Penrose Virtual Directory 2.0

Admin Guide

Publication date: Released March 27, 2009

Penrose Virtual Directory 2.0 Admin Guide

1801 Varsity Drive
Raleigh, NC 27606-2072USAPhone: +1 919 754 3700
Phone: 888 733 4281
Fax: +1 919 754 3701
PO Box 13588Research Triangle Park, NC 27709USA

About This Guide

Penrose Virtual Directory is a lightweight and flexible server which creates a bridge between LDAP, database, Active Directory, and NIS servers, and any application which stores its content in databases or directory services, and combines the information from all of these separate data sources into a single, LDAP-style directory. The entries and hierarchy of this directory tree are created dynamically as operations come into the server, based on predefined directory configuration parameters and mapping rules for entries and attributes. Since data are not copied to a central source, just pulled as necessary, this is a *virtual* directory.

Penrose Virtual Directory is simple to configure and use. Its interface, Penrose Studio, makes it easy to navigate both the configuration and the virtual directory. Penrose Server is based on widely-known standard protocols and easily edited XML files.

This guide is intended for experienced system administrators who are deploying and managing a virtual directory.

1. Content Overview

This admin guide describes how to create virtual directories, set up identity federation between NIS and LDAP sources, manage data sources, install the Penrose Server and Penrose Studio, and configure the server.

The first section in this guide contains general concepts, installation, and basic usage for Penrose Server and Penrose Studio:

- *Chapter 1, Overview of Penrose Virtual Directory* describes the general concepts behind virtual directories, metadirectories, and proxies. This also covers planning considerations for deploying Penrose Virtual Directory.

- *Chapter 2, Installing Penrose Virtual Directory* contains installation procedures and requirements for Penrose Server.

- *Chapter 3, Basic Usage* contains basic configuration for the Penrose Server, such as configuring SSL, checking logs, and changing the server properties.

- *Chapter 4, Using Penrose Studio* contains a walkthrough of Penrose Studio and common actions for managing virtual directory entries in Penrose Studio.

The second section covers setting up the virtual directory and its required entries and configuring identity federation:

- *Chapter 5, Managing Partitions* describes creating *partition* entries, the container for a virtual directory and all of its elements.

- *Chapter 6, Configuring Connections* describes how to create *connections*, entries which define server machines that Penrose Virtual Directory can access.

- *Chapter 7, Configuring Data Sources* describes how to configure data *sources*, applications, servers, or databases which contain information polled by Penrose Server to create virtual directory entries.

- *Chapter 8, Configuring the Virtual Directory* describes how to configure the virtual directory tree.

- *Chapter 9, Mapping Entries and Attributes* describes how to configure mappings between source attributes and virtual entry attributes.

- *Chapter 10, Configuring Identity Federation* describes how to configure identity federation, a way of uniting NIS, Active Directory, and LDAP sources in a global repository. This method can also be used for NIS migrations to LDAP or LDAP migrations to Red Hat IPA.

The next section contains additional and advanced configuration options for Penrose Server:

- *Chapter 11, Configuring Modules* describes how to add additional modules to extend the functionality of Penrose Server.

- *Chapter 12, Using Services with Penrose Virtual Directory* describes the default services for Penrose Virtual Directory, additional LDAP and database servers integrated with Penrose Virtual Directory to communicate with sources.

- *Chapter 13, Customizing Schema* contains information on the default schema files contained with Penrose Server and procedures for creating custom schema to use for virtual directory entries.

- *Chapter 14, Configuring Cache* describes the different cache settings for Penrose Virtual Directory and the different performance impacts, depending on the load and operations for the virtual directory.

The appendix covers the management tools included with Penrose Virtual Directory (*Appendix A, Using Penrose Virtual Directory Command-Line Tools*).

2. Examples and Formatting

All of the examples for Penrose Virtual Directory commands, file locations, and other usage are given for Red Hat Enterprise Linux 5 32-bit systems. Be certain to use the appropriate commands and files for your platform.

To start the Penrose Virtual Directory:

```
service vd-server start
```

Example 1. Example Command

All of the tools for Penrose Virtual Directory are located in the **`/opt/vd-server-2.0/bin`** directory on Red Hat Enterprise Linux 5 32-bit systems. These tools can be run from any location without specifying the tool location.

Certain words are represented in different fonts, styles, and weights. Different character formatting is used to indicate the function or purpose of the phrase being highlighted.

Formatting Style	Purpose
`Monospace font`	Monospace is used for commands, package names, files and directory paths, and any text displayed in a prompt.
`Monospace with a background`	This type of formatting is used for anything entered or returned in a command prompt.
Italicized text	Any text which is italicized is a variable, such as *instance_name* or *hostname*. Occasionally, this is also used to emphasize a new term or other phrase.
Bolded text	Most phrases which are in bold are application names, such as **Cygwin**, or are fields or options in a user interface, such as a **User Name Here:** field or **Save** button.

Other formatting styles draw attention to important text.

NOTE
A note provides additional information that can help illustrate the behavior of the system or provide more detail for a specific issue.

IMPORTANT
Important information is necessary, but possibly unexpected, such as a configuration change that will not persist after a reboot.

WARNING
A warning indicates potential data loss, as may happen when tuning hardware for maximum performance.

3. Additional Reading

The *Penrose Admin's Guide* describes how to install, configure, and administer Penrose Virtual Direct-

ory services, as well as set up other services to integrate with the virtual directory, browse and search the directory, and organize information. This guide is targeted for Penrose administrators.

The documentation for Penrose Virtual Directory includes the following guides:

- *Penrose Admin's Guide* explains all administrative functions for the Penrose, such as adding users, mapping entries, adding services, and using Penrose Server.

- *Release Notes* contains important information on new features, fixed bugs, known issues and work-arounds, and other important deployment information for Penrose Virtual Directory 2.0.

4. Samples Files

There are dozens of example configurations in the **`/opt/vd-server-2.0/samples`**, including files for identity federation and synchronization, access controls, Active Directory schema, referrals and proxies, and different types of sources. This is an excellent reference to use for setting up many of the features and scenarios in Penrose Virtual Directory, so refer to those files frequently as you configure the virtual directory.

Overview of Penrose Virtual Directory

A virtual directory creates a consolidated, high-level directory view from different sources of information. Penrose Virtual Directory is a simple and flexible way to make accessing information across a network environment easier, whether this involves a new view of LDAP and database sources, bridging between Active Directory and other LDAP servers, or performing an easier NIS migration.

This chapter provides an overview of the purpose and opportunities of using a virtual directory, the components in Penrose Virtual Directory, and considerations when planning Penrose Virtual Directory deployment.

1.1. Explaining Virtual Directories

Most organizations have a "hetergenous" network environment — meaning there is a messy combination of databases, NIS servers, Window and Unix LDAP servers which all contain important information. In heterogenous environments, these servers and applications usually cannot communicate directly with one another, or cannot communicate effectively, so the information get partitioned off into its respective server, even usernames and passwords.

Virtual directories consolidate that information without having to perform migrations, find complex software options, or try to reconfigure thousands of different entries. Virtual directories create a view or picture of a single, consolidated LDAP server and function as a type of lightweight middleware between the different servers and users. LDAP clients from email programs to servers communicate with the virtual directory. The virtual directory, in turn, talks to each server or database in its native protocol.

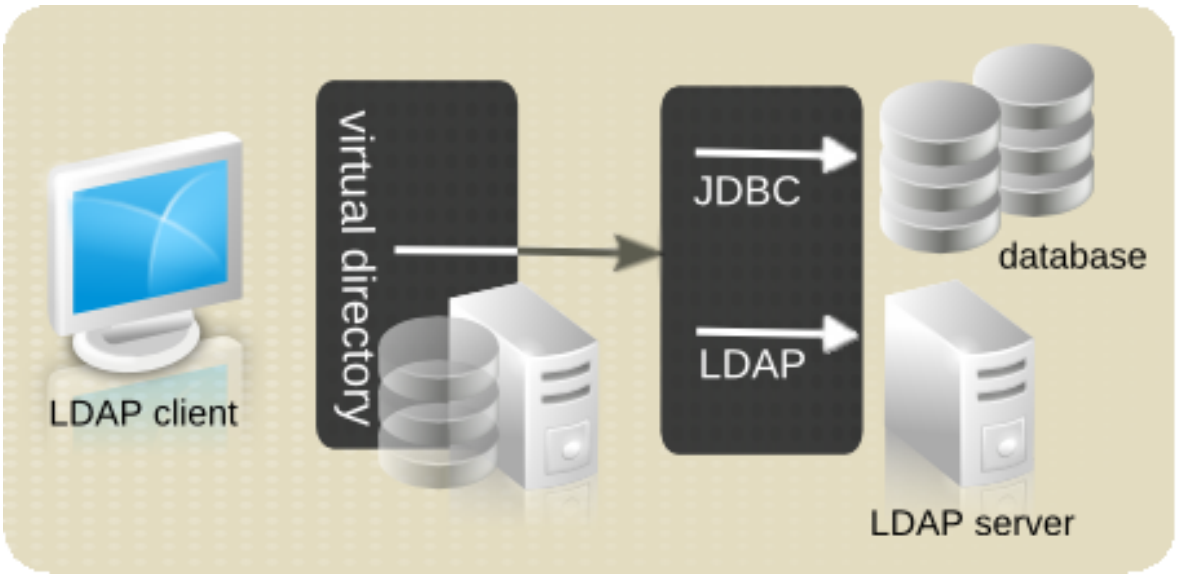


Figure 1.1. Overview of Penrose Virtual Directory

A virtual directory performs two essential functions: it provides a single source for accessing information and it creates a flattened, standardized namespace. Those two features of a virtual directory make it ideal for consolidating data in different formats from different applications.

A virtual directory is a single source because that is the only directory users need to access to get complete access to information. The virtual directory is configured to connect to each data source

(without having to reconfigure the data source itself). The user or client sees the virtual directory as an LDAP directory service.

Since the virtual directory simulates an LDAP server, it has a hierarchy of subtrees and entries which correspond to actual entries in the data sources. The virtual subtrees can be named whatever the administrator wants. Even the names of the entries and the names of their attributes can be translated according to the new virtual hierarchy. Standardizing distinguished entry names is called *namespace flattening*.

There is an important distinction between a virtual directory and *metadirectory*. A virtual directory loosely couples identity data and applications; this is a real-time, dynamically-generated view, not a separate directory. A metadirectory, while also giving a consolidated view of data, adds a layer of infrastructure above native repositories. A metadirectory actually draws data out from the source and copies it into a new consolidated directory. A virtual directory is much more agile to used and configure and easier to synchronize data.

By functioning similarly to an LDAP directory, Penrose Virtual Directory has several advantages that make deploying the solution ideal:

- Centralized configuration
- Robust authentication
- Solid integration with other applications, such as mail servers and email clients
- Easily configured replication
- Widely-utilized standard protocol

1.2. Looking at Penrose Virtual Directory

Penrose Virtual Directory is comprised of several components:

- Penrose Virtual Directory Common, the set of classes by used all other components, including base classes, configuration classes, XML readers and writers, DTD definitions, utilities
- Penrose Virtual Directory Core, the primary virtual directory component which handles partitions, source configuration, directory entries, and modules

NOTE

Penrose Virtual Directory Core can be embedded in another application without running an LDAP service.

- Penrose Server, a lightweight application server which combines several different LDAP services to communicate with clients through OpenDS or ApacheDS and a JMX service to communication with Java clients

- Penrose Studio, an intuitive GUI to manage Penrose Server configuration

- Command-line tools and classes to manage Penrose Server remotely

Penrose Server uses Java MBeans to instantiate and manage all aspects of the virtual directory configuration, so Penrose Server can run on any platform which supports Java. Penrose Server is the primary component that users will use to create the virtual directory:

- Mapping attributes and values from multiple sources into virtual entries, and converting and manipulating attribute values

- Flattening the namespace and intelligently routing queries

- Configurable caching

- Access control information

- Load-balancing and failover

- Bidirectional synchronization between different LDAP servers

The Penrose Studio is a powerful yet simple tool to make managing Penrose Server easier. This provides wizards to configure partition and identitty federation entries, set up access control, and configure mapping. The Penrose Studio also includes an integrated LDAP browser to view the virtual directory and browsers to see the hierarchy of the real data sources.

1.3. Planning Penrose Virtual Directory

A virtual directory at a high level combines entries (identities), a process called *identity joining*, and also maps the attributes in the source to the virtual entry. Penrose Virtual Directory has several different ways that identity joining and mapping can be configured.

The following sections cover issues that should be considered while planning a virtual directory so that information and entries are effectively joined and the virtual hierarchy and configuration meet your needs.

1.3.1. Identifying Current Data Sources

Identify what sources in the infrastructure need to be consolidated.

- *What servers need to be joined?*

  Perform a survey of your complete infrastructure, across geographical locations and departments. Identify any applications or areas where the same person or identity has multiple entries; these can be joined in the virtual directory.

- *What tables or subtrees will supply information for the virtual directory?*

  Identify what parts of the infrastructure need to be included, such as the subtrees of an LDAP directory. This can also be part of auditing the information which will go into the virtual directory to help plan the virtual directory tree.

NOTE

Penrose Virtual Directory does not distinguish between virtual views in an LDAP directory or database and real subtrees and tables. Therefore, virtual views can be treated as parts of the data sources.

- *What protocols are used?*

  Penrose Virtual Directory, by default, communicates with JDBC and LDAPv3 protocols, so the server can communicate with any database or LDAPv3 compliant server, such as Red Hat Directory Server, OpenLDAP, and Active Directory. Penrose Virtual Directory can be configured to communicate to NIS servers, as well.

  The protocols used for the servers can also influence the identity joining method (described in *Section 1.3.2, "Configuring a Virtual Directory through Partitions"* and *Section 1.3.3, "Joining LDAP, NIS, and Active Directory Servers through Identity Federation"*). For example, for an environment with several database applications and LDAP services, a partition virtual directory. For a NIS migration, use identity federation.

- *What attributes should be mapped?*

  Joining is configured per attribute as well as per identity, so identify what information should be included in the consolidated entry. Look for each attribute to include and what data source will supply the value.

- *What schema elements will be used?*

  Since the virtual directory is strucutred as an LDAP directory, it requires defined object classes and attributes for each kind of entry. Identify what kind of LDAP schema or custom schema elements to include.

1.3.2. Configuring a Virtual Directory through Partitions

The most common method of creating a virtual directory in Penrose Virtual Directory is through creat-

ing *partitions*. A partition defines virtual directory for LDAP servers and databases. Individual entries are created for each host server, application, subtree and entry, and mapped attribute. Each piece of information is defined separately:

- The directory, database, or other application which contains the information (the *source*)
- The connection information to the server machine which hosts the data source (the *connection*)
- The hierarchy of the new, virtual directory being created (the *directory*)
- The relationships between the entries and attributes in the different data sources (the *mapping*)

NOTE

The entry mapping — the way the disparate entries and attributes are combined into the virtual entries — is also configured within the partition, but this is more complex and has many different options. Entry mapping is covered in *Chapter 8, Configuring the Virtual Directory*.

- Additional code which affects the partition behavior, such as additional mapping functionality (the *module*, described in *Chapter 11, Configuring Modules*)

These configurations, collectively, comprise the server *partition* entry in Penrose Server. There can be an unlimited number of partitions configured for a single Penrose Server server instance; though the default is to use a single partition. Likewise, there can be an unlimited number of sources, connections, and mappings for a single partition.

A virtual directory partition is illustrated in *Figure 1.2, “A Virtual Directory Partition”*.

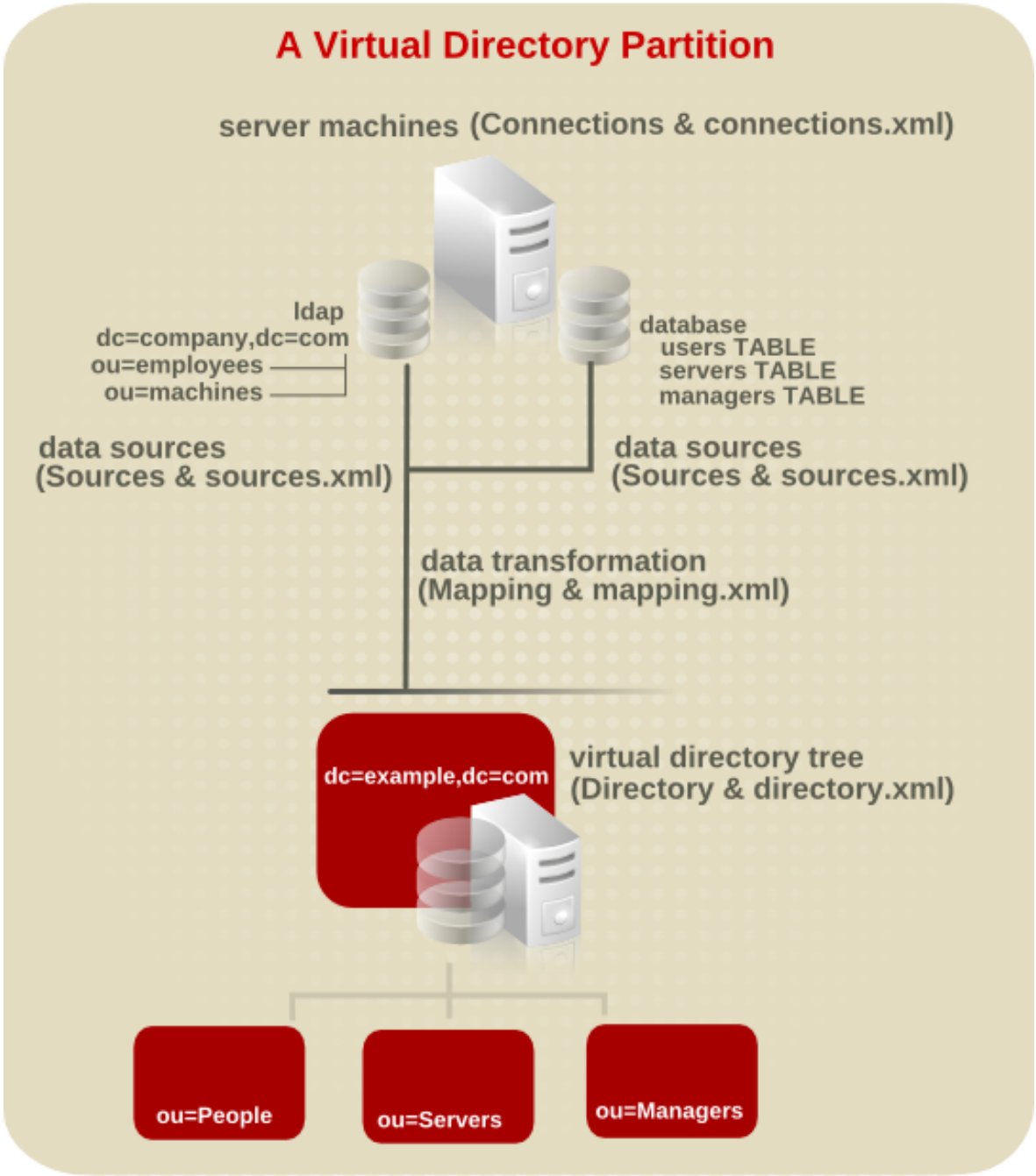


Figure 1.2. A Virtual Directory Partition

In *Figure 1.2, "A Virtual Directory Partition"*, each component in the partition has two areas specified, such as the connection having both **`connections.xml`** and **Connection**. All of the Penrose Server partition components are defined in XML files, which can be edited directly. These components can also be configured through Penrose Studio, as shown in *Figure 1.3, "Partition Configuration in Penrose Studio"*. For a connection, then, **`connections.xml`** is the configuration file and **Connection** is the folder in the Penrose Studio hierarchy for changes.

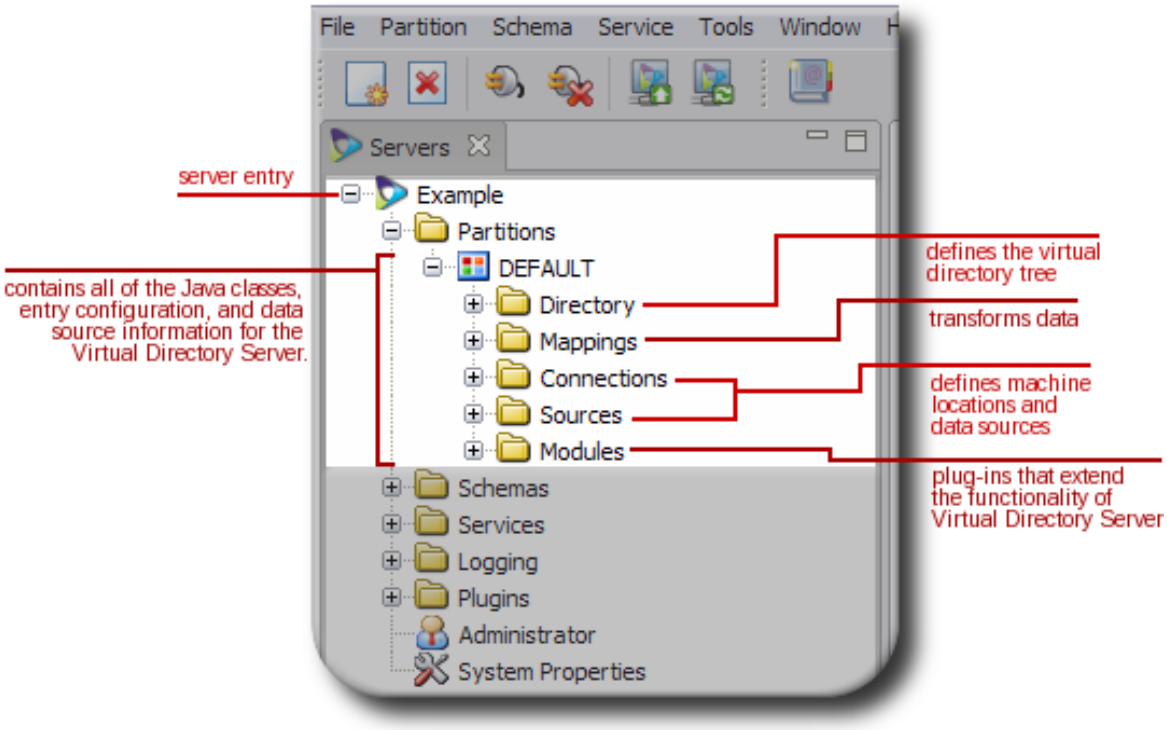


Figure 1.3. Partition Configuration in Penrose Studio

1.3.3. Joining LDAP, NIS, and Active Directory Servers through Identity Federation

Identity federation or *identity linking* is a method to unite entries from Active Directory, LDAPv3, or NIS servers. This method can be used to synchronize these types of servers or to simplify a NIS migration.

Identity federation, like partition directories, takes source entries to create a virtual directory entry, but is for different source environments.

Identity linking (or federation) creates a hybrid virtual directory and meta directory by using a global repository to combine and synchronize entries from the different sources. Local repositories (analogous to sources in the virtual directory configuration) are linked to the global repository. Using a mapping tool, the local identities are then matched to entries in the global repository.

With identity federation, there is a *global repository*, a separate and authoritative source for entries. In this figure, a NIS source is used to create the initial global entry. After that global repository is defined, entries from other sources (*local repositories*) are linked to the global entry either manually or through matching rules.

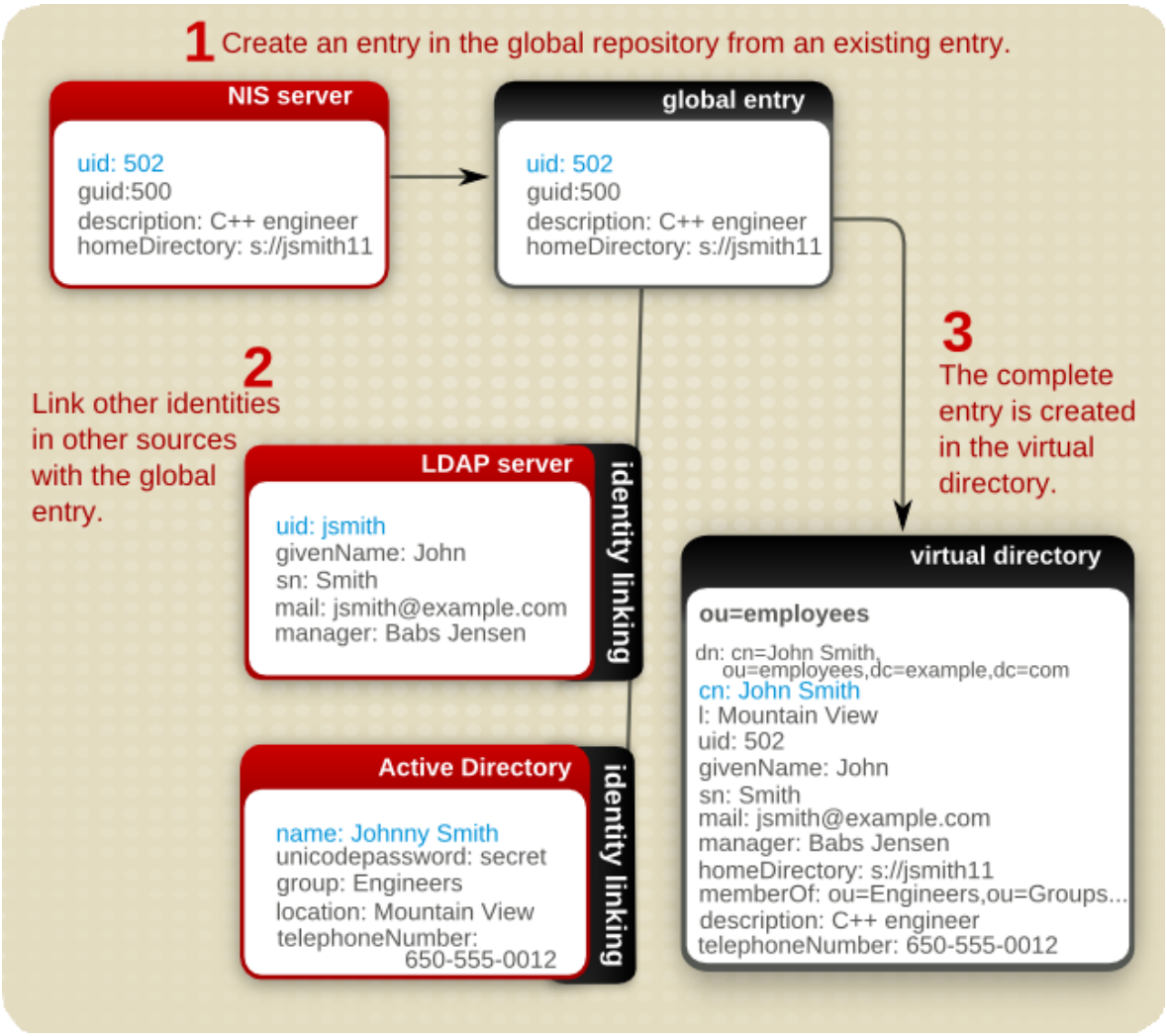


Figure 1.4. Merging Entries through Identity Linking

1.3.4. Synchronizing Active Directory and Other LDAP Services

Active Directory and other LDAPv3 compliant directory services cannot communicate or synchronize data easily. Some directory services have tools which try to bridge the differences in schema, entry naming, and directory hiearchy, such as Windows Sync in Red Hat Directory Server. However, these tools are usually limited to a small subset of entry attributes (without any support for custom or non-standard attributes) and have restrictions on the kinds of entries and locations in the subtrees where sycnhronization can occur.

Penrose Virtual Directory aids synchronization between Active Directory and LDAPv3 compliant directory services, including support for all Active Directory schema and custom schema.

Active Directory and LDAP synchronization can be configured through a partition or through identity federation; the method does not really matter. The key is in mapping the attributes between the Active Directory and LDAP servers.

The synchronization with Penrose Virtual Directory will check activity on one server and then write those changes over to the other server. There are two ways that Penrose Virtual Directory keeps track

of changes:

- A *full synchronization*, which periodically takes snapshots of one server, compares it to the data on the target server, and writes over any changes.

- An *incremental update*, which uses a change log to track and copy over changes to the target server.

This method can also be used to synchronize other LDAPv3 servers which normally cannot be synchronized, such as OpenLDAP and Red Hat Directory Server, using identity federation.

1.3.5. Migrating from NIS Servers to LDAP Servers

Penrose Virtual Directory simplifies NIS server migrations to LDAP, without interrupting the NIS service and remaining transparent to users.

Penrose Virtual Directory uses a *lazy migration* process, there there are two views of the data, one in the old location and one in the new location. In Penrose Virtual Directory, the old NIS location is a *local repository*. All of the NIS entries are migrated over to a new LDAP server, called a *global repository*. This migration can take several weeks, with a mix of entries in the old and new data formats. Al entries will be available in the new data source.

At any point in the migration process, data can be accessed on either the old or new repositories.

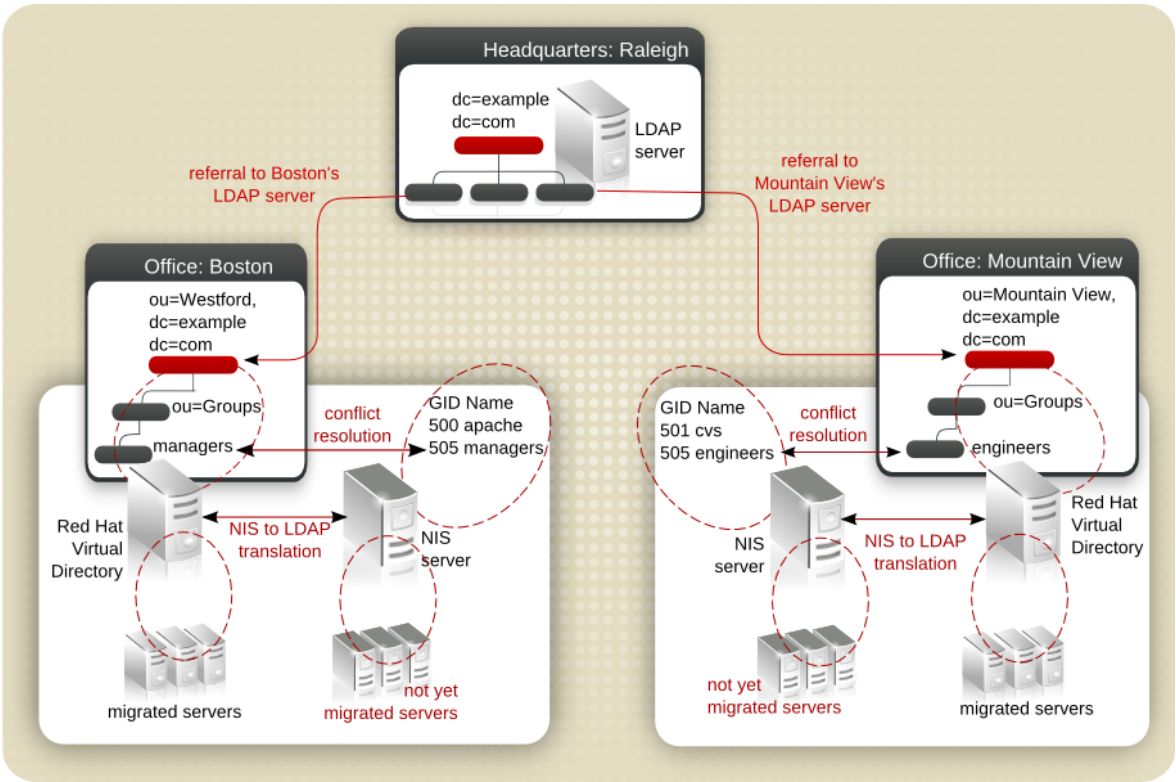


Figure 1.5. Example of a NIS to LDAP Migration

A lazy migration is much simpler than trying to force a reconfiguration of all applications at the same time. Applications and clients can be switched over incrementally. This also eliminates almost all of the downtime of the migration, since the switchover takes just a few seconds. There is no need to cram a migration into a rigid outage window.

Penrose Virtual Directory supplies several tools which make is easy to match entries between NIS and LDAP repositories and to resolve UID and GID conflicts, including reassigning file ownerships. The NIS migration process using Penrose Virtual Directory is as follows:

1. Copy the information in the NIS servers to a central directory server, a *global repository*.

2. Configure the global repository subtrees to contain three branches:

   - Create a subtree for the current NIS servers, such as **ou=nis**. This will be mapped to the current, old NIS server entries.

   - Create a subtree for the new global repository, such as **ou=global**. This is mapped to the new global LDAP repository; any new entries are written to the global repository, and any changes write the NIS entries over to the global repository.

   - Create a subtree for users to access, such as **ou=nss**. Before the migration begins, **ou=nss** has a referral to the NIS subtree, **ou=nis**. After the NIS migration begins, this has a referral to the global repository subtree, **ou=global**.

3. Link the identities in the NIS server to entries in the global repository.

4. If there are multiple NIS servers, then the same unique ID number many have been assigned to more than one user. These UID, as well as group ID, conflicts can be resolved using Penrose Virtual Directory's UID/GID Conflict Resolution tool.

5. After any UID/GID conflicts are resolved, new UIDs will be reassigned. After reassigning UIDs, then the file ownership for any files associated with that UID or GID must also be reassigned. This is done using the *Ownership Alignment Tool*.

6. The migration of entries begins. Any changes to the NIS entries are written over to the global repository.

7. As the migration for a server is completed, change the PAM modules on the NIS server hosts to point to the global repository rather than the NIS server.

8. Switch from performing operations against the NIS server (**ou=nis**) to running against the global LDAP repository (**ou=global**). Users have always performed operations against **ou=nss**, so change the referral for that subtree from **ou=nis** to **ou=global**.

1.3.6. Migrating to Red Hat IPA

Many applications do not have a native migration path to more advanced servers; Penrose Virtual Directory can fill in the gap for migration and allow NIS servers and LDAP servers to migrate to Red Hat IPA. This migration is even possible for LDAP servers in a Kerberos/SASL environment, which would otherwise be trickier to manage.

As with a NIS to LDAP migration, Penrose Virtual Directory can use a lazy migration to allow migration to happen incrementally by supply two different resources, the original source and the new IPA source, and working as a pass-through application. Penrose Virtual Directory also resolves entry, UID number, group ID number, and file ownership conflicts through its toolset.

The process for migrating to Red Hat IPA is very similar to migrating to NIS to LDAP, and as in that migration, this uses identity federation to unify and resolve entries, as shown in *Figure 1.4, "Merging Entries through Identity Linking"*:

1. Copy the initial entry information into Red Hat IPA, and set up Red Hat IPA as the *global repository*.

2. Configure the global repository subtrees to contain three branches: one for the current server (for example, **ou=ldap**), one for the IPA server (for example, **ou=ipa**), and one to which to point clients (for example, **ou=federation**). Having a third subtree makes the process easier on clients; they simply access that subtree at any point in the migration. The **ou=federation** subtree contains a referral to the appropriate source. Before migration, this referral directs to the original source, **ou=ldap**, and after migration, it points to the IPA subtree, **ou=ipa**. The referral can be changed without affecting users.

3. Link the identities in the old NIS or LDAP server to entries in the IPA global repository.

4. Resolve any conflicts. If there are multiple servers being migrated, more than one entry may have the same user or group ID. Two tools helps resolve the UID/GID conflicts and then align the file ownership with the new UID numbers.

5. Begin migrating entries, clients, and applications.

6. When migration is complete, switch from performing operations against the old server subtree (**ou=ldap**) to running against the global IPA repository (**ou=ipa**). Users have always performed operations against **ou=global**, so change the referral for that subtree from **ou=ldap** to **ou=ipa**.

Using Penrose Virtual Directory to assist with the incremental migration to IPA has several benefits:

- Synchronized data between old servers and new IPA servers.

- Integrated tools to reconcile UID conflicts and to simplify federating entries.

- No lost time or outage windows to coordinate simultaneous migrations.

- Transparent to users.

- Extensible support for legacy schema which may be required or hardcoded in legacy applications; by using Penrose Virtual Directory as a proxy, these legacy applications can communicate with IPA.

- Synchronization between IPA and other applications, such as Active Directory.

1.3.7. Planning Authentication

The most common reason administrators want a virtual directory is to provide LDAP-style uathentication to databases or to provide a single sign on service when a user has multiple logins.

Penrose Virtual Directory can perform simple authentication through both partition-style hierarchies and identity federation. Identity federation, with the different relationship established between repositiories, also can be configured for *stacking authentication*. Stacking authentication can verify a user's credentials against several repositories in a single operation. A user's credentials are first presented to the global repository, then cascades down all of the configured local repositories.

