sequence] sequence name
```

where `-sequence` is the flag indicating a sequence name follows. *sequence name* is the name to be assigned to the current sequence. This name must contain alphabetic characters only.

- The following options are used to obtain sequence status details, and/or to clear sequences:

sequence name

specifies the target sequence. If no sequence is specified, the current sequence is selected.

`-all`

The operation (reset or status) is applied to all user sequences.

`-reset`

removes all the Distinguished Names from the sequence(s), and frees any memory used by them. On subsequent operations using the sequence(s), reference numbers will restart from 1.

`-status`

gives the sequence name(s) and reports the number of entries in the sequence(s).

The following example shows the information output by this operation:

```
323 entries in sequence <GB>
40 entries in sequence <test>
1 entries in sequence *internal*
10 entries in sequence <default>
```

GB and test are user defined sequences. default is the default Tcldish sequence, and is the current sequence when Tcldish is started. *internal* shows how many Distinguished Names are being used by Tcldish. This sequence is not accessible by the user.

G.6.2 dshowname

```
dshowname [object] [-ufn]
```

dshowname will display the Distinguished Name of the current entry. A name is displayed in Internet format.

-ufn

This flag requests that the Distinguished Name be displayed in a User Friendly Naming style (as described in *RFC 1781*). For example:

```
Jim Beasley, Widget Ltd, GB
```

G.6.3 dstatus

```
dstatus [-user] [-syntax] [-dsa]
```

The command **dstatus** with no parameters will return information about the current bound state of Tcldish, and about the Directory Server it has contacted. For example:

```
Connected to : <cn=DSA,o=Widget Ltd,c=GB> at
             Internet=224.40.16.220
Current position : <o=Widget Ltd,c=GB>
User name : <cn=Manager,o=Widget Ltd,c=GB>
Current sequence : default
Authentication level : none
Signed ops : Not using signed ops
```

-user

Prints out the name under which the user is currently bound.

-syntax

This flag prints out a list of the syntaxes configured in the Directory Server. Syntaxes are listed in [Appendix C, Attribute Syntaxes](#).

-dsa

If connected to an M-Vault Server, Directory Server statistics can be printed using this flag. These are described in [Chapter 11, Monitoring the Directory](#).

G.6.4 dmanager

This command is only applicable when using DAP as the access protocol (via Tcldish).

```
dmanager -subentry object -grant -user object
```

```
dmanager -subentry object -revoke -user object
```

The **dmanager** command may only be used by the DSA Manager, interacting with an M-Vault Server.

-subentry object

This flag identifies the subentry to be modified. The subentry specified should be immediately subordinate to a naming context mastered by this Directory Server, and can be found using the **dlist** command with the **-subentries** flag.

-grant

Authorization to act as a DSA Manager is to be given to the user identified by the argument to the **-user** flag. That user will be able to add, modify and remove entries, but will not be able to use the **dmanager** command.

-revoke

This flag revokes Directory management authorization for the user identified by the argument to the **-user** flag. Only the DSA Manager may revoke authorization.

-user object

The target user for granting or revoking Directory management authorization.

An example of using this command might be:

```
dmoveto "<ou=Research,o=Widget Ltd,c=GB>"
dmanager -subentry "cn=ac-subentry" -grant -user "cn=DSA Manager"
```

G.7 Bulk data utilities

There are three Tcldish commands which provide facilities for managing bulk data in the Directory: **dbulclean**, **dbulkload** and **dbulkdump**:

dbulclean

is a useful utility for clearing out all entries below a specified location. You would typically run this utility, to clear out old entries, before running the **bulkload** utility to add new entries. You can find more details of this command in [Section G.7.2, “The dbulclean command”](#).

dbulkload

is for loading multiple entries from a data file into the Directory. The data file format can be LDAP Data Interchange Format (LDIF) or Comma Separated Value (CSV). You can find more details of this command in [Section G.7.1, “The Tcldish dbulkload command”](#).

dbulkdump

allows you to write Directory entries to an LDIF file. Using this utility and the **dbulkload** utility with LDIF files provides a useful facility for backing up your Directory data, or moving entries in a Directory subtree to a different system. You can find more details of this command in [Section G.7.3, “The dbulkdump command”](#).

G.7.1 The Tcldish dbulkload command

The Tcldish dbulkload command has two formats, depending on the format of the file holding the data to be loaded: one for CSV files and one for LDIF.

Although data stored in CSV format is not as portable to other systems as the LDIF format, CSV has the following advantages:

- CSV is simple to generate from database programs.
- DNs can be specified as relative DNs, so it is possible to load entries into a different location in the Directory. In LDIF files, absolute DNs are specified making it more difficult to load data into a location other than the one for which it was originally created.

LDIF files have the following advantages:

- LDIF is a widely accepted format for importing and exporting Directory information between LDAP or X.500 servers.
- LDIF format is portable between different systems.
- Entries for an entire subtree can be described in an LDIF file. A CSV file can describe the entries for one level only.
- LDIF files can provide a simple Directory data backup facility.

G.7.1.1 The Tcldish dbulkload command for CSV data

The format of the Tcldish dbulkload command for CSV data is:

```
dbulkload [object] [-[no]overwrite]
-csvdata filename -template filename
[-rdn [attribute type...]] [-[no]usefirst]
[any service control flag] [-help]
```

Where:

object

specifies the Directory location under which the data is to be loaded. The default value is the current location in the Directory.

`-[no]overwrite`

specifies whether **dbulkload** should try to overwrite existing entries. `-nooverwrite` reports an error if it tries to add an entry which already exists. `-overwrite` is the default.

The `-nooverwrite` option involves fewer operations per entry than `-overwrite`. Therefore, if efficiency is a concern, run **dbulclean** (see Section F.7.2 on page 328) to clear out existing entries and then run **dbulkload** with the `-nooverwrite` option.

`-csvdata data filename`

specifies a CSV file (see Section G.7.1.1.1, “The CSV data file”) which contains attribute values for the entries to be added.

`-template template filename`

specifies the file to be used as a template file (see Section G.7.1.1.2, “The CSV template file”) to translate data entries into the required format for adding to the Directory.

`-rdn [attribute type [attribute type ...]]*`

is a list of attributes which form the Relative Distinguished Name of the entries to be added. If no attributes are specified, **cn** is selected by default. Multi-valued RDNs can be specified.

`-[no]usefirst`

If there are multiple values for an attribute type which is to be included in the RDN, the `-usefirst` flag directs **dbulkload** to use the first value in the template file for the attribute type. Thus if a data file line included two values for **cn**:

```
Adam Alexander, A.N. Alexander...
```

and you wanted to use the second value as the RDN, the template file should be set up something like:

```
objectClass= top
objectClass= person
objectClass= organizationalPerson
```

```
cn= $2
cn= $1
```

You would then run **dbulkload** with the arguments:

```
dbulkload -csvdata filename -template filename -rdn cn -usefirst
```

The default value, `-nousefirst`, indicates that if multiple values exist for an RDN attribute type, an error is to be generated.

`-help`
displays help information.

G.7.1.1.1 The CSV data file

The CSV data file is a standard comma separated value file, that is, each entry is on a new line and values are delimited by a comma. The following is a CSV formatted data file.

```
Adam Alexander, Alexander,555-0442, aa@Widget.com, aa, {FILE}/path/to/adam.jpg
Bob Brown, Brown, , bb@Widget.com, bb, {FILE}/path/to/bob.jpg
Carl Carpenter, Carpenter,555-0443, , cc,{FILE}/path/to/carl.jpg
Dave Davenport, Davenport,555-0446, dd@Widget.com, dd, {FILE}/path/to/dave.jpg
Eric Edmonson, Edmonson, 555-0448, ee@Widget.com,, {FILE}/path/to/eric.jpg
Fred Ford, Ford, 555-0449, ff@Widget.com, ff, {FILE}/path/to/fred.jpg
Geoff Green, Green, 555-0388, gg@Widget.com, gg, {FILE}/path/to/geoff.jpg
```

A value containing a comma or quote character should be surrounded by quotes. A literal quote is specified by two quotes in a row. For example:

```
employee,"Ford "Fred""","3rd floor, room 12",0449
```

has the following four fields:

- employee
- Ford "Fred"
- 3rd floor, room 12
- 0449

G.7.1.1.2 The CSV template file

The CSV template file is a simple EDB format file. The example below shows the contents of a sample template file for the data given [Section G.7.1.1.1, “The CSV data file”](#).

```
objectClass= top
objectClass= person
objectClass= organizationalPerson
objectClass= inetOrgPerson
sn= $2
cn= $1
telephoneNumber= $3
postalAddress= Widget Ltd. $ 5293 Butchers Lane $ York $ GB
rfc822Mailbox= $4
userid= $5
jpegPhoto= $6
```

In this file, a line is given in the format:

```
attribute= $number
```

where *number* denotes the field of the data file entry to be substituted. The line should contain one substitution only, and no additional text. For example: `sn= $2` indicates that the second field of each entry in the data file is to be added as a **sn** attribute.

Fields in the data file can be reused or even ignored; for example, you could specify `userid= $2`, to make the **userid** attribute the same as the **sn** attribute.

Attributes which are common to all entries, **postalAddress** in this example, can be included in the template file, and omitted from the data file.

G.7.1.1.3 Examples of interactive bulk loading

dbulkload can be run from Tcldish or Ltclidish. As loading from an LDIF file is straightforward, examples are not included in this section. Instead, the following examples use the CSV sample data and template files given above.