For example, a user attempts to bind to a Red Hat Directory Server instance through Penrose Virtual Directory with the following command:

```
ldapsearch -h rhvd.example.com -p 10389 -D
uid=jsmith,ou=users,ou=nisdomain,ou=nss,dc=example,dc=com -w secret -x -b
"" -s subtree
```

The authentication operation proceeds as follows:

1. Penrose Virtual Directory searches the local repository for a user with the *uid* **jsmith**.

2. If the local entry is found in the local repository, then Penrose Virtual Directory searches the global repository for an entry which contains a link to the local entry, meaning an entry with a *seeAlso* attribute which contains the local entry DN.

3. If there is a global entry, Penrose Virtual Directory checks the other entries contained in the *seeAlso* attribute for any account disable, account lockout, or account expiration attributes.

4. If the account is still active, then Penrose Server checks the global identity for a *userPassword* attribute.

5. Penrose Server uses the local identity's *uid* and global identtity's *userPassword* to bind to the specified Red Hat Directory Server.

TIP

Stacking authentication is ideal for NIS migration because it allows users to authenticate against a virtual NSS subtree, and that authentication is then performed against all of the federated NIS identities.

Installing Penrose Virtual Directory

This chapter describes the complete procedure to install Penrose Virtual Directory, along with dependencies.

2.1. Supported Platforms

Penrose Server 2.0 is supported on the following platforms:

- Red Hat Enterprise Linux 4.7 i386 (32-bit)
- Red Hat Enterprise Linux 4.7 x86_64 (64-bit)
- Red Hat Enterprise Linux 5.2 i386 (32-bit)
- Red Hat Enterprise Linux 5.2 x86_64 (64-bit)

NOTE

Penrose Virtual Directory 2.0 is supported running on a virtual guest on Red Hat Enterprise Linux 5 Virtualization Server.

2.2. Required Software

Penrose Virtual Directory requires Sun Java Development Kit (JDK) version 1.5.0.

1. Download Sun JDK 5.0 from *http://1java.sun.com/1javase/1downloads/1index_jdk5.jsp*.
2. Install the JDK. This may be downloaded as an RPM or as an executable binary. For example:

```
./jdk-1_5_0_17-linux-amd64.bin
```

2.3. Installing Penrose Server

A virtual directory maps information from disparate data sources, such as directory services and databases, into a single location for users to access, while keeping the virtual service lightweight and simple to administer.

1. Download the RPMs. There are two packages for Penrose Server, one for the core server and the

other for client tools.

2. Install the packages. For example:

```
rpm -i vd-server-2.0-build#.el5.noarch.rpm
vd-client-2.0-build#.el5.noarch.rpm
```

The Penrose Server components are installed in the **/opt/vd-server-2.0** and **/opt/vd-client-2.0** directories.

3. Make sure that the proper JDK is configured for Penrose Server to use:

   a. Open the **vd.conf** file with the Penrose Server directory.

```
vim /opt/vd-server-2.0/etc/vd.conf
```

   b. Add the **JAVA_HOME** environment variable, pointing to Sun JDK 1.5.0. For example:

```
JAVA_HOME=/usr/lib/jdk1.5.0_17/
```

   c. After editing the **vd.conf** file, copy it into the host's **/etc** directory.

```
cp /opt/vd-server-2.0/etc/vd.conf /etc
```

4. Optionally, configure the Penrose Server to run as a service:

   a. Open the init script directory.

```
cd /opt/vd-server-2.0/etc/init.d
```

   b. Edit the **vd-server** script so that the Penrose Server home and script locations are correct. For example:

```
VD_SERVER_HOME=/opt/vd-server-2.0
VD_SERVER_SCRIPT=$VD_SERVER_HOME/bin/vd-server.sh
```

   c. Copy the init file to the **/etc/init.d** directory.

```
cp /opt/vd-server-2.0/etc/init.d/vd-server /etc/init.d
```

d. Make the init script executable.

```
chmod +x /etc/init.d/vd-server
```

5. Run a configuration script to reset the server hostname, give the admin username and password, and set the port numbers and other information for the associated LDAP and JMX services. Hitting **`Enter`** accepts the defaults in the brackets.

   For example:

```
[root@server bin]# ./vd-config.sh
Configuring VD Server:
----------------------

Hostname [server.example.com]:
Root DN [uid=admin,ou=system]:
Root Password [*****]: secret12
User account [root]:
Group account [root]:

Configuring OpenDS Service:
---------------------------

LDAP Enabled [true]:
LDAP Port [389]:
Secure LDAP Enabled [true]:
Secure LDAP Port [636]:
SSL Certificate Name [server-cert]:
Key Store Type (JKS/PKCS12) [PKCS12]:
Key Store File [config/keystore.p12]:
Key Store PIN File [config/keystore.pin]:


Configuring JMX Service:
------------------------

RMI Port [1099]:
RMI Transport Port [40888]:
```

2.4. Installing Additional Libraries

Additional libraries can be installed on Penrose Server so that extended functions can be deployed. These libraries cover a range of different Penrose functions, including JDBC drivers, custom adapters, custom modules, and other third party libraries.

NOTE
Any additional libraries *must* be JAR files.

1. Copy the JAR files into the **`lib/ext/`** directory in the Penrose Server home directory; for example:

```
cp /export/example.jar /opt/vd-server-2.0/lib/ext/example.jar
```

2. Restart Penrose Server.

```
service vd-server restart
```

2.5. Installing Additional Security Providers

Although some encryption support is included with the required JDK, some directory resources or clients may require additional encryption support.

2.5.1. Bouncy Castle Security Provider

Bouncy Castle Crypto package provides additional encryption methods, including **SHA** algorithms used by Penrose Virtual Directory, through a Java implementation. The Bouncy Castle security provider is required to run Penrose Virtual Directory and provides additional encryption and security services for Penrose Server.

NOTE
JDK 1.5 or higher must be installed in the Java home directory specified in the Bouncy Castle installation.

1. Download the Bouncy Castle provider for the JDK from *http://1www.bouncycastle.org/1latest_releases.html*.

2. Copy the Bouncy Castle JAR file, such as **`bcprov-jdk15-140.jar`**, to the Java home directory, such as **`/usr/lib/jdk1.5.0_17/`**.

3. Open the **`java.security`** file. For example:

```
vim /usr/lib/jdk1.5.0_17//jre/lib/security/java.security
```

4. Add the Bouncy Castle provider to the list of registered security providers. For example:

```
security.provider.1=sun.security.provider.Sun
security.provider.2=com.sun.net.ssl.internal.ssl.Provider
security.provider.3=com.sun.rsajca.Provider
security.provider.4=com.sun.crypto.provider.SunJCE
security.provider.5=sun.security.jgss.SunProvider
security.provider.6=org.bouncycastle.jce.provider.BouncyCastleProvider
```

2.5.2. JCE Unlimited Strength Jurisdiction Policy Files

Because of some governmental restrictions, some countries do not allow unlimited encryption strength. The JDK can be extended from its standard *strong but limited* encryption to unlimited strength by installing Java Cryptography Extension (JCE) files.

1. Download the JCE Unlimited Strength Jurisdiction Policy Files for the JDK.
   - *http://1java.sun.com/1j2se/11.4.2/1download.html*
   - *http://1java.sun.com/1j2se/11.5.0/1download.jsp*
2. Back up the existing policy files in the JDK home directory. For example:

```
zip security_files.zip /usr/lib/jdk1.5.0_17//jre/lib/security/
```

3. Copy the **`local_policy.jar`** and **`US_export_policy.jar`** files into the **`jre/lib/security`** directory in the JDK directory.

2.6. Uninstalling Penrose Server

The Penrose Server packages can be uninstalled using Red Hat's package management tools, the same as used to install it. To remove Penrose Server, use the `-e` option with **`rpm`**:

```
rpm -ev vd-server-2.0-build#.el5.noarch vd-client-2.0-build#.el5.noarch
```

2.7. Upgrading Penrose Virtual Directory

There is no direct upgrade or migration path to Penrose Virtual Directory 2.0. Updating the version requires updating the packages:

1. Stop the Penrose Server process, and verify the server is stopped.

```
service vd-server stop

ps -ef | grep vd-server
```

2. Back up or copy the **`/opt/vd-server-2.0/partitions/`** directory to save any custom partitions. If there are templates used for identity linking, save the templates and the federation domain's **`federation.xml`** file and configuration; it's not necessary to save the generated federation partitions.

   Also back up or copy the **`server.xml`** file in the **`/opt/vd-server-2.0/conf/`** directory. If SSL has been configured, save the **`cacerts`** file in the **`/opt/vd-server-2.0/conf/`** directory.

   Be sure to save these files to a different directory.

3. Uninstall all Penrose Virtual Directory RPMs. For example:

```
rpm -e vd-server
rpm -e vd-studio
rpm -e vd-client
```

4. Remove the remaining files.

```
rm -rf /opt/vd-server-2.0
rm -rf /opt/vd-studio-2.0
rm -rf /opt/vd-client-2.0
```

5. Install the new RPMs, and configure the services and configuation files as described in *Section 2.3, "Installing Penrose Server"*.

6. Copy in the saved **`partitions/`** directory and the **`server.xml`** and **`cacerts`** files.

   If there are templates used for identity linking, the restart the server to generate the federation partitions.

```
service vd-server restart
```

7. If NIS synchronization or identity federation was configured in the original Penrose Virtual Directory instance, then re-configure the **`/etc/yp.conf`** file.

   a. Open Penrose Studio.

```
/opt/vd-studio-2.0/vd-studio
```

b. Expand the **Federation** folder in the lower left of the navigation tree.

c. Select the federation domain.

d. Open the **NIS** folder.

e. There is a tab in the main window labeled **yp.conf**. Copy the information there into the **`/etc/yp.conf`** file on the Penrose Server host machine.

8. Check that you can connect to the Penrose directory by checking the directories using LDAP tools.

   - For example, verify a regular virtual directory configuration:

```
ldapsearch -h localhost -p 389 -x -b "" -s base
```

   - If identity federation is configured, check both the local and global directories. For example:

```
ldapsearch -h localhost -p 389 -x -b
"uid=jsmith,ou=Users,ou=local-nis,ou=NSS,dc=example,dc=com"

ldapsearch -h localhost -p 389 -x -b
"uid=jsmith,ou=Users,ou=Global,dc=example,dc=com"
```

   - If Penrose Server is used for user authentication, then attempt to log into a server using the proper credentials:

```
ldapsearch -h localhost -p 389 -x -D
"uid=jsmith,ou=Users,ou=local-nis,ou=NSS,dc=example,dc=com" -w secret -b
"" -s base
```

   - Last, check that NIS synchronization is working. For example:

```
/opt/vd-server-2.0/bin/nis.sh D uid=admin,ou=system -w secret synchronize
exampleDomain otherDomain
```

Basic Usage

This chapter contains common information about the configuration files and directories used in Penrose Studio and information on basic server configuration options, such as starting and stopping the server, establishing secure connections, and changing the root user and host information.

3.1. Starting the Service

Penrose Server is started by running a shell script, **`vd-server.sh`**, in the **`/opt/vd-server-2.0/bin`** directory. For example:

```
cd /opt/vd-server-2.0/bin

./vd-server.sh
PenroseServer        [ 126] Penrose Server is ready.
```

To stop the server, simply close the script.

3.2. Configuring Penrose Server to Run as a Service

Penrose Server can also stopped, started, and restarted using system tools on Red Hat Enterprise Linux. Init scripts are included with the configuration files with Penrose Server. and then copy them over to the **`/usr/sbin`** directory.

1. Log into the Penrose Server host machine as the **`root`** user.

2. Open the Penrose Virtual Directory init script directory.

```
cd /opt/vd-server-2.0/etc/init.d
```

3. Edit the **`vd-server`** script so that the Penrose Server home and script locations are correct. For example:

```
VD_SERVER_HOME=/opt/vd-server-2.0
VD_SERVER_SCRIPT=$VD_SERVER_HOME/bin/vd-server.sh
```

4. Copy the init file to the **`/etc/init.d`** directory.

```
cp /opt/vd-server-2.0/etc/init.d/vd-server /etc/init.d
```

5. Make the init script executable.

```
chmod +x /etc/init.d/vd-server
```

6. Test the new Penrose Virtual Directory service.

```
service vd-server start
```

After setting Penrose Server up as a service, it can be managed using the **`service`** on Red Hat Enterprise Linux:

```
service vd-server {start|stop|restart}
```

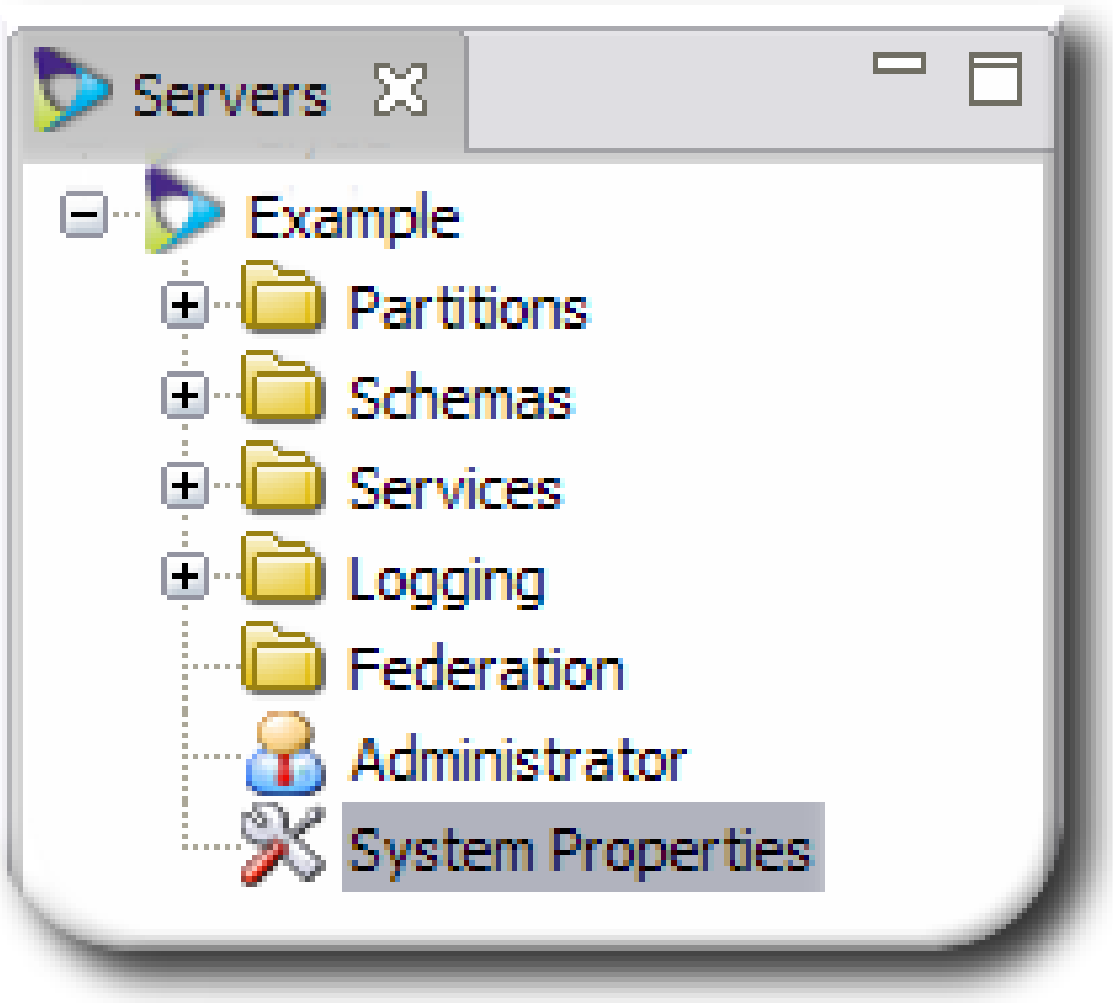
3.3. Configuring Penrose Server Host System Properties

The system properties for Penrose Virtual Directory define the connection information to access the Penrose Server. This is used by clients such as Penrose Studio and LDAP browsers.

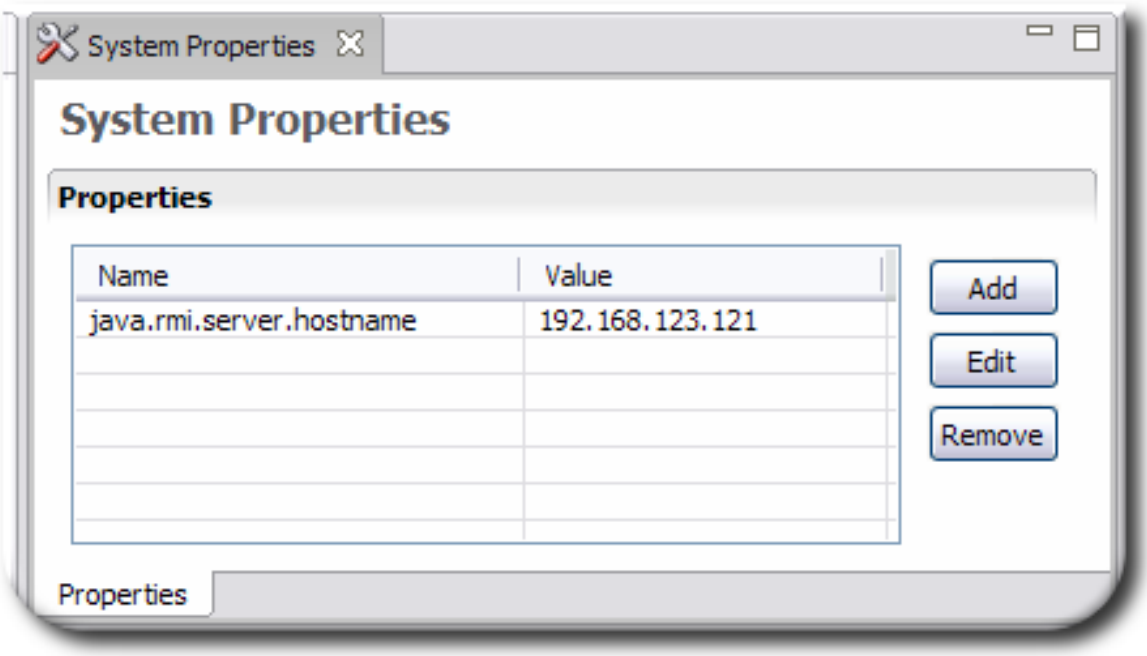
3.3.1. Configuring the Host System Properties in Penrose Studio

To set the host system information in Penrose Studio:

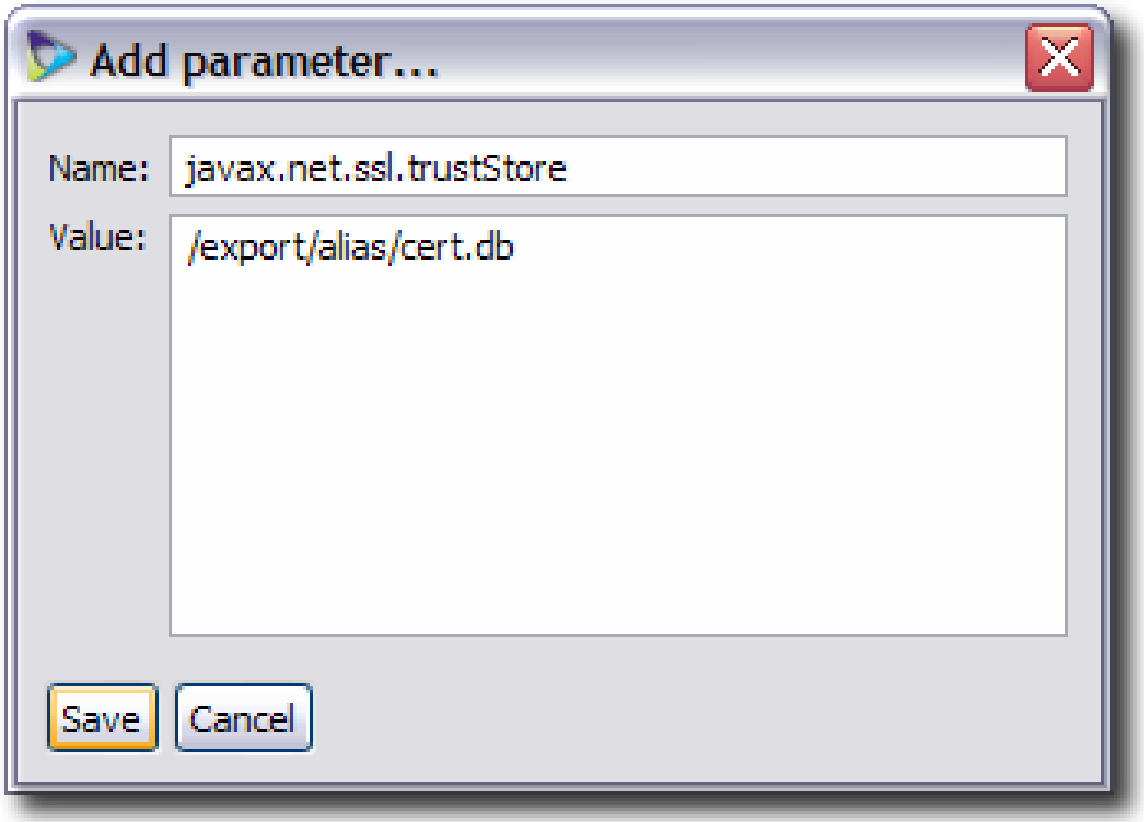
1. Click the **System Properties** link at the bottom of the server navigation window.



2. Click the **Add** button to add new system properties, or select and existing property and click **Edit** to change it.



3. Fill in the properties and values.



The hostname (**`java.rmi.server.hostname`**) is required. There are four optional parameters,

which are set in pairs:

- **javax.net.ssl.trustStore** and **javax.net.ssl.trustStorePwd**, which give the path to the server's certificate database and the password to access the database.

- **http.proxyHost** and **http.proxyPort**, which give the hostname and port of a proxy web server to use to access the server.

3.3.2. Configuring the Host System Properties in the Configuration File

The main file for configuring the Penrose service is **server.xml** in the **/opt/vd-server-2.0/conf** directory. This file defines important parameters such as the Java system properties, adapters, and the server's root user.

NOTE

The **server.xml** file is only loaded when the Penrose Server starts, so the server must be restarted after any changes to this file.

The Java properties define the host connection information, and the information set in the **server.xml** file are automatically loaded when Penrose Server starts. These properties include the host's connection information.

For example:

```
<server>

  <system-property>
    <property-name>java.rmi.server.hostname</property-name>
    <property-value>server.example.com</property-value>
  </system-property>

</server>
```

The *java.rmi.server.hostname* system property allows Penrose Studio to connect to Penrose Server.

Property	Description
java.rmi.server.hostname	Used by JMX clients to connect to Penrose Server; the default value is **localhost** and must be changed to the full hostname of the server.
javax.net.ssl.trustStore	Contains the path to the certificate database for Penrose Server.
javax.net.ssl.trustStorePwd	Contains the password to access the certificate

Property	Description
	database.
http.proxyHost	Contains the hostname of an HTTP proxy server.
http.proxyPort	Contains the port number of an HTTP proxy server.

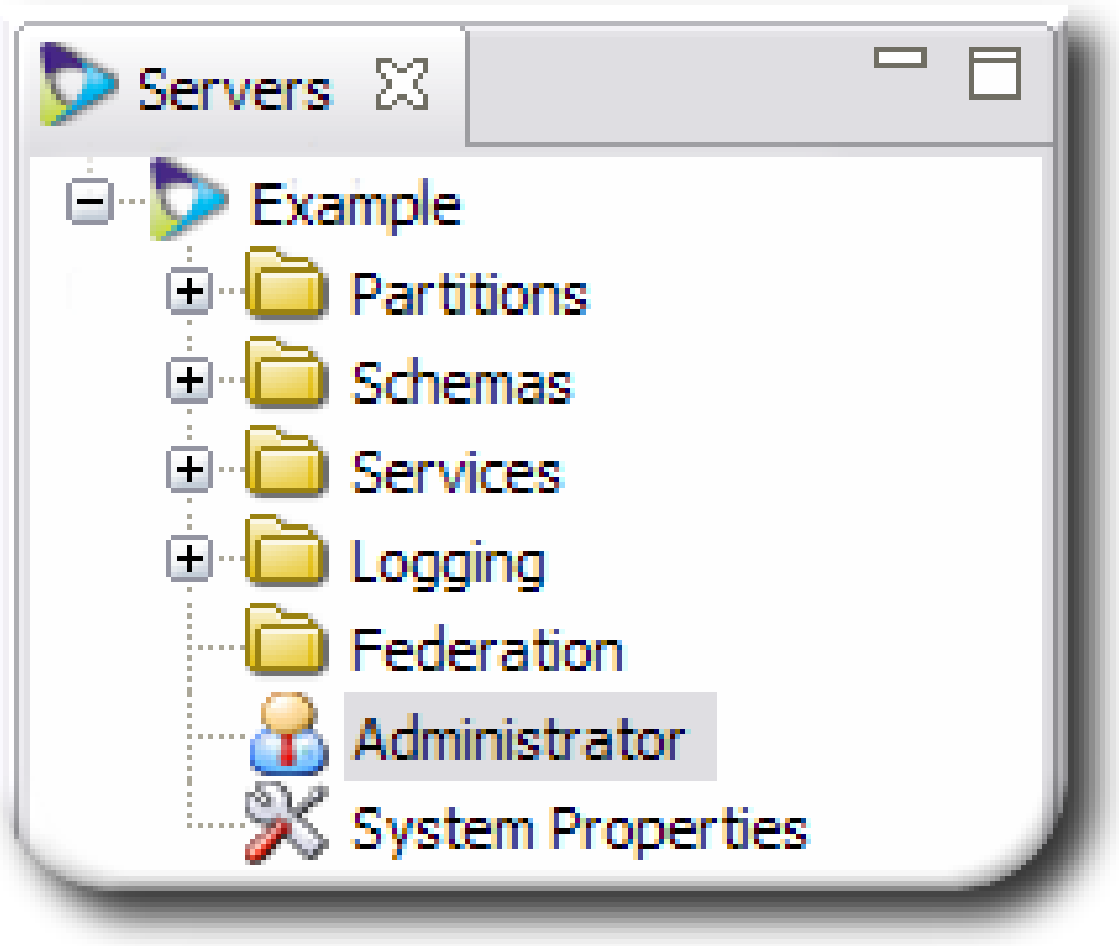
Table 3.1. Penrose Server Java System Properties

3.4. Setting up the Admin User

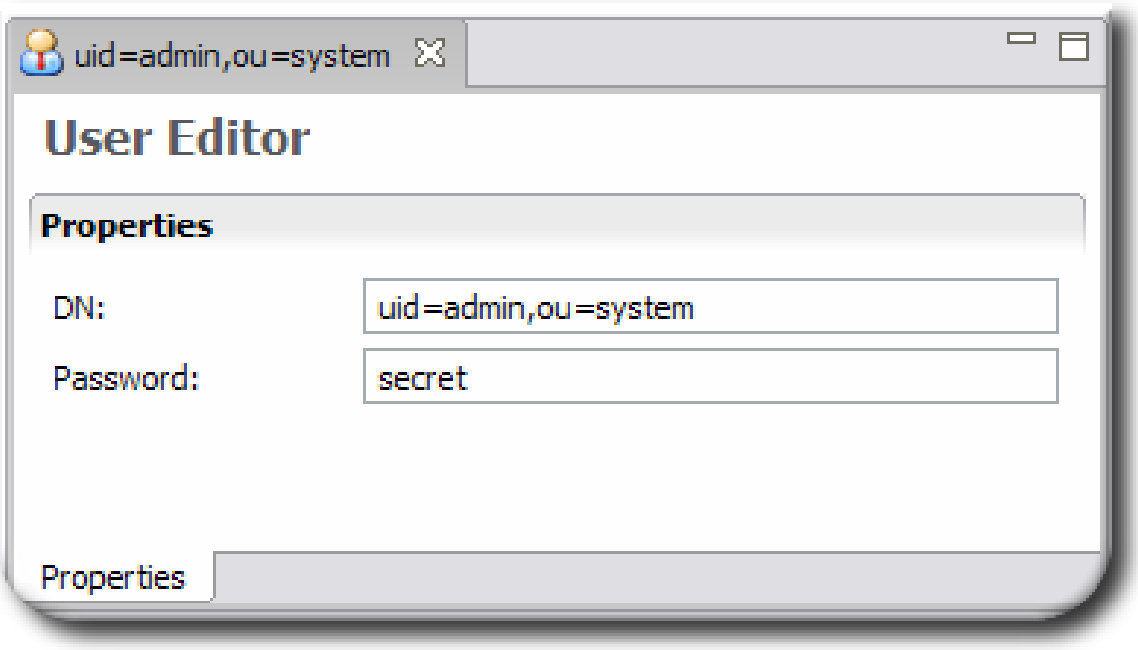
The root user for Penrose Server has full access to the Penrose Server application. LDAP clients which bind to Penrose Server as the configured root user bypass all access control settings for the virtual directory; therefore, any management clients, including Penrose Studio, must bind to the Penrose Server as the (Penrose Server) root user.

3.4.1. Editing the Admin User in Penrose Studio

1. Click the **Administrator** link at the bottom of the server navigation window.



2. Change the admin user's name or password in the appropriate fields.



The root DN can be any valid distinguished name (DN), and this DN does not have to be present in the virtual directory tree. The default root DN is **`uid=admin,ou=system`**, and the default password is **`secret`**.

3.4.2. Editing the Admin User in the Configuration File

Like the host system properties, the root user is configured in the **`/opt/vd-server-2.0/conf/server.xml`** file.

The root DN can be any valid distinguished name (DN), and this DN does not have to be present in the virtual directory tree. The default root DN is **`uid=admin,ou=system`**, and the default password is **`secret`**.

For example:

```
<server>

  <root>
    <root-dn>uid=admin,ou=system</root-dn>
    <root-password>secret</root-password>
  </root>

</server>
```

3.5. Configuring and Viewing Logs

There are three logs kept by Penrose Server:

- Access logs
- Error logs
- Debug logs

These logs are stored in the **/opt/vd-server-2.0/logs** directory. Since these files apply to access to Penrose Server and Penrose Studio, one set of log files is kept, regardless of the number of partitions.

The log configuration file is located in the configuration directory, **/opt/vd-server-2.0/conf**.

3.5.1. Configuring Log Settings

The logging parameters for all logs are configured in the **log4j.xml** file in the **/opt/vd-server-2.0/conf** directory. Several parameters can be set for the logs individually, including the file location, maximum file size, and the number of backup files to keep. The parameters are listed in *Table 3.2, "log4j Parameters"*.

NOTE

All three log files — access, error, and debug logs — have the same configuration parameters available to them.

Parameter	Description	Example
File	Gives the path, relative to the **/opt/vd-server-2.0**, of the log file.	param name="File" value="logs/access.log"
MaxFileSize	Sets the threshold for the log file before beginning a new log file.	param name="MaxFileSize" value="10MB"
MaxBackupIndex	Sets the number of old log files to keep as archives.	param name="MaxBackupIndex" value="100"
ConversionPattern	Sets the pattern to use in the log entries, both for the timestamp and the log message. This parameter is set within **<layout>** tags.	param name="ConversionPattern" value="[%d{MM/dd/yyyy HH:mm:ss}] %m%n"

Table 3.2. log4j Parameters

For example:

```
  <appender name="access" class="org.apache.log4j.RollingFileAppender">
    <param name="File" value="logs/access.log"/>
    <param name="MaxFileSize" value="10MB"/>
    <param name="MaxBackupIndex" value="100"/>
    <layout class="org.apache.log4j.PatternLayout">
      <param name="ConversionPattern" value="[%d{MM/dd/yyyy HH:mm:ss}]
%m%n"/>
    </layout>
  </appender>
```

3.5.2. Access Logs

The access logs record every transation that is performed through Penrose Server on the virtual directory or its sources. The access log entry has the following general format:

```
[date timestamp] session_ID type_of_operation operation_details
```

The *session_ID* gives the number for the connection thread.

There are several different types of transaction logged in the access log:

- **`CONNECT`** shows the initial connection. The operation details include the IP address of the client, the IP address of the Penrose Server host, and the protocol used to connect to the server (either LDAP or JMX).

- **`BIND`** shows that a user has authenticated to the server when the session was opened. This contains the DN of the user.

  After the initial log entry for the **`BIND`** operation, the result is shown in the **`result=`** tag and then the DN which connected (**`authDN`**).

- **`SEARCH`** shows the search details for a search of the virtual directory. This contains the base DN, scope, and filter for the search, as well as the entry attributes returned.

  After the initial log entry for the **`SEARCH`** operation, the result is shown in the **`result=`** tag and, for a successful operation, the number of entries returned.

- **`DISCONNECT`** shows when the connection is closed.

For example:

```
[10/29/2008 12:47:31] session="5" - CONNECT from="192.168.123.122"
to="192.168.123.121" protocol="LDAP"
[10/29/2008 12:47:31] session="5" - BIND dn="uid=admin,ou=system"
```

```
[10/29/2008 12:47:31] session="5" - BIND result="Success" au\
thDn="uid=admin,ou=system"
[10/29/2008 12:47:33] session="5" - SEARCH base="dc=example,dc=com"
scope="one level" filter="(objectClass=*)" attrs="*,+"
[10/29/2008 12:47:45] session="5" - SEARCH result="Success" entries="2"
[10/29/2008 14:58:59] session="5" - DISCONNECT
```

Example 3.1. Access Logs

3.5.3. Error Logs

The error logs have a more simple format than the access logs, showing the time of the error and the error message.

```
[date timestamp] error_message
```

Many error messages contain Java exceptions, which can indicate which areas of the Penrose Virtual Directory service have a problem. For example:

```
[10/08/2008 19:19:40] An error occurred while attempting to calculate a
SHA-1 digest of file /
usr/local/penrose-server-2.0RC2/services/OpenDS/config/config.ldif:  SHA-
1 MessageDigest not available (GetInstance.java:158 Security.java:691
MessageDigest.java:145 ConfigFileHandler.java:773 ConfigFileHand\
ler.java:274 DirectoryServer.java:1190 DirectoryServer.java:1139 OpenD\
SLDAPService.java:66 Service.java:54 ServiceManager.java:113 PenroseServ\
er.java:119 PenroseServer.java:307)
org.opends.server.types.InitializationException: An error occurred while
attempting to calculate a SHA-1 digest of file /
usr/local/penrose-server-2.0RC2/services/OpenDS/config/config.ldif:  SHA-
1 MessageDigest not available (GetInstance.java:158 Security.java:691
MessageDigest.java:145 ConfigFileHandler.java:773 ConfigFileHand\
ler.java:274 DirectoryServer.java:1190 DirectoryServer.java:1139 OpenD\
SLDAPService.java:66 Service.java:54 ServiceManager.java:113 PenroseServ\
er.java:119 PenroseServer.java:307)
        at
(Co
nfi
gFi
org.opends.server.extensions.ConfigFileHandler.initializeConfigHandler\
Handler.java:278)
        at
org.opends.server.core.DirectoryServer.initializeConfiguration(DirectoryS
erver.java:1190)
        at
org.opends.server.core.DirectoryServer.initializeConfiguration(DirectoryS
erver.java:1139)
        at
(OpenDSLDAPService.java
org.safehaus.penrose.opends.OpenDSLDAPService.init:66)
        at org.safehaus.penrose.service.Service.init(Service.java:54)
```

```
        at
org.safehaus.penrose.service.ServiceManager.startService(ServiceManager.j
ava:113)
        at
org.safehaus.penrose.server.PenroseServer.start(PenroseServer.java:119)
        at
org.safehaus.penrose.server.PenroseServer.main(PenroseServer.java:307)
```

Example 3.2. Error Logs

3.5.4. Debug Logs

The debug log contains all of the information recorded in the access and error logs, including all of the error message output and bind attempts, and additional information, such as notices on when the server starts and stops.

The format of the debug log is slightly different than the access or error logs. The log entry references only the Java file and line number which is accessed, then a message. The message can be relatively short or very long, depending on the response from the server.

```
Java_class [source_line_number] message.
```

For example:

```
PenroseServer        [ 147] Penrose Server has been shutdown.
PenroseServer        [ 126] Penrose Server is ready.
```

Example 3.3. Debug Logs

3.6. Configuring Penrose Virtual Directory for SSL

SSL can be configured two ways for Penrose Server. Front-end SSL means that Penrose Server uses SSL-secured connections to communicate with its OpenDS client services. Back-end SSL means that Penrose Server uses SSL-secured connections to communicate with LDAP and JDBC data sources.

3.6.1. Configuring SSL for Backend Sources

NOTE

The source must be configured to run over SSL as well for Penrose Virtual Directory to connect to it using SSL.

1. Configure the source to run in SSL.

2. Configure the connection for the source, as described in *Chapter 6, Configuring Connections*, with the protocol and ports set for SSL.

3. When the source is configured for SSL, an SSL client certificate is issued to the source. This certificate has to be imported into Penrose Server's certificate database for Penrose Server to connect to the source over SSL.

   a. Export the source's SSL client certificate to a PEM or PKCS #12 file.

   b. Open the Penrose Server configuration directory.

```
cd /opt/vd-server-2.0/conf
```

   c. Import the certificate into the certificate database. This can be done using tools like **`certutil`** or **`keytool`**, depending on how the database was created. For example:

```
keytool -import -trustcacerts -alias mydb -file certificate.pem -keystore
conf/cacerts
Enter keystore password:  secret
Owner: CN=source1.example.com, OU=Engineering, L=Raleigh, ST=North Caro\
lina, C=US
Issuer: CN=mydb, OU=Engineering, L=Raleigh, ST=North Carolina, C=US
Serial number: 499ee484
Valid from: Fri Feb 20 11:12:36 CST 2009 until: Thu May 21 12:12:36 CDT
2009
Certificate fingerprints:
         MD5:  37:45:60:73:A8:DD:BB:1D:C0:7A:F3:82:49:99:B9:04
         SHA1:
CE:2C:21:D4:91:D1:7E:94:92:29:24:33:EE:59:06:DD:F0:B5:B8:EA
Trust this certificate? [no]:  yes
Certificate was added to keystore
```

   d. Edit the **`server.xml`** file. Add two parameters, **`javax.net.ssl.trustStore`** and **`javax.net.ssl.trustStorePassword`**, to give the location and password to access the certificate database.

```
<system-property>
  <property-name>javax.net.ssl.trustStore</property-name>
  <property-value>conf/cacerts</property-value>
</system-property>

<system-property>
  <property-name>javax.net.ssl.trustStorePassword</property-name>
  <property-value>secret</property-value>
</system-property>
```

To verify that SSL is properly configured for the backend, open Penrose Studio and open the source. If the source opens and can be browsed, then SSL is working.

3.6.2. Configuring SSL for Frontend Services

1. Request or generate a certificate for Penrose Server to use.

   Penrose Server supports a JKS or PKCS #12 certificate. Certificates can be request from a third-party authority or can be self-generated using a tool such as **`keytool`** or **`certutil`** and **`pk12util`**.

   However the certificate is generated, make sure to use the hostname of the Penrose Server host machine as the common name (**`CN`**) part of the certificate.

   For example, this generates a JKS certificate:

```
keytool -genkey -keyalg RSA -alias server-cert -keystore conf/keystore
Enter keystore password:
Re-enter new password:
What is your first and last name?
  [Unknown]:  server.example.com
What is the name of your organizational unit?
  [Unknown]:  Engineering
What is the name of your organization?
  [Unknown]:  Example Corp.
What is the name of your City or Locality?
  [Unknown]:  Raleigh
What is the name of your State or Province?
  [Unknown]:  North Carolina
What is the two-letter country code for this unit?
  [Unknown]:  US
Is CN=server.example.com, OU=Engineering, O=Example Corp., L=Raleigh,
ST=North Carolina, C=US correct?
  [no]:  yes

Enter key password for <server-cert>
        (RETURN if same as keystore password):
```

   For more information on **`keytool`**, see *http://1java.sun.com/1j2se/11.3/1docs/1tooldocs/1win32/1keytool.html*.

2. Edit the front-end service to support SSL. The only service which currently supports SSL for Penrose Server is the OpenDS service.

   a. Open the OpenDS directory.

```
/opt/vd-server-2.0/services/OpenDS/config/
```

   b. Copy the keystore from the Penrose Server **`conf`** directory to the OpenDS service configuration directory.

```
cp /opt/vd-server-2.0/conf/keystore .
```

c. In the OpenDS service configuration directory, there is a file called **keystore.pin**. Add the password for the keystore database to this file. For example:

```
echo secret > keystore.pin
```

d. The OpenDS **config.ldif** file. This contains entries which enable and configure SSL for the service.

```
vim config.ldif
```

e. The OpenDS **config.ldif** file has an entry for a JKS key manager and a PKCS #12 key manager. Edit the appropriate entry so that it is enabled, depending on the type of certificate created. For example:

```
dn: cn=JKS,cn=Key Manager Providers,cn=config
objectClass: ds-cfg-file-based-key-manager-provider
objectClass: ds-cfg-key-manager-provider
objectClass: top
ds-cfg-key-store-type: JKS
cn: JKS
ds-cfg-key-store-file: config/keystore
ds-cfg-key-store-pin-file: config/keystore.pin
ds-cfg-enabled: true
```

If necessary, edit the other parameters in the entry to make sure it matches the certificate information. Make sure the directories given for the keystore and PIN files are relative to the *OpenDS* service directory, not the Penrose Server home directory.

f. To accept client certificates, make sure the blind trust entry is enabled.

```
dn: cn=Blind Trust,cn=Trust Manager Providers,cn=config
objectClass: ds-cfg-blind-trust-manager-provider
objectClass: top
objectClass: ds-cfg-trust-manager-provider
cn: Blind Trust
ds-cfg-enabled: true
```

g. Make sure that LDAPS connections are allowed. The entry should be enabled, SSL should be allowed, and Start TLS should be denied. Additionally, the certificate nickname in the entry should match the nickname of the certificate generated for Penrose Server.

```
dn: cn=LDAPS Connection Handler,cn=Connection Handlers,cn=config
.. snip...
ds-cfg-allow-start-tls: false
ds-cfg-listen-address: 0.0.0.0
ds-cfg-use-tcp-keep-alive: true
ds-cfg-ssl-cert-nickname: server-cert
.. snip...
ds-cfg-enabled: true
.. snip...
cn: LDAPS Connection Handler
.. snip...
ds-cfg-listen-port: 636
.. snip...
ds-cfg-use-ssl: true
```

3. Restart Penrose Server.

```
service vd-server restart
```

4. The Penrose Server certificate has to be available in the certificate database of any LDAP tools used to manage Penrose Server. For example, for OpenLDAP tools:

   a. For a self-signed certificate, export a PEM file.

```
keytool -export -alias server-cert -keystore conf/keystore -file pen\
rose.pem -rfc
```

   b. Import the certificate into the certificate database, such as the default **`cacerts`**.

```
cp penrose.pem /etc/openldap/cacerts

c_rehash /etc/openldap/cacerts
```

5. To use OpenLDAP's LDAP client tools with Penrose Server, edit the OpenLDAP configuration file, **`ldap.conf`**, to recognize Penrose Server's certificate.

```
vim /etc/openldap/ldap.conf

TLS_CACERTDIR /etc/openldap/cacerts
```

To test the frontend SSL configuration for Penrose Server, run an **`ldapsearch`** using the secure port, 636, and the secure protocol, **`ldaps`**.

```
ldapsearch -H ldaps://localhost:636 -x -b "" -s base
```

If SSL is properly configured the search returns the root DSE for Penrose Server.

3.7. Running Penrose Server Outside a Firewall

Penrose Server can run outside a firewall for additional network security, but, in that case, the firewall must be configured to allow access for the different client ports required for Penrose Server.

Port Number	Client Description
1099	An RMI port used by Penrose Studio.
40888	An RMI transport port used by Penrose Studio.
8122	A web port used by web browsers.
10389	The LDAP standard port used by directory services.
10636	The LDAP secure (SSL) port used by directory services.

NOTE

The ports used by these services can be changed; see *Chapter 12, Using Services with Penrose Virtual Directory*. Before changing the services ports or firewall settings, make sure that the client itself is configured to use a different port number.

Using Penrose Studio

Penrose Studio is the graphical interface to access, modify, and manage Penrose Server and its entries. This is a simple, elegant interface, making it much easier to configure complex mappings, virtual directory hierarchies, and identity federation, synchronize NIS and LDAP environments, resolve UID conflicts, and migrate data between servers.

This chapter describes how to install and start Penrose Studio and gives an overview of its menus and different areas.

4.1. Installing Penrose Studio

IMPORTANT

Penrose Studio requires JDK 1.5.0 or higher be installed. See *Section 2.2, "Required Software"* for instructions.

4.1.1. Supported Platforms

Penrose Studio is supported on three platforms:

- Red Hat Enterprise Linux 4.7, 32-bit and 64-bit
- Red Hat Enterprise Linux 5.2, 32-bit and 64-bit, including virtual guests
- Microsoft Windows XP and Server 2003

4.1.2. Installing Penrose Studio on Red Hat Enterprise Linux

1. Download the RPM from Red Hat Network.
2. Install the packages.

```
rpm -ivh vd-studio-2.0-build#.el5.i386.rpm
```

3. Select the installation directory to use; by default, this is **`/opt/vd-studio-2.0`**.

4.1.3. Installing Penrose Studio on Windows

1. Download the Windows **`.msi`** package from Red Hat Network.

2. Double-click the **`.msi`** file to launch the installer.

3. Select the installation directory to use; by default, this is **`C:\Program Files\Identyx Corporation\VD Studio 2.0`**.

4.2. Starting Penrose Studio

IMPORTANT

If you have problems connecting to Penrose Server from Penrose Studio, check the Penrose Server configuration and make sure that the *`java.rmi.server.hostname`* in the **`/opt/vd-server-2.0/conf/server.xml`** files has been changed to the specific hostname or IP address of the Penrose Server host machine, rather than using the default value of **`localhost`**.

4.2.1. Starting Penrose Studio on Red Hat Enterprise Linux

To start Penrose Studio on Red Hat Enterprise Linux:

1. Open the Penrose Studio home directory.

```
cd /opt/vd-studio-2.0
```

2. Run the **`studio`** command. For example:

```
./vd-studio
```

3. Expand the server listed in the left to connect to a Penrose Server instance.

4.2.2. Starting Penrose Studio on Windows

To launch Penrose Studio on Windows:

1. Open the **Start** menu.

2. Select **Identyx**, then the **Studio** menu item.

NOTE

The executable file for Penrose Studio is named **`studio.exe`** in the **`C:\Program Files\Identyx Corporation\VD Studio 2.0`** directory.

3. Expand the server listed in the left to connect to a Penrose Server instance.

4.3. Looking at Penrose Studio

Penrose Studio is very simple to navigate. There are two main sections. The navigation tree on the left organizes all of the partitions, directories, mappings, and other configurations for a Penrose Studio instanc. The main window displays and edits entries and contains the integrated LDAP directory browser.

Figure 4.1, "Penrose Studio Window" shows the major sections of the Penrose Studio interface.

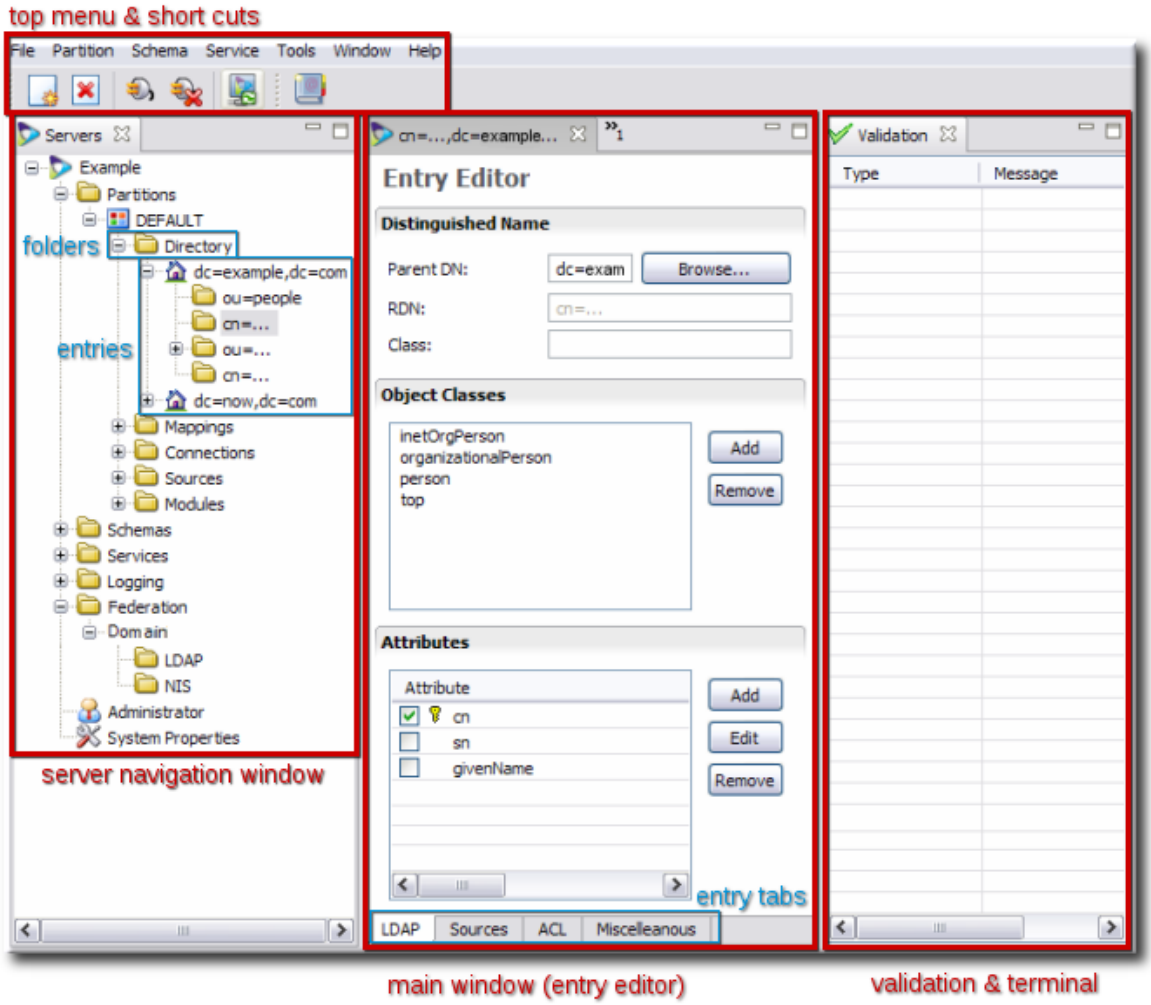


Figure 4.1. Penrose Studio Window

The **File** manages servers. All if the actions in the file menu are mirrored in the icons below it, managing the Penrose Server instance by stopping and starting it, synchronizing the Penrose Studio changes, and adding and deleting server entries.

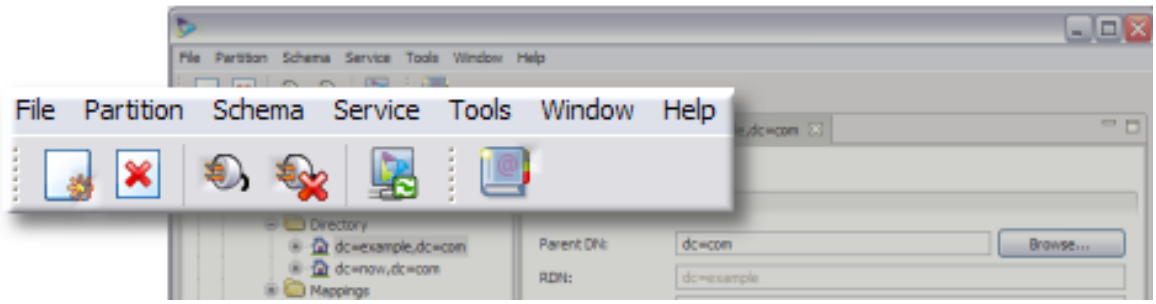


Figure 4.2. Top Menu

Three other menus in the top menu, the **Partition**, **Schema**, and **Service** menus, have the same operations that are available by right-clicking on the corresponding folders in the server navigation areas, such as exprting a partition, importing a schema, or configuring a new service.

Both the **Tools** menu and the last icon below the top menu open Penrose Virtual Directory's LDAP browser.

The last menu, **Window**, controls which of the three sections in the lower window are visible, meaning the server navigation area, main entry window, and validation window.

The server window, shown in *Figure 4.3, "Server and Entry Navigation"*, is a hierarchical organization of the server, its partitions, mappings, sources, connections, and repositories along with its backend configuration settings, schema, logging, plug-ins, and services.

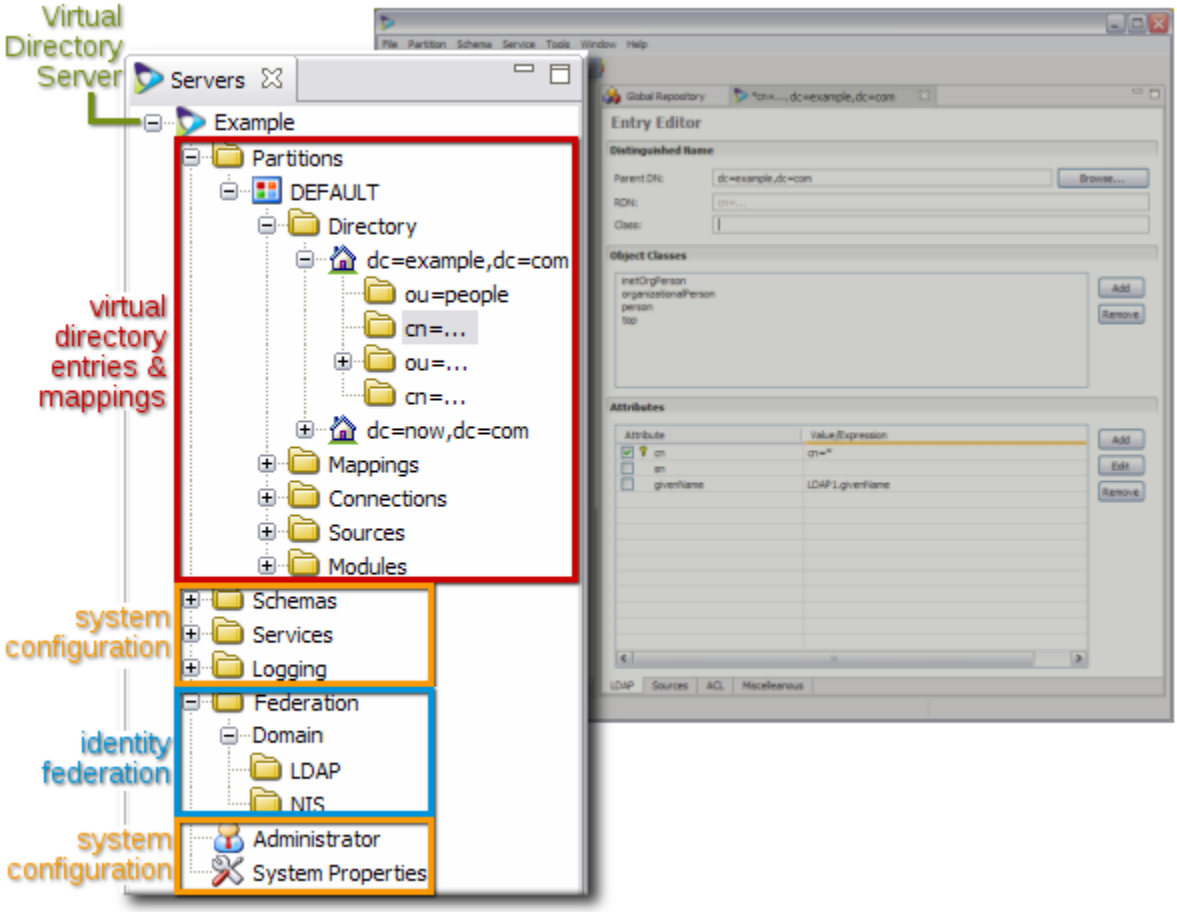


Figure 4.3. Server and Entry Navigation

Double-clicking or opening any entry in the server window opens the editor for that entry in the main window, as in *Figure 4.4, "Main Window for Editing Entries"*.

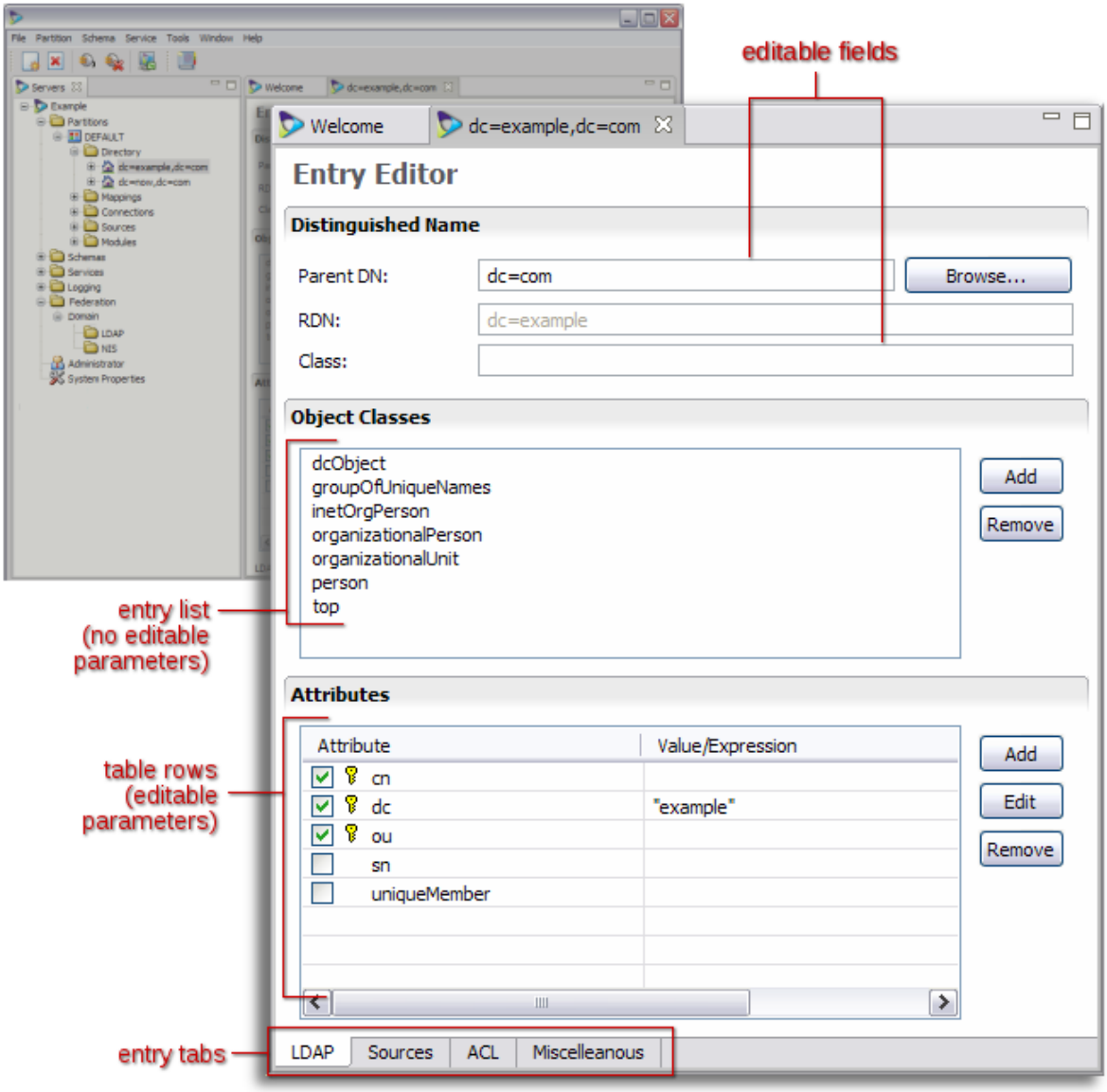


Figure 4.4. Main Window for Editing Entries

The configuration of the entry can be changed in several areas:

- Changing the descriptions in fields and adding or removing Java classes which define functionality
- Adding or removing attributes
- Changing the parameters set for attributes; entries in table rows can be double-clicked to allow additional configuration options

Depending on the complexity or type of entry, there may be tabs at the bottom of the main window which open additional editing windows for the entry or browsers to display the virtual directory or entry hierarchy.

All of the windows and the entries within the windows are tabbed, to keep the overall navigation clean and easy to view.

Every window, meaning the server, entry editor, and validation windows, is controlled independently. To minimize or maximize the window in the Penrose Studio space, click the small line and box icons in the right corner of the entry. To close the window, click the X icon by the window name.

Figure 4.5. Window Controls

NOTE
Changes to an entry are saved when the window is closed.

4.4. Editing, Copying, and Deleting Entries

Almost every entry for the virtual directory configuration — such as mappings, subtrees, repoositories, sources, and connections — has several options available to open, edit, delete, copy, or paste the entry. This is illustrated for a mapping in *Figure 4.6, "Entry Menu Options"*. Some kinds of entries, including partitions and schema, can be imported and exported.

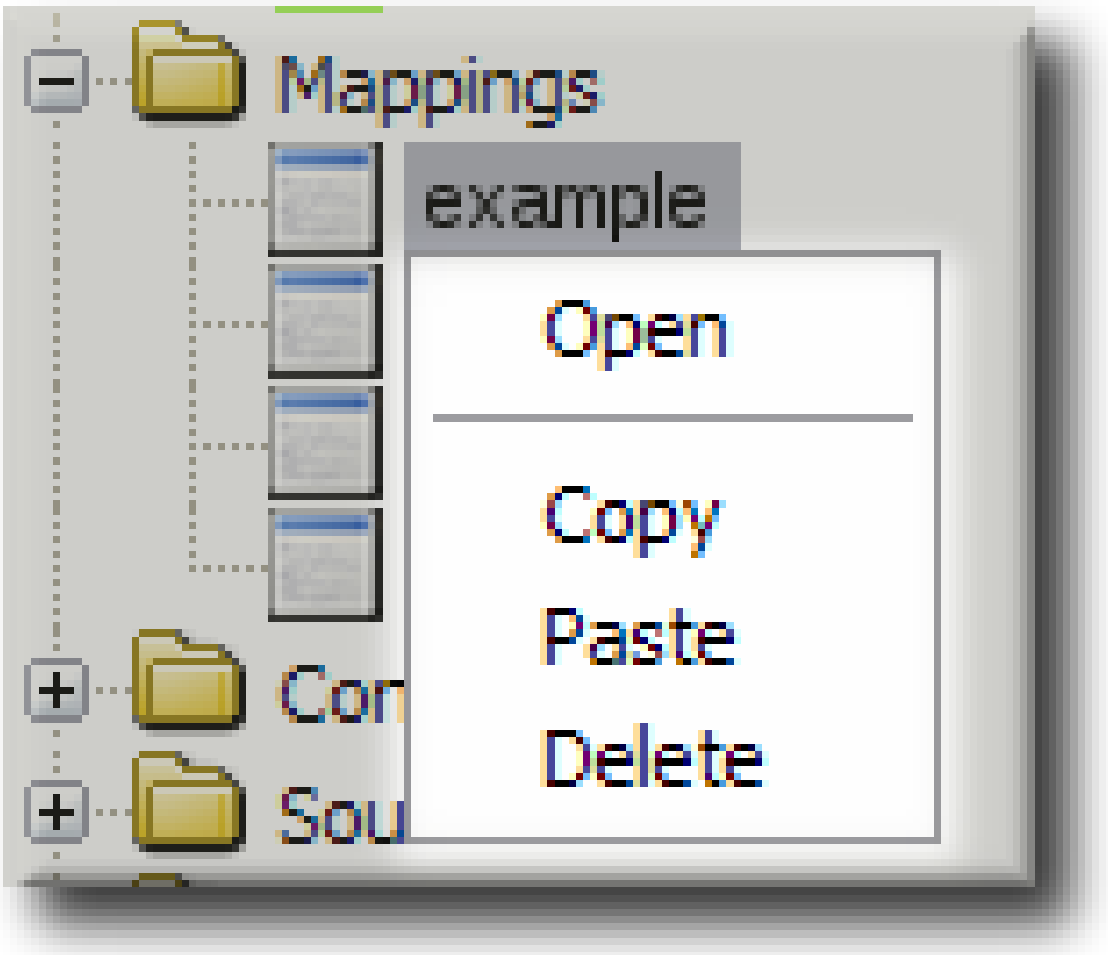


Figure 4.6. Entry Menu Options

4.4.1. Viewing and Editing Entries

The editor for any entry is also the way to view the entry. To open, view, or edit an entry:

1. Navigate to the entry in the server window.

2. Right-click the entry to view or edit.

3. Select **Open** from the menu.

4. The **Entry Editor** opens in the main window.

4.4.2. Deleting Entries

To delete any entry (and any sub-entries):

1. Navigate to the entry in the server window.
2. Right-click the entry to delete.
3. Select **Delete** from the menu.
4. Confirm the deletion when prompted.

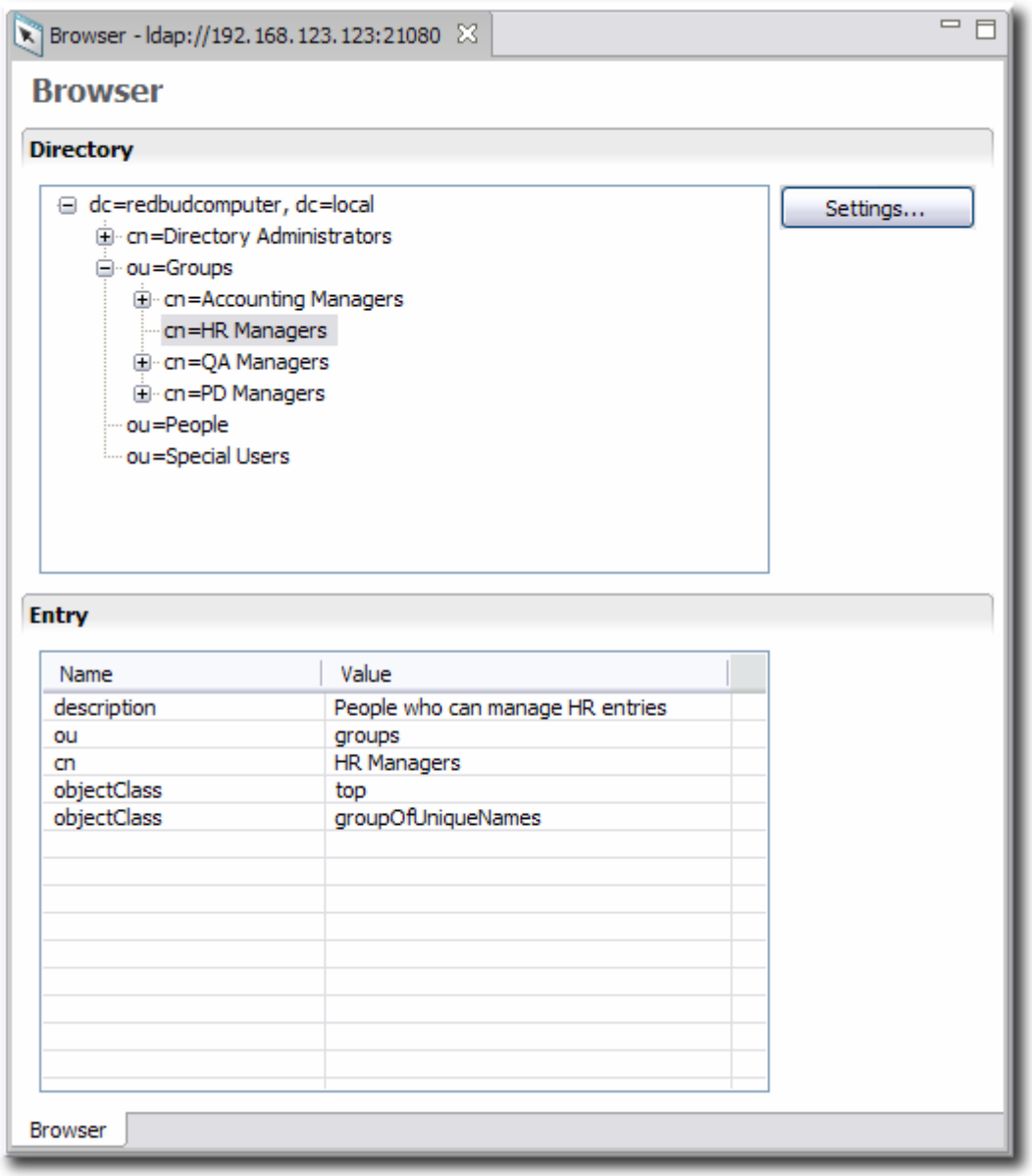
4.4.3. Copying Entries to Another Penrose Server Instance

The copy and import options in Penrose Studio make it very simple to duplicate the exact configuration of one Penrose Server instance to another Penrose Server without having to edit multiple files.

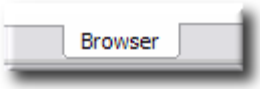
1. Navigate to the entry in the server window.
2. Right-click the entry to copy.
3. Select **Copy** from the menu.
4. Open the second Penrose Server entry and navigate to the area to paste the duplicate entry.
5. Right-click the folder or entry to paste the copied entry.
6. Select **Paste** from the menu.
7. The pasted entry is listed with the second Penrose Server entries.

4.5. Browsing the LDAP Directory

Penrose Studio also provides a built-in LDAP directory browser, which can be used to view the virtual directory or any of the sources or connections for the virtual directory.



Almost every virtual directory configuration entry has a browser tab at the bottom of its entry window, to view the entries within the source or connection.



The browser can also be opened in the top menu or by clicking the browser icon in the top menu.

When the ldap browser is opened through the menu options, the connection information has to be entered in a dialog box, then the browser opens. When choosing a virtual directory entry to browse, the connection information is already given.

Managing Partitions

In Penrose Virtual Directory, a virtual directory is defined through the relationships of different server entities: servers (connections), applications (sources), individual entries (identities), and links between entities (mappings). All of these relationships are contained in the partition entry.

This chapter describes how to create and manage partition entries.

5.1. About Partitions

The complete virtual directory set up is defined in a container entry called a *partition*. The partition is a Java object which controls the operations and lives of all of the other components of the virtual directory configuration, including sources, connections, and mappings. The partition contains all of the Java classes and common files used by these virtual directory components. In this way, a partition is functionally similar to an application server like Tomcat.

A default partition is created with every Penrose Server instance, and this is usually sufficent to define the complete Penrose Virtual Directory deployment. It is also possible, depending on the deployment requirements, to have multiple partitions within a single Penrose Server instance.

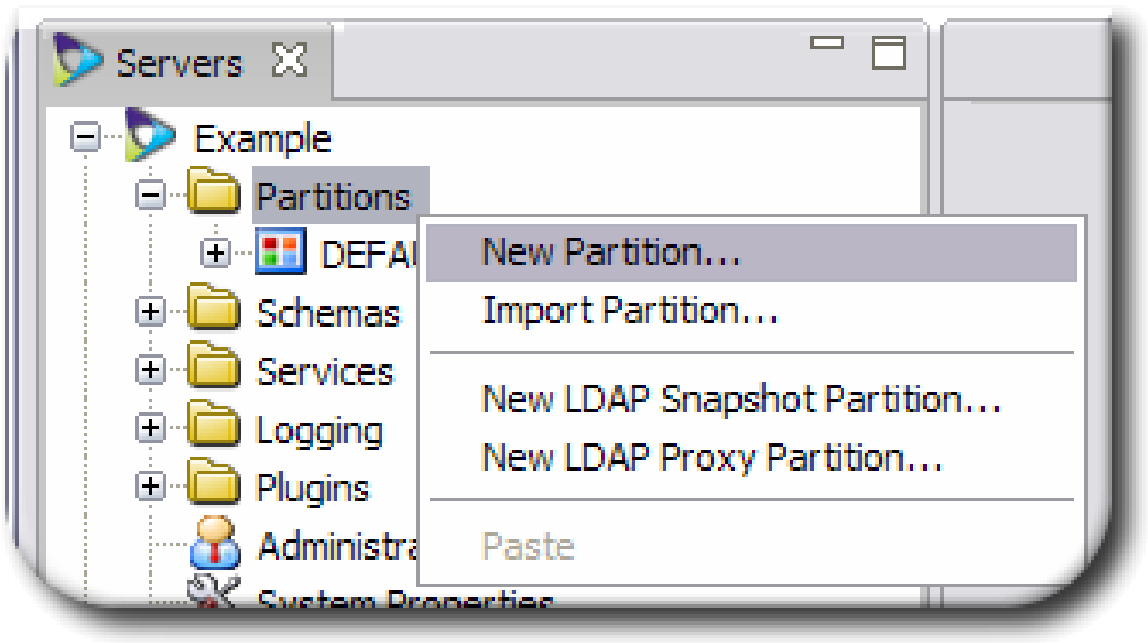
It can also be necessary to have multiple Penrose Server instances with the same virtual directory configuration. Penrose Virtual Directory partitions can be easily exported from one Penrose Server instance and imported on another, making it easy to have a consistent virtual directory on different servers.