```
$ dbulkload -csvdata /path/to/data.csv
  -template /path/to/template -rdn cn

Added <cn=Adam Alexander,o=Widget Ltd,c=GB>
Added <cn=Bob Brown,o=Widget Ltd,c=GB>
Added <cn=Carl Carpenter,o=Widget Ltd,c=GB>
Added <cn=Dave Davenport,o=Widget Ltd,c=GB>
Added <cn=Eric Edmonson,o=Widget Ltd,c=GB>
Added <cn=Fred Ford,o=Widget Ltd,c=GB>
Added <cn=Geoff Green,o=Widget Ltd,c=GB>
```

If the command is issued a second time (with the default, `-overwrite`), notices will be issued for both add and remove operations:

```
$ dbulkload -csvdata /path/to/data.csv
  -template /path/to/template -rdn cn

Removed <cn=Adam Alexander,o=Widget Ltd,c=GB>
Added <cn=Adam Alexander,o=Widget Ltd,c=GB>
Removed <cn=Bob Brown,o=Widget Ltd,c=GB>
Added <cn=Bob Brown,o=Widget Ltd,c=GB>
Removed <cn=Carl Carpenter,o=Widget Ltd,c=GB>
Added <cn=Carl Carpenter,o=Widget Ltd,c=GB>
Removed <cn=Dave Davenport,o=Widget Ltd,c=GB>
Added <cn=Dave Davenport,o=Widget Ltd,c=GB>
Removed <cn=Eric Edmonson,o=Widget Ltd,c=GB>
Added <cn=Eric Edmonson,o=Widget Ltd,c=GB>
Removed <cn=Fred Ford,o=Widget Ltd,c=GB>
Added <cn=Fred Ford,o=Widget Ltd,c=GB>
Removed <cn=Geoff Green,o=Widget Ltd,c=GB>
Added <cn=Geoff Green,o=Widget Ltd,c=GB>
```

If the command is issued again, with `-nooverwrite` specified, errors are reported, since the entries already exist from the previous operations:

```
$ dbulkload -csvdata /path/to/data.csv
  -template /path/to/template -rdn cn -nooverwrite

Unable to add <cn=Adam Alexander,o=Widget Ltd,c=GB>
:*** Update error - Already exists ***
Unable to add <cn=Bob Brown,o=Widget Ltd,c=GB>
:*** Update error - Already exists ***
```

```

Unable to add <cn=Carl Carpenter,o=Widget Ltd,c=GB>
::*** Update error - Already exists ***
Unable to add <cn=Dave Davenport,o=Widget Ltd,c=GB>
::*** Update error - Already exists ***
Unable to add <cn=Eric Edmonson,o=Widget Ltd,c=GB>
::*** Update error - Already exists ***
Unable to add <cn=Fred Ford,o=Widget Ltd,c=GB>
::*** Update error - Already exists ***
Unable to add <cn=Geoff Green,o=Widget Ltd,c=GB>
::*** Update error - Already exists ***

```

The next example shows the inclusion of multi-valued entries, combining the attributes **cn** and **userid**:

```

$ dbulkload -csvdata /path/to/data.csv
  -template /path/to/template -rdn cn userid

Added <userid=aa+cn=Adam Alexander,o=Widget Ltd,c=GB>
Added <userid=bb+cn=Bob Brown,o=Widget Ltd,c=GB>
Added <userid=cc+cn=Carl Carpenter,o=Widget Ltd,c=GB>
Added <userid=dd+cn=Dave Davenport,o=Widget Ltd,c=GB>
data.ex:5 error processing record: entry doesn't have the rdn attribute userid
Added <userid=ff+cn=Fred Ford,o=Widget Ltd,c=GB>
Added <userid=gg+cn=Geoff Green,o=Widget Ltd,c=GB>

```

Here, an error is reported on line 5 of the data file, because that line did not contain a value in the “userid” field, which is required when it is an RDN attribute. It is also important to note that the entries added by previous **dbulkload** calls were not removed. The RDN of these entries specified the **cn** attribute on its own, which is considered to be different from the multi-valued RDN of **userid+cn**.

G.7.1.1.4 Running bulk data utilities from a Tclish script

The bulk data utilities can be used interactively, or as a part of a simple Tclish script. This section contains a sample Tclish script which bulkloads data into the DIT location, **ou=Marketing,o=Widget Ltd,c=GB**, from a CSV data file. You would need to set the variables **BASE** and **OU** to the required Directory location where the data is to be loaded, and the **CSVDATA** and **TEMPLATE** variables to the correct paths for the data and template files.

```

#!/opt/isode/bin/tclish -f
# allow username to be specified in the command line
eval [concat dbind $argv]

set BASE "<o=Widget Ltd,c=GB>"
set OU "ou=Marketing"
set CSVDATA /path/to/data.csv
set TEMPLATE /path/to/template

proc bulkload_ou {base ou csvdata template} {
  # move to the level where we want to create the ou
  puts "Moving to $base"
  dmoveto $base

  # create a draft for the ou
  set filename [glob ~]/.dishdraft
  set fd [open $filename w]

  puts $fd "objectclass= top"
  puts $fd "objectclass= organizationalUnit"

  close $fd
}

```

```

# add the ou and go there
puts "Creating $ou"
dadd $ou -draft $filename -noedit
dmoveto $ou

# perform the bulkload
puts "Bulkloading entries into $ou"
puts [dbulkload -csvdata $csvdata -template $template -overwrite]
}
puts "Script started at [exec date]"
puts "-----"

set command [list bulkload_ou $BASE $OU $CSVDATA $TEMPLATE]
catch $command result
puts $result

puts "-----"
puts "Script ended at [exec date]"

```

G.7.1.2 The Tcldish dbulkload command for LDIF data

Note: **dbulkload** currently supports LDIF entry specification records, but not LDIF change records, which describe a set of changes to be applied to the Directory.

The format of the **dbulkload** command for LDIF data is as follows:

```

dbulkload [object] [-[no]overwrite]
-ldif filename
[any service control flag] [-help]

```

object

specifies the Directory location under which the data is to be loaded. The default value is the current location in the Directory.

-[no]overwrite

specifies whether **dbulkload** should try to overwrite existing entries. **-nooverwrite** reports an error if it tries to add an entry which already exists. **-overwrite** is the default.

The **-nooverwrite** option involves fewer operations per entry than **-overwrite**. Therefore, if efficiency is a concern, run **dbulkclean** (see [Section G.7.2, “The dbulkclean command”](#)) to clear out existing entries and then run **dbulkload** with the **-nooverwrite** option.

-ldif *filename*

defines the pathname of the LDIF file containing the attribute specification records.

-dontdereferencealias

is the service control flag. **dbulkload** sets **-dontdereferencealias** as the default.

Note: For most commands, the default for this service control is **-dereferencealias**.

G.7.2 The dbulkclean command

The **dbulkclean** command removes (recursively) all entries below the current entry. The format of the **dbulkclean** command is:

```
dbulclean [object] [-[no]prompt] [-[no]base] [-[no]below]
[-sequence sequence name]
[any service control flag] [-help]
```

Where:

object

The default value is the current location in the Directory.

`-[no]prompt`

is a flag to specify whether Tcldish should issue an initial prompt, before removing the entries. For non-interactive use `-noprompt` should be specified. `-prompt` is the default.

`-[no]base`

If `-base` is specified, the entry specified by *object* or entries specified by *sequence name* will be removed. `-nobase`, the default, does not delete these entries.

`-below`

All entries below the entry specified by *object* or entries specified by *sequence name* will be (recursively) removed. This flag is set by default.

`-sequence sequence name`

allows the **bulkclean** operation to act on all entries in a given sequence rather than an *object* specified on the command line. To ensure that the actual entries in the sequence are removed, in addition to the entries below them, use both the `-base` and `-sequence` flags.

`-help`

displays help information.

On completion, **dbulclean** reports the following information:

- How many entries were deleted.
- How many entries could not be deleted.
- Any conditions which prohibited the command from being able to find all the entries which should have been deleted.

G.7.3 The dbulkdump command

```
dbulkdump [object] [-sequence sequence name] [-file filename]
[-[no]header] [-[no]base] [-[no]below]
[any service control flag] [-help]
```

The **dbulkdump** command writes entries from the Directory to a file or standard output in LDIF format.

object

The location in the DIT from which entries are to be dumped. If neither an object nor a sequence is specified, the current location in the DIT is used.

`-sequence sequence name`

The name of the sequence of entries to be dumped.

`-file filename`

The file to which the entries will be written in LDIF format. If a file is not specified, the entries will be output to the standard output device.

`-[no]header`

This flag controls whether the LDIF header information (currently only a version specifier) is output. `-header` is the default setting.

-[no]base

The default setting, `-base`, causes the entry specified by *object* or entries specified by *sequence name* to be dumped.

-[no]below

The default setting, `-below`, dumps the entire subtree below the specified location. To dump a single level of the DIT, use:

```
dlist -sequence sequence name
dbulkdump -sequence sequence name -nobelow
```

dlist holds the entries in a sequence, and **dbulkdump** dumps the sequence.

When dumping a large number of entries, the limit on the number of entries which can be returned may be exceeded. If the Tcldish `-sizelimit` service flag is defaulted, the maximum number of entries is controlled by the **adminSizeLimit** and **adminLookthroughLimit** attributes in the GDAM entry. If a limit is encountered, not all entries will be dumped, and an error will be logged in the output.

Appendix H Dmish Scripting Interface

This chapter describes the scripting interface **Dmish**. The Directory Management Shell (**Dmish**) is an extended Tcl (Tool Command Language) shell for use by Directory administrators and systems integrators.

Dmish does not require a graphics display to operate and can also be used in batch mode. Because it is a Tcl application, the user has all the features of the Tcl interpreter available for writing scripts. The following operations are available for displaying and updating managed objects:

- **List** – list the objects in the Directory Server.
- **Add** – add a managed object to the Directory Server.
- **Modify** – modify an object in the Directory Server.
- **Remove** – remove an object from the Directory Server.

These operations can be applied to the supported managed object types. Additionally, there are commands to:

- Create a Directory Server.
- Start a Directory Server.
- Stop a Directory Server.
- Open a management connection with a Directory Server.
- Close a management connection with a Directory Server.
- Set or display the base of a subtree for the list commands.
- Show a single managed object.

H.1 Using Dmish

When creating new Directory Servers, or starting or stopping local Directory Servers, the Server Manager should login as the user of the Directory Server account before running the **Dmish** tool, as the Directory Server account must be the owner of the Directory Server configuration directory.

Under Unix, **Dmish** is started by running *dmish* which will have been installed into (*SBINDIR*).

Under Windows, run the executable file *dmish.exe*, which will have been installed by default into the folder (*SBINDIR*) on the installation drive.

When running, **Dmish** uses the following prompt:

```
dmish%
```

H.2 Creating a new Directory Server

A new Directory Server can be created on a local host machine using the **dmi create** command.

Note: While this command is available deployers should note that servers created in this way are not initialized with any form of access control, and we recommend use of M-Vault Console or dsa-setup which employ templating mechanisms to create a practical and fully initialized DSA.