5.2. Adding Partitions

By default, every new server already has a partition named **DEFAULT**. Server entries can have an unlimited number of partitions. To create a new partition:

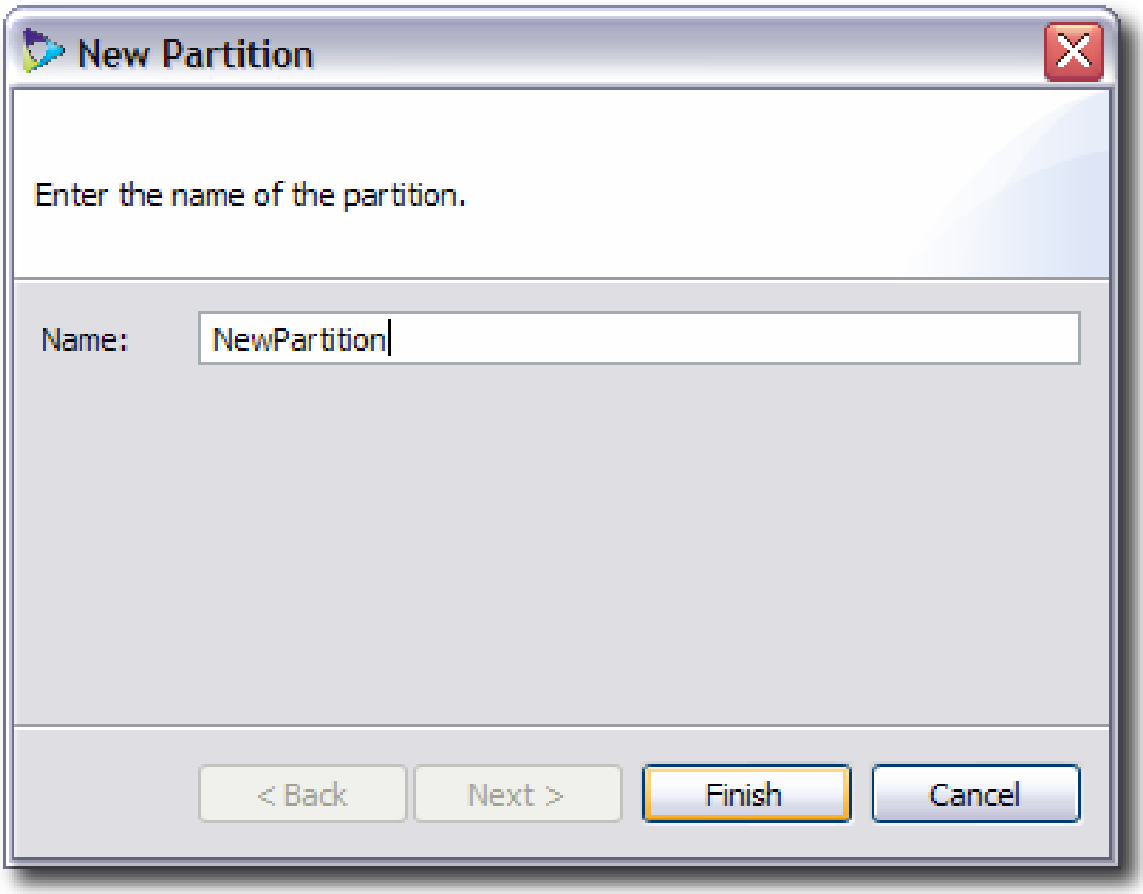
1. In Penrose Server, open the server entry.

2. In the top menu, click the **Partitions** menu item, and select the **New Partition...** option.

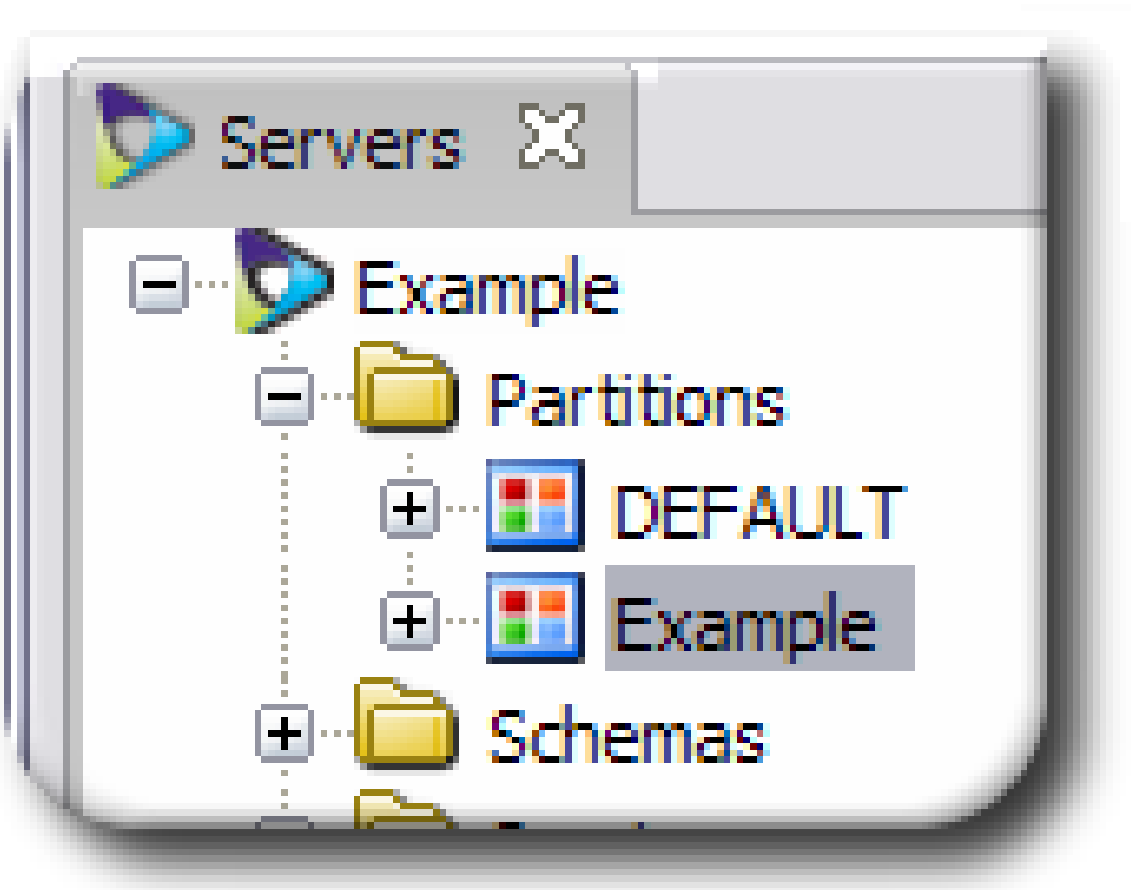


Alternatively, right-click the **Partitions** folder, and select the **New Partition...** option.

3. Fill in a name for the new partition, and click **Finish**.



The new partition, with all of its subfolders, is listed in the **Partitions** section.



5.3. Exporting and Importing Partitions

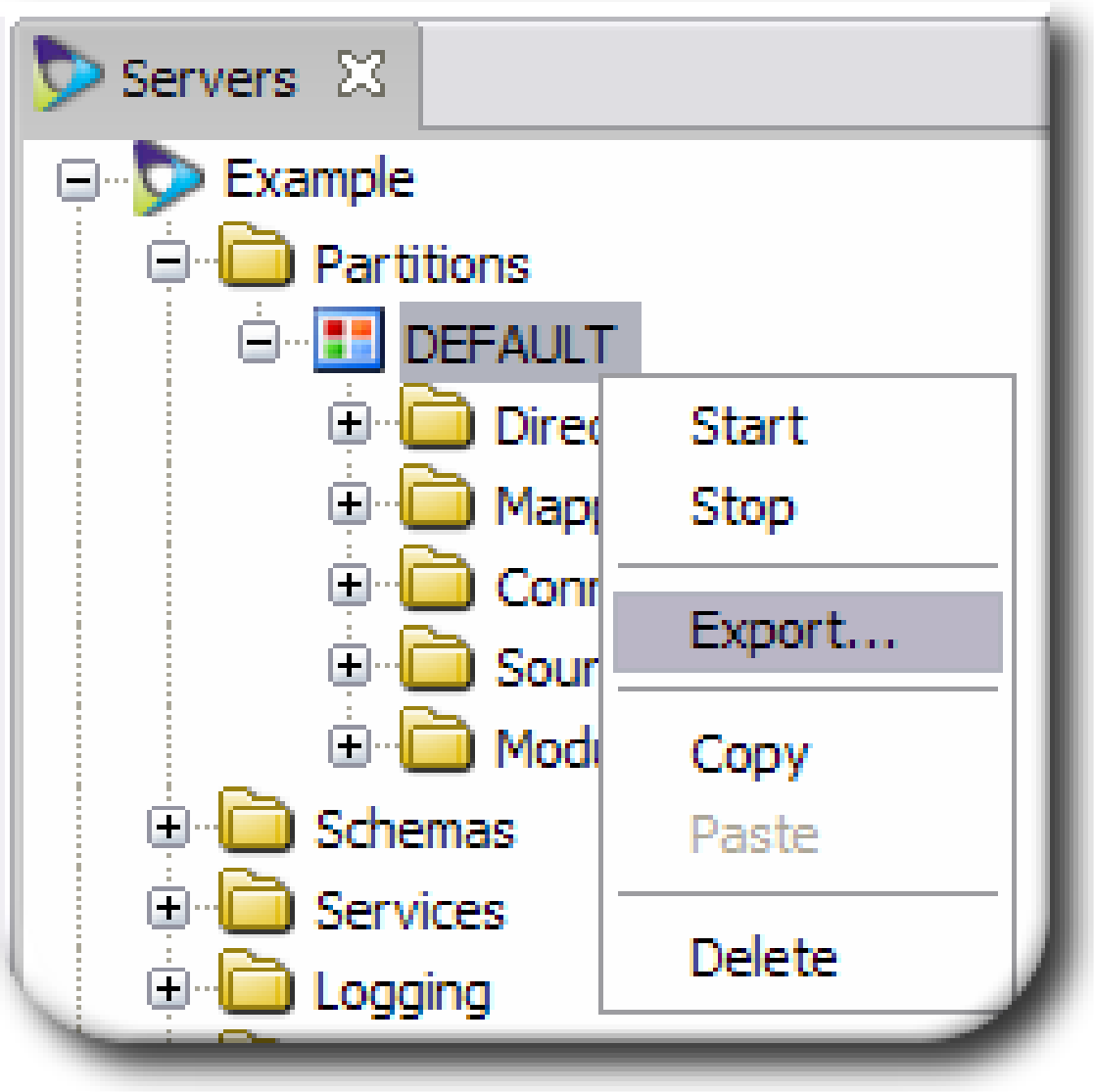
The partition configuration can be exported from one Penrose Server and imported into another Penrose Server. This makes it much easier to have the same virtual directory mirrored on multiple servers.

5.3.1. Exporting Partitions in Penrose Studio

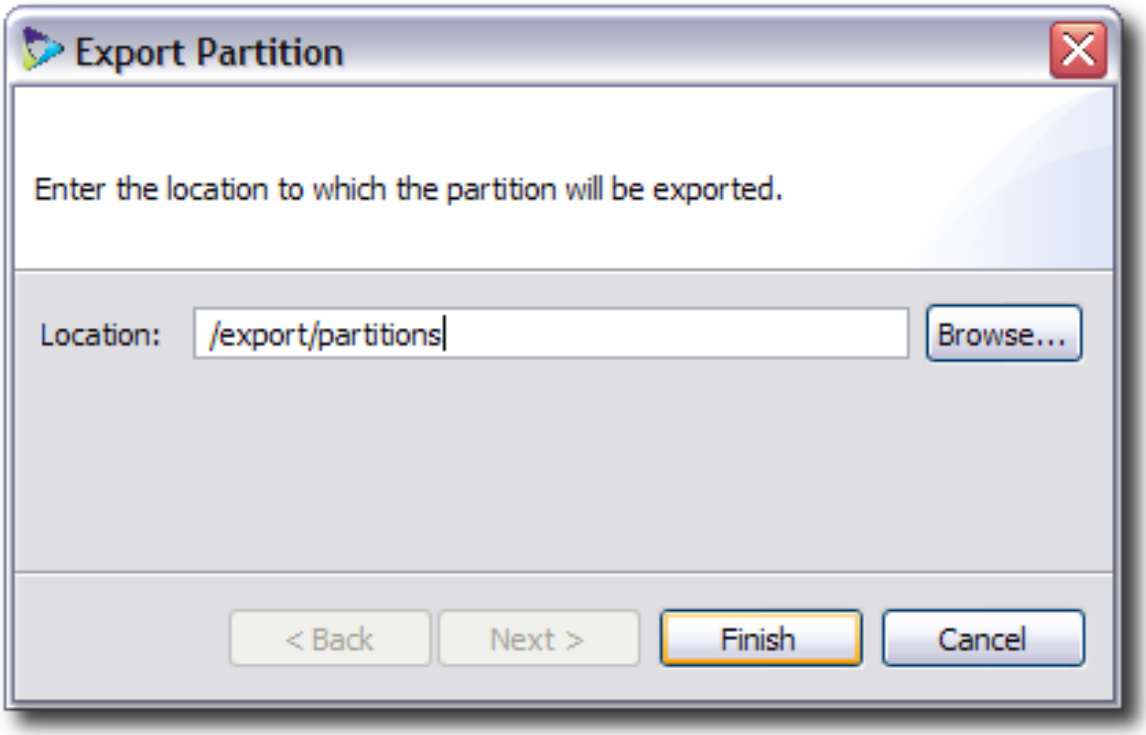
1. Open the server entry in Penrose Studio, and expand the **Partitions** folder.

2. Right-click the partition to export under the **Partitions** folder.

3. Select **Export...** from the drop-down menu.



4. Fill in the path to the directory to which to save the partition configuration file, and click the **Finish** button.



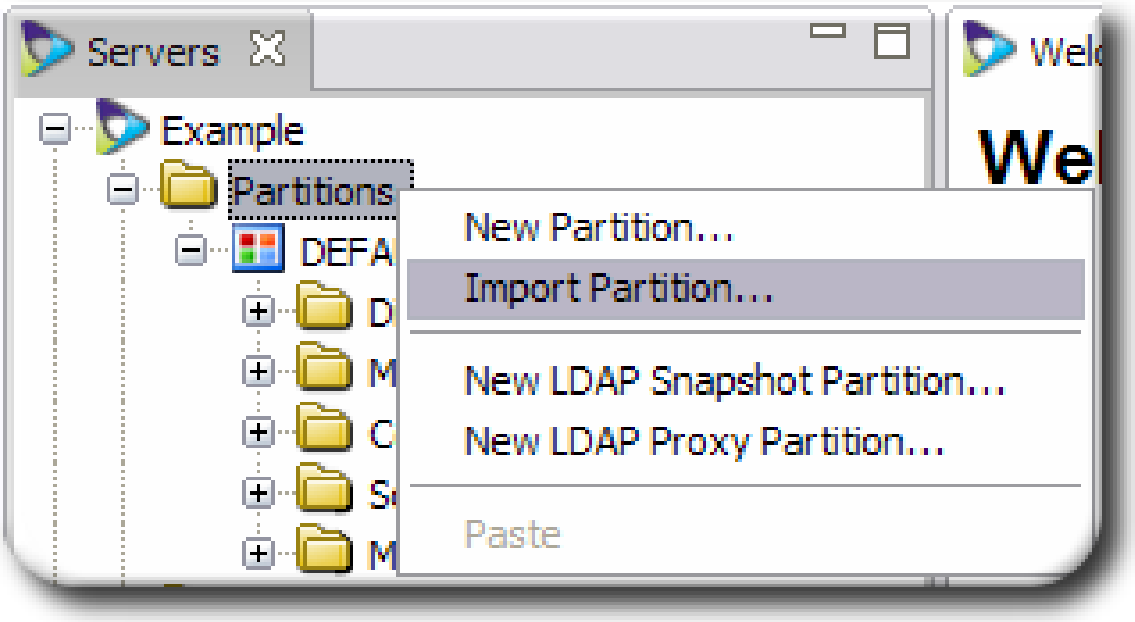
5.3.2. Importing Partitions in Penrose Studio

NOTE
The import partitions directory must be local to the machine where Penrose Studio is being run.

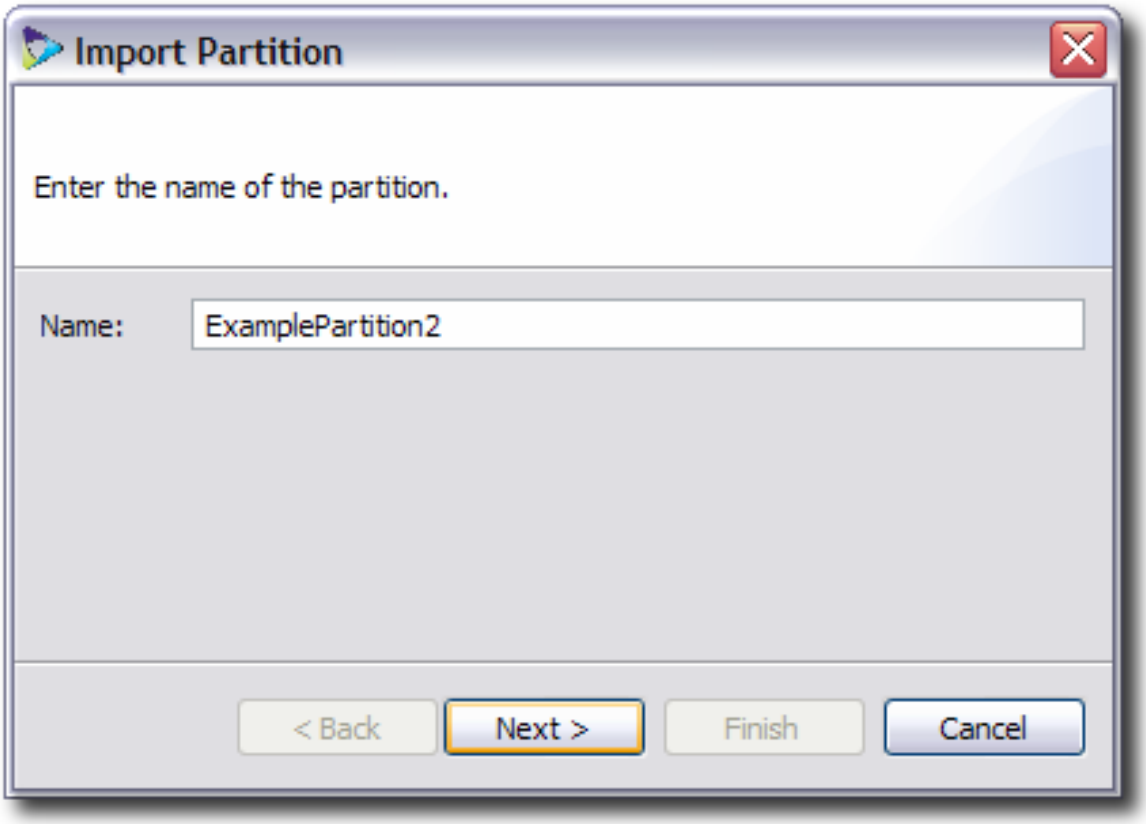
1. Open the server entry in Penrose Studio, and expand the **Partitions** folder.

2. Right-click the **Partitions** folder.

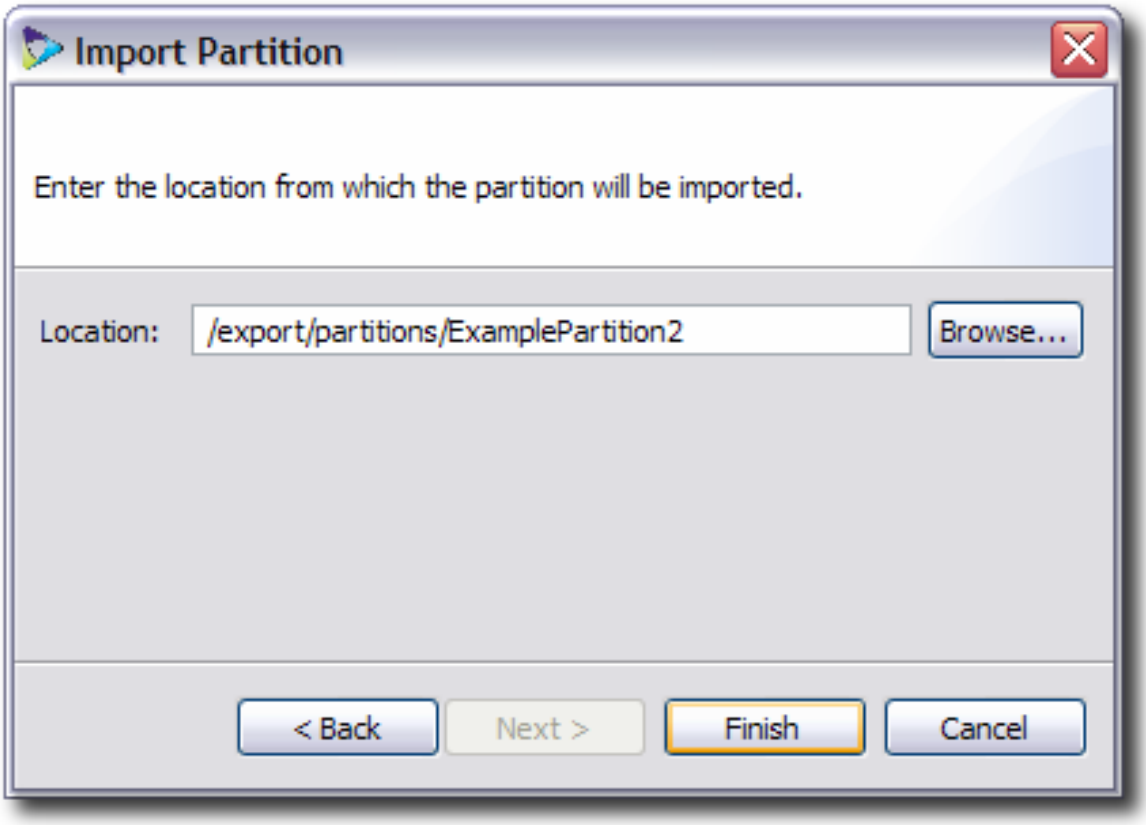
3. Select **Import Partition...** from the drop-down menu.



4. Fill in the name for the new partition. The name can be different than the original name. Click **Next**.



5. Fill in the path to the directory to the partition directory, and click **Finish**.



The imported partition is listed beneath the **Partitions** folder.

5.3.3. Exporting and Importing Partitions in the Command Line

Penrose Server has a tool, **`partition.sh`**, which can manage partitions, and this tool is used to import and export partitions.

1. On the first server, export the partition. For example:

```
/opt/vd-server-2.0/bin/partition.sh export partition partition_name /
path/to/exported/partition
```

2. The partition must be imported from a directory which is local to the Penrose Server instance, so the exported partition must be available on a fileserver or the local machine. For example, copy the exported partition to the target Penrose Server machine:

```
scp -r ExamplePartition2/
root@server2.example.com:/opt/vd-server-2.0/partitions/ExamplePartition2/
```

3. Import the partition into the target Penrose Server.

```
/opt/vd-server-2.0/bin/partition.sh import partition partition_name /
local/path/to/exported/partition
```

5.4. Starting and Stopping Partitions

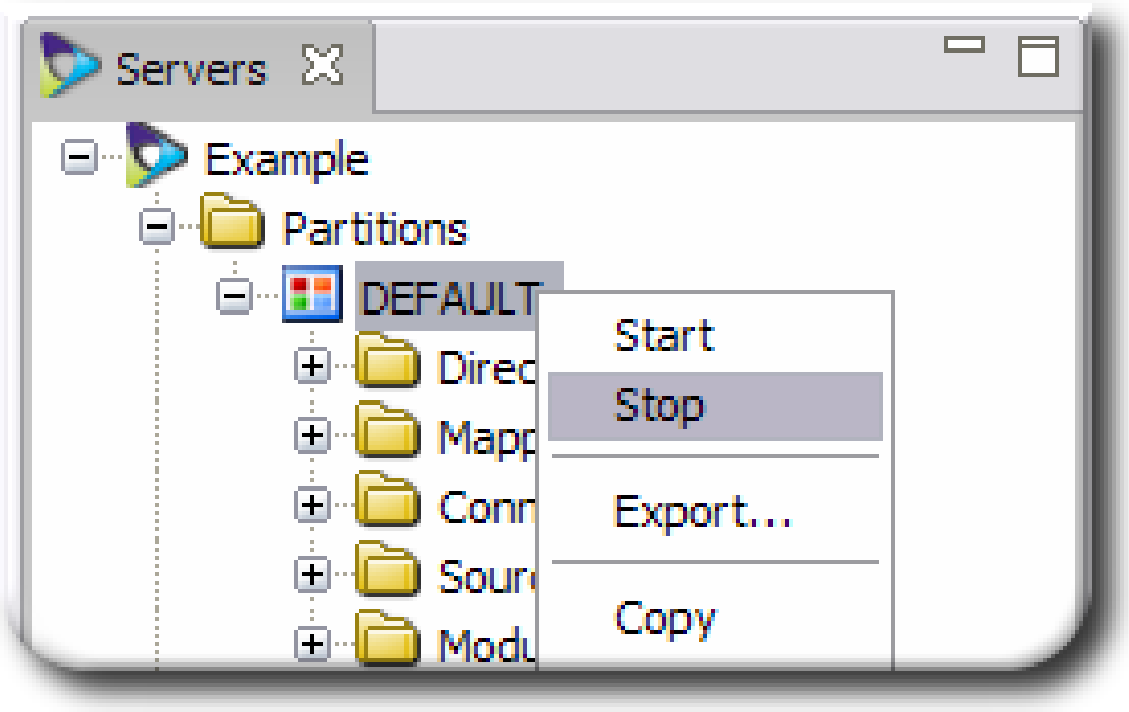
The partition is a Java MBean object which starts and runs automatically with Penrose Server. Each source, connection, directory, and mapping is also an MBean object, controlled by the partition object. The partition can be started and stopped, which also starts or stops all of the sources, connections, directories, and mappings configured under it.

To start or stop a partition from Penrose Studio:

1. Open the server entry in Penrose Studio, and expand the **Partitions** folder.

2. Right-click the partition to start or stop under the **Partitions** folder.

3. Select **Start** or **Stop** from the drop-down menu.



To start or stop a partition from the command line, run the **`partition.sh`** command. For example:

```
/opt/vd-server-2.0/bin/partition.sh stop partition DEFAULT
```

The **`partition.sh`** command is explained in *Section A.7, "partition.sh"*.

5.5. Using Custom Java Classes

Penrose Virtual Directory comes with many different Java libraries which define different entry types, such as **`org.safehaus.penrose.partition.Partition`** for a partition and **`org.safehaus.penrose.directory.Entry`** for virtual directory entries. It is possible to add custom classes to support special entries or to define specific subtree elements quickly.

To use a class for a specific partition only, create a **`lib/`** directory in the partition directory, **`/opt/vd-server-2.0/partitions/`*`partition_name`*`/DIR-INF`**.

To make a custom class available for any partition, put the JAR file in the **`/opt/vd-server-2.0/lib/ext`** directory.

NOTE

Do not put any custom libraries in **`/opt/vd-server-2.0/lib/`**. This directory is reserved for built-in libraries, and any custom files may be deleted or overwritten during upgrades.

Configuring Connections

A virtual directory in Penrose Virtual Directory establishes relationships between different data sources. The source exists on a connection which is, most simply, a server host and application.

This chapter covers how to add new server connections to Penrose Virtual Directory.

6.1. About Connections

A virtual directory establishes relationships between different data sources, which may not speak the same native protocol or have shared attributes, even shared entries. In Penrose Virtual Directory, the virtual directory is defined by configuring the Penrose Server to recognize data sources.

A *connection* is the connection information and parameters to the host machine of a data source for the virtual directory. One connection is configured automatically for the server, and there can be an unlimited number of connections for failover and for other data sources.

A connection defines kind of data source (and the service for the Penrose Server to use to connect) through *adapters*, a backend service used by the server. There are two adapters defined in Penrose Server for LDAP and JDBC (which includes JDBC, Microsoft SQL Server, Oracle, MySQL, PostgreSQL, ODBC, and Sybase). A NIS adapter can be added to the Penrose Server to handle NIS migrations and integration.

6.2. Adding a NIS Adapter

Adapters are abstraction layers to access data sources. LDAP and JDBC adapters are configured by default, and a NIS adapter can be configured.

1. Open the configuration file for the partition. For the default partition, this is in **`/opt/vd-server-2.0/conf/server.xml`**; for added partitions, it is in **`/opt/vd-server-2.0/partitions/`***partition_name*`/DIR-INF/partition.xml`**.

```
vim /opt/vd-server-2.0/conf/server.xml
```

2. Add the NIS adapter entry.

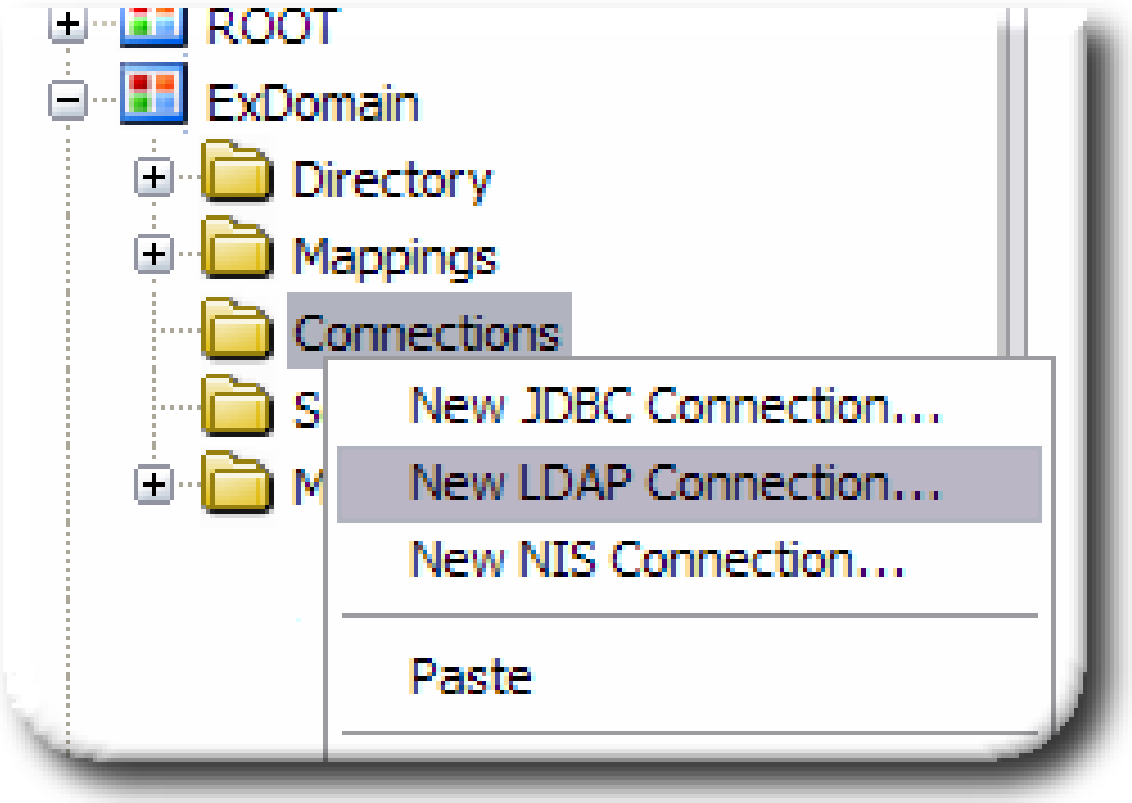
```
<adapter name="NIS">
<adapter-class>org.safehaus.penrose.nis.adapter.NISAdapter</adapter-class
>
</adapter>
```

6.3. Creating Connections in Penrose Studio

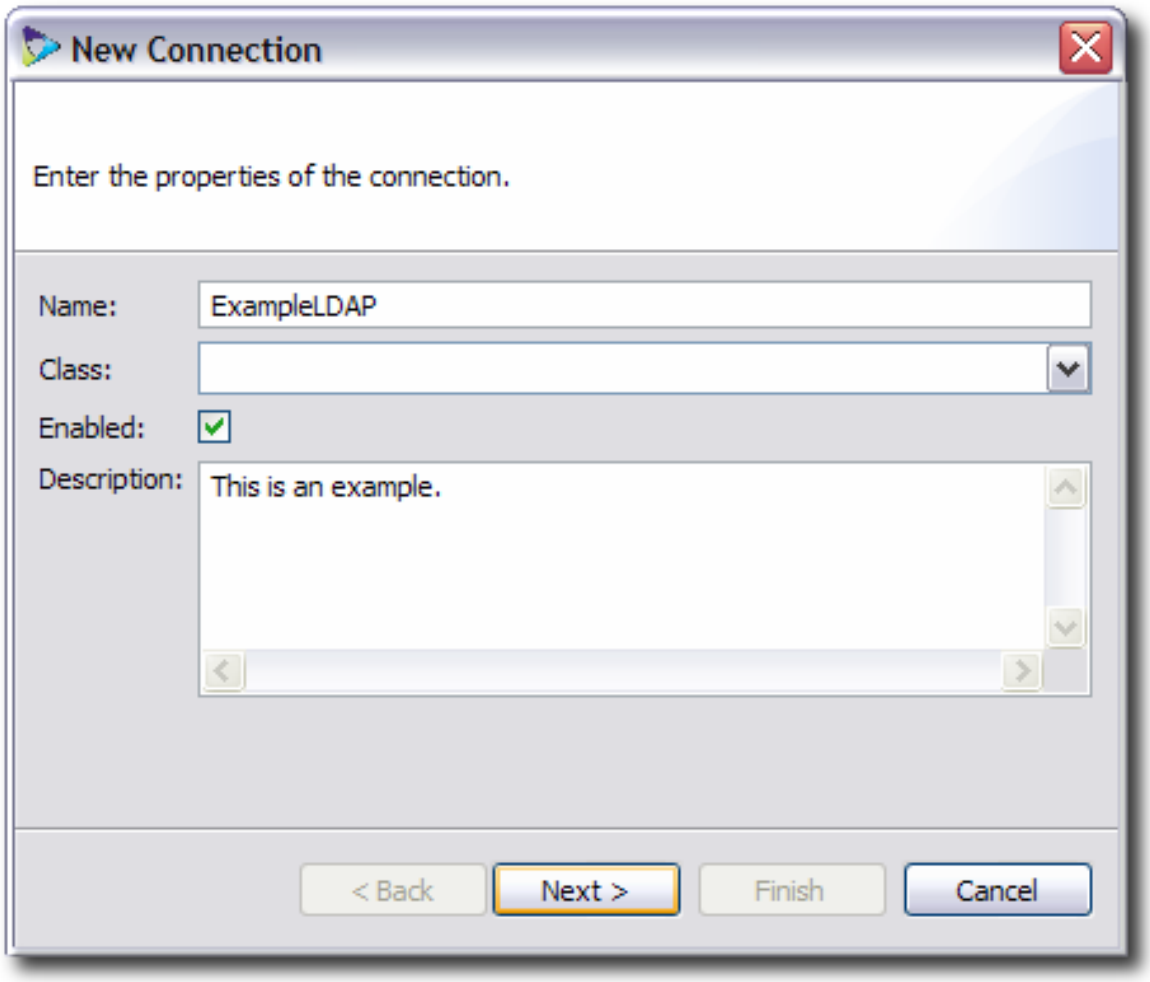
1. Open the server entry.

2. In the top menu, expand the **Partitions** menu item, and select the **Connections** folder.

3. Right-click **Connections**, and select the type of new connection to create from the menu.

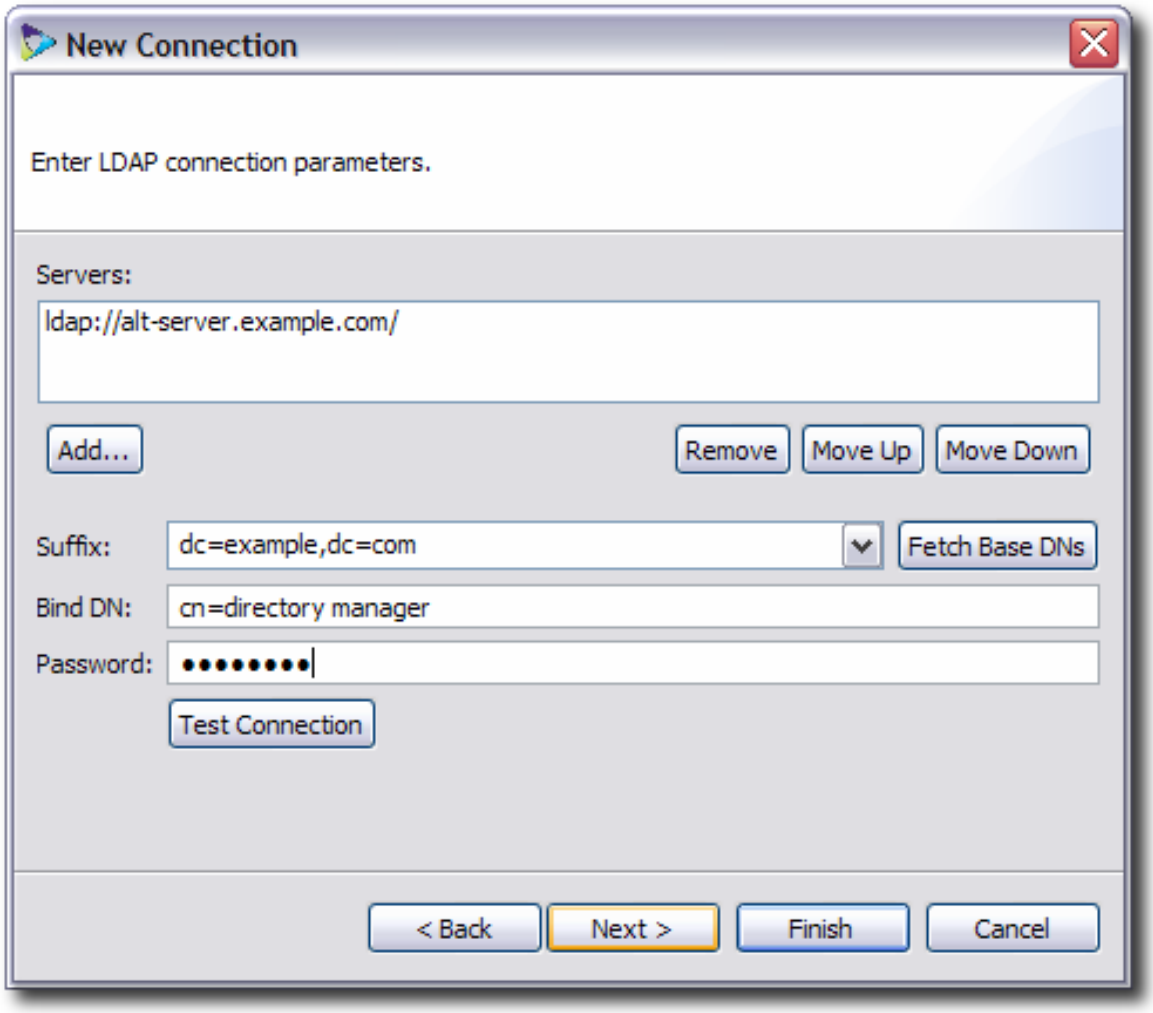


4. Name the connection and, if necessary, give a brief description.



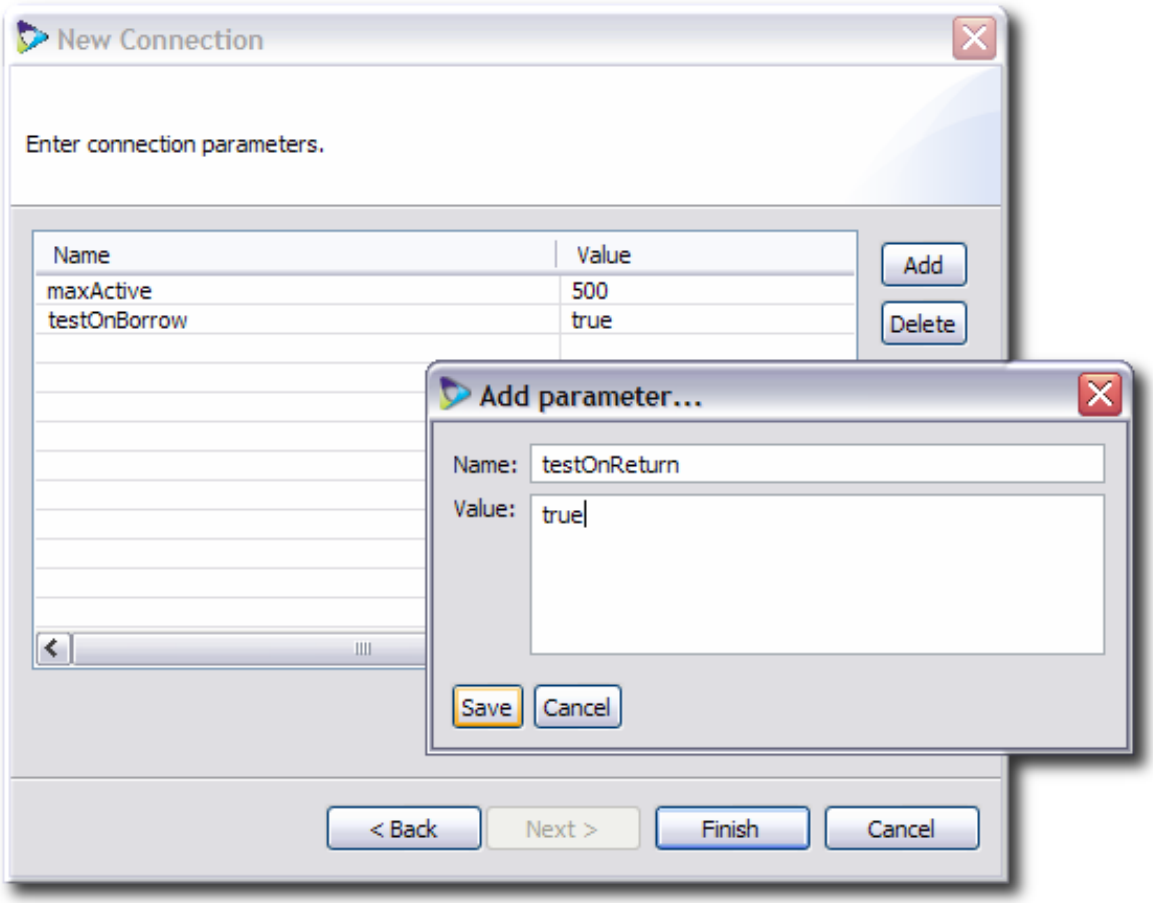
The **Class** is available for applying a custom Java class to the entry, but, if it is not given, the default class is used. The default is fine for almost all applications.

5. Fill in all of the connection parameters; this is different for every kind of connection. For example, for an LDAP connection:



Use the **Test Connection** button to make sure that the correction parameters are entered properly and that the connection server or application is available.

6. Optionally, fill in connection pool parameters for the connection. The LDAP and JDBC connection parameters are listed in *Table 6.2, "Connection Pool Options for LDAP and JDBC"*.



7. Click **Finish**.

6.4. Editing Connections in Penrose Studio

A connection can be viewed and edited in Penrose Studio. There are several tabs which correspond to a different area of the connection entry:

- There is a tab named for the type of connection, such as **LDAP**, which allows the connection information to be changed.

- The **Properties** tab contains all optional configuration entries for the connection, like connection pool parameters.

- The **Schema** tab shows the object classes and attributes which are accessed through the connection.

There is also a **Browser** tab, which displays the entries configured within the connection.

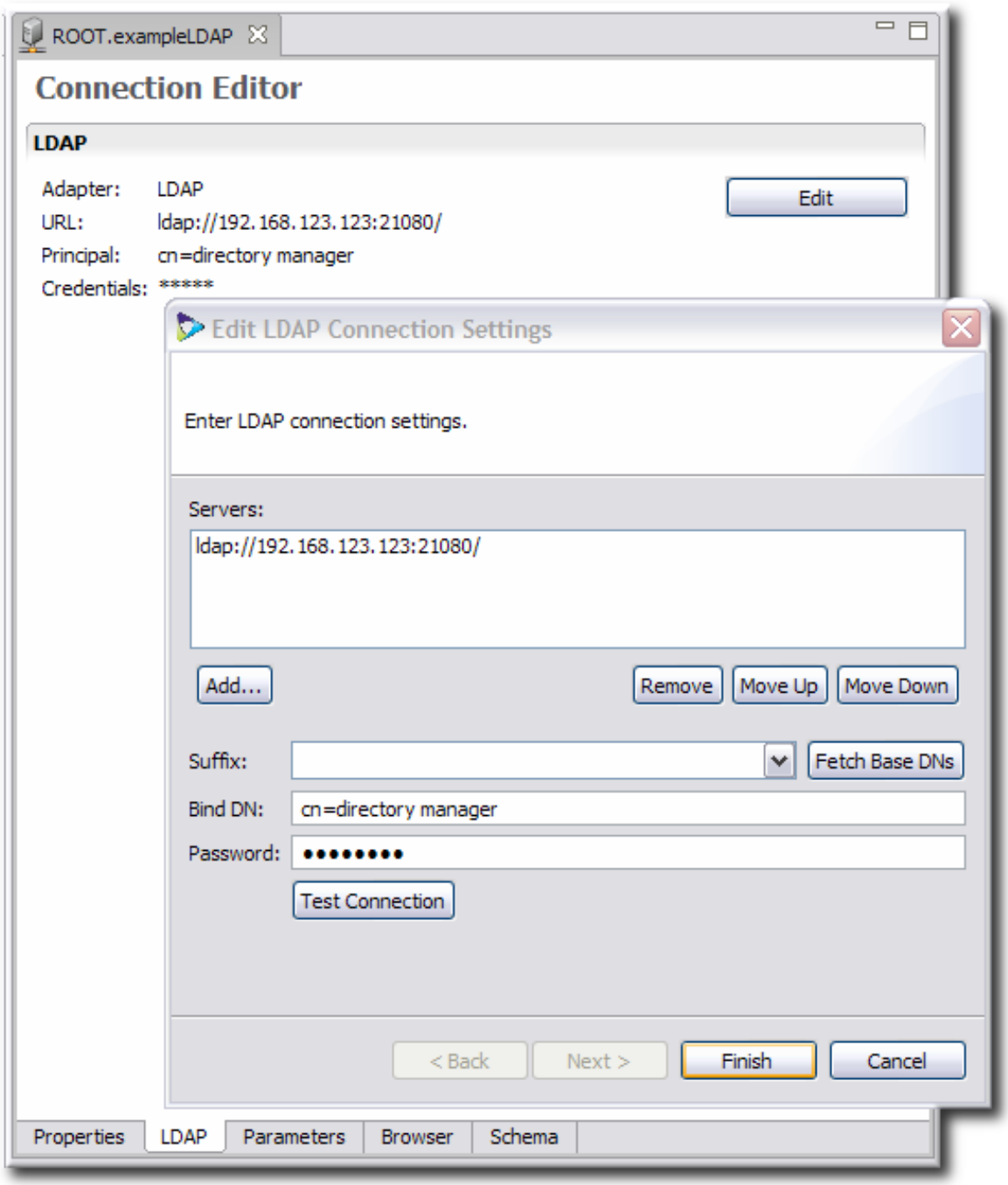
To edit the connection:

1. Open the server entry.

2. Expand the **Partitions** menu item, and open the **Connections** folder.

3. Double-click the connection entry or right-click and select **Open** to open the entry in the main window.

4. In the **LDAP|JDBC|NIS** tab, depending on the type of connection, and change the connection parameters.



5. Close the main tab and save the changes when prompted.

6.5. Creating and Editing Connections Manually

Connection settings are defined in the file **`connections.xml`**. For the default partition, this file is in **`/opt/vd-server-2.0/conf`**; for additional partitions, this file is in **`/`**

opt/vd-server-2.0/partitions/*partition_name***/DIR-INF**.

```
 <connections> main file tag

   <connection name="..."> begins the connection entry

     <adapter-name>...</adapter-name> the connection type, LDAP, JDBC, or
NIS

     <parameter> the configuration settings, in attribute-value pairs
          <param-name>...</param-name>
          <param-value>...</param-value>
     </parameter>

   </connection>

 </connections>
```

Example 6.1. Annotated connections.xml File

To create a new connection, add a new connection entry to the **connections.xml** file. To edit a connection, add, remove, or edit parameters within the entry. *Example 6.2, "Example connections.xml File"* shows three different connection entries for a partition, for each adapter type.

IMPORTANT

Always restart Penrose Server after editing the configuration file. For example:

```
service vd-server restart
```

```
<connections>

<connection name="DirectoryServer">
  <adapter-name>LDAP</adapter-name>
  <parameter>
    <param-name>maxActive</param-name>
    <param-value>500</param-value>
  </parameter>
  <parameter>
    <param-name>testOnBorrow</param-name>
    <param-value>true</param-value>
  </parameter>
  <parameter>
    <param-name>java.naming.provider.url</param-name>
    <param-value>ldap://localhost/</param-value>
  </parameter>
  <parameter>
    <param-name>java.naming.security.principal</param-name>
    <param-value>cn=Directory Manager</param-value>
```

```
  </parameter>
  <parameter>
    <param-name>java.naming.security.credentials</param-name>
    <param-value>secret</param-value>
  </parameter>
</connection>

<connection name="MySQL">
    <adapter-name>JDBC</adapter-name>
    <parameter>
      <param-name>user</param-name>
      <param-value>exampleuser</param-value>
    </parameter>
    <parameter>
      <param-name>password</param-name>
      <param-value>secret</param-value>
    </parameter>
    <parameter>
      <param-name>url</param-name>
<param-value>jdbc:mysql://localhost/example?autoReconnect=true</param-val
ue>
    </parameter>
    <parameter>
      <param-name>driver</param-name>
      <param-value>com.mysql.jdbc.Driver</param-value>
    </parameter>
  </connection>

<connection name="NIS">
    <adapter-name>NIS</adapter-name>
    <parameter>
      <param-name>java.naming.factory.initial</param-name>
      <param-value>com.sun.jndi.nis.NISCtxFactory</param-value>
    </parameter>
    <parameter>
      <param-name>java.naming.provider.url</param-name>
      <param-value>nis://localhost/</param-value>
    </parameter>
    <parameter>
      <param-name>com.sun.jndi.nis.mailaliases</param-name>
      <param-value>nonull</param-value>
    </parameter>
    <parameter>
      <param-name>method</param-name>
      <param-value>yp</param-value>
    </parameter>
  </connection>

</connections>
```

Example 6.2. Example connections.xml File

The different configuration settings for the file are in *Table 6.1, "Connection Configuration Values"*.

Configuration Setting	Description	Example
LDAP Connections		

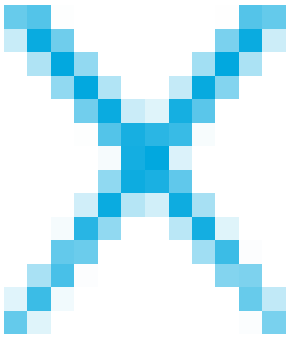
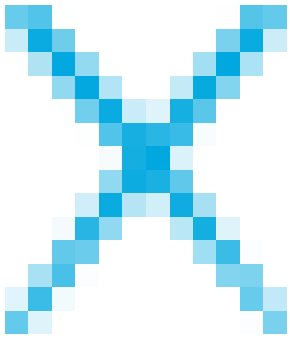
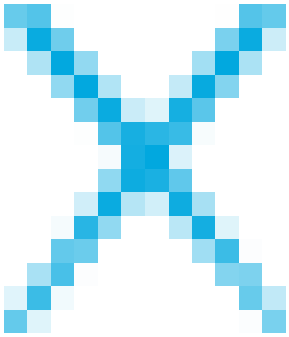
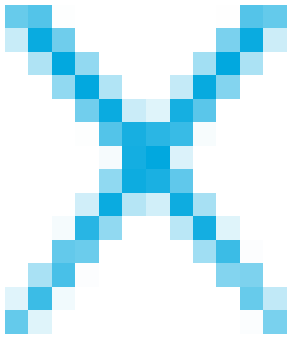
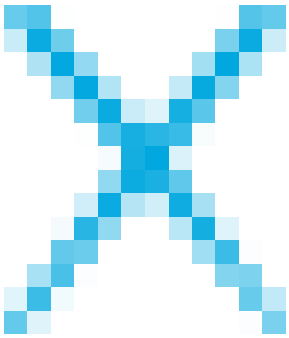
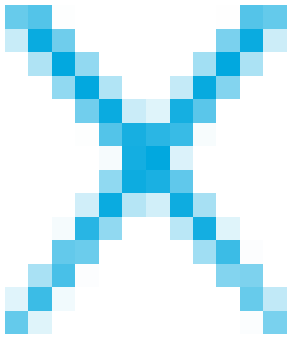
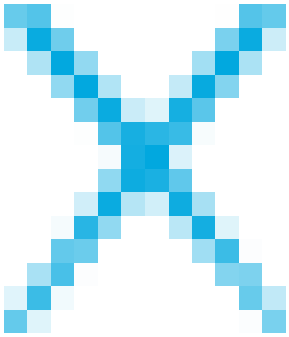
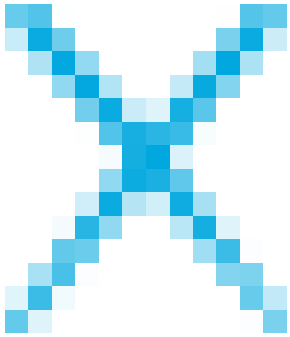
Configuration Setting	Description	Example
java.naming.provider.url	Gives a URL to a naming service on a remote machine. To specify failover machines, separate the URLs with a space, and the Penrose Server will try them in order.	ldap://remote.example.com/
java.naming.security.principal	Gives the bind DN to use to connect to the LDAP service.	cn=Directory Manager
java.naming.security.credentials	Contains the password associated with the bind DN.	secret
JDBC Connections		
driver	Gives the JDBC driver class name for the database.	com.mysql.jdbc.Driver
url	Gives the URL to connect to the database.	jdbc:mysql://localhost/example?autoReconnect=true
user	Gives the username to use to connect to the database.	exampleuser
password	Gives the password associated with the username.	secret
quote	Delimiter for quoting identifiers.	
queryTimeout	Sets the JDBC query timeout limit, in seconds.	30
NIS Connections		
java.naming.factory.initial	Sets the naming service for Penrose Server to use. The NIS JNDI service class is **`com.sun.jndi.nis.NISCtxFactory`**.	com.sun.jndi.nis.NISCtxFactory
java.naming.provider.url	Gives a URL to a naming service on a remote machine.	nis://localhost/
method	Defines the way to connect to the NIS source. There are three options: • YP clients (**`yp`**) • JNDI (**`jndi`**) • Local files (**`local`**)	jndi

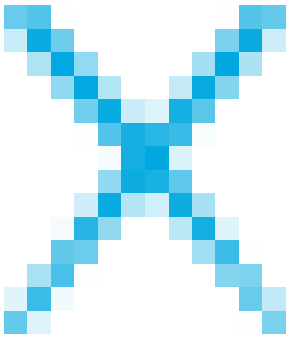
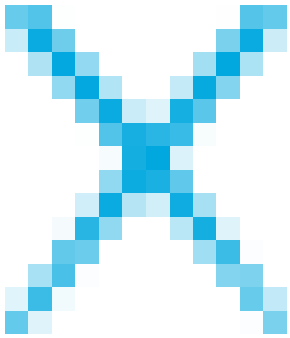
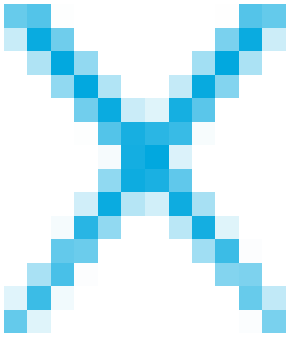
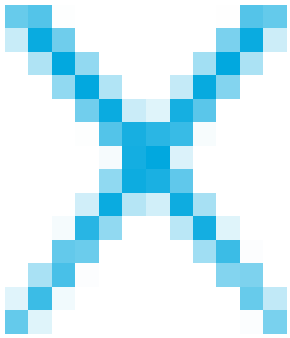
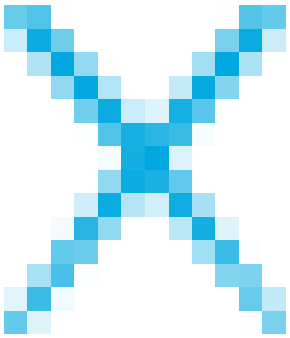
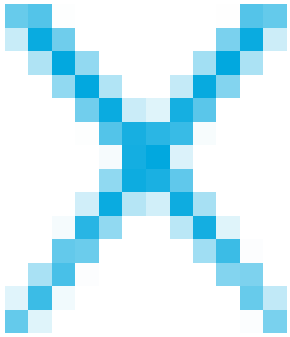
Table 6.1. Connection Configuration Values

LDAP and JDBC connections have addtional, optional parameters to configure the connection pool

settings for the connection to the source.

Parameter	Definition	LDAP Connection	JDBC Connection
maxActive	The maximum number of active connections that can be allocated from this pool at the same time, or zero for no limit.		
maxIdle	The maximum number of active connections that can remain idle in the pool, without extra ones being released, or zero for no limit.		
minIdle	The minimum number of active connections that can remain idle in the pool, without extra ones being created, or zero to create none.		
maxWait	The maximum number of milliseconds that the pool will wait (when there are no available connections) for a connection to be returned before throwing an exception, or -1 to wait indefinitely.		
testOnBorrow	Indicates whether objects will be validated before being borrowed from the pool. If the object fails to validate, it		

Parameter	Definition	LDAP Connection	JDBC Connection
	will be dropped from the pool, and Penrose Server attempts to borrow another.		
testOnReturn	Indicates whether objects will be validated before being returned to the pool.		
testWhileIdle	Indicates whether objects will be validated by the idle object evictor (if any). If an object fails to validate, it will be dropped from the pool.		
minEvictableIdleTimeMillis	The minimum amount of time an object may sit idle in the pool before it is eligable for eviction by the idle object evictor (if any).		
numTestsPerEvictionRun	The number of objects to examine during each run of the idle object evictor thread (if any).		

Parameter	Definition	LDAP Connection	JDBC Connection
			
timeBetweenEviction-RunsMillis	The number of milli-seconds to sleep between runs of the idle object evictor thread. When non-positive, no idle object evictor thread will be run.		
whenExhaustedAction	fail, block, or grow		
initialSize	The initial size of the connection pool.		
validationQuery	The SQL query that will be used to validate connections from this pool before returning them to the caller. If specified, this query MUST be an SQL SE-		

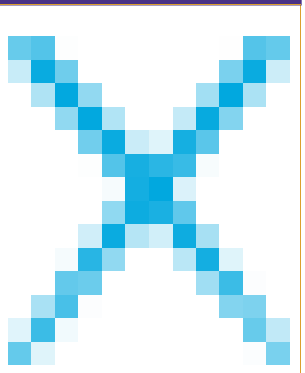
Parameter	Definition	LDAP Connection	JDBC Connection
	LECT statement that returns at least one row.		

Table 6.2. Connection Pool Options for LDAP and JDBC

Configuring Data Sources

Data sources are applications or servers which can be accessed by Penrose Server; the source entry defines the data set and the reverse mappings going from the source data to the virtual directory entry attribute. This chapter describes how to create and edit sources.

7.1. About Data Sources

Data sources are applications or servers which can be accessed by Penrose Server. A connection gives a connection to a server machine; a source is an application on that host machine. A single connection could theoretically have several databases, LDAP servers, and NIS servers. A source is a single instance of Directory Server or a single Active Directory server.

To define the source, the entry includes information to access the source, such as a search base and filter to access an LDAP server. It also lists relevant object classes and attributes for the entries contained in the source.

The source entry also contains *fields*. These fields are *reverse mappings*. Basically, Penrose Virtual Directory creates a directory by creating a pair of cross-references. The first mapping (described in *Chapter 8, Configuring the Virtual Directory*) links the virtual directory entry attribute to a source attribute. The field in the source entry is a reverse mapping, going from the source attribute to a virtual directory attribute.

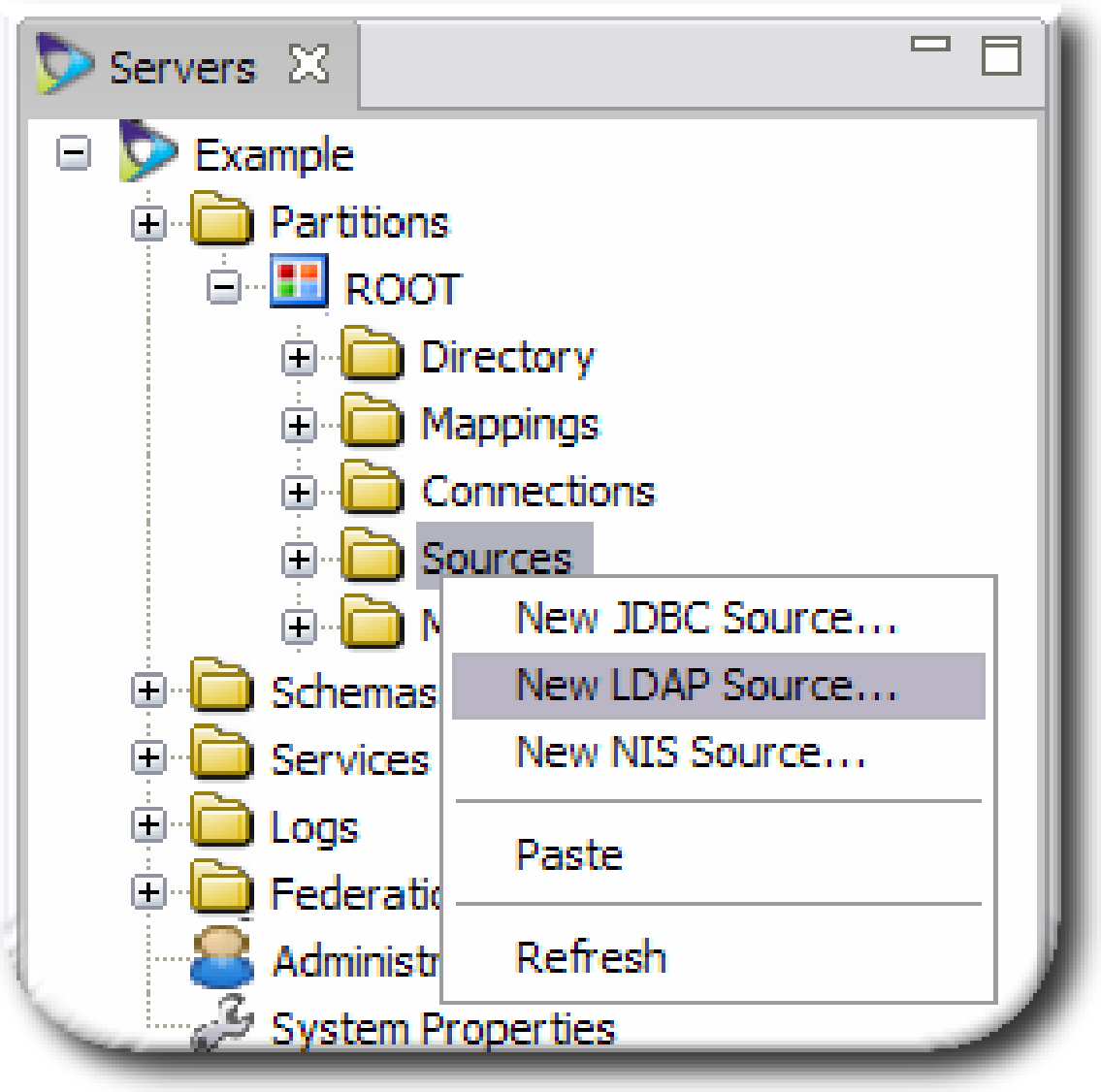
The information referenced in a field depends on the type of data source:

- In a JDBC data source, the source is a database table, and the fields are the table columns.
- For an LDAP data source, the source is a directory subtree, and the fields are its attributes.
- For an NIS data source, the source is an NIS map, and the fields are the columns in that map.

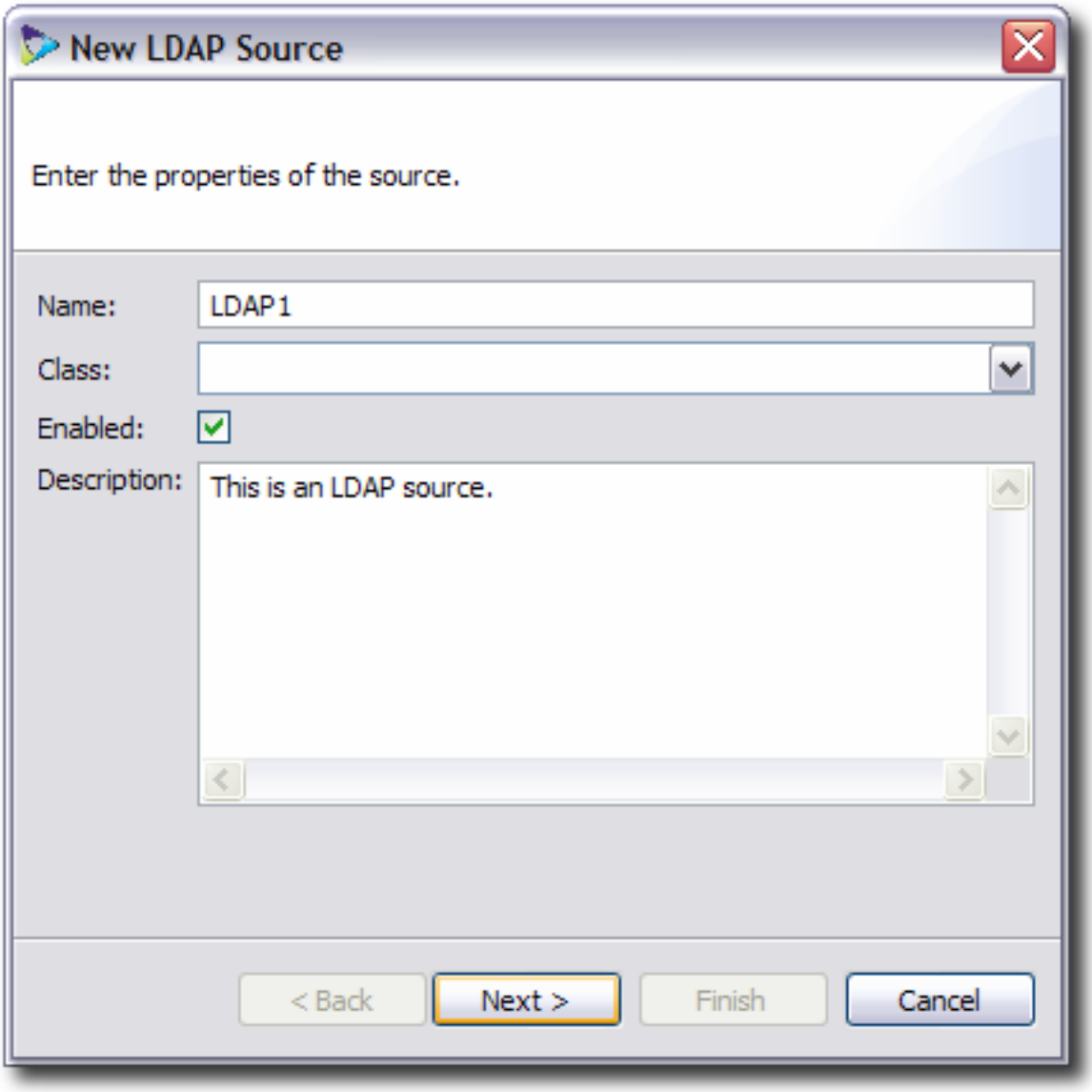
The data within sources can be normalized on the source before being processed by Penrose Virtual Directory mappings. Penrose Virtual Directory can also normalize data, but this is done through Penrose Virtual Directory, which uses Java, rather than the native tools on the source itself, so normalizing on Penrose Virtual Directory can be much slower than normalizing data on the source. What kind of normalization can be performed depends on the source schema.

7.2. Configuring Sources in Penrose Studio

1. Open the server entry in Penrose Studio, and expand the **Partitions** folder.
2. Right-click the **Sources** folder, and select **New ... Source...** from the menu. There are three options — LDAP, JDBC, and NIS — for each supported source type.

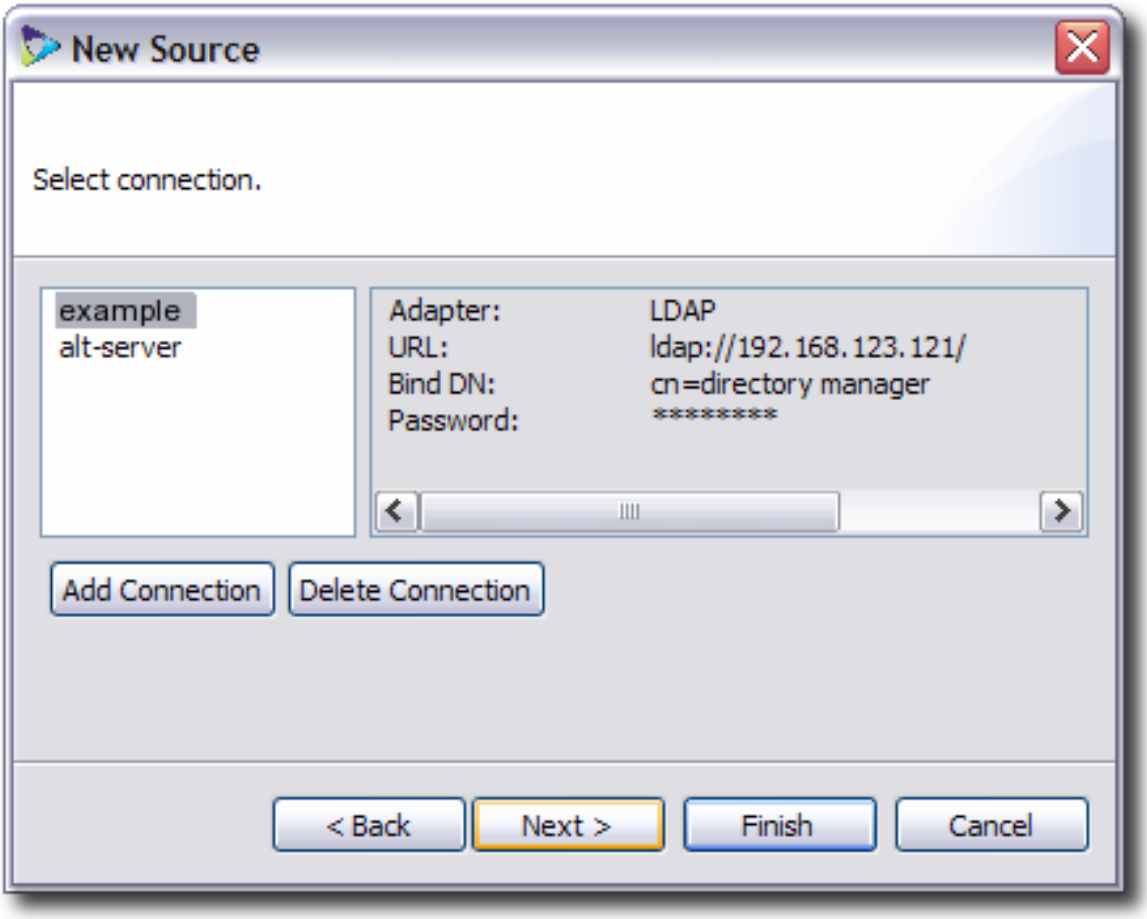


3. Name the source and, if necessary, give a brief description.

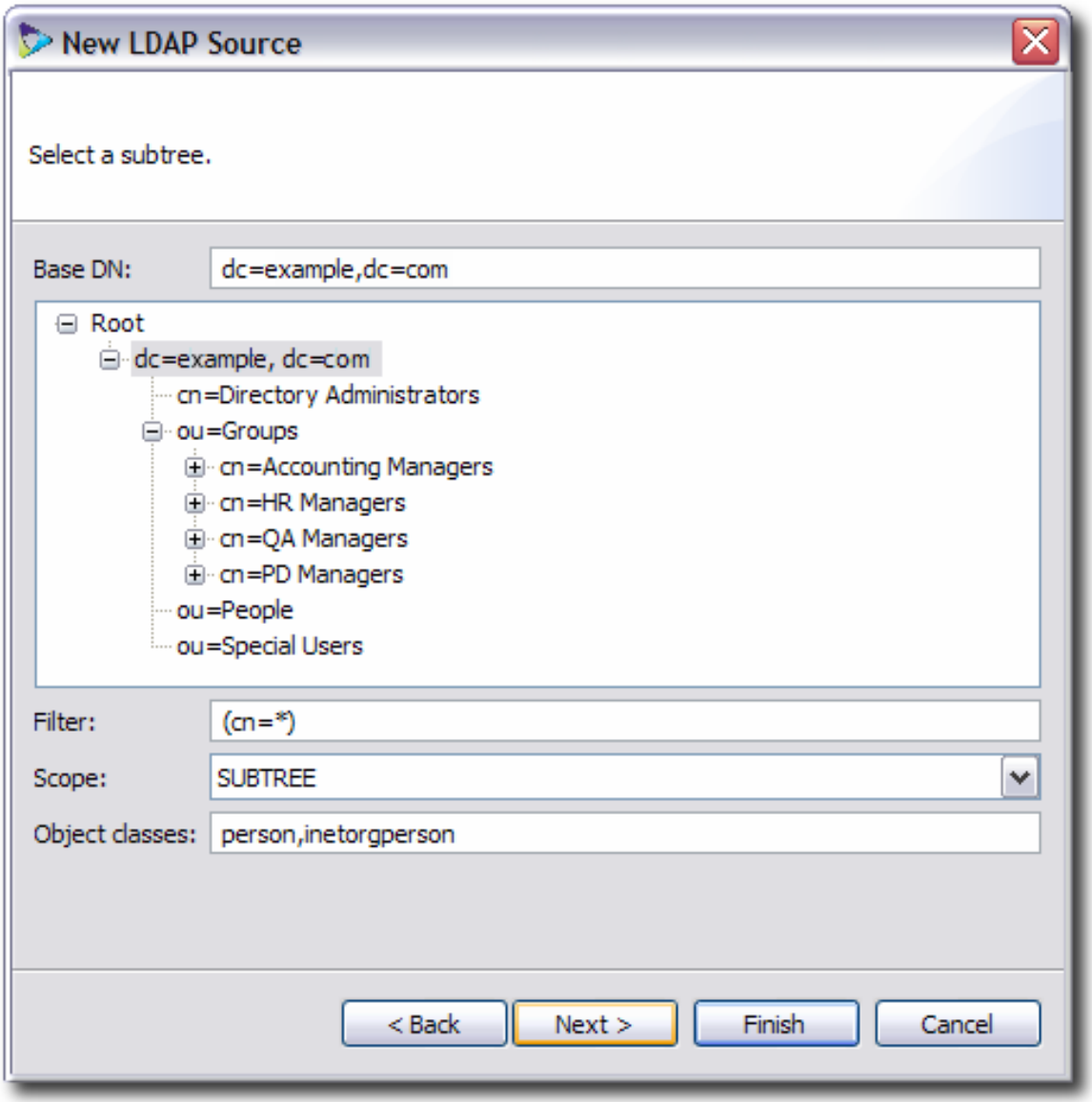


The **Class** is available for applying a custom Java class to the entry, but, if it is not given, the default class is used. The default is fine for almost all applications.