```
dmi create -dsa_dir path
-name dnstring
-address addrstring
-password password
-prefix dnstring
[ -superior_name dnstring -superior_address addrstring ]
| -help
```

H.2.1 Arguments

- dsa_dir *path*
sets the pathname of the file system directory in which the new Directory Server will run and create its database files and subdirectories.
- name *dnstring*
sets the new Directory Server's own name to the Distinguished Name with string representation *dnstring*.
- address *addrstring*
sets the address of the new Directory Server to the Presentation Address with string representation *addrstring*.
- password *password*
sets the manager password of the new Directory Server to the string *password*.
- prefix *dnstring*
sets the context prefix of the primary naming context that the new Directory Server will master to the Distinguished Name with the string representation *dnstring*. This variable is mandatory.
- superior_name *dnstring*
sets the name of the superior reference to the Distinguished Name with the string representation *dnstring*.
- superior_address *addrstring*
sets the address of the superior reference Directory Server to the Presentation Address with the string representation *addrstring*.
- help
displays a synopsis of the commands available.

H.2.2 Result

On successful creation of the new Directory Server this command returns an empty Tcl string.

H.2.3 Errors

If the Directory Server cannot be created or if any other failure occurs, this command provides an error describing the problem.

H.3 Starting a Directory Server

A Directory Server that is not already running can be started using the following command:

```
dmi start [-dsa_dir path] | -help
```

H.3.1 Arguments

`-dsa_dir path`

specifies the pathname of the file system directory in which the Directory Server was created and will run. If this switch and argument are not supplied a default of (*DSADIR*) is used.

`-help`

displays a synopsis of the commands available.

H.3.2 Result

On successfully starting the Directory Server, the command returns an empty Tcl string.

H.3.3 Errors

On failing to start the Directory Server for any reason, the command provides an error message describing the problem.

H.4 Stopping a Directory Server

A Directory Server that is already running can be stopped using the following command:

```
dmi stop [-dsa_dir path] | -help
```

H.4.1 Arguments

`-dsa_dir path`

specifies the pathname of the file system directory in which the Directory Server was created and is running. If this switch and argument are not supplied a default of (*DSADIR*) is used.

`-help`

displays a synopsis of the commands available.

H.4.2 Result

On successfully stopping the Directory Server, the command returns an empty Tcl string.

H.4.3 Errors

On failing to stop the Directory Server for any reason, the command provides an error message describing the problem.

H.5 Opening a management connection

To open a management association with an available Directory Server, use the following command:

```
dmi open [-call dsa] -user dn -password password
[ -simple | -strong [-protected] ]
| -help
```

H.5.1 Arguments

`-call dsa`

specifies the name of the Directory Server to be managed. The argument may be a Directory Server name configured in *dsaptailor*, or a Directory Server Presentation Address. If this switch is not supplied it defaults to the first Directory Server address configured in *dsaptailor*.

`-user dn`

is the Distinguished Name of the DSA Manager.

`-password password`

is the password for the DSA Manager. In the case of strong authentication, this is the passphrase used to encrypt the PKCS#12 file containing the certificate and private key corresponding to the user dn. [Section 3.10, “Managing identities”](#).

`-simple`

indicates that simple authentication is to be used.

`-strong`

indicates that strong authentication is to be used.

`-protected`

indicates that signed operations are to be used.

`-help`

displays a synopsis of the commands available.

H.5.2 Result

On success, this command returns a Directory Server managed object. When run interactively, the Tcl interpreter prints the result to standard output. The name of this object is also stored in the global variable `DMIID` and becomes the default connection for future **Dmish** commands to which the `-id` argument is not supplied.

H.5.3 Errors

This command propagates any error from `dmiLib` and provides its own error for any other failure.

H.6 Closing a management connection

When a connection to the Directory Server is no longer required it should be closed using the following command:

```
dmi close [-id id] | -help
```

This releases resources in both the management application and in the Directory Server being managed. After this command has completed successfully, no further commands may be issued on that connection. If the current default connection is closed, the variable `DMId` is set to a special value meaning that there is no longer any default.

H.6.1 Arguments

`-id id`

if this argument is supplied, *id* identifies the management connection to be closed. If it is omitted, the connection returned by the last successful open command is closed.

`-help`

displays a synopsis of the commands available.

H.6.2 Result

On success this command returns the empty Tcl string.

H.6.3 Errors

On failure this command provides an error message describing the problem.

H.7 Manipulating managed objects

The following commands operate over an established management association with a Directory Server.

Note: The implementation is currently limited to a single management connection at a time.

H.7.1 Types of objects that can be managed

The following switches are used in order to identify the types of object that can be managed:

- `-dsa` manage Directory Server objects.
- `-db` manage database (GDAM) objects.
- `-nc` manage naming context objects.
- `-adm` manage administrative point objects.
- `-subr` manage subordinate reference objects.

- `-xr` manage cross-reference objects.
- `-nssr` manage non-specific subordinate reference objects. This option is currently not supported.
- `-supr` manage superior reference objects.
- `-supplier` manage shadow supplier objects.
- `-consumer` manage shadow consumer objects.

H.7.1.1 Managing indexes

Indexes can also be managed as objects in **Dmish**, with the following limitations:

- The **list** command does not retrieve index objects.
- You cannot **modify** index objects – you must remove the index and create a new one.
- You can **add** multiple indexes at once, but you can only remove a single index at a time. For this reason, the index managed object makes use of two different switches:
 - `-indexes` add index objects
 - `-index` perform other operations on index objects, i.e. show, remove.

H.7.2 Default list base

The following command sets or displays the default base object for the list command:

```
dmi list [-id id] [dnstring] | -help
```

H.7.2.1 Arguments

`-id id`

if this argument is supplied, *id* identifies a management connection previously returned by the open command whose default is to be set. If it is omitted, the connection returned by the last successful open command is used.

`dnstring`

if this argument is supplied it must be the string representation of a Distinguished Name to which the default base will be set. If this argument is omitted, the current default base is returned.

`-help`

displays a synopsis of the commands available.

H.7.2.2 Results

On success, this command returns the string DN of the default base that is in effect when the command completes.

H.7.2.3 Errors

On failure, this command provides an error message describing the problem. Possible errors include an invalid argument to the `-id` switch or an invalid `dnstring`.

H.7.3 List managed objects

Managed objects can be retrieved from the Directory Server and displayed using the following command:

```
dmi list [-id id] [-base dnstring] [type] [ -show | -noshow ]
| -help
```

H.7.3.1 Arguments

`-id id`

if this argument is supplied, *id* identifies a management connection previously returned by the open command to which the operation should be applied. If it is omitted, the connection returned by the last successful open command is used.

`-base dnstring`

if this argument is supplied it must be the string representation of a Distinguished Name from which the base object for the list operation is generated. If this argument is omitted, the current default base as set by the base command is used.

`type`

by default, information about all types of managed objects are returned, except for indexes. This can be constrained to managed objects of particular kinds by specifying one or more of the switches in [Section H.7.1, “Types of objects that can be managed”](#).

By default, only the name and type of each managed object are returned. All attributes can be returned by making the appropriate selection below:

`-show`

show all attributes.

`-noshow`

show only the name and type of the managed object (the default).

`-help`

displays a synopsis of the commands available.

H.7.3.2 Results

On success, this command returns a sequence number, the type, name, and properties of each managed object found, as a string formatted similar to a Tclish search result. Managed objects are separated in the result by a blank line following each managed object except the last. Each managed object begins with a line of the format:

```
Sequence-number MO-type : MO name
```

The properties of the managed object are listed next if the `-show` option was selected, one per line, as though they were attributes of an entry:

```
Property-type = Property value
```

The property values are in the format defined by the string command of the type used to represent the corresponding managed object property in `dmilib`. This is commonly the string format for a Directory attribute syntax used to store the property. The sequence number may be used with the **show** command to identify that specific managed object, without having to supply the managed object identification. When run interactively the Tcl interpreter prints the result string to standard output.

H.7.3.3 Errors

On failure, the command provides an error message describing the problem.

H.7.4 Show managed objects

A specific managed object can be retrieved from the Directory Server and displayed using the following command:

```
dmi show [-id id] type name | sequence-number | -help
```

H.7.4.1 Arguments

`-id id`

if this argument is supplied, *id* identifies a management connection previously returned by the open command to which the operation should be applied. If it is omitted, the connection returned by the last successful open command is used.

`type`

a managed object is identified by its type which can be chosen by selecting one of the switches in [Section H.7.1, “Types of objects that can be managed”](#) and its name, or a sequence number generated by the list command.

`name`

if this argument is supplied it must be the string representation of the Distinguished Name of the entry.

`sequence-number`

is a number listed by the **list** command.

`-help`

displays a synopsis of the commands available.

Note: All attributes of the managed object are returned.

H.7.4.2 Results

On success, this command returns the type, name, and properties of the managed object as a string formatted similar to a **Tcl** search result. The managed object begins with a line of the format:

```
MO-type : MO name
```

The properties of the managed object are listed next, one per line, as though they were attributes of an entry:

```
Property-type = Property value
```

The property values are in the format defined by the string command of the type used to represent the corresponding managed object property in `dmilib`. This is commonly the string format for a Directory attribute syntax used to store the property. The sequence number may be used with the show command to identify that specific managed object, without having to supply the managed object identification. When run interactively the **Tcl** interpreter prints the result string to standard output.

H.7.4.3 Errors

On failure, the command provides an error message describing the problem.

H.7.5 Add managed object

A new managed object can be added to the Directory Server using the following command:

```
dmi add [-id id] type
[-draft draft]
[-newdraft]
[-template file]
[-noedit]
[-editor prog]
| -help
```

This command creates a draft property file for the new managed object and displays it in an editor. The draft contains a line for each property of the managed object in the format described for the list command result except that the property values are absent. When the user has finished editing the properties they save the draft and quit the editor. **Dmish** then constructs a new managed object with the properties contained in the draft file and adds it to the Directory Server.

H.7.5.1 Arguments

- id *id*
if this argument is supplied, *id* identifies a management connection previously returned by the open command to which the operation should be applied. If it is omitted, the connection returned by the last successful open command is used.
- type*
this argument specifies the type of managed object to be added. It must be one of the managed object type switches described in [Section H.7.1, “Types of objects that can be managed”](#). For index objects, see [the section called “Adding one or more index objects”](#).
- draft *draft*
if this argument is supplied, *draft* is the pathname of the draft file to use. Otherwise, a file called *.dmidraft* in the user’s home directory is used.
- newdraft
normally, if the draft file already exists when the add command is issued, the user is asked whether they wish to use the existing file and contents. This is useful for correcting errors in the draft after a failed add command. If this argument is supplied, an existing draft file is overwritten by the new template without prompting.
- template *file*
normally the add command creates the draft file with a set of properties appropriate to the type of managed object. If this argument is supplied the draft file is created from *file*.
- noedit
if this argument is supplied, the editor is not invoked on the draft file before attempting to add the new managed object. This is useful if there is already an existing draft file containing the desired properties, or in conjunction with the `-template` switch for adding prepared managed objects.
- editor *prog*
normally the name of the editor program to use is taken from the user’s EDITOR environment variable if it exists, or else a system default is used. If this switch is supplied, the program *prog* is used instead.
- help
displays a synopsis of the commands available.

H.7.5.2 Results

On success, this command deletes the draft file and returns the empty Tcl string.

H.7.5.3 Errors

On failure, this command does not delete the draft file and provides an error message describing the problem.

H.7.5.4 Adding one or more index objects

Note: The index is the only object of which multiples can be created at once.

To indicate this, the *type* used to add an index is `-indexes`. Therefore, the command to add an index would be:

```
dmi add [-id id] -indexes
[-draft draft]
[-newdraft]
[-template file]
[-noedit]
[-editor prog]
| -help
```

When you enter this command, you are asked to complete three fields: the name of the database the index or indexes are to be created for; the index name(s); and the index file, if appropriate. The index name is entered as the abbreviated attribute name and index type, separated by a colon; for example, to create an equality index on the surname attribute, you would enter:

```
index_name = sn:e
```

The substring and approximate index types are represented by *s* and *a* respectively. To create more than one index, enter more index names on the same line separated by spaces; for example, to create both a substring and equality index for the surname attribute, you would enter:

```
index_name = sn:s sn:e
```

H.7.6 Modify managed object

The properties of an existing managed object in the Directory Server can be modified using the following command:

```
dmi modify [-id id] type name | sequence-number
[-draft draft]
[-newdraft]
[-noedit]
[-editor prog]
| -help
```

This command reads the managed object with the specified type and name from the DSA, creates a draft property file, and displays it in an editor. The draft contains a line showing the existing value of each managed object property in the format described for the list command. The user modifies the properties by editing the values, saving the file, and quitting the editor. **Dmish** then constructs a managed object containing the new property values and modifies it in the Directory Server.

H.7.6.1 Arguments

-id id

if this argument is supplied, *id* identifies a management connection previously returned by the open command to which the operation should be applied. If it is omitted, the connection returned by the last successful open command is used.

type

this argument specifies the type of managed object to be modified. It must be one of the managed object type switches described in [Section H.7.1, “Types of objects that can be managed”](#).

Note: You cannot modify an index object; you must remove the index and create a new one.

name

if this argument is supplied it must be the string representation of the Distinguished Name of the entry.

sequence-number

is a number listed by the **list** command.

`-draft draft`

if this argument is supplied, *draft* is the pathname of the draft file to use. Otherwise, a file called *.dmidraft* in the user's home directory is used.

`-newdraft`

normally, if the draft file already exists when the modify command is issued, the user is asked whether they wish to use the existing file and contents. This is useful for correcting errors in the draft after a failed modify command. If this argument is supplied, an existing draft file is overwritten by the new template without prompting.

`-noedit`

if this argument is supplied, the editor is not invoked on the draft file before attempting to add the new managed object. This is useful if there is already an existing draft file containing the desired properties, or in conjunction with the `-draft` switch for modifying prepared managed objects.

`-editor prog`

normally the name of the editor program to use is taken from the user's `EDITOR` environment variable if it exists, or else a system default is used. If this switch is supplied, the program *prog* is used instead.

`-help`

displays a synopsis of the commands available.

H.7.6.2 Results

On success, this command deletes the draft file and returns the empty Tcl string.

H.7.6.3 Errors

On failure, this command does not delete the draft file and provides an error message describing the problem.

H.7.7 Remove managed object

An existing managed object can be removed from the Directory Server using the following command:

```
dmi remove [-id id] type name | sequence-number | -help
```

H.7.7.1 Arguments

`-id id`

if this argument is supplied, *id* identifies a management connection previously returned by the open command to which the operation should be applied. If it is omitted, the connection returned by the last successful open command is used.

type

this argument specifies the type of managed object to be removed. It must be one of the managed object type switches described in [Section H.7.1, "Types of objects that can be managed"](#)

name

this argument specifies the name of the managed object to be removed.

sequence-number

is a number listed by the **list** command.

`-help`
displays a synopsis of the commands available.

H.7.7.2 Results

On success this command returns the empty Tcl string.

H.7.7.3 Errors

On failure, the command provides an error message describing the problem.

H.8 Deleting entire subtrees

The `dbulk` tool can be used in clean mode to delete entire subtrees.

Caution: Read the caveats given in [Section 4.8.2.1](#), “Using `dbulk` to Import and Export Data”.

H.8.1 Using `dbulk clean`

To remove a subtree of the database, enter the following command line:

```
dbulk clean baseDN -db_directory pathname
[-maxfail n] [-force] [-descendants]
```

Note: The line is folded here for clarity.

The meanings of the parameters are as follows:

baseDN

This identifies the subtree to be removed (cleaned). The root entry may not be removed, hence the base may be "" to indicate the root, only in conjunction with `-descendants`.

`-db_directory pathname`

This specifies the location of the GDAM database, for example,
`/var/isode/dsa-db/gdam1`

`-maxfail n`

This gives the maximum number of failed records which should be tolerated. When this is exceeded the program aborts. The default is 9. Use 0 to abort on the first error.

`-force`

Entries that cannot be decoded normally cannot be deleted, as attribute indexes cannot then be properly updated. This flag overrides that, possibly leaving indexes in an inconsistent state.

`-descendants`

This removes all the subordinates of the base entry down to the bottom of the tree, but does not remove the base entry itself.

H.9 Examples

H.9.1 Creating and opening a Directory

```
#!/bin/sh
# -*- tcl -*- \
exec dmish "$0" ${1+"$@"}

#This script sequentially:
#  o Creates a first level dsa
#  o Starts the newly created dsa
#  o Opens a management connection to dsa using simple
#    credentials
#  o Performs a simple (list all objects) operation
#  o Closes the management connection to dsa
#  o Stops the dsa

set dir      "/tmp/dsa-db-example"
set name     "cn=DSA,cn=eric,o=Isode Ltd,c=US"
set addr     "Internet=eric+19991"
set user     "cn=DSA Manager,$name"
set passwd  "GARY123"
set prefix  "c=US"

#Create a first level dsa
if {[catch {dmi create \
    -dsa_dir "$dir" \
    -name    "$name" \
    -address "$addr" \
    -password "$passwd" \
    -prefix  "$prefix"} err]} {
    puts "dsa ($dir) not created: $err"
    return "$err"
} else {
    puts "dsa ($dir) created"
}

#Start the newly created dsa
if {[catch {dmi start -dsa_dir "$dir"} err]} {
    puts "dsa ($dir) not started: $err"
    return "$err"
} else {
    puts "dsa ($dir) started"
}

#Sleep to allow dsa to start
after 5000

#Open management connection to dsa using simple credentials
if {[catch {dmi open \
    -call    "$addr" \
    -user    "$user" \
    -password "$passwd" \
    -simple} err]} {
    puts "dsa connection not opened ($addr) : $err"
    return "$err"
} else {
    puts "dsa connection opened ($addr)"
}
}
```

```

#Perform operation
if {[catch {dmi list} err]} {
    puts "dmi list failed: $err"
    return "$err"
} else {
    puts "$err"
}

#Close management connection to dsa
if {[catch {dmi close} err]} {
    puts "dsa ($dir) not closed: $err"
    return "$err"
} else {
    puts "dsa ($dir) closed"
}

#Stop the dsa
if {[catch {dmi stop \
-dsa_dir "$dir"} err]} {
    puts "dsa ($dir) not stopped: $err"
    return "$err"
} else {
    puts "dsa ($dir) stopped"
}

```

H.9.2 Establishing a consumer shadowing agreement

```

#!/bin/sh
# -*- tcl -*- \
exec dmish "$0" ${1+"$@"}

#Pick up authcon information
lappend auto_path $isode_libpath/authconlib/scripts

#Consumer host information
set consumer(setup) Internet=eric+29995|LDAP=eric+29990
set consumer(addr) Internet=eric+29995
set consumer(user) "<cn=DSA Manager, cn=DSAC, cn=eric, o=Isode Ltd, c=US>"
set consumer(passwd) "GARY123"
set consumer(name) "cn=DSAC, cn=eric, o=Isode Ltd, c=US"
set consumer(dir) "/var/isode/dsa-dbc"
set consumer(prefix) "cn=DSAC, cn=eric, o=Isode Ltd, c=US"
set consumer(supnme) "<c=US>"
set consumer(supadr) "Internet=eric+29996"

#Location of templates used in adding objects
set template_dir "/path/to/script/dmish/templates"

#Shadow departments to be consumed from master. A template MUST
#already exist in template_dir for each of these departments.
set departments {{Engineering}\
                {Sales}\
                }

#Create a consumer dsa
if {[catch {dmi create \
    -dsa_dir "$consumer(dir)" \
    -name "$consumer(name)" \
    -address "$consumer(setup)" \
    -password "$consumer(passwd)" \
    -prefix "$consumer(prefix)" \
    -superior_name "$consumer(supnme)" \
    -superior_address "$consumer(supadr)"} err]} {

```

```

    puts "dsa ($consumer(dir)) not created: $err"
    return "$err"
} else {
    puts "dsa ($consumer(dir)) created"
}

#Start the newly created consumer dsa
if {[catch {dmi start \
-dsa_dir "$consumer(dir)" } err]} {
    puts "dsa ($consumer(dir)) not started: $err"
    return "$err"
} else {
    puts "dsa ($consumer(dir)) started"
}

#Sleep to allow dsa to start
after 5000

#Open management connection to dsa using simple credentials
if {[catch {dmi open \
    -call      "$consumer(addr)" \
    -user      "$consumer(user)" \
    -password  "$consumer(passwd)" \
    -simple} err]} {
    puts "dsa connection not opened ($consumer(addr)) : $err"
    return "$err"
} else {
    puts "dsa connection opened ($consumer(addr))"
    set id "$err"
}

# Read the local authcon object for this DSA
set myconsumer [pacm mapLocal [dname "$consumer(name)"]]

#Set the auth level for supplier-initiated as simple
$myconsumer configure -prot disp -req resp {simple}

#Set password information for consumer
$myconsumer configure -prot disp -password remote_init "supplier"
$myconsumer configure -prot disp -password resp "consumer"

#Commit authentication information into dsa
$myconsumer commit

#Add a GDAM (database) object for consumed data to be stored
if {[catch {dmi add \
    -id $id \
    -db \
    -template "$template_dir/consumer/Consumer-DB.template" \
    -noedit {-noedit} } err]} {
    puts "object (db) not added: $err"
    return "$err"
} else {
    puts "object (db) added"
}

#Add a shadow consumer agreement for each department
foreach dept $departments {
    if {[catch {dmi add \
        -id $id \
        -consumer \
        -template "$template_dir/consumer/$dept-CONSUMER.template" \
        -noedit {-noedit} } err]} {
        puts "object (consumer $dept) not added: $err"
        return "$err"
    } else {

```

```

        puts "object (consumer $dept) added"
    }
}

#Close management connection to dsa
if {[catch {dmi close} err]} {
    puts "dsa ($consumer(dir)) not closed: $err"
    return "$err"
} else {
    puts "dsa ($consumer(dir)) closed"
}
}

```

H.9.3 Establishing a supplier shadowing agreement

```

#!/bin/sh
# -*- tcl -*- \
exec dmish "$0" ${1+"$@"}

#Pick up authcon information
lappend auto_path $isode_libpath/authconlib/scripts

#Supplier host information
set supplier(setup) Internet=eric+29996|LDAP=eric+29991
set supplier(addr) Internet=eric+29996
set supplier(user) "<cn=DSA Manager,cn=DSAS,cn=eric,o=Isode Ltd, c=US>"
set supplier(passwd) "GARY123"
set supplier(name) "cn=DSAS, cn=eric, o=Isode Ltd, c=US"
set supplier(dir) "/var/isode/dsa-dbs"
set supplier(prefix) "c=US"

#Location of templates used in adding objects
set template_dir "/path/to/script/dmish/templates"

#Shadow departments to be supplied from master. A template MUST
#already exist in template_dir for each of these departments.
set departments {{Engineering}\
    {Sales}\
}

#Create a supplier dsa
if {[catch {dmi create \
    -dsa_dir "$supplier(dir)" \
    -name "$supplier(name)" \
    -address "$supplier(setup)" \
    -password "$supplier(passwd)" \
    -prefix "$supplier(prefix)"} err]} {
    puts "dsa ($supplier(dir)) not created: $err"
    return "$err"
} else {
    puts "dsa ($supplier(dir)) created"
}