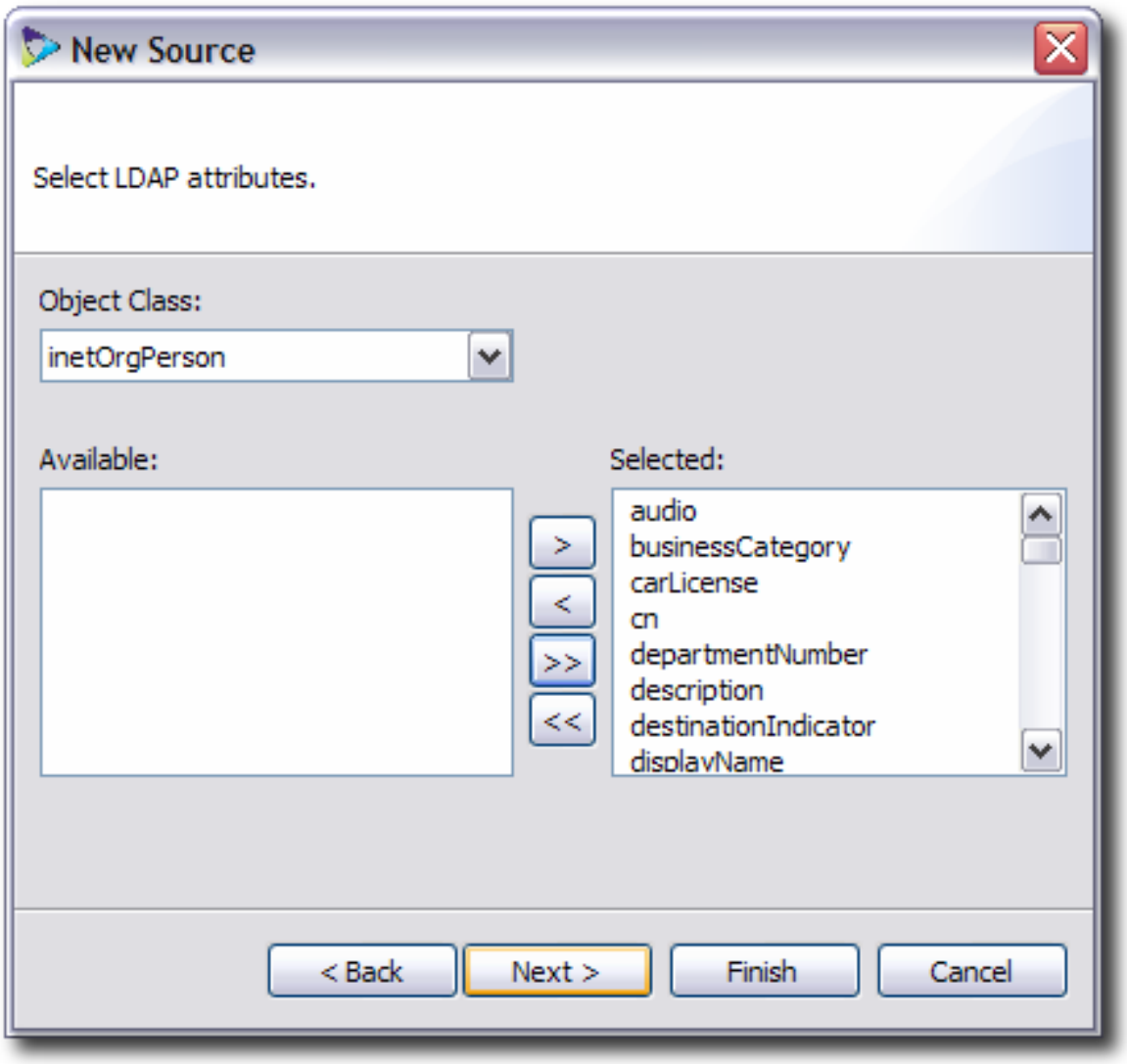
4. Choose the connection to use for the source. You can also add a new connection to use a different host or to use a different adapter to connect to a host.



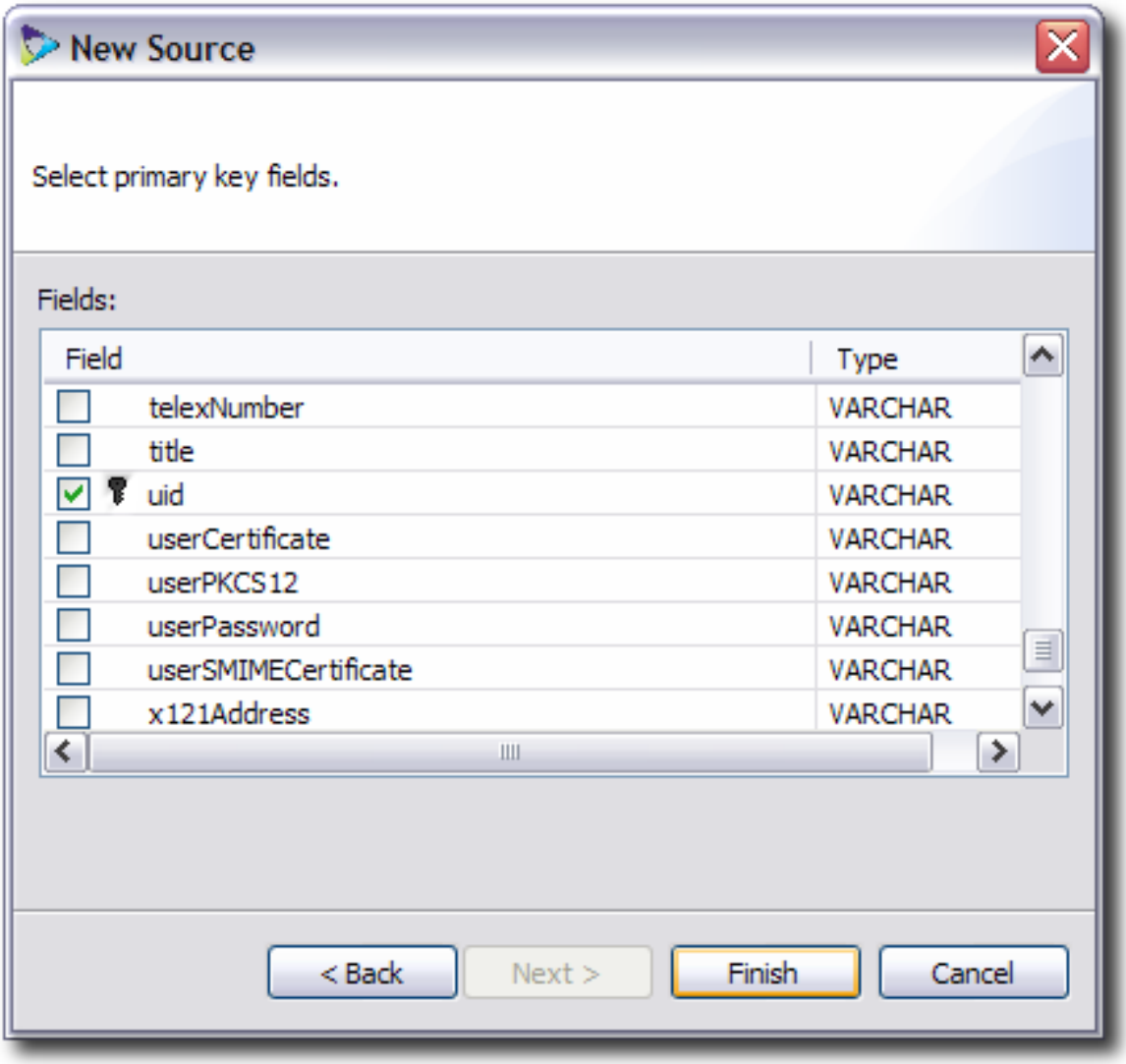
5. Fill in the data source information. For example, for an LDAP source, supply parameters for LDAP operations (base DN, search filter, and search scope). The parameters for LDAP, JDBC, and NIS sources are described in *Table 7.2, "Parameters for LDAP, JDBC, and NIS Sources"*.



6. Select the attributes allowed with the object class which will be displayed when the data source is browsed.



7. Select the checkbox by the attributes which will be used for naming or identifying the entries.



8. Click **Finish** to save the new data source.

7.3. Creating and Editing Sources Manually

Data sources are configured in the file **`sources.xml`**, and, as with other partition configuration files, the **`sources.xml`** file is in **`/opt/vd-server-2.0/conf`** for the default partition and in **`/opt/vd-server-2.0/partitions/`***partition_name*`/DIR-INF`** directory for additional partitions. This file is illustrated in *Example 7.1, "Annotated sources.xml File"*.

```
<sources>  main file tag
  <source name="..."> begins the connection entry
    <partition-name>...</partition-name> the partition to which the
```

```
source belongs

     <connection-name>...</connection-name> the connection to use with
the source

     <field name="..." primaryKey="..."/> the names of attributes which
will be viewed


     <parameter> the configuration settings, in attribute-value pairs
        <param-name>...</param-name>
        <param-value>...</param-value>
     </parameter>

   </source>

 </sources>
```

Example 7.1. Annotated sources.xml File

To create a new source, add a new source entry to the **`sources.xml`** file. To edit a source, add, remove, or edit parameters within the entry. *Example 7.2, "Example sources.xml File"* shows three different source entries for LDAP, JDBC, and NIS sources.

IMPORTANT

Always restart Penrose Server after editing the configuration file. For example:

```
service vd-server restart
```

```
<sources>

   <source name="LDAP1">
     <partition-name>ExamplePartition2</partition-name>
     <connection-name>example</connection-name>
     <field name="uid" primaryKey="uid" />
     <field name="givenName" />
     <field name="surName" />
     <field name="cn" />
     <field name="mail" />
     <parameter>
               <param-name>baseDn</param-name>
               <param-value>dc=example,dc=com</param-value>
     </parameter>
     <parameter>
               <param-name>scope</param-name>
               <param-value>SUBTREE</param-value>
     </parameter>
     <parameter>
               <param-name>filter</param-name>
```

```
            <param-value>(objectClass=*)</param-value>
      </parameter>
      <parameter>
            <param-name>objectClasses</param-name>
            <param-value>inetorgperson,person</param-value>
      </parameter>
    </source>

    <source name="MySQL">
      <partition-name>ExamplePartition2</partition-name>
      <connection-name>example</connection-name>
      <field name="username" />
      <field name="firstname" />
      <field name="lastname" />
      <field name="fullname" />
      <field name="email" />
      <parameter>
            <param-name>catalog</param-name>
            <param-value>employees</param-value>
      </parameter>
      <parameter>
            <param-name>schema</param-name>
            <param-value>system</param-value>
      </parameter>
      <parameter>
            <param-name>table</param-name>
            <param-value>employees</param-value>
      </parameter>
      <parameter>
            <param-name>filter</param-name>
            <param-value>u.lastname = ''</param-value>
      </parameter>
    </source>

    <source name="NISUsers">
      <connection-name>NIS</connection-name>
      <field name="uid" primaryKey="true"/>
      <field name="uidNumber"/>
      <field name="gidNumber"/>
      <field name="homeDirectory"/>
      <field name="userPassword"/>
      <field name="loginShell"/>
      <field name="gecos"/>
      <field name="description"/>
      <parameter>
            <param-name>objectClasses</param-name>
            <param-value>posixAccount</param-value>
      </parameter>
      <parameter>
            <param-name>base</param-name>
            <param-value>passwd</param-value>
      </parameter>
   </source>
```

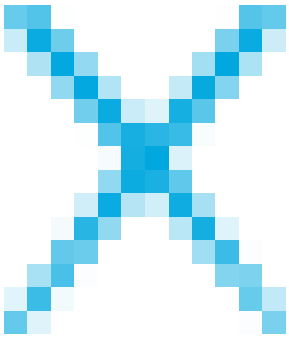
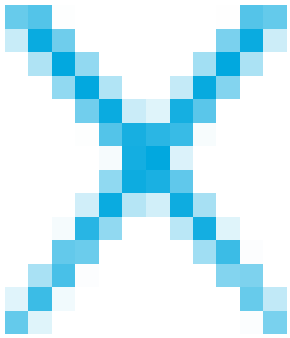
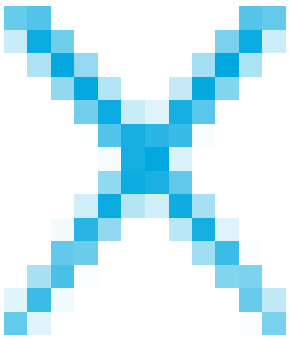
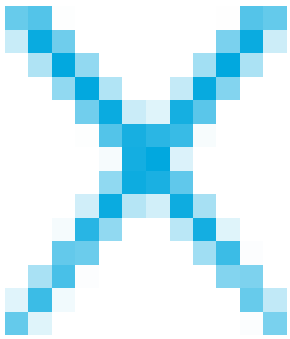
Example 7.2. Example sources.xml File

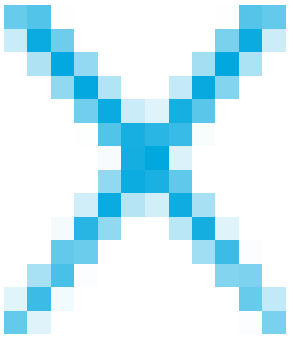
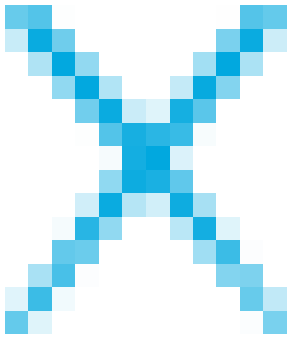
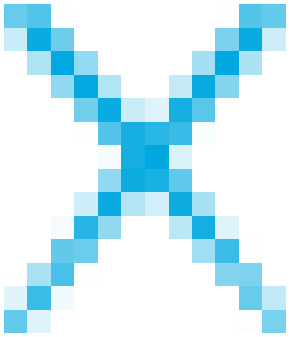
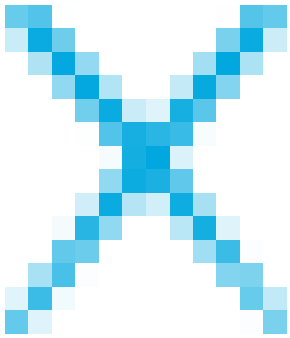
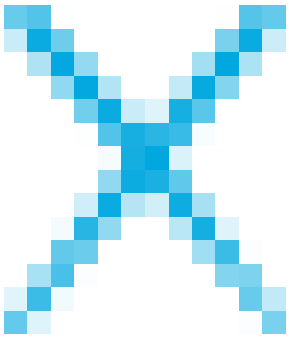
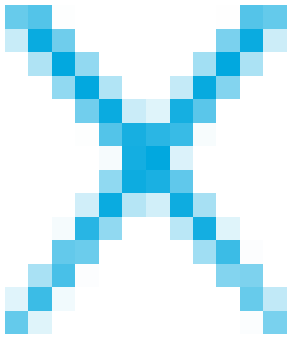
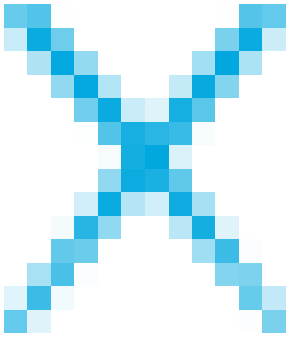
Two important parameters are set in the **`sources.xml`** file: *fields* and *parameters*.

- *Fields* are reverse mappings, rules for matching source attributes with corresponding attributes in the virtual directory entries. Each field represents the attribute, accepted attribute value, and whether it is required or if there are any preconditions for processing the rule. The order of the fields is the order in which Penrose Server will process the attributes, so changing the order changes the attribute precedence.

  Fields can have arguments which further control how Penrose Server processes the attribute; for example, the **`primaryKey`** argument signals that a certain attribute is a naming attribute and should be loaded first. These arguments can also be used to normalize the attribute information before it is displayed in the virtual entry. The arguments are listed in *Table 7.1, "Field Arguments for LDAP and JDBC"*.

- *Parameters* vary for the three different types of sources (JDBC, LDAP, and NIS). In general, parameters define where Penrose Server can access the data source, such as containing parameters for LDAP operations. The parameters are listed in *Table 7.2, "Parameters for LDAP, JDBC, and NIS Sources"*.

Argument	Description	JDBC	LDAP
primaryKey	The naming attribute for the entry.		
originalName	The original DN or name of the entry in the source.		
type	The type of entry.		
NIS sources do not take any additional arguments with their fields.			

Argument	Description	JDBC	LDAP
			
originalType	The orignal type of entry in the source.		
castType	Method for changing the value type.		
length	The field length in the database table.		
precision	A parameter to define the precision of number-related data.		
NIS sources do not take any additional arguments with their fields.			

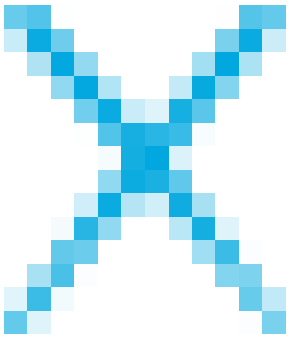
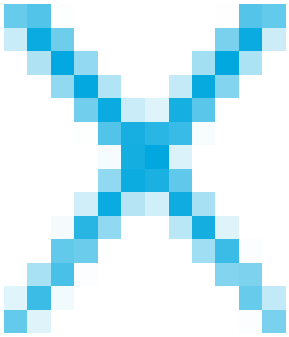
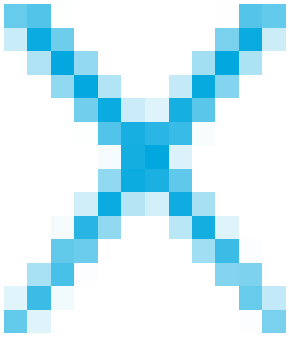
Argument	Description	JDBC	LDAP
			
caseSensitive	Whether the information is case-sensitive.		
autoIncrement	Automatically increments values in table rows.		
NIS sources do not take any additional arguments with their fields.			

Table 7.1. Field Arguments for LDAP and JDBC

Source Parameters	Description	Example
LDAP Sources		
baseDn	As with an LDAP command like **`ldapsearch`**, the entry name of the entry or subtree to use as a search base.	dc=example,dc=com
scope	The scope of the operation, how far below the search base to search for entries. The possible values are **`OBJECT`** (only the base DN), **`ONELEVEL`** (the base DN and all of its immediate chil-	SUBTREE

Source Parameters	Description	Example
	dren), and **SUBTREE** (every entry below the base DN).	
filter	The LDAP filter to use to search for and identify matching entries.	(objectClass=*)
objectClasses	A comma-separated list of object classes which are assigned to the newly-added entries.	person,organizationalPerson,inetOrgPerson
JDBC Sources		
catalog	The name of the database's catalog.	example
schema	The name of the schema used by the database.	system
table	The name of the specified database table.	users
filter	A search filter to use, based on the database field name. The source is identified by prefixing its alias to the filter.	u.lastName = 'Smith'
NIS Sources		
objectClasses	A comma-separated list of object classes which are assigned to the newly-added entries.	posixAccount
base	A search base for NIS attributes.	passwd.adjunct.byname

Table 7.2. Parameters for LDAP, JDBC, and NIS Sources

Configuring the Virtual Directory

The purpose of a Penrose Virtual Directory is to establish relationships between existing servers, databases, and applications which contain data and to make that unified information easily accessible. This is done by creating a new, centralized organization for the entire data set (*virtual directory tree*), then establishing how the different pieces of information in different data sets relate to the new directory (*mappings*).

This chapter describes how to create and manage virtual directory trees and create relationships (mappings) between entries in data sources to create virtual directory entries.

8.1. About the Virtual Directory Tree and Handling Entries

Different data sources may not speak the same native protocol or have shared attributes, or even shared entries. The virtual directory "creates" a new hierarchy, analogous to the directory tree in a directory service, which names and organizes entries. However, this tree view does not create entries or configure a directory in reality; it is a virtual, dynamic way of viewing that information.

A virtual directory mirrors an LDAP directory. It has a hierarchical structure with a base DN, subtrees, and entries which can be searched and modified by LDAP operations and return data in LDIF. This is true regardless of the applications which store the data.

To create the unified view of all the disparate data sources, then, first define the structure of the directory tree and the way that information in a source is collected in the corresponding virtual directory entries.

As with an LDAP service, the first step is to decide what kind of hierarchy the directory tree will have. Each directory tree is laid out in forks and branches, similar to a genealogical tree. Since Penrose Virtual Directory creates a new view of data, the virtual directory does not need to imitate an existing directory hierarchy.

One option is to create a separate subtree for each data source. For example, if there are several different databases which hold employee data, each one could be carried into separate user subtrees, as shown in *Figure 8.1, “Keeping Data in Separate Subtrees”*. The entries are still searchable in a single location by searching the base DN.

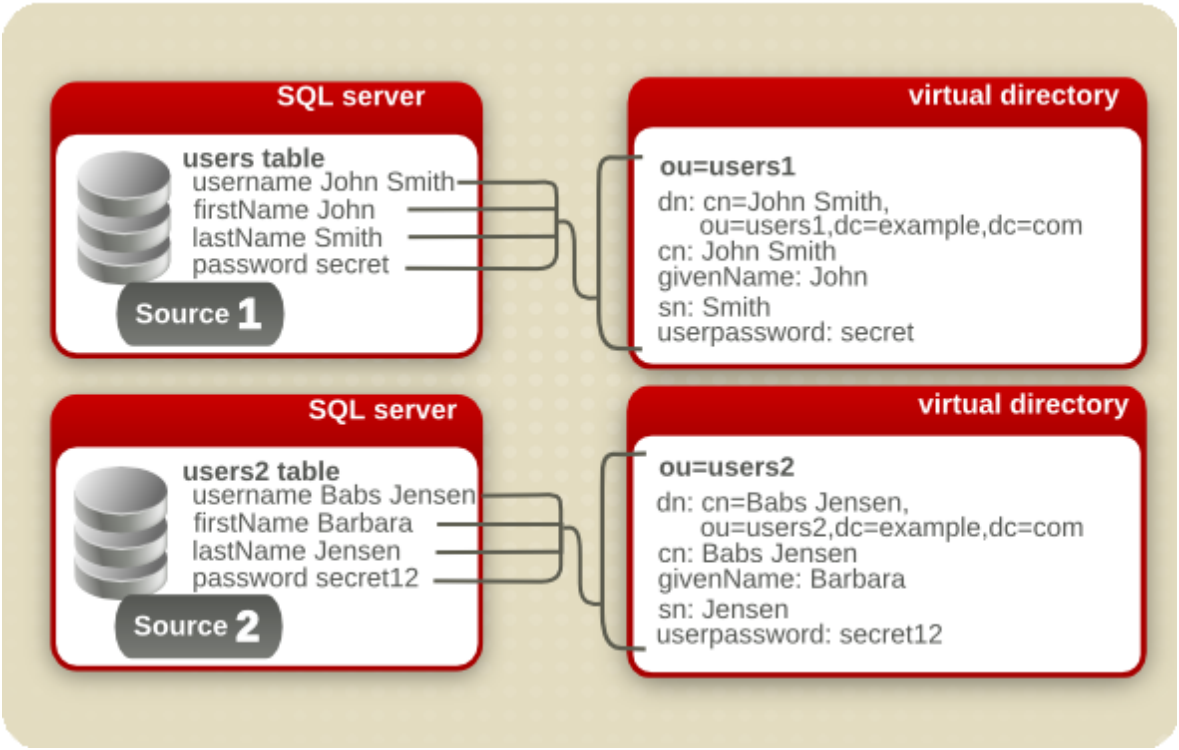


Figure 8.1. Keeping Data in Separate Subtrees

However, the obvious problem is that there can be dozens, even hundreds, of data sources for a single company, and having a dispersed directory tree is unnecessarily unwieldy. There can also be a lot of overlap between the information in one database (or directories or human resources applications), and it makes sense to arrange the grouping of the directory according to the types of information in it. Penrose Virtual Directory can unify all of that information, allowing it to be reorganized in a more logical and easier to find way. To re-group the information in different sources, Penrose Virtual Directory can merge the entries into a single subtree, simply by pointing the configuration for each data source to the same subtree, as shown in *Figure 8.2, "Merging Data Sources into a Single Subtree"*. Even if a single sources holds different kinds of information (such as users and machines), the information can still be mapped to the appropriate subtree.

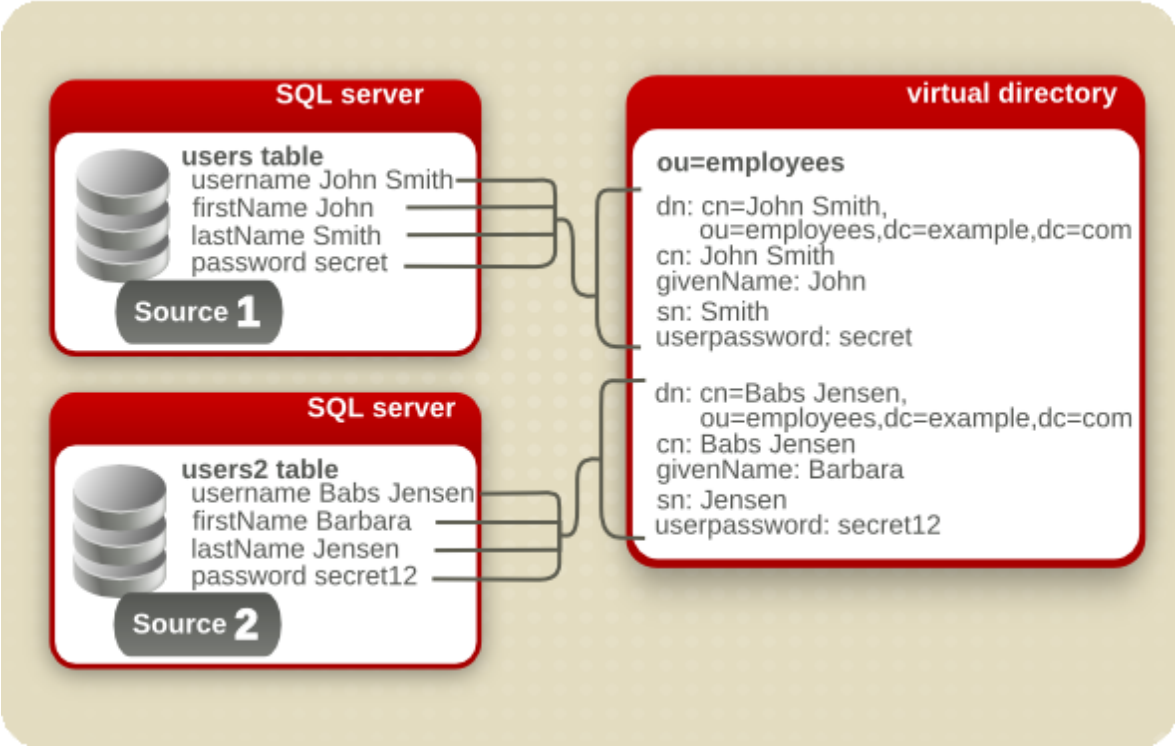


Figure 8.2. Merging Data Sources into a Single Subtree

For example, Example Company has several major applications: NIS servers, Active Directory domains, and several different kinds of SQL databases. All employees in the company have entries in the Active Directory domain and one of the SQL servers, while customer information is in a SQL server, and engineering and support have NIS entries.

Example Company decides to create an employees subtree, a customer subtree, and a server machine subtree. The data will still reside on the individual data sources (sparing the IT department the migration) but all of the relevant information will be unified into a single directory view.

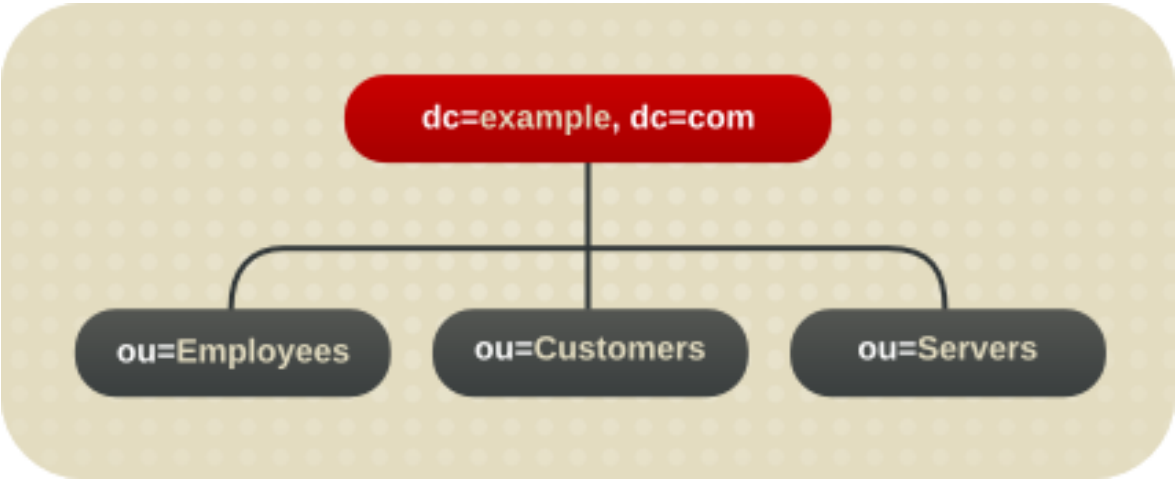


Figure 8.3. Example Company's Virtual Directory Tree

8.2. Creating and Editing the Virtual Subtrees

The virtual directory tree is a hierarchical view, created by a Penrose Server administrator, to help organize the data from different sources. A directory entry is created in Penrose Studio in the **Directory** folder in the **Partitions** section. The configuration file is **`directory.xml`** in **`/opt/vd-server-2.0/conf/`** for the default partition and **`/opt/vd-server-2.0/partitions/`***partition_name*`/DIR-INF`** for additional partitions.

A virtual directory contains seven pieces of information:

- The distinguished name (DN) of the subtree or branch in the virtual directory tree; this is the only required parameter
- Optionally, a class file to use to process operations against the entry
- The sources to use for the directory
- The object classes allowed for virtual directory entries
- Attribute mappings, which define as what virtual directory attributes the source attributes are rendered
- Source fields which map virtual directory attributes back to the original source attributes (reverse mappings)
- Access control instructions (ACIs) that control user permissions to *virtual directory* entries

8.2.1. Creating the Virtual Directory in Penrose Studio

The virtual directory tree is a hierarchical view, created by a Penrose Server administrator, to help organize the data from different sources.

1. The first step is adding a root entry, creating the base DN as in a regular directory, in *Section 8.2.1.1, “Adding a Base DN (Root Entry)”*.
2. The next step involves specifying the subtrees, either by explicitly creating them (*Section 8.2.1.2, “Adding a Static Directory Entry”*) or by creating rules which dynamically determine what entries to include (*Section 8.2.1.3, “Adding a Dynamic Directory Entry”*).

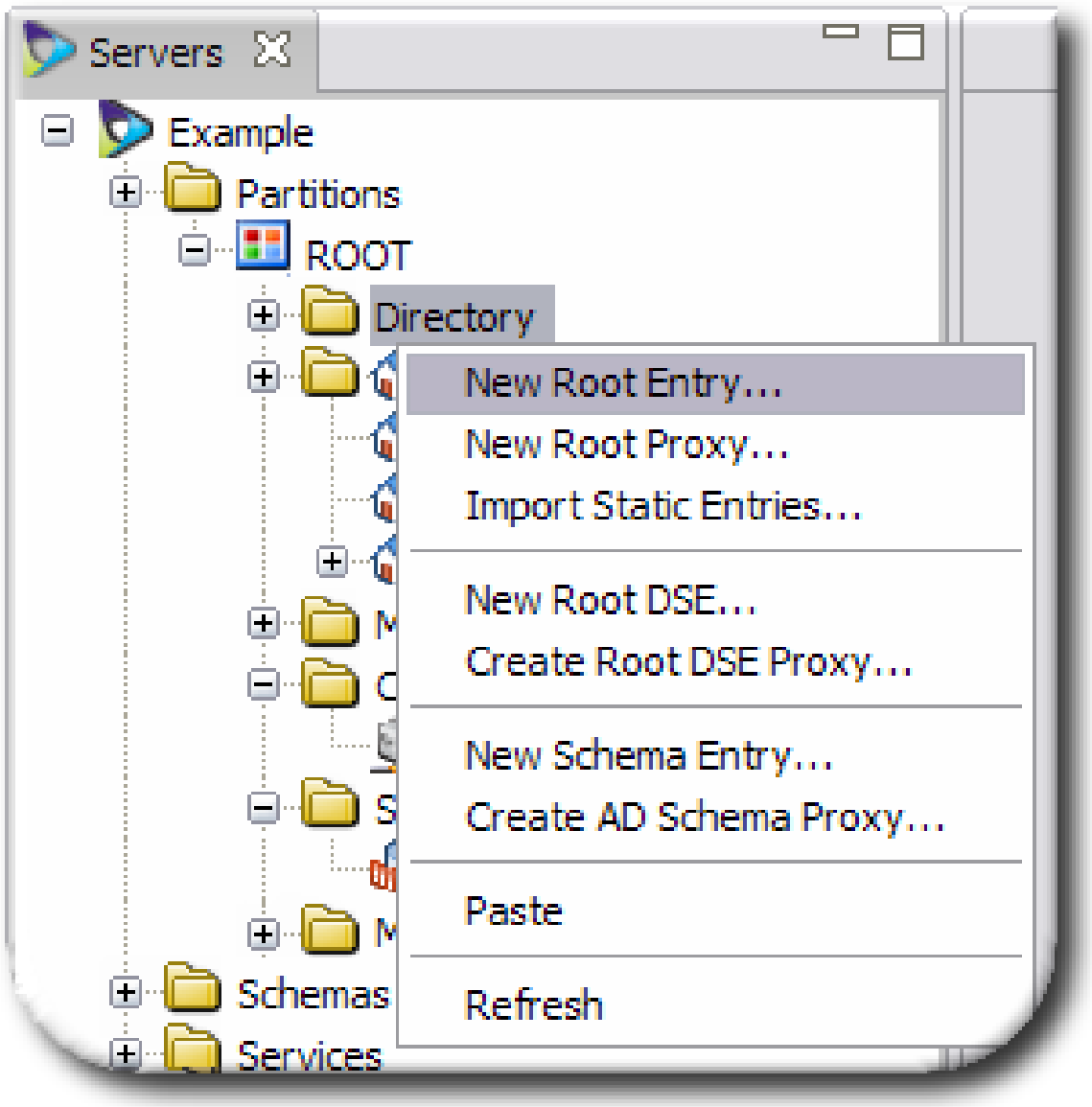
A new directory tree can also be started by taking a snapshot image of an existing LDAP or by imitating an existing LDAP or Active Directory tree (called *mapping* the tree). See *Section 8.4, “Duplicating Existing LDAP Servers”* for more information on LDAP snapshots.

8.2.1.1. Adding a Base DN (Root Entry)

NOTE
The base DN entry is the highest entry in the directory tree.

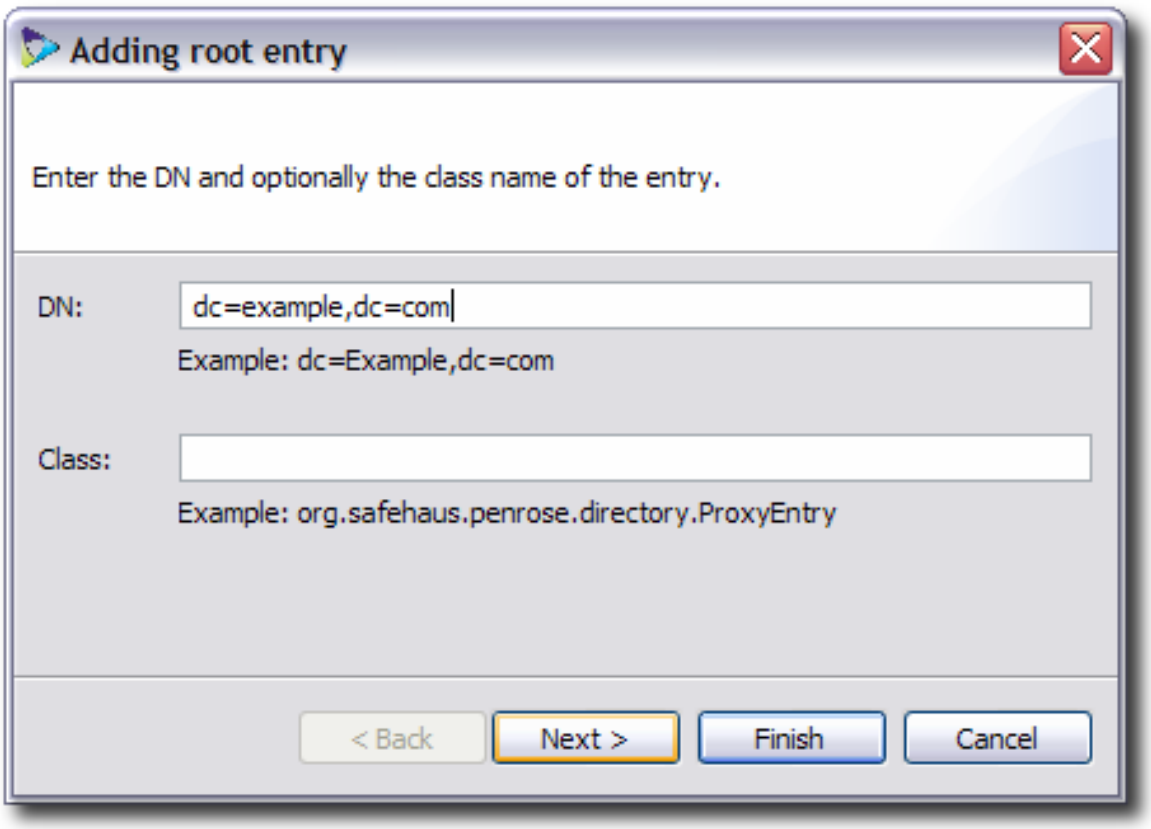
1. In Penrose Studio, open the server entry, then expand the **Partitions** folder.

2. Right-click the **Directory** folder, and select the **New Root Entry...** option.



3.

Fill in the distinguished name (DN) for the entry. This usually has the format **dc=***domainComponent1*, **dc=***domainComponent2*.



Optionally, supply the path name to a class for the static entry. There are five options:

- **`org.safehaus.penrose.directory.Entry`**, the default class, which generates a directory without referencing an external source.

- **`org.safehaus.penrose.directory.Entry.DynamicEntry`**, which creates and joins entries from different external sources. For LDAP servers, this will also flatten the directory namespaces.

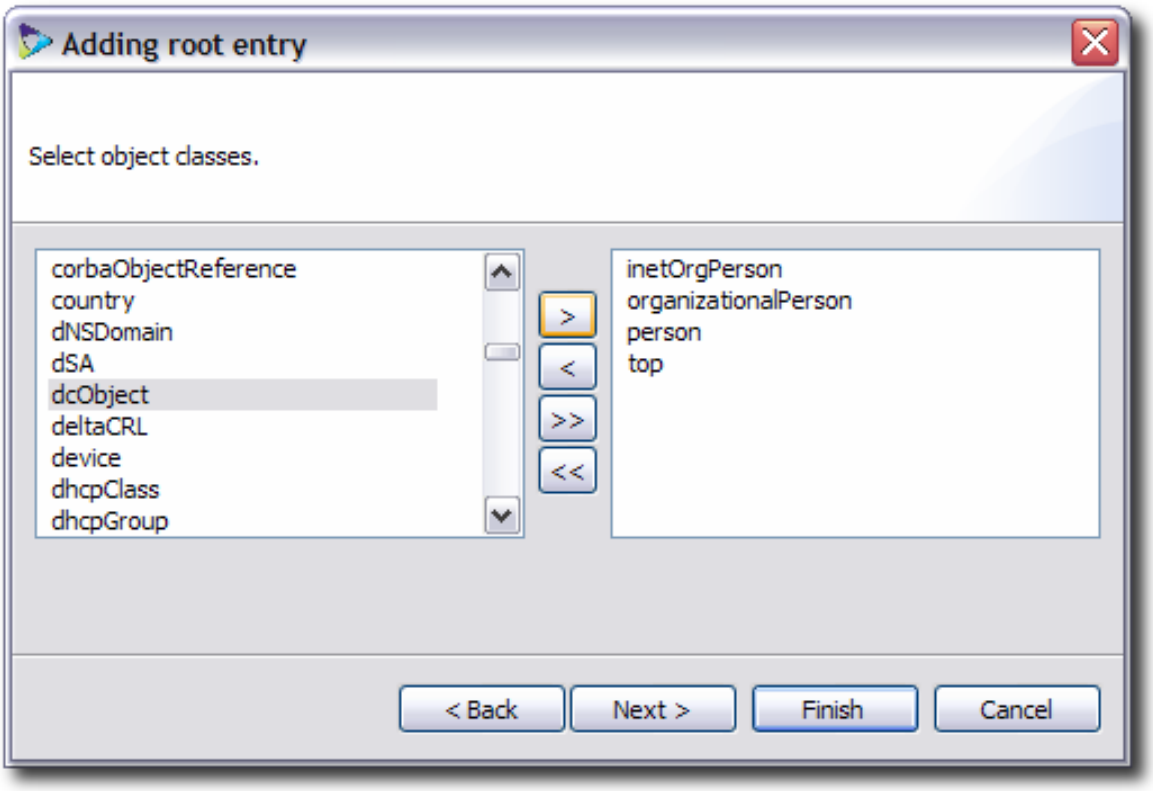
- **`org.safehaus.penrose.directory.Entry.ProxyEntry`**, which preserves the existing structure of an LDAP source, although DNs can be renamed. A proxy entry can only be used with LDAP sources and cannot be used for joining multiple sources.

- **`org.safehaus.penrose.directory.Entry.SchemaEntry`**, which creates a **`cn=subschema`** subtree, as in a regular LDAP server.

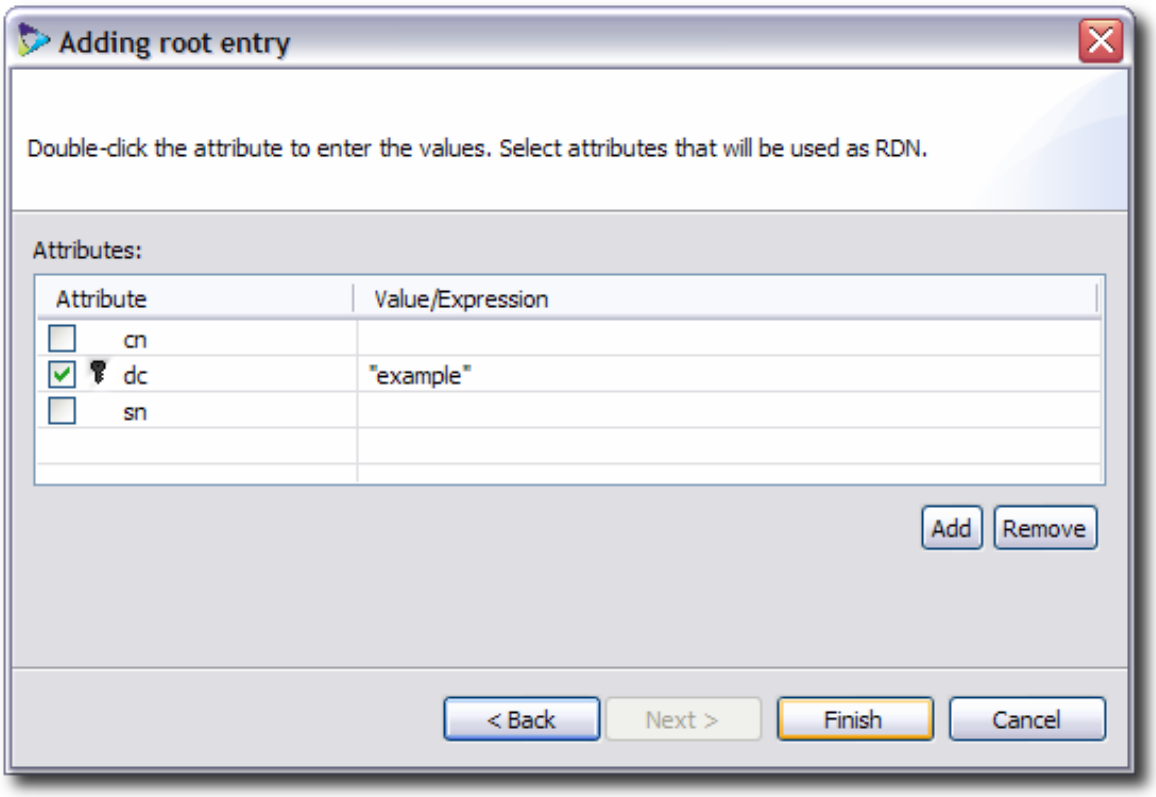
- **`org.safehaus.penrose.directory.Entry.RootEntry`**, which creates a root DSE, a special entry which contains information related to a server instance.

If no class is given, then Penrose Server uses the base class, **`org.safehaus.penrose.directory.Entry`**.

4. Select the object classes for the entries in the virtual directory tree on the left, then click the **>** arrows to move the object class to the list on the right. These object classes are used to define the entries displayed in the virtual directory, which also defines what attributes are allowed and required for the entries and how they are named.



5. Select which attribute will be used as the relative DN (RDN). This is the attribute that appears first in the entry name, also called the *naming attribute*. The selected attributes are *primary keys*, meaning those attributes must be present in the source entry for it to be included in the virtual directory.



The naming attribute determines what the far left component of the name for an entry is. For example, if the naming attribute is *dc*, then the name of an entry could be **`dc=example`**; if it is *ou*, the name could be **`ou=People`**; or if it is *cn*, the name could be **`cn=John Smith`**.

6. Select attributes which will be available to the entries. By default, only the required attributes for the previously-selected object classes are listed. To add more attributes, click the **Add** button. In the window, select the object class and then attributes from its list.[1]

7. Optionally, double-click an attribute, and set rules for the possible attribute values. These are mappings which define the values of the virtual directory entry attributes.

   - *text* is an exact string of what the attribute value should be.

   - *binary* means that the attribute values are binary, such as for the `jpegPhoto`, `objectGID`, or `userCertificate` attributes. When **`binary`** is selected, no value needs to be supplied.

   - *variable* means to use whatever value is in a given attribute in a specific data source. For example, for the *cn* attribute, use the *cn* attribute from the LDAP1 source, **`LDAP1.cn`**.

   - *expression* is a BeanShell script which transforms the attribute value in some way, such as generating it from other attribute values.

[1] It is possible to select an attribute which does not belong to any of the selected object classes for the DN. Penrose Virtual Directory does not enforce or validate schema elements.

8. Click **Finish** at the bottom of the RDN window to save the new root entry.

NOTE

Choosing the object classes and primary keys can automatically create subtrees beneath the root DN. For example:

1. When setting up the root DN, an admin selects the **`dcObject`** object class (for the base DN entry) and also the **`organizationalUnit`** and **`inetorgperson`** object classes.

2. Along with selecting the *dc* attribute as a naming attribute, the admin selects the *ou* attribute and *cn* attribute.

3. The base DN is created. Penrose Virtual Directory also creates a static **`ou=people`** subtree entry and a dynamic **`cn=...`** entry.

When the entry is created, the base DN entry is not configured with any sources. To configure sources, see *Section 8.2.2.2, "Editing the Directory Sources"*.

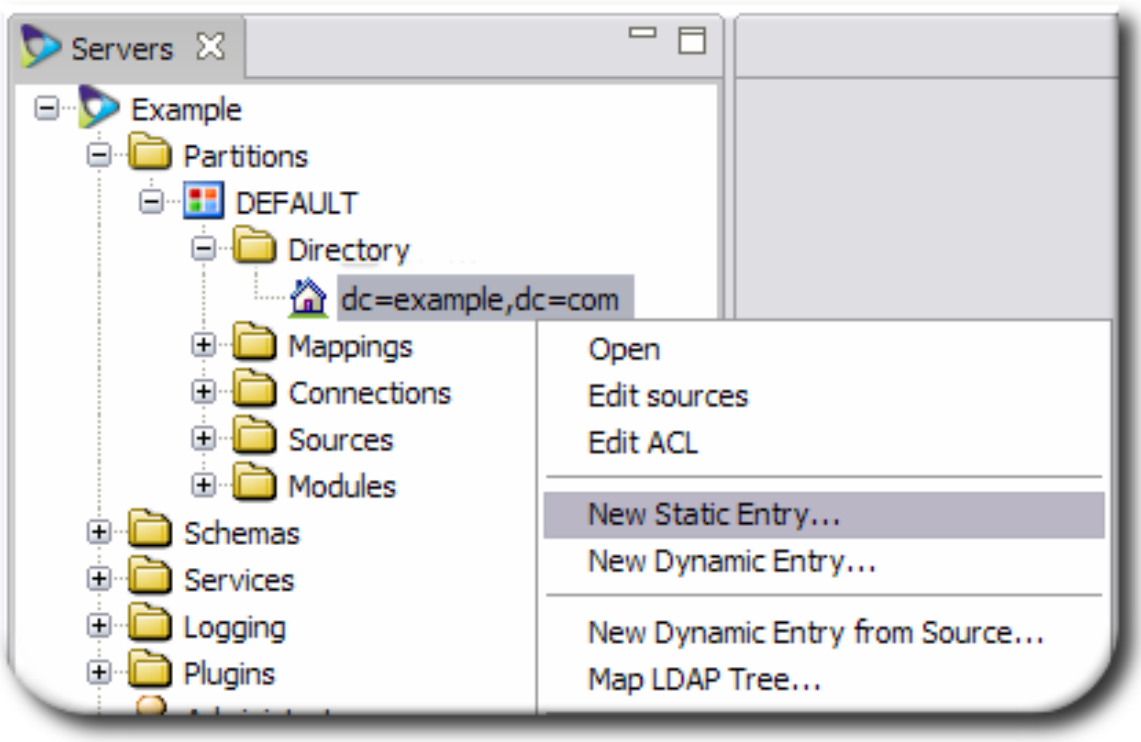
8.2.1.2. Adding a Static Directory Entry

A static entry is for a clearly-defined subtree with constant values, such as **`ou=People`**.

1. In Penrose Studio, open the server entry, then expand the **Partitions** folder.

2. Open the **Directory** folder.

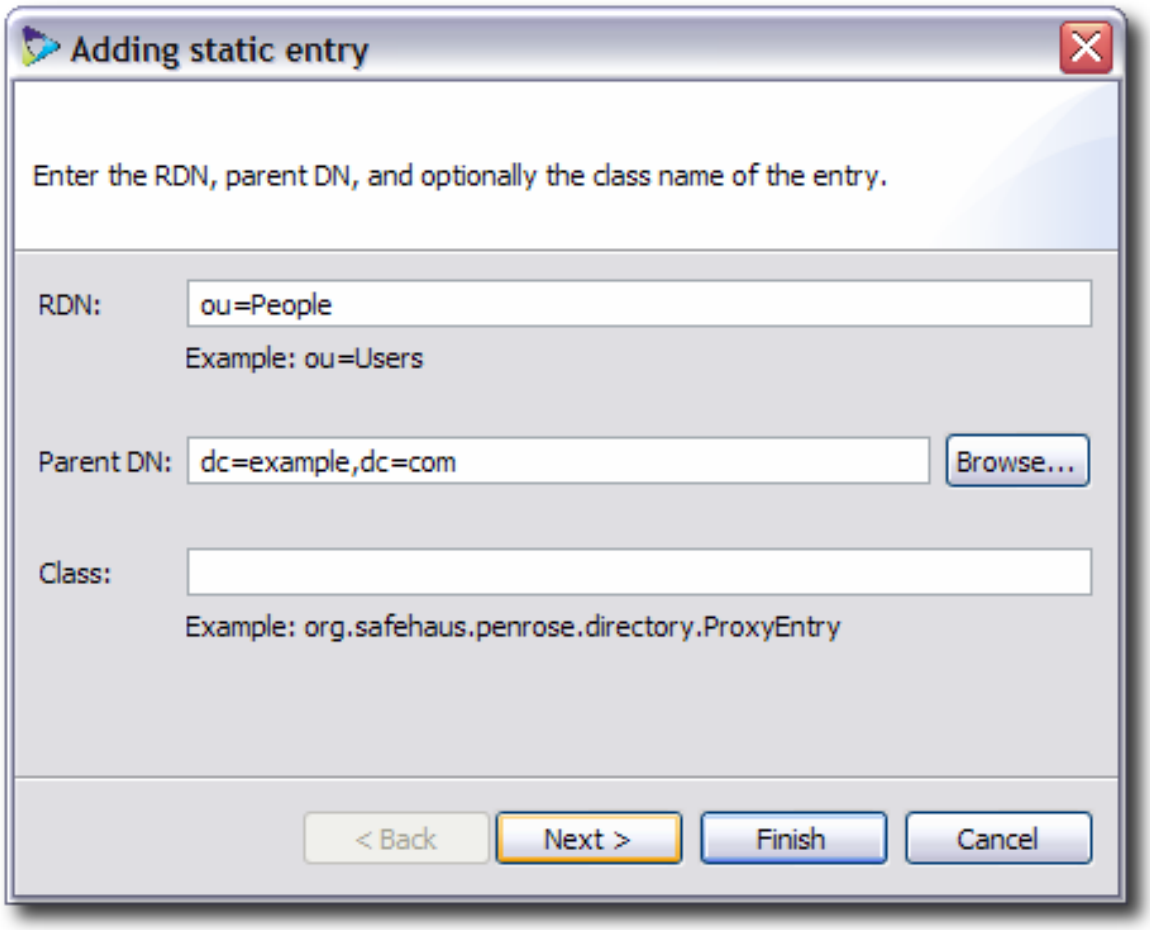
3. Right-click the virtual directory entry under which to create the static subtree, and select the **New Static Entry...** option.



NOTE

A static entry can be created under the root entry, under another static subtree, or under a dynamic subtree, depending on the planned virtual directory tree.

4. Enter the relative distinguished name (RDN) for the subtree entry (the name which is farthest on the left) and the parent DN. For example, if the subtree is named **`ou=People,dc=example,dc=com`**, the RDN is **`ou=People`** and the parent DN is **`dc=example,dc=com`**.



Optionally, supply the path name to a class for the static entry. There are five options:

- **`org.safehaus.penrose.directory.Entry`**, the default class, which generates a directory without referencing an external source.

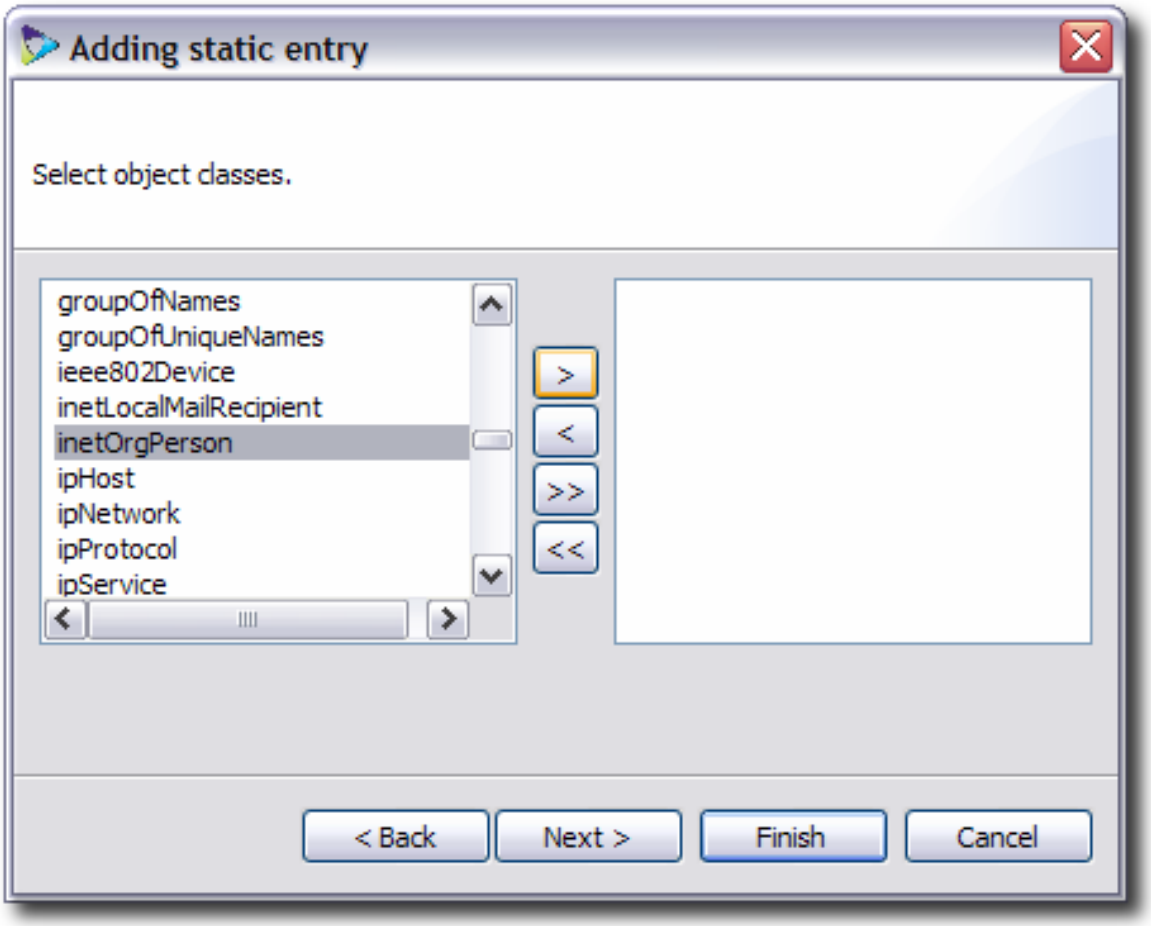
- **`org.safehaus.penrose.directory.Entry.DynamicEntry`**, which creates and joins entries from different external sources. For LDAP servers, this will also flatten the directory namespaces.

- **`org.safehaus.penrose.directory.Entry.ProxyEntry`**, which preserves the existing structure of an LDAP source, although DNs can be renamed. A proxy entry can only be used with LDAP sources and cannot be used for joining multiple sources.

- **`org.safehaus.penrose.directory.Entry.SchemaEntry`**, which creates a **`cn=subschema`** subtree, as in a regular LDAP server.

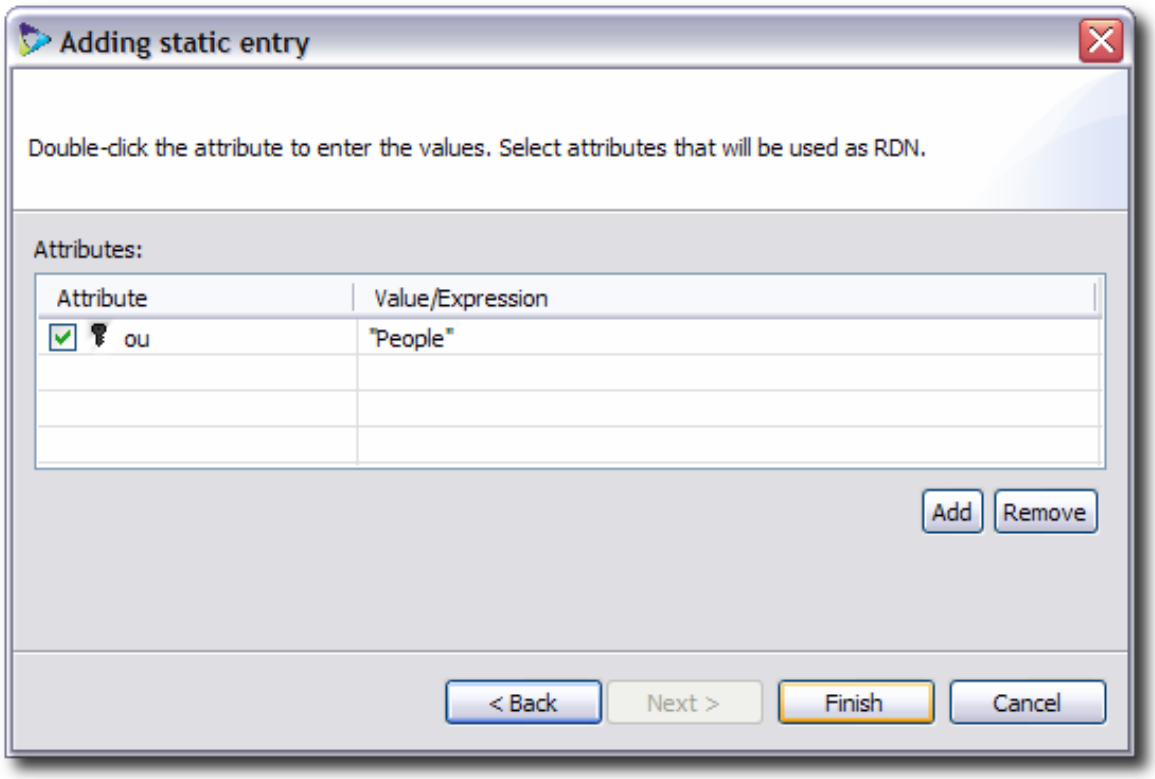
- **`org.safehaus.penrose.directory.Entry.RootEntry`**, which creates a root DSE, a special entry which contains information related to a server instance.

5. Enter the object classes to use or to allow for entries in this subtree. Click the object class in the list on the left, then click the **>** arrows to move the object class to the list on the right.

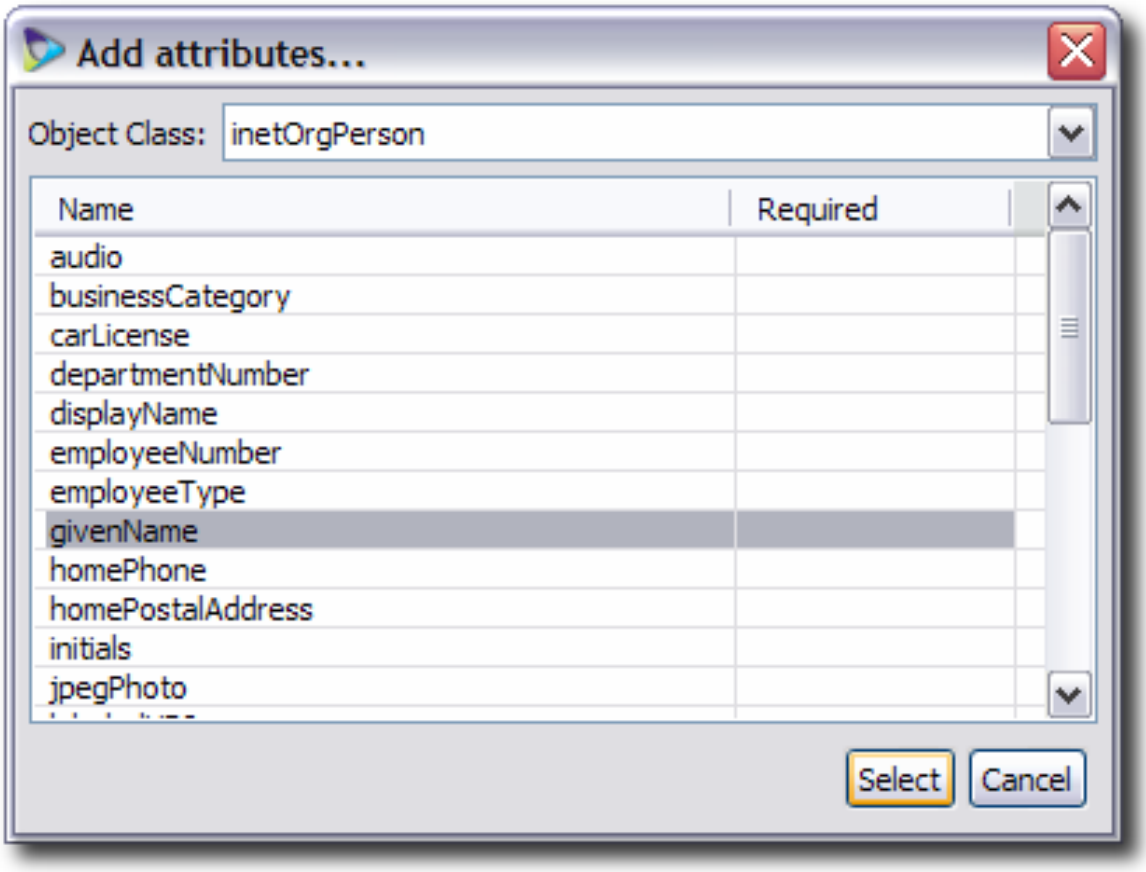


6. Check the checkbox by an attribute to set the relative DN (RDN). This is the attribute that appears first in the entry name, also called the *naming attribute*. The selected attributes are *primary keys*, meaning those attributes must be present in the source entry for it to be included in the virtual directory.

   The naming attribute determines what the far left component of the name for an entry is. For example, if the naming attribute is *dc*, then the name of an entry could be **`dc=example`**; if it is *ou*, the name could be **`ou=People`**; or if it is *cn*, the name could be **`cn=John Smith`**.



7. Select attributes which will be available to the entries. By default, only the required attributes for the previously-selected object classes are listed. To add more attributes, click the **Add** button. In the window, select the object class and then attributes from its list.1



8. Optionally, double-click an attribute, and set rules for the possible attribute values. These are mappings which define the values of the virtual directory entry attributes.

   The attribute value can be any of the following:

   - *text*, an exact string of what the attribute value should be.
   - *binary* means that the attribute values are binary, such as for the `jpegPhoto`, `objectGID`, or `userCertificate` attributes. When **`binary`** is selected, no value needs to be supplied.
   - *variable*, using whatever value is in a given attribute in a specific data source. For example, for the *cn* attribute, use the *cn* attribute from the LDAP1 source, **`LDAP1.cn`**.
   - *expression*, a BeanShell script which transforms the attribute value in some way, such as generating it from other attribute values.

9. Click the **Finish** button at the bottom of the RDN window to save the new static subtree.

When the entry is created, the static entry is not configured with any sources. To configure sources,

see *Section 8.2.2.2, "Editing the Directory Sources"*.

8.2.1.3. Adding a Dynamic Directory Entry

A dynamic DN is used for RDNs which are constantly changing or which are numerous, such as user entries. The RDNs which are contained in a dynamic subtree are determined by the actual data being processed.

1. In Penrose Studio, open the server entry, then expand the **Partitions** folder.

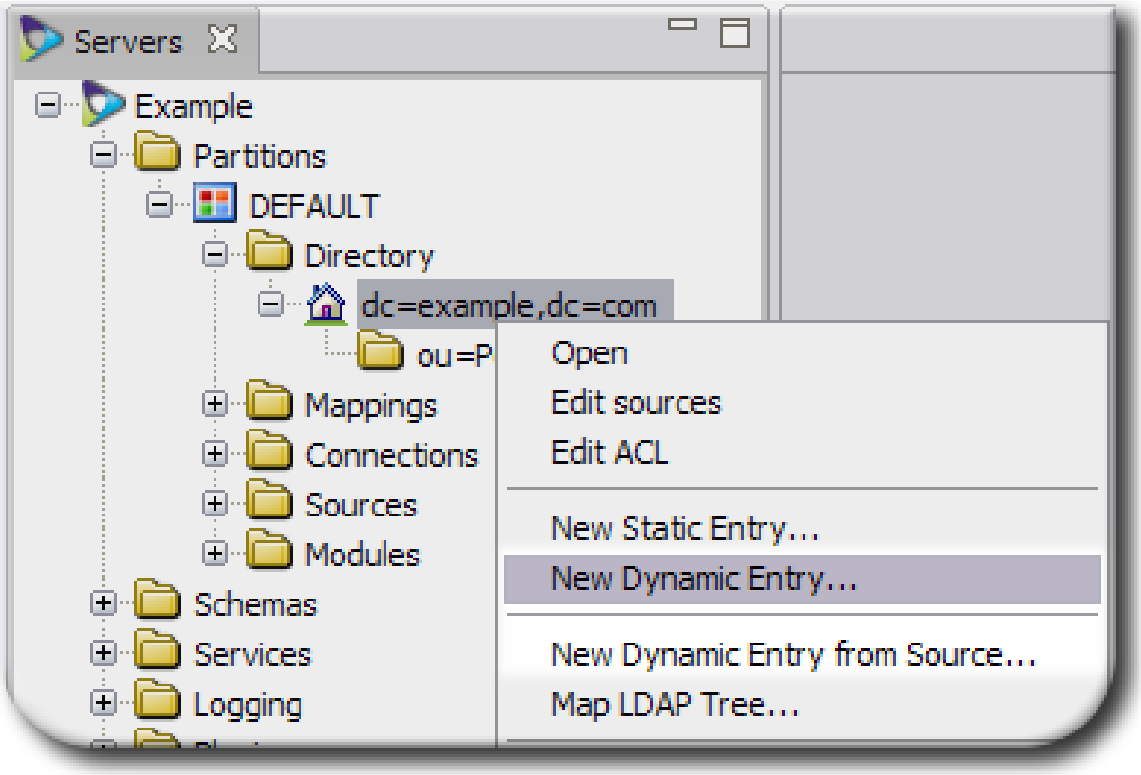
2. Open the **Directory** folder.

3. Right-click the directory entry under which to create the dynamic entry, and select the **New Dynamic Entry...** option.

   A dynamic entry can be created under the root entry, under a static subtree, or under another dynamic subtree, depending on the planned virtual directory tree. More probably, this will go under a static entry, such as putting both directory and database user entries under a single **`ou=People`** subtree.

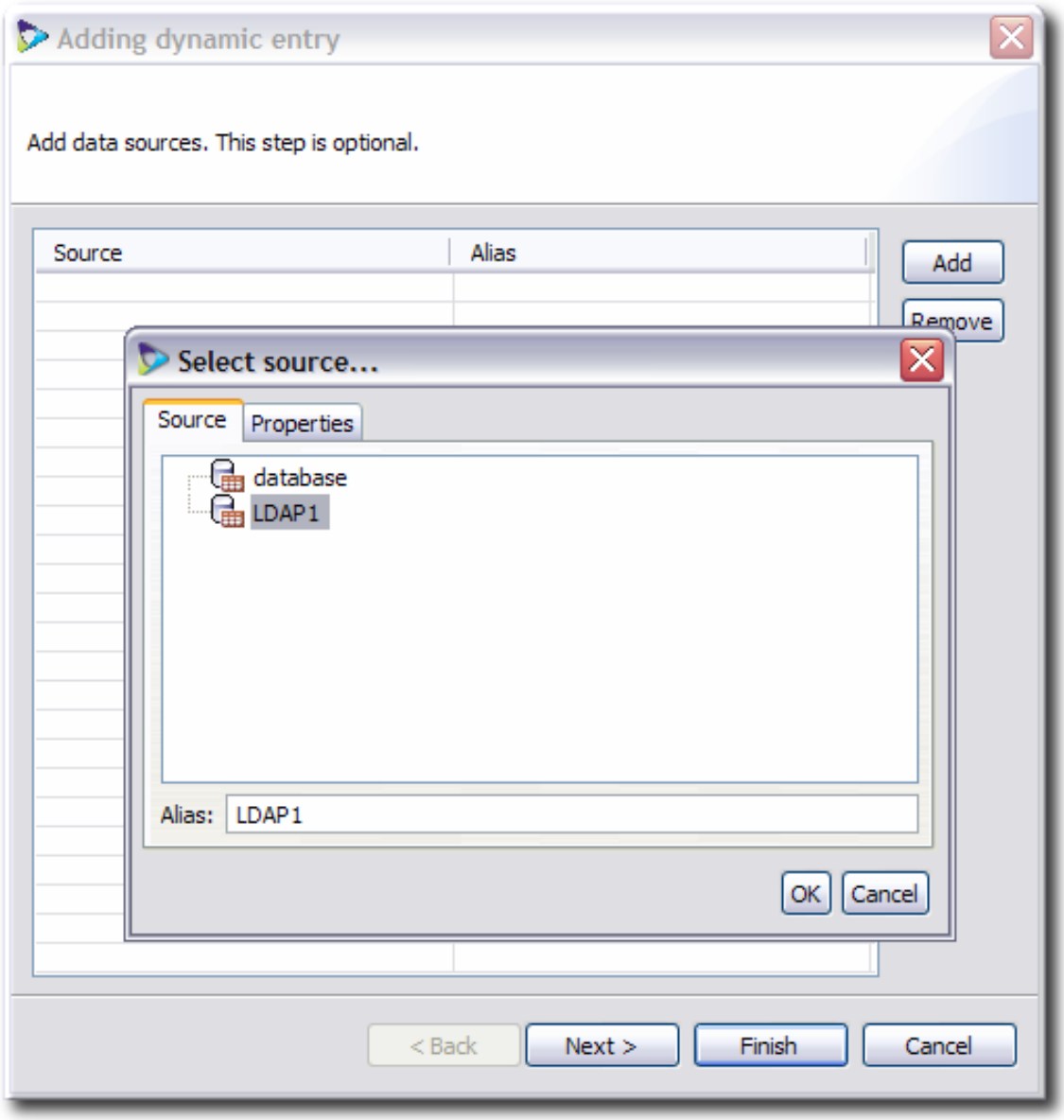
NOTE

Alternatively, select **New Dynamic Entry from Source...**. This automatically creates a subtree based on the configuration of the selected source, such as using the configured object classes and using the source's primary key for the RDN, and skips source configuration steps.