#Start the newly created supplier dsa
if {[catch {dmi start \
    -dsa_dir "$supplier(dir)"} err]} {
    puts "dsa ($supplier(dir)) not started: $err"
    return "$err"
} else {
    puts "dsa ($supplier(dir)) started"
}

#Sleep to allow dsa to start
after 5000

```

```

#Open management connection to dsa using simple credentials
if {[catch {dmi open \
    -call      "$supplier(addr)" \
    -user      "$supplier(user)" \
    -password  "$supplier(passwd)" \
    -simple} err]} {
    puts "dsa connection not opened ($supplier(addr)) : $err"
    return "$err"
} else {
    puts "dsa connection opened ($supplier(addr))"
    set id "$err"
}

# Read the local authcon object for this DSA
set mysupplier [pacm mapLocal [dname "$supplier(name)"]]

#Set the auth level for supplier-initiated as simple
$mysupplier configure -prot disp -req remote_resp {simple}

#Set password information for supplier
$mysupplier configure -prot disp -password remote_resp "consumer"
$mysupplier configure -prot disp -password init "supplier"

#Commit authentication information into dsa
$mysupplier commit

#Add a GDAM (database) object for consumed data to be stored
if {[catch {dmi add \
    -id $id \
    -db \
    -template "$template_dir/supplier/Supplier-DB.template" \
    -noedit {-noedit} } err]} {
    puts "object (db) not added: $err"
    return "$err"
} else {
    puts "object (db) added"
}

#Add a naming context for each department to be shadowed
foreach dept $departments {
    if {[catch {dmi add \
        -id $id \
        -nc \
        -template "$template_dir/supplier/$dept-NC.template" \
        -noedit {-noedit} } err]} {
        puts "object (nc $dept) not added: $err"
        return "$err"
    } else {
        puts "object (nc $dept) added"
    }
}

#Convert each naming context to an administrative point ACSA/CASA
foreach dept $departments {
    if {[catch {dmi add \
        -id $id \
        -adm \
        -template "$template_dir/supplier/$dept-ACSA.template" \
        -noedit {-noedit} } err]} {
        puts "object (adm $dept) not added: $err"
        return "$err"
    } else {
        puts "object (adm $dept) added"
    }
}

```

```
#Add a shadow supplier agreement for each administrative point
foreach dept $departments {
  if {[catch {dmi add \
    -id $id \
    -supplier \
    -template "$template_dir/supplier/$dept-SUPPLIER.template" \
    -noedit {-noedit} } err]} {
    puts "object (supplier $dept) not added: $err"
    return "$err"
  } else {
    puts "object (supplier $dept) added"
  }
}

#Close management connection to dsa
if {[catch {dmi close} err]} {
  puts "dsa ($supplier(dir)) not closed: $err"
  return "$err"
} else {
  puts "dsa ($supplier(dir)) closed"
}
```

H.10 Templated DSA creation

M-Vault Console uses a template-based mechanism to determine the structure and initial content when creating a Directory. The same mechanism is used by the **dsa-setup** utility, which can be run from the command-line. Because the utility does not rely on a graphics display, it is suitable for use in script or batch files, or even when used to create configurations where no operator interaction is required at all.

The process of templated DSA creation, whether using M-Vault Console or **dsa-setup** involves the following steps:

1. Creating and starting an empty DSA that has no accounts other than the *superuser* account enabled, which requires:
 - the directory server's name and presentation address
 - the file system location for the Directory Server
2. Utilising the *superuser* account, connecting to the DSA and load information into it, including:
 - initial naming contexts
 - initial set of user data
 - passwords for accounts other than the superuser
 - Global Access Control information
3. creating bind profiles that may be used by applications such as M-Vault Console and Sodium to connect to the directory
4. (optionally) disabling the *superuser* account

The information used in the above process is derived from a set of files which together form the DSA creation template. The files form a *template*, since the information they contain is subject to being overridden or modified by input provided by the operator at DSA creation time.

The following section provides a reference for the template files which are used when creating DSAs.

H.10.1 Files used by the template mechanism

The following files are used to determine the initial configuration of a Directory Server created using either M-Vault Console or the **dsa-setup** script (note that when creating *mirror* or *shadow* directory servers, the templates are not used, since in these cases the structure and contents of the new directory are copied from the old one):

- an XML file containing the *Directory Server creation template*
- an XML file containing *Global Access Control (GAC)* information
- an file containing *LDIF data* which represents entries that are to be loaded into the Directory once it has been created

The creation mechanism searches for any template files in the directory *dsa-setup*, first below (*ETCDIR*) and then below (*SHAREDIR*). Some predefined templates (those which appear when you use M-Vault Console to create a new DSA) are supplied in (*SHAREDIR*)/*dsa-setup*. If you want to modify the existing templates, or create new ones, then you should create files in (*ETCDIR*)/*dsa-setup*; these will be used in preference to any files of the same name that are in (*SHAREDIR*)/*dsa-setup*.

Depending on what other Isode products are installed on the system, there may be other directories in (*SHAREDIR*) which contain templates, such as *mhs-dsa-setup*.

H.10.1.1 Directory Server Creation Template

The Directory Server Creation template contains information about the structure of the directory, and also contains references to the GAC and LDIF files. The contents of the template file are described below:

H.10.1.1.1 <dsa-creation-template> element

This is the root XML element. It has a single mandatory attribute of `label`. For example:

```
<dsa-creation-template label="Default DSA Configuration">
```

The XML file must contain a single instance of this element.

H.10.1.1.2 <ldif> element

This element specifies the name of an LDIF file which is to be loaded into the directory once the server has been created. There is a single mandatory attribute of `file` which should be the filename (no path; the file is assumed to be in the same folder as the template file).

The DNs in the file are subject to relocation according to information provided by the operator during the creation process. For example, if the initial "Groups subtree location" specified in the template is set to "o=groups,c=xx", and the operator overrides this value to "o=groups,cn=test,c=gb", then all the groups in the LDIF file will be created underneath "o=groups,cn=test,c=gb".

The information in the LDIF file must be consistent with the information in the template and GAC files. For example, if the initial user is missing in this file, then no initial user will be created and the operator will be unable to bind to the directory and configure other users. Or if a group is present in the GAC file but not in the LDIF file, then that group will not be created and the access control defined for that group may be lost.

Example:

```
<ldif file="simple_dump.ldif" />
```

The XML file must contain a single instance of this element.

H.10.1.1.3 <gac> element

This specifies the name of an XML file containing Global Access Control that is to be loaded into the directory once the it has been created. There is a single mandatory attribute of `file` which should be the filename (no path; the file is assumed to be in the same folder as the template file).

DNs inside this file are relocated in the same way as for the `<ldif>` file. For example:

```
<gac file="simple_gac.xml" />
```

The XML file must contain a single instance of this element.

H.10.1.1.4 <dn-param> element

The data loaded into the directory after the server has been created is specified in the template, LDIF and GAC files. The `dn-param` element provides a way to prompt and allow the operator to override the default location specified in these files with a different location.

Any part of a DN specified in the files can be relocated. Subtrees may be relocated pretty much without limit. It is not necessary to maintain the default hierarchy. However, if it is required that the DN is restricted to a certain subtree, then the attribute `sub` can be used with the value being the subtree that this DN needs to be a part of.

It is also possible to relocate individual entries, either to put them in some other location, or just to rename them - for example, to rename individual users.

It is also possible to rename from one RDN-type to another - for example, to rename from **c=GB** to **o=Isode**. RDN-type changes can be prevented by specifying the attribute `<ocs="fixed" />`. If the objectclass specified in the LDIF for a DN is not appropriate for certain RDN types, then care needs to be taken when allowing the operator to modify these types, since it could lead to objectclass errors.

When the data is loaded into the directory, any dummy entries that are required but not specified will also be loaded.

This element may appear multiple times, once for each relocation. It has the following attributes:

- `label` (mandatory). The label used to identify the parameter when requesting operator input
- `init` (mandatory). The DN to relocate. The operator can override this DN with a different DN, which will replace the original DN with the new DN wherever it occurs in the template, LDIF and GAC files.
- `sub` (optional). If it is required that the DN is restricted to a certain subtree, then this attribute should be used with a value of the subtree that the DN needs to be a part of.
- `ocs` (optional). To prevent RDN-type changes, this attribute should be used with a value of `"fixed"`.

Examples:

```
<dn-param label="Base DN" init="c=xx" />
<dn-param label="Group DN" init="o=group,c=xx" ocs="fixed" />
```

```
<dn-param label="Initial Directory User"
init="cn=Thomas Atkins,o=users,c=xx" sub="o=users,c=xx" />
```

The XML file may contain any number of these elements.

H.10.1.15 <dsa> element

This is the distinguished name of the DSA itself. The name is subject to being overridden by the operator. It is not necessary for there to be a real entry in the DSA with this DN. The element has a single mandatory attribute of `dn` which must be a valid DN.

Example:

```
<dsa dn="cn=dsa,c=xx" />
```

The XML file must contain a single instance of this element.

H.10.1.16 <naming-context> element

This element identifies the location of a naming context inside the directory. If the DN contains multiple RDNs, then glue entries will be created for any non-existing parent entries of the naming context. For example, creation of the naming context `o=Isode,c=GB` requires that the existence of `c=GB`, and so if no such entry exists, then a glue entry for `c=GB` will be created.

Note that any (non glue) entry which is created directly below the root entry will be a naming context, whether or not it is specified as a `<naming-context>`.

This element has the following attributes:

- `dn` (mandatory). The DN of the naming context. This may not be the root DN
- `gdam` (optional). The name of a GDAM to be used for the naming context, which may be overridden by the operator. If not present, the GDAM will be created. If this attribute is absent, then the default GDAM "gdam1" will be used.

Examples:

```
<naming-context dn="c=xx" />
<naming-context dn="ou=Staff,o=Isode,c=GB" />
<naming-context dn="o=Isode" gdam="isodegdam" />
```

The XML file may contain any number of these elements.

H.10.1.17 <create-entry> element

This element identifies the distinguished name of an entry that must be present in the Directory. If the DN contains multiple RDNs, then glue entries will be created for any non-existing parent entries of the naming context. For example, creation of the naming context `o=Isode,c=GB` requires that the existence of `c=GB`, and so if no such entry exists, then a glue entry for `c=GB` will be created.

This element is ignored if the entry already exists as a result of a `naming-context` element for the same DN, or as a result of another relocation.

This element has the following attribute

- `dn` (mandatory). The DN of the entry.

Examples:

```
<create-entry dn="c=xx" />
<create-entry dn="ou=Staff,o=Isode,c=GB" />
<create-entry dn="o=Isode" gdam="isodegdam" />
```

The XML file may contain any number of these elements.

H.10.1.1.8 <admin-point> element

This element identifies a DN which is to be converted to an admin point. The DN of the entry must already exist as a result of a <naming-context> or a <create-entry> or another relocation.

Note that naming contexts are by default admin points, even when the <admin-point> element is not used.

This element has the following attributes:

- dn (mandatory). The DN of the admin point.
- bac (optional). Should be set to "true" for Basic Access Control. If the attribute is absent, or set to "false", then the admin point is created with Simple Access Control.

Examples:

```
<admin-point dn="c=xx" />
<admin-point dn="ou=Staff,o=Isode,c=GB" bac="true" />
<admin-point dn="o=Isode" bac="false" />
```

The XML file may contain any number of these elements.

H.10.1.1.9 <manager> element

This element identifies the user DN for the bind profile used to manage the DSA. If no <manager> is specified, then the superuser is left active and the bind profile used to manage the DSA will contain the DN of the superuser (see below).

This element has the following attributes:

- dn (mandatory). The DN used to identify the manager.

Example:

```
<manager dn="cn=DSA Manager,o=users,c=xx" />
```

The XML file may contain only one of these elements.

H.10.1.1.10 <create-profile> element

This element identifies a user DN for a bind profile that can be used to bind in Sodium etc., but is not intended to be used for managing the DSA using M-Vault Console.

This element has the following attributes:

- dn (mandatory). The DN used to identify user DN.

Example:

```
<manager dn="cn=DSA Manager,o=users,c=xx" />
```

The XML file may contain any number of these elements.

H101.1.11 <pw-param> element

This element can be used to generate passwords for any user entries loaded from the LDIF file.

This element may appear multiple times; once for each user entry. If a user entry is present in the LDIF file and there is no matching <pw-param> element, then the user entry will be loaded with no password. However, in the case of <manager> or <create-profile> elements, it is mandatory that a corresponding <pw-param> be specified.

If a <pw-param> is specified for a DN which has no matching entry in the LDIF file, then it is ignored.

This element has the following attributes:

- label (mandatory). The label used to identify the user when requesting operator input.
- dn (mandatory). The user DN which this password applies to.
- length (optional). The length of the generated random password. If this is not specified, a default value of 9 is used.

Examples:

```
<pw-param label="Initial Directory User"
dn="cn=Fred Smith,o=Users,c=xx" />
<pw-param label="General User"
dn="cn=Fred Smith,o=Users,c=xx" length="5" />
```

The XML file may contain any number of these elements.

H101.1.12 <superuser> element

For the purposes of DSA creation, a special *superuser* role is enabled in the DSA, which is not subject to any access control. If a <manager> is specified in the template file, then by default the superuser account will be removed.

If no <manager> is specified, then the superuser account is *not* removed, and will be used in the bind profile used to manage the DSA.

The <superuser> element has two purposes: firstly it allows you to prevent the superuser account from being removed, even when a <manager> is specified, and secondly it allows you to specify what the superuser password is (whether or not a <manager> is specified).

This element has the following attributes:

- pw (mandatory). The password to be used for the superuser.

Examples:

```
<superuser pw=" " />
<superuser pw="secret" />
```

The XML file may contain no more than one of these elements.

H101.1.13 <opt-group> element

This element is used to allow the operator to choose to omit certain groups defined in the LDIF or GAC files when creating a new Directory Server. Stripping out a group may lead to the stripping out of items that use the groups and rules used in such items.

This element has the following attributes:

- dn (mandatory). The DN of the group. If this DN does not exist in the LDIF and/or GAC file, then no action is taken.
- sel (optional). The initial selection status of this group. A value of "true" means that the group is to be selected when first shown. If the attribute is absent, or has a value of "false", then group will be de-selected when first shown. The operator can override the initial selection.

Examples:

```
<opt-group dn="cn=Data Managers,o=groups,c=xx" />
<opt-group dn="cn=CRL Writers,o=groups,c=xx" sel="true" />
```

The XML file may contain any number of these elements, one for each group that is optional.

H101.1.14 <opt-rule> element

This element is used to allow the operator to choose to omit certain items defined in the GAC file when creating a new Directory Server.

This element has the following attributes:

- label (mandatory). The label used to identify the item when prompting the operator for input.
- prec (mandatory). The integer precedence of the item in the GAC file. This is used to uniquely identify the item from all the items in the GAC file, which means that all optional items in a GAC file should have mutually distinct precedences. If no item exists in the GAC file with this precedence, then the element is ignored.
- sel (optional). The initial selection status of the item. A value of "true" means that the item should be selected when first shown. If the attribute is absent, or has a value of "false", then the item will be de-selected when first shown.

Examples:

```
<opt-rule
label="Everyone has read access by default"
prec="0"
sel="true" />
<opt-rule
label="Only authenticated users have read access by default"
prec="2" />
```

The XML file may contain any number of these elements, one for each item that is optional.

H101.1.15 <opt-rule-limit> element

This element is used to restrict the operator from selecting conflicting items, or to force the operator to select certain items defined in the <opt-rule> elements.

This element has the following attributes, at least one of which must be present:

- at-least-one (optional). A comma separated list of precedence values, of which *at least* one must be selected by the operator. Each value in the list must be present in an <opt-rule> element.
- at-most-one (optional). A comma separated list of precedence values, of which *only* one may be selected by the operator. Each item in the list must be present in an <opt-rule> element.

Examples:

```
<opt-rule-limit at-least-one="0,2" />
<opt-rule-limit at-most-one="1,3,4" />
<opt-rule-limit at-least-one="0,2" at-most-one="5,6" />
```

The XML file may contain any number of these elements.

H101.1.16 <attr-to-index> element

This element is used to create an index in the initial GDAM database. Indexes are described in [Section 4.6.4, "Database indexes"](#). It has the following attributes:

- `type` (mandatory). The name of the attribute for which an index should be created
- `match` (optional). The type of index to create, which may be:
 - `eq` for an *Equality* index
 - `sub` for a *Substring* index
 - `approx` for an *Approximate* index
 - `pres` for a *Presence* index

If no `match` is specified, then an *Equality* index will be created.

Examples:

```
<attr-to-index type="mail" />
<attr-to-index type="surname" match="pres" />
```

The XML file may contain any number of these elements.

H.10.1.2 Global Access Control information

Global Access Control (GAC) provides a means of describing a set of rules and roles which control access to data inside the directory. A GAC configuration can be dumped to or loaded from an XML file, and will be translated by Isode client applications such as M-Vault Console into *Simplified Access Control* which are used inside the directory.

During templated DSA creation, an XML file containing GAC configuration is used to specify the initial Global Access Control (from which is derived Simplified Access Control) inside the directory. The DNs inside the LDIF file are subject to relocation, based on responses that may be given by the operator.

The name of the GAC file is specified by the `<gac>` element inside the Directory Server Creation template file.

H.10.1.3 LDIF data file

A set of data may be loaded into the directory at creation time, by loading up the contents of an LDIF file. The DNs inside the LDIF file are subject to relocation, based on responses that may be given by the operator.

The name of the LDIF file is specified by the `<ldif>` element inside the Directory Server Creation template file.

H.10.2 dsa-setup utility

`dsa-setup` is implemented as a script which is invoked from the command line. It has the following options:

- *create* - create and start a standalone Directory Server. This will also load specific data and global access control in to the Directory from the template files, and update the bind profile file.
- *delete* - shut down and delete a Directory Server, removing the databases from disk, as well as removing its bind profile.
- *shadow* - create a new Directory Server which has consumer agreements that shadow all or some of a "supplier" Directory Server.
- *mirror* - create a new Directory Server which is a new failover mirror of an existing failover master Directory Server.

When using the *create* option, then the information from the specified template files will be used. Unless the *-noprompt* option is used, then the operator will be prompted to accept or modify various options.

Under Windows, open a command window (using "Run as Administrator" on those platforms which require it), and run the script *dsa-setup.bat*, which will have been installed by default into the folder (*SBINDIR*) on the installation drive. The example below uses *-noprompt* which means that no user input is required during the creation process:

```
C:\Program Files\Isode\bin>dsa-setup.bat create
  -ppfile secret.txt -noprompt "C:\mydsa" simple.xml
CREATING A STANDALONE DSA WITH THE FOLLOWING DATA:
Bind profile passphrase stored in file
 [secret.txt]
DSA folder
 [C:\mydsa]
XML template file
 [C:\Program Files\Isode\share\dsa-setup\simple.xml]
Presentation address
 [URI+0000+URL+itot://server:19999|URI+0001+URL+ldap://server:19389]
Base DN, for example: Country (c=gb),
International Organization (o=isode),
Organization (o=isode,c=gb), or
Internet Domain (dc=isode,dc=com)
 [c=xx]
DSA DN: This identifies the DSA when working with several DSAs.
It must be unique within the group of DSAs that you plan to
interact with using M-Vault Console. For a standalone DSA,
you may simply call it cn=DSA. It is not necessary for there
to be a real entry with this DN.
 [cn=DSA,c=xx]
Users subtree location: This can be put either under the base
DN (as below), or if you prefer, at the root (o=users)
 [o=users,c=xx]
Groups subtree location: This can be put either under the base
DN (as below), or if you prefer, at the root (o=groups)
 [o=groups,c=xx]
Initial Directory User: This user is put into all the initial
roles, and the bind profile created will bind as this user.
Afterwards you can create more users and change which users
are put in which roles. You should change the common name
field to a suitable value for a real person.
 [cn=Thomas Atkins,o=users,c=xx]
Optional rules:
 1: [x] Everyone has read access by default
 2: [ ] Only authenticated users have read access by default
 3: [x] Allow users to modify their own entry's attributes
 4: [x] Allow users to modify their own password
Optional groups:
 1: [x] Data Managers Group
 2: [ ] CRL Writers Group
 3: [ ] Certificate Writers Group
 4: [ ] CA Managers Group
```

```
Password Hashing
  [Clear Text]
Manager bind profile name
  [cn=DSA,c=xx / Thomas Atkins]
Password for Initial Directory User (cn=Thomas Atkins,o=users,c=xx)
  [PtjjY4pQs]
CREATING ...
SUCCESS

C:\Program Files\Isode\bin>
```

Under Unix, **dsa-setup** is started by running *dsa-setup* which will have been installed into (*SBINDIR*). For example:

```
% /opt/isode/sbin/dsa-setup delete
  -ppfile secret.txt
  -dsa "cn=dsa,o=isode" /var/isode/dsa-db
SHUTTING DOWN AND DELETING THE DSA WITH THE FOLLOWING DATA:
Bind profile passphrase stored in file
  [secret.txt]
DSA Folder
  [/var/isode/dsa-db]
DSA DN
  [cn=dsa,o=isode]
Press 'q' to stop now or any other key to continue:

DELETING ...
SUCCESS
```

Appendix I References

The documents listed in this appendix provide references to the appropriate standards and other sources of information.

If documents can be obtained electronically, the location is stated as part of the reference. For other documents, please see [Section I.4, “Obtaining documents”](#).

I.1 RFCs

RFC 5246

The Transport Layer Security (TLS) Protocol Version 1.2. T. Dierks, August 2008

RFC 2696

LDAP Control Extension for Simple Paged Results Manipulation. C. Weider, A. Herron, A. Anantha, T. Howes, September 1999

RFC 2849

The LDAP Data Interchange Format (LDIF) - Technical Specification. G. Good. June 2000.

RFC 2891

LDAP Control Extension for Server Side Sorting of Search Results. T. Howes, M. Wahl, A. Anantha. August 2000.

RFC 3045

Storing Vendor Information in the LDAP root DSE. M. Meredith. January 2001.

RFC 3062

LDAP Password Modify Extended Operation. K. Zeilenga. February 2001.

RFC 3268

Advanced Encryption Standard (AES) Ciphersuites for Transport Layer Security (TLS). P. Chown. June 2002.

RFC 3296

Named Subordinate References in Lightweight Directory Access Protocol (LDAP) Directories. K. Zeilenga. July 2002.

RFC 3672

Subentries in the Lightweight Directory Access Protocol (LDAP). K. Zeilenga. December 2003.

RFC 3673

Lightweight Directory Access Protocol version 3 (LDAPv3): All Operational Attributes. K. Zeilenga. December 2003.

RFC 3986

Uniform Resource Identifier (URI): Generic Syntax. T. Berners-Lee, R. Fielding, L. Masinter. January 2005.

RFC 4422

Simple Authentication and Security Layer (SASL). A. Melnikov, K. Zeilenga. June 2006.

RFC 4511

Lightweight Directory Access Protocol (LDAP): The Protocol. J. Sermersheim. June 2006.

RFC 4512

Lightweight Directory Access Protocol (LDAP): Directory Information Models. K. Zeilenga. June 2006.

- RFC 4513
Lightweight Directory Access Protocol (LDAP): Authentication Methods and Security Mechanisms. R. Harrison. June 2006.
- RFC 4515
Lightweight Directory Access Protocol (LDAP): String Representation of Search Filters. M. Smith, T. Howes. June 2006.
- RFC 4517
(LDAP): Syntaxes and Matching Rules. S. Legg, Ed. June 2006
- RFC 4519
Lightweight Directory Access Protocol (LDAP): Schema for User Applications. A. Sciberras. June 2006.
- RFC 4524
COSINE LDAP/X.500 Schema. K. Zeilenga. June 2006.
- RFC 4526
Lightweight Directory Access Protocol (LDAP) Absolute True and False Filters. K. Zeilenga. June 2006.
- RFC 4529
Requesting Attributes by Object Class in the Lightweight Directory Access Protocol. K. Zeilenga. June 2006.
- RFC 4532
Lightweight Directory Access Protocol (LDAP) "Who am I?" Operation. K. Zeilenga. June 2006.
- RFC 5020
Lightweight Directory Access Protocol (LDAP) entryDN Operational Attribute. K. Zeilenga. August 2007.
- RFC 5803
Lightweight Directory Access Protocol (LDAP) Schema for Storing Salted Challenge Response Authentication Mechanism (SCRAM) Secrets. A. Melnikov. July 2010.

I.2 Recommendations and standards

- ISO 639:1988
Code for the Representation of Names of Languages.
- ISO 3166:1988
Codes for the Representation of Names of Countries.
- ISO/IEC 9075-3:1992 (ANSI X3.135-1992)
Database Language SQL - Part 3: Call-Level Interface.
- ISO/IEC DIS 10181-2.2
Information Technology — Open Systems Interconnection— Security Frameworks for Open Systems: Authentication Framework.
- ISO/IEC DIS 11586-1
Information technology — Open Systems Interconnection — Generic Upper Layers Security — Part 1: Overview, Models and Notations.
- ISO/IEC JTC1/SC 21/N 9294
Draft Technical Corrigenda to Rec. X.500 | ISO/IEC 9504 resulting from Defect Reports 9594/128.
- ITU-T Rec. X.208(1988) | ISO 8824
Information Processing Systems — Open Systems Interconnection — Specification of Abstract Syntax Notation One (ASN.1).

- ITU-T Rec. X.500(2008) | ISO/IEC 9594-1:2008
Information Technology — Open Systems Interconnection — The Directory: Overview of Concepts, Models, and Services.
- ITU-T Rec. X.501(2008) | ISO/IEC 9594-2:2008
Information Technology — Open Systems Interconnection — The Directory: Models.
- ITU-T Rec. X.511(2008) | ISO/IEC 9594-3:2008
Information Technology — Open Systems Interconnection — The Directory: Abstract Service Definition.
- ITU-T Rec. X.518(2008) | ISO/IEC 9594-4:2008
Information Technology — Open Systems Interconnection — The Directory: Procedures for Distributed Operations.
- ITU-T Rec. X.519(2008) | ISO/IEC 9594-5:2008
Information Technology — Open Systems Interconnection — The Directory: Protocol Specifications.
- ITU-T Rec. X.520(2008) | ISO/IEC 9594-6:2008
Information Technology — Open Systems Interconnection — The Directory: Selected Attribute Types.
- ITU-T Rec. X.521(2008) | ISO/IEC 9594-7:2008
Information Technology — Open Systems Interconnection — The Directory: Selected Object Classes.
- ITU-T Rec. X.509(2008) | ISO/IEC 9594-8:2008
Information Technology — Open Systems Interconnection — The Directory: Authentication Framework.
- ITU-T Rec. X.525(2008) | ISO/IEC 9594-9:2008
Information Technology — Open Systems Interconnection — The Directory: Replication.
- ITU-T Rec. X.721(1992) | ISO/IEC 10165-2
Information Technology — Open Systems Interconnection — Structure of Management Information: Definition of Management Information.
- ITU-T Rec. X.800(1991) | ISO 7498-2
Information processing systems — Open Systems Interconnection — Basic Reference Model — Part 2: Security architecture.

I.3 Other publications

Chadwick, D. W. *Understanding X.500 (The Directory)*. International Thompson Publishing, July 1996. ISBN 1-85032-281-3.

Gardner, Ella and Ginsburg, Elliot. *Defense Message System Unclassified Directory Schema*. Mitre Corporation, Washington C3 Center. 5 February 1993.

MIT. *Kerberos: The Network Authentication Protocol*. <http://web.mit.edu/Kerberos/>.

OIW Implementor's Workshop. *Stable Implementation Agreements for Open Systems Interconnection Protocols: Part 12 - OS Security*. September 1994.
ftp://nemo.nsl.nist.gov/pub/oiw/agreements/12S_9409.ps

Ousterhout, J. *An X11 Toolkit Based on the Tcl Language*. Winter 1991 USENIX Conference Proceedings. <ftp://ftp.scripatics.com/pub/tcl/doc/tkUsenix91.ps>

Ousterhout, J. *Tcl: An Embeddable Command Language*. Winter 1990 USENIX Conference Proceedings. <ftp://ftp.scripatics.com/pub/tcl/doc/tclUsenix90.ps>

Ousterhout, J. *Tcl and the Tk Toolkit*. Addison-Wesley, 1994. ISBN 0-201-63337-X.

Roe, M. *PASSWORD R2.5: Certification Authority Requirements*. Nov. 1992.
<ftp://cs.ucl.ac.uk/password/r25.ps>

RSA Data Security Inc. *PKCS #1: RSA Encryption Standard*. Nov. 1993.

RSA Data Security Inc. *PKCS #6: Extended-Certificate Syntax Standard*. Nov. 1993.

RSA Data Security Inc. *PKCS #7: Cryptographic Message Syntax Standard*. Nov. 1993.

RSA Data Security Inc. *PKCS #8: Private-Key Information Syntax Standard*. Nov. 1993.

RSA Data Security Inc. *PKCS #9: Selected Attribute Types*. Nov. 1993.

versit Consortium. *vCard. The Electronic Business Card Version 2.1*. September 18, 1996.
<http://www.imc.org/pdi/vcard-21.ps>

Welch, B. B. *Practical Programming in Tcl and Tk*. Prentice Hall. ISBN 0136168302.

I.4 Obtaining documents

I.4.1 ISO/IEC documents

ISO/IEC standards and draft documents may be obtained from:

ISO Central Secretariat
International Organization for Standardization (ISO)
1, rue de Varembe
Case postale 56
CH-1211 Geneva 20
Switzerland

Telephone: +41 22 749 01 11

Fax: +41 22 733 34 30

Web: <http://www.iso.org/>

I.4.2 ITU-T (CCITT) documents

International Telecommunications Union
Place des Nations
CH-1211 Geneva 20
Switzerland

Telephone: +41 22 730 61 41 Fax: +41 22 730 51 94

Email: sales@itu.int

Web: <http://www.itu.int/>

I.4.3 RFCs

Electronic copies of RFCs are available from the following servers:

- <http://ftp.isi.edu/in-notes/>
- <http://www.rfc-editor.org/>

Glossary

AAA

Autonomous Administrative Area. See [Administrative area](#).

AAP

Autonomous Administrative Point. See [Administrative point](#).

Abstract object class

An object class used only by the Directory, not a classification of objects in the real world. See [Alias](#) and [Top](#).

Abstract service

See [Directory abstract service \(DAS\)](#).

Abstract Syntax Notation One (ASN.1)

A notation for describing and defining data syntax.

Access control

A security service aimed at preventing unauthorized access to a capability. Once an identity has been established (see [Authentication](#)), access control determines what data and operations may be accessed by that identity.

Access Control Information (ACI)

ACI items define the access control policy for an entry. See also [Access control](#).

Access Control Inner Area (ACIA)

A subtree of the DIT inside a specific administrative area whose entries share the same access control administration.

Access Control Specific Area (ACSA)

A subtree of the DIT whose entries share the same access control administration.

Access Point

This uniquely identifies the DSA. It contains the DSA's Distinguished Name, Presentation Address and an optional set of protocol information. See also [Knowledge reference](#).

ACI

See [Access control information](#).

ACIA

See [Access control inner area](#).

ACSA

See [Access control specific area](#).

Address

A label which identifies an object by its location. See [IP address](#) and [Presentation Address](#).

ADMD

See [Administration Management Domain](#).

Administration Management Domain (ADMD)

An arrangement of MTAs, usually managed by a public service provider such as a PTT authority.

Administrative area

A subtree of the DIT whose entries share administration.

Administrative authority

The agent of a domain management organization which has responsibility for administering the domain, or the authority under which the agent operates.

Administrative model

The model used to define how authority is delegated in the DIT.

Administrative point

The root of the subtree formed by an administrative area.

AET

Application Entity Title. See [Application entity](#).

Alias

An abstract object class covering those entries which have an aliased entry name; this aliased entry name contains the Distinguished Name of another entry to which the alias points.

Application context

A set of rules governing the interactions between application entities over an association.

Application entity

The part of an application process concerned with interactions with others.

ARPA

Advanced Research Projects Agency. A Department of Defense Agency in the USA.

ASN.1

See [Abstract Syntax Notation One](#).

Association

A communication relationship established between application entities.

Attribute

Information in an entry describing some aspect of the object. Also a component of an X.400 address, for example, organization.

Attribute type

Classification of the aspects used to describe objects.

Authentication

The process of determining the identity of a communications partner.

Authorization

See [Access control](#).

Autonomous Administrative Area (AAA)

See [Administrative area](#).

Autonomous Administrative Point (AAP)

See [Administrative point](#).

Auxiliary object class

An object class whose members have some common characteristics, despite (potentially) belonging to different structural object classes.

BER

Basic Encoding Rules. Definition of a concrete syntax for ASN.1, mainly used when exchanging data (defined in ASN.1) between applications on different systems.

Bind

An action to establish an association between processes for the purpose of invoking or performing operations.

BNF

Bachus-Naur Format. A type of notation used when defining encodings for objects or attributes.

C

Abbreviation for Country in the DIT.

CAIA

See [Collective Attribute Inner Area](#) .

CASA

See *Collective Attribute Specific Area* .

CCITT

The International Telegraph and Telephone Consultative Committee, now known as the *ITU-T*.

Chaining

A style of interaction whereby the DSA satisfies a request by invoking another DSA and passing back the reply.

Collective Attribute Inner Area (CAIA)

An inner area of the DIT used to administer a group of attributes. See also *Inner administrative area*.

Collective Attribute Specific Area (CASA)

A specific area of the DIT used to administer a group of attributes. See also *Specific administrative area*.

Common Name

A *Directory* attribute.

Consumer

The DSA receiving (or consuming) information in a shadowing agreement.

Context prefix

The Distinguished Name of the entry at the Root of a naming context.

Cross reference

A reference allowing direct access to the DSA which holds a particular naming context.

DAP

See *Directory Access Protocol*.

DAS

See *Directory Abstract Service*.

DIB

See *Directory Information Base*.

Directory

When written with a capital D, this is a distributed database built to X.500 standards.

Directory Abstract Service (DAS)

A service providing operations to access and update the Directory.

Directory Access Protocol (DAP)

An OSI application layer protocol used by the DUA for communicating with the Directory via the DSA.

Directory Information Base (DIB)

A collection of information about objects, which is stored by the Directory.

Directory Information Model

The fundamental structures of the *DIB*, standardised to provide a basis for defining the Directory Service and its use.

Directory Information Shadowing Protocol (DISP)

An OSI application layer protocol used for the exchange of Directory information in support of shadowing agreements.

Directory Information Tree (DIT)

The *DIB*, considered as a tree, whose vertices are entries and whose arcs are relationships between the corresponding objects. A way of defining relationships between, and access routes to, the complete set of information contained in the Directory.

Directory Management Domain (DMD)

The set of *DUAs* and *DSAs* managed by a single organization.

Directory name

A term often used for the *Distinguished Name (DN)* of an entry in the Directory. Although an entry can have only one *DN* it may have more than one Directory name, as it may have an *alias*.

Directory Operational Bindings Management Protocol (DOP)

A protocol used to configure the binding of a cooperation agreement between *DSAs*. Not implemented.

Directory schema

The set of rules enforced by the Directory which governs the contents of the *DIB*; the consistency rules applying to the object classes making up the *DIB*

Directory Service

The service provided by the Directory to its users.

Directory System Agent (DSA)

A server process which maintains and provides access to defined parts of the *DIT*.

Directory System Protocol (DSP)

An OSI application layer protocol used for communication between *DSAs*.

Directory User Agent (DUA)

A client application which represents a user in accessing the Directory.

DISP

See *Directory Information Shadowing Protocol*.

Distinguished Name (DN)

The name for an object, based upon the unique path through the *DIT* from the Root to the object's entry.

Distribution model

The model which defines how an authority can locate the entries which make up the *DIT* among the *DSAs* it operates.

DIT

See *Directory Information Tree*.

DIT domain

A subset of the *DIT* held by a particular *Directory Management Domain*.

DMD

See *Directory Management Domain*.

DNS

See *Domain Name Service*.

DN

See *Distinguished Name*.

Domain Name Service (DNS)

A service for providing a mapping between domain names (for example, *isode.com*) and IP addresses.

DSA

See *Directory System Agent*.

DSA Abstract Service

The capability for chaining requests from one *DSA* to another.

DSA Information Model

A *DSA*'s internal model of the information it holds.

DSA Information Tree

A tree, held by the *DSA*, containing all the names in the *DIT* known to this *DSA*.

DSA Specific Entry (DSE)

A private, internal object held for each name in the *DSA* Information Tree, which contains the attributes for those entries mastered by the DSA.

DSE

See *DSA Specific Entry*.

DSP

See *Directory System Protocol*.

DUA

See *Directory User Agent*.

Entry

A unit in the Directory representing one object and identified by its Distinguished Name.

Entry type

See *Object class*.

First level DSA

A *DSA* which holds a first level (below the *Root*) *naming context* and knowledge of all other naming contexts which are immediate subordinates of the Root.

First level entry

An entry in the Directory immediately below the *Root*.

Fragment

That part of the *DIB* held by a single *DSA*; it can contain several naming contexts.

GDAM

Generic Directory Access Module. In the M-Vault Server, GDAMs provide an abstract interface to back-end database services.

Glue

A placeholder entry in the *DIT* with no attributes.

Hierarchical operational binding

An operational binding whose participants are the master *DSAs* for a pair of adjacent *naming contexts*, one holding a subordinate reference to the other.

IANA

Internet Assigned Numbers Authority.

Inner administrative area

An overlapping area of the *DIT* inside a specific administrative area, used for specific purposes, for example, *access control*.

Integrity

Protection against unauthorized access to information, typically while that information is in transit on the network.

Intranet

A local network based on Internet standards.

IP address

Internet Protocol address. A dotted number list (for example 193.133.227.19) which identifies a host machine on an Internet network.

ISO

International Standards Organization.

ITU-T

International Telecommunication Union - Telecommunication Standardisation Bureau, which issues recommendations to regulate the national and international operation of telecommunications authorities, public network operators and other interested parties. Formerly known as CCITT.

Knowledge

Operational information allowing a DSA to route requests to other DSAs.

Knowledge reference

An item of knowledge identifying a DSA Access Point by name and address.

LDAP

See *Lightweight Directory Access Protocol*.

LDAP Data Interchange Format (LDIF)

A widely accepted format for importing and exporting Directory information between LDAP or X.500 servers.

LDIF

See *LDAP Data Interchange Format*.

Leaf entry

An entry in the *DIT* with no other entries beneath it.

Lightweight Directory Access Protocol (LDAP)

An Internet protocol, passed directly over *TCP/IP*, used to provide access to the Directory.

Ltldish

A command line interface *DUA* for data managers, operating over *LDAP*. See also *ITcldish*.

MHS-DS

An abbreviation used to refer to a method of configuring MTAs and Message Stores using an X.500 Directory.

Model

See *Administrative model*, *Directory Information Model*, *Distribution model* and *DSA Information Model*.

M-Vault Console

Isode's graphical *DUA* for managing the Directory.

Name

The identifier of an object, used as the basis for finding its entry in the *DIB*. See also *Distinguished Name* and *Relative Distinguished Name*.

Naming authority

The organization given the task of allocating names and ensuring they are unambiguous.

Naming context

The subtree of the *DIT* held by a single *DSA*.

O

Abbreviation for Organization in the *DIT*.

Object

Something or someone (for example, a machine, application or user) which is described by the information in the Directory.

Object class

A collection of actual or conceivable objects sharing certain characteristics. See also *Abstract object class*, *Auxiliary object class* and *Structural object class*.

Object entry

An entry in the Directory representing an object.

Object identifier (OID)

A labelling mechanism, in the form of a hierarchical arrangement of integers, used to identify the representation of objects.

Open Systems Interconnection (OSI)

Set of design rules and protocol specifications defined to allow systems from any source to talk to each other, thus evading problems of hardware and software incompatibility.

Operation

A requested action upon the Directory, for example, read, search or list.

Operational attribute

An attribute of a Directory entry which contains operational information.

Operational binding

An agreement between two processes to engage in some kind of interaction.

OSI

See *Open System Interconnection*.

OU

Abbreviation for Organizational Unit in the DIT.

PA

See *Presentation Address*.

Presentation Address (PA)

The fundamental mechanism used by applications to address other application entities.

Protocol

The set form in which data must be presented to be handled by a particular computer configuration or process.

RDN

See *Relative Distinguished Name*.

Referral

A reply to an operation indicating that the DSA was unable to carry out the request and identifying another DSA which should be contacted directly.

Relative Distinguished Name (RDN)

The lowest level of Distinguished Name identifying an entry in the DIT.

Replication

The process of copying a section of the DIT from one DSA to another. See also *Shadowing*.

RFC

Request For Comments. Standards documents defining Internet protocols. Relevant RFCs are listed in the [Appendix I, References](#) section.

Role

A job function within an organization. More than one individual can be assigned to a single role.

Root

The top of the DIT tree; this is not an object, and does not have an entry.

Root DSE

A conceptual element at the Root of the DSA, used for holding operational information. See also *DSA Specific Entry*.

SASL

See *Simple Authentication and Security Layer*.

Schema

Consistency rules for object classes in the Directory. See also *Directory schema, Subschema, System schema*.

Secure socket layer

See [TLS](#) .

Shadowing

The process of replicating part of the [DIT](#) from one [DSA](#) to another under the terms of a shadowing agreement.

Shadowing agreement

A contract between two [DSAs](#), one to supply information and the other to receive (or consume) it. The information is always updated by the shadow supplier and changes (or a new copy) are supplied by the shadow supplier at agreed intervals.

Simple authentication and security layer (SASL)

A protocol-independent mechanism which supports different secure authentication mechanisms in an easily extensible fashion.

Specific administrative area

A non-overlapping area of the [DIT](#) inside an administrative area, used for specific purposes, for example, access control.

Structural object class

The classification of an object which gives its position in the object class hierarchy.

Subentry

An entry in the [DIT](#) which must be placed immediately below an administrative point entry. Subentries contain policy information, and are used by the Directory itself.

Subschema

Information to regulate the entries held by a [DSA](#). Part of the total [Directory schema](#).

Superior reference

A reference forming part of a reference path from its holder to the Root. A superior reference is required for all [DSAs](#) which do not master a first level naming context. It is the Access Point of a [DSA](#) to contact if there is no other suitable reference to progress the operation.

Supplier

The [DSA](#) supplying information in a shadowing agreement.

System schema

The set of rules governing operational information.

Tcl

Tool Command Language.

Tclnish

A command line interface [DUA](#) for data managers, operating over [DAP](#). See also [Ltclnish](#).

TCP/IP

Transport Control Protocol/Internet Protocol.

TLS

Transport Layer Security protocol. The [TLS](#) protocol provides communications privacy over the Internet.

Top

The entry at the top of a naming context, that is, the context prefix entry. It belongs to an abstract object class, and is therefore only used by the Directory.

X.500

A set of standards devised for the Directory, developed jointly by the [ITU-T](#) and [ISO/IEC](#), also known as [ISO 9594](#).

X.501

A set of standards for [X.500](#) models.

X.509

A set of standards for X.500 authentication.

X.511

A set of standards for X.500 abstract service.

X.518

A set of standards for X.500 distributed operations.

X.520

A set of standards for X.500 selected attribute types.

X.521

A set of standards for X.500 selected object classes.

X.525

A set of standards for X.500 replication.