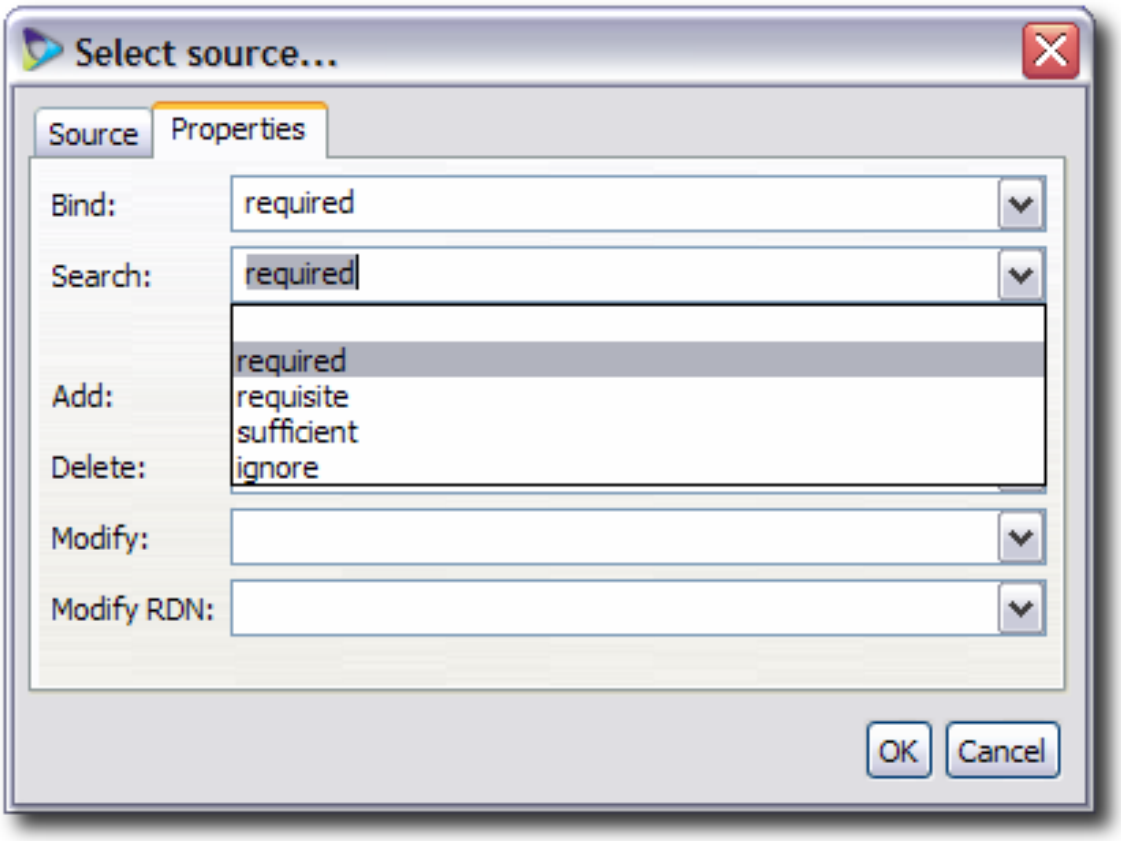


4. Click the **Add** to add data sources for the virtual tree; there can be more than one source for the tree.

   a. Select the source to use from the list in the **Sources** tab.



b. Click the **Properties** tab, and select the required way to handle the source for the different operations performed against the virtual directory, such as bind, search, and modify.



The properties for a bind operation can be any of the following:

- *required*, which means that the users must bind against this source to connect to the virtual directory.

- *sufficient*, which means that the user can be authenticated against any of the allowed (**`sufficient`**) sources.

- *ignore*, which means that Penrose Virtual Directory will not use that source for authentication.

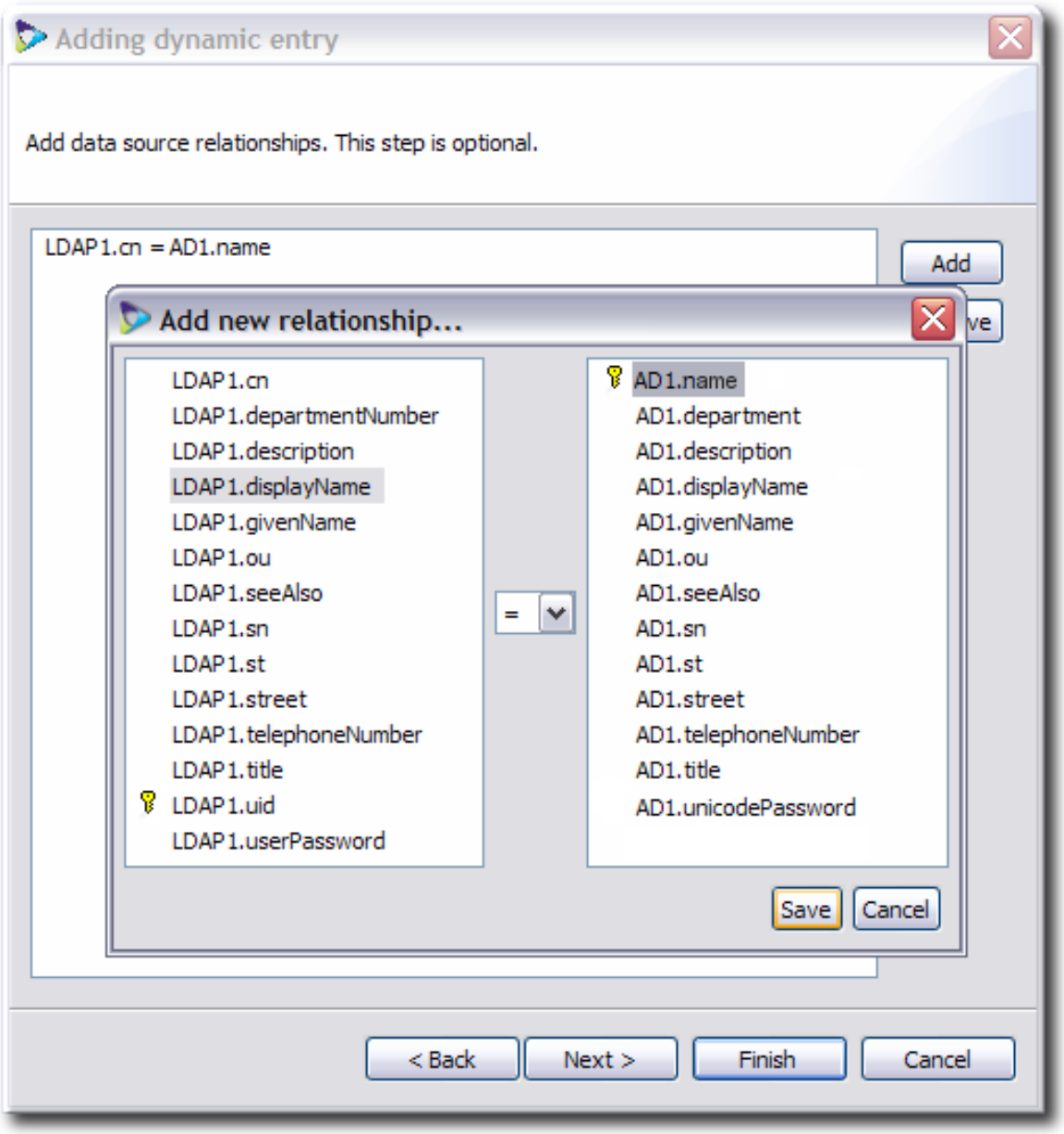
All other LDAP operations have the following options:

- *required*, which means that the source must have the identity present for the entry to exist in the virtual directory. This is the default.

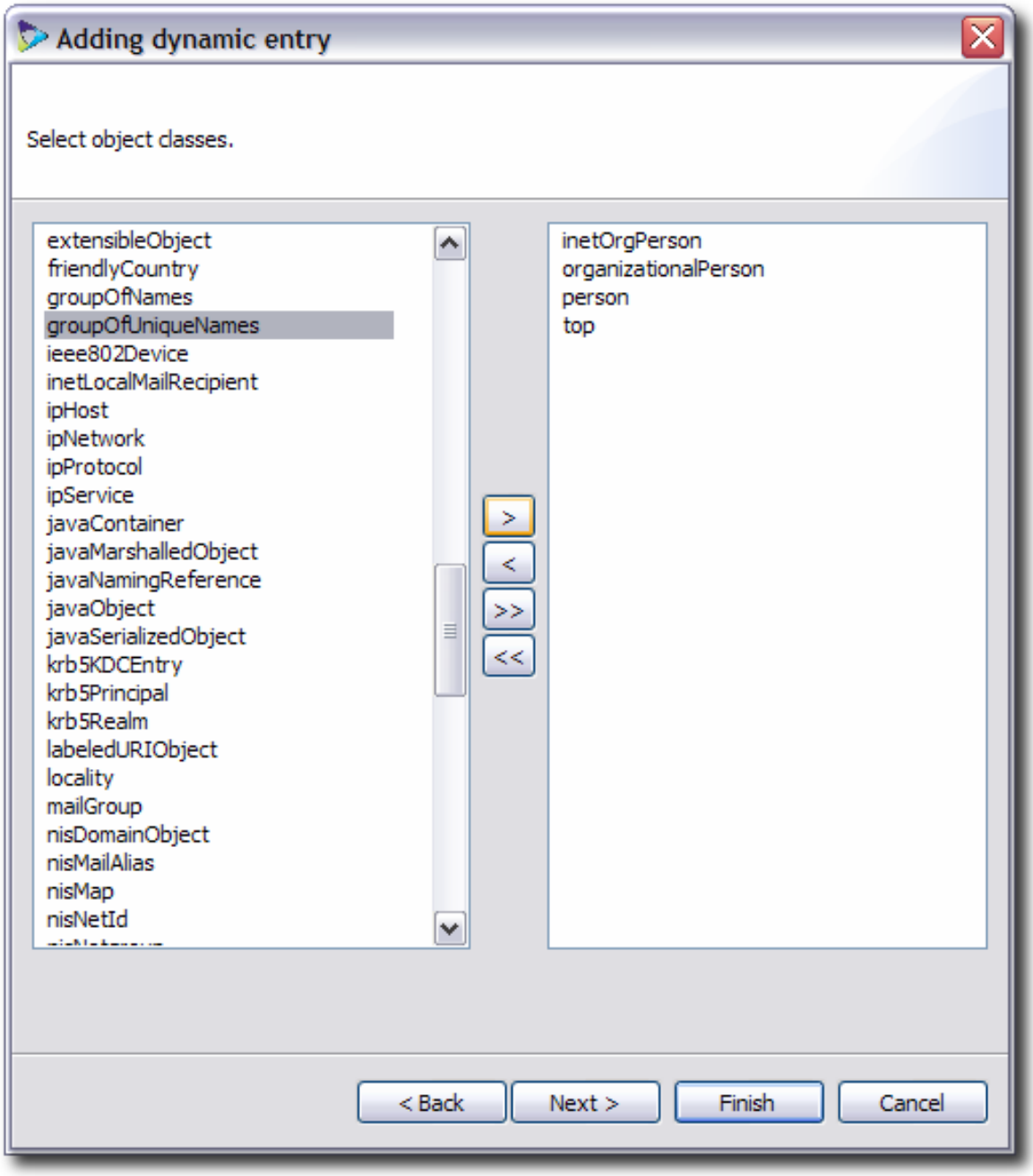
- *optional*, which means that Penrose Virtual Directory will create a virtual entry out of all available sources, even if the specific source does not have the identity.

- *ignored*, which means that Penrose Virtual Directory will not used this source to perform that specific operation.

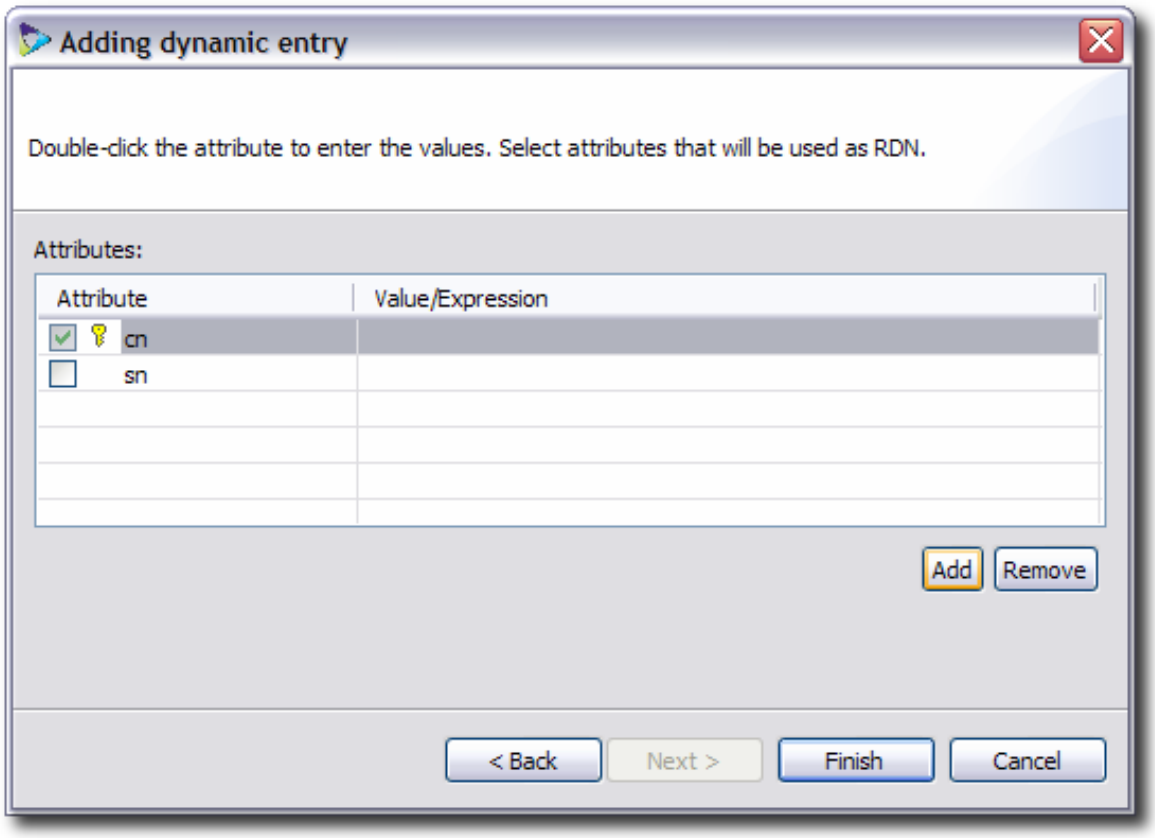
5. If there is more than one data source selected for the directory, map the relationships between the different source entry attributes. For example, link the *cn* attribute in a Red Hat Directory Server source to the *name* attribute in a Microsoft Active Directory source.



6. Select the object classes for the entries in the virtual directory subtree on the left, then click the **>** arrows to move the object class to the list on the right. These object classes are used to define the entries displayed in the virtual directory, which also defines what attributes are allowed and required for the entries and how they are named.

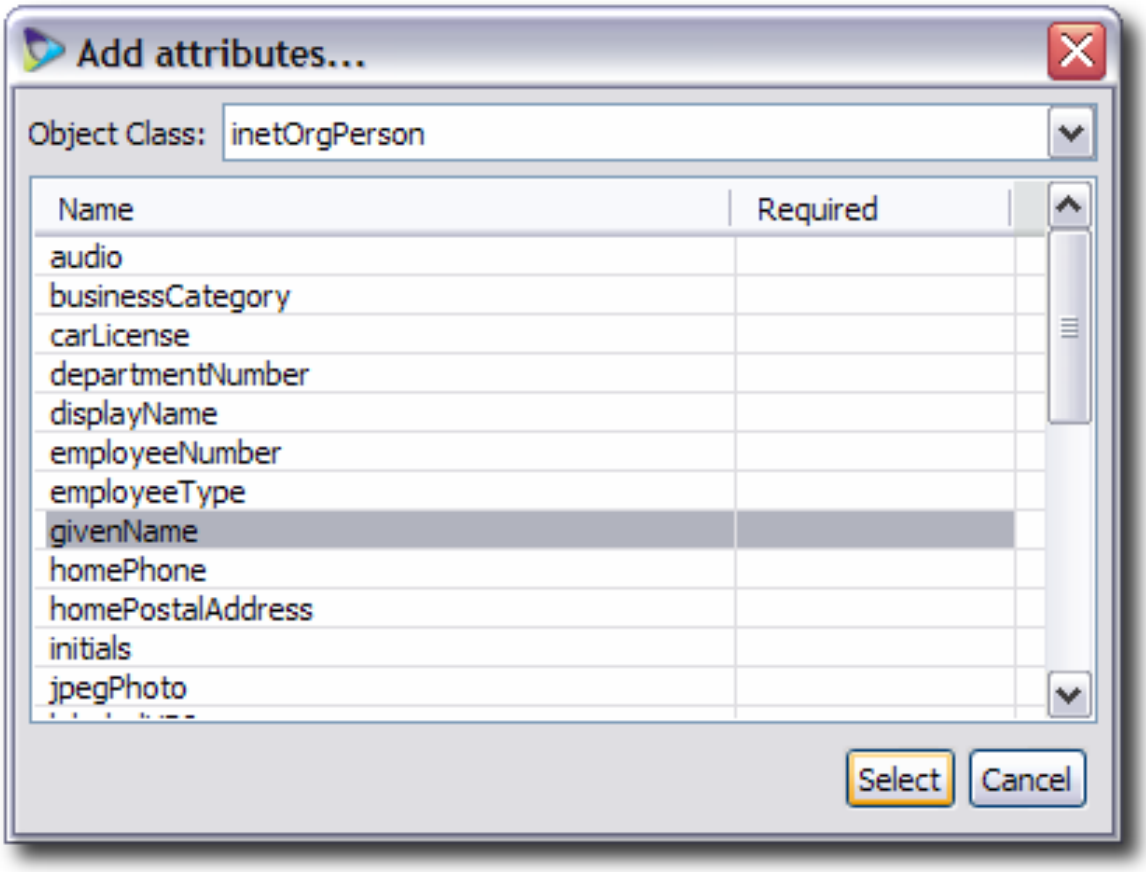


7. Check the checkbox by an attribute to set the relative DN (RDN). This is the attribute that appears first in the entry name, also called the *naming attribute*. The selected attributes are *primary keys*, meaning those attributes must be present in the source entry for it to be included in the virtual directory.

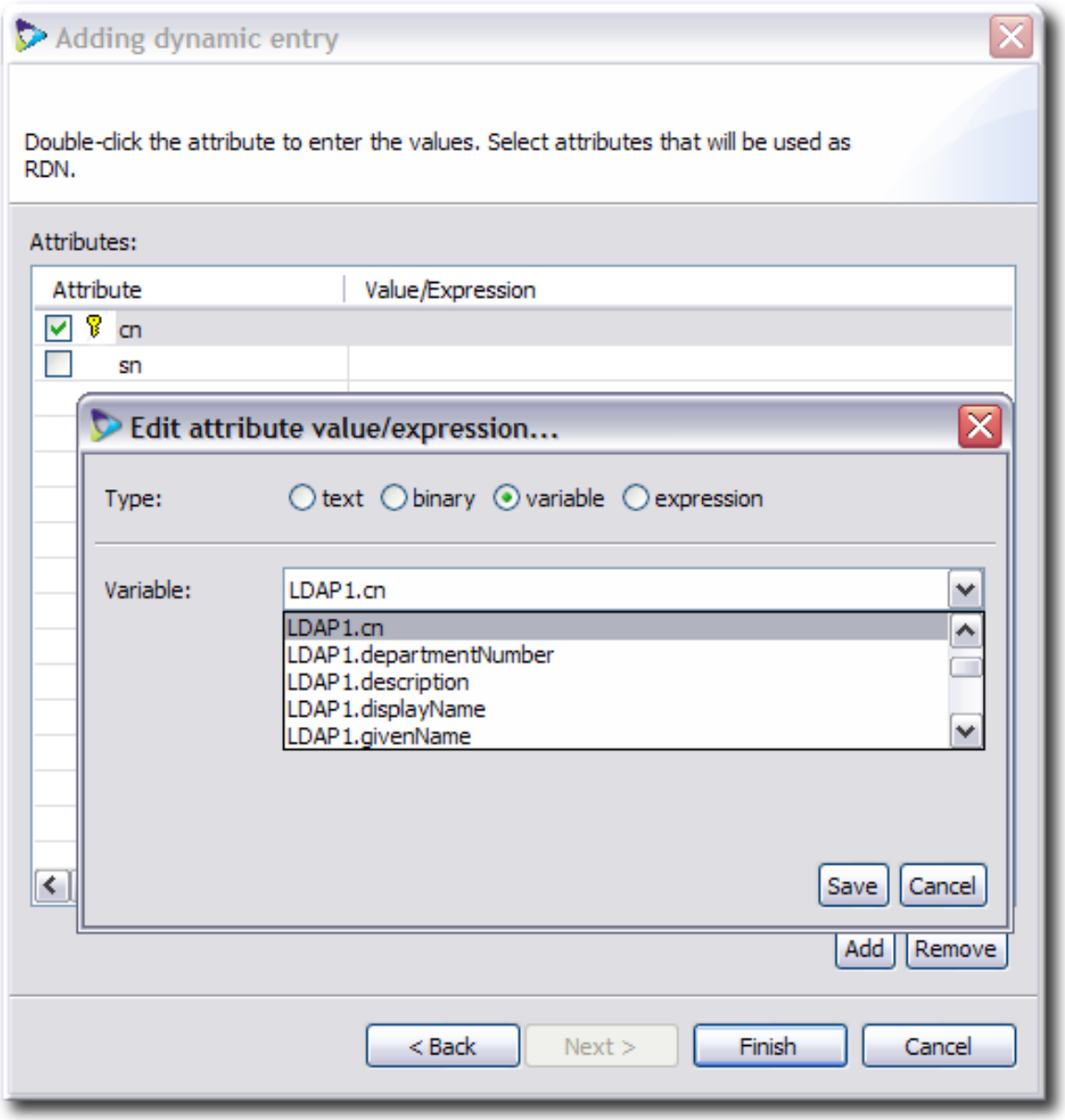


The naming attribute determines what entry attribute is when entries are displayed in the tree. For example, if the naming attribute is *dc*, then the name of an entry could be **`dc=example`**; if it is *ou*, the name could be **`ou=People`**; or if it is *cn*, the name could be **`cn=John Smith`**.

8. Select attributes which will be available to the entries. By default, only the required attributes for the previously-selected object classes are listed. To add more attributes, click the **Add** button. In the window, select the object class and then attributes from its list.[1]



9. Optionally, double-click an attribute, and set rules for the possible attribute values. These are mappings which define the values of the virtual directory entry attributes.



- *text* is an exact string of what the attribute value should be.
- *binary* means that the attribute values are binary, such as for the `jpegPhoto`, `objectGID`, or `userCertificate` attributes. When **`binary`** is selected, no value needs to be supplied.
- *variable* means to use whatever value is in a given attribute in a specific data source. For example, for the *cn* attribute, use the *cn* attribute from the LDAP1 source, **`LDAP1.cn`**.
- *expression* is a BeanShell script which transforms the attribute value in some way, such as generating it from other attribute values.

10. Click the **Finish** button at the bottom of the attributes window to save the new dynamic entry.

8.2.2. Editing the Virtual Directory in Penrose Studio

Editing a virtual directory's base DN or a static or dynamic subtree can edit the sources and attributes properties that the entry was created with.

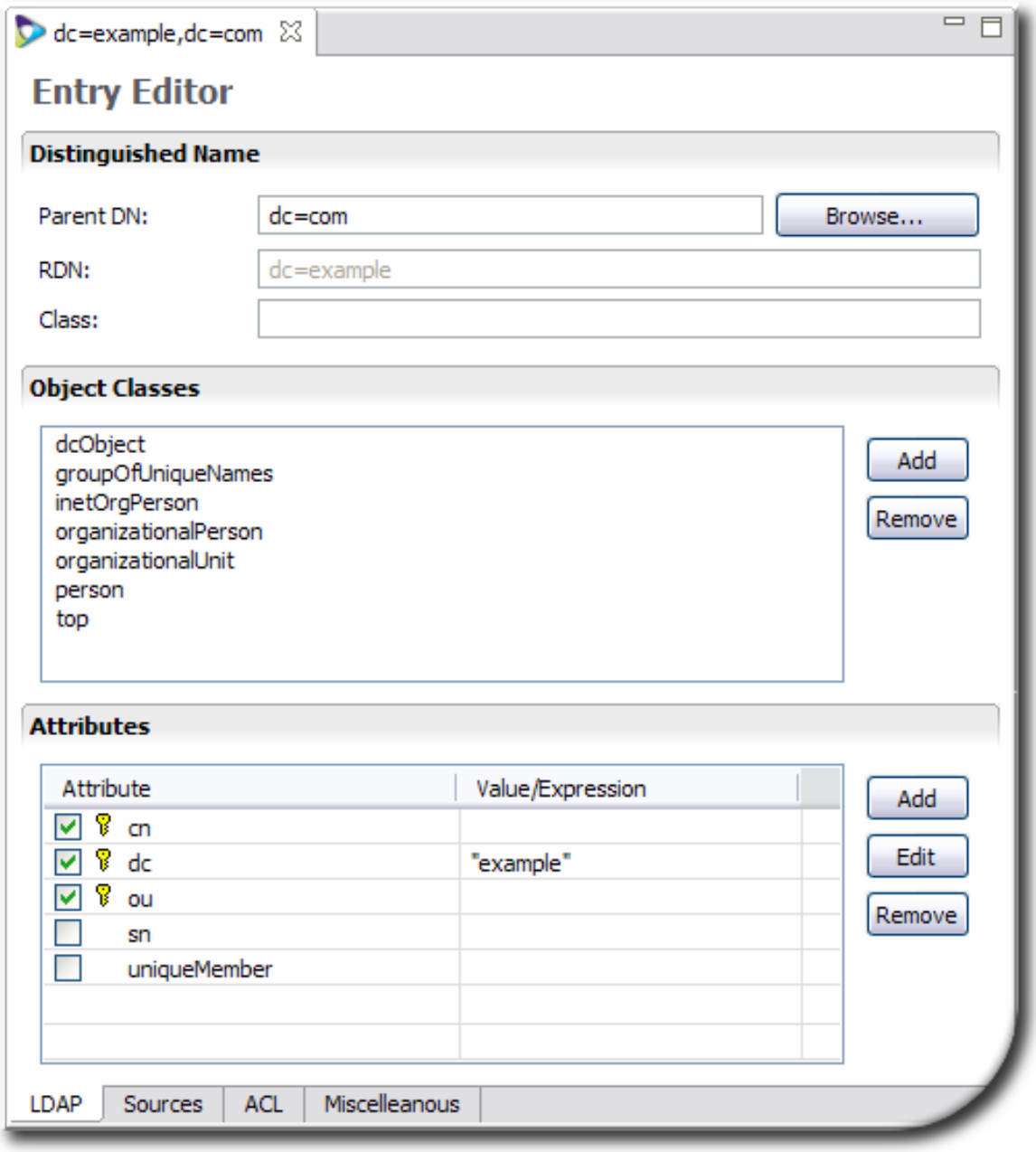
8.2.2.1. Editing the Directory Configuration

1. In Penrose Studio, open the server entry, then expand the **Partitions** folder.

2. Open the **Directory** folder.

3. Double-click the directory entry to edit.

4. Edit the virtual directory configuration in the **LDAP** tab.



The virtual tree can be edited in several ways:

- Changing the parent DN or the class associated with the entry.
- Changing the object classes assigned to the entry.
- Adding or removing attributes, creating mappings to sources, and setting naming attributes

5. Close the window and save the changes when prompted.

8.2.2.2. Editing the Directory Sources

NOTE

Dynamic entries automatically have sources configured for them, but base DN and static entries do not, so sources need to be added by editing the entry.

1. In Penrose Studio, open the server entry, then expand the **Partitions** folder.

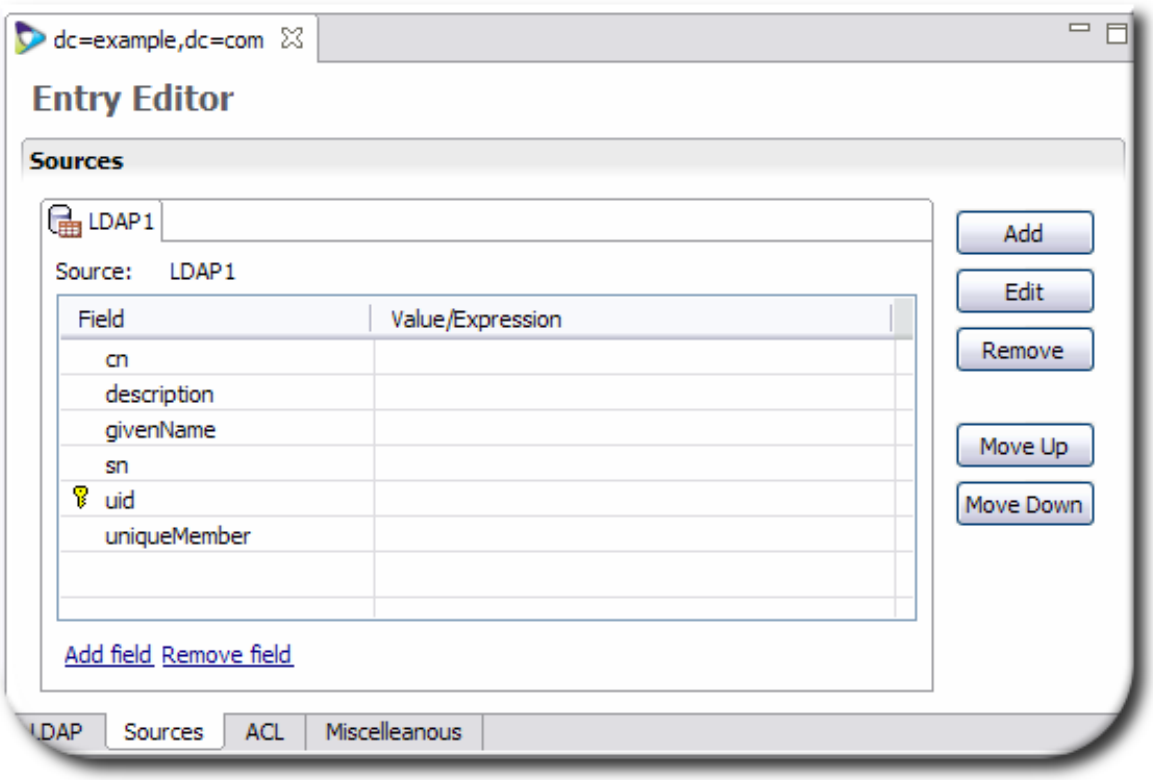
2. Open the **Directory** folder.

3. Double-click the directory entry to edit.

   Alternatively, right-click the subtree entry and select **Edit Sources**.

4. Click the **Sources** tab.

5. Add new sources or edit existing sources.



Additional fields or mappings can be added or edited to the sources, as in *Section 9.3, "Configuring Basic Mapping"*, and new sources can be created, as described in *Section 7.2, "Configuring*

Sources in Penrose Studio".

6. Close the window and save the changes when prompted.

8.2.3. Configuring the Virtual Directory Manually

There is a single directory file. Each root DN or subtree within the partition is added as an entry in this file. This is **`directory.xml`**, which is located in **`/opt/vd-server-2.0/conf`** for the default partition and in **`/opt/vd-server-2.0/partitions/`***partition_name*`/DIR-INF`** for additional partitions. This file is illustrated in *Example 8.1, "Annotated directory.xml File"*.

```
 <directory>  main file tag
  <entry dn="...">  the subtree dn
    <oc>...</oc>  an object class for the subtree entry
    <at name="..." rdn="...">  an attribute with the subtree
entry
      <mapping_type>...</mapping_type>  the mapping for the attribute
    </at>

    <aci>   an access control instruction for the subtree
      <permission>...</permission>
    </aci>

    <source>  a source entry to use with the subtree
      <source-name>...</source-name>  the source name, as it is in the
Penrose Virtual Directory source entry
      <field name="...">  a reverse mapping entry for the source attrib\
ute
        <mapping_type>...</mapping_type>  the mapping for the attribute

      </field>
    </source>
  </entry>
 </directory>
```

Example 8.1. Annotated directory.xml File

Not every parameter in *Example 8.1, "Annotated directory.xml File"* is required simply to create the entry. For example, mapping attributes requires the **`<field>`** parameter, but the entry can exist without the mapping. Likewise, a dynamic subtree requires a **`<source>`** definition, but a static one does not.

To create a subtree, add a new entry to the **`directory.xml`** file. To edit a subtree, add, remove, or edit parameters within the entry. *Example 8.2, "Example directory.xml File"* shows three different subtree entries for a base DN, static subtree, and dynamic subtree.

IMPORTANT

Always restart Penrose Server after editing the configuration file. For example:

```
service vd-server restart
```

```
<directory>
base DN entry
  <entry dn="dc=example,dc=com">
    <oc>dcObject</oc>
    <oc>groupOfUniqueNames</oc>
    <oc>inetOrgPerson</oc>
    <oc>organizationalPerson</oc>
    <oc>organizationalUnit</oc>
    <oc>person</oc>
    <oc>top</oc>
    <at name="dc" rdn="true">
      <constant>example</constant>
    </at>
    <aci>
      <permission>rs</permission>
    </aci>
  </entry>

static subtree entry
  <entry dn="ou=people,dc=example,dc=com">
    <oc>organizationalUnit</oc>
    <oc>top</oc>
    <at name="ou" rdn="true">
      <variable>LDAP1.ou</variable>
      <expression var="person">source_name.attribute</expression>
    </at>
  </entry>

dynamic subtree entry
  <entry dn="cn=...,ou=people,dc=example,dc=com">
    <oc>inetOrgPerson</oc>
    <oc>organizationalPerson</oc>
    <oc>person</oc>
    <oc>top</oc>
    <at name="cn" rdn="true">
      <expression var="person">cn=*</expression>
    </at>
    <source alias="LDAP1" search="required" bind="required"
add="required" delete="required" modify="required" modrdn="required">
      <source-name>LDAP1</source-name>
      <field name="ou">
        <constant>example</constant>
      </field>
    </source>
  </entry>

</directory>
```

Example 8.2. Example directory.xml File

The parameters available for any kind of Penrose Virtual Directory subtree entry are listed in *Table 8.1, "Parameters for Subtree Entries"*.

A base DN entry must contain enough information to define the DN, required object classes and attributes for the virtual directory tree, and the ACIs for the entry:

- An **<entry>** tag with the `dn="..."` argument
- An **<oc>** tag
- An **<at>** tag with the `name="..."` argument and some kind of mapping sub-tag, identified as *<mapping_type>* in *Example 8.2, "Example directory.xml File"*
- An **<aci>** tag with a **<permission>** sub-tag to define directory access (more specific access control is possible, as described in *Section 8.6.4, "Setting Access Controls Manually"*)

A static entry has similar definition requirements as the base DN, only it does not have an ACI by default:

- An **<entry>** tag with the `dn="..."` argument
- An **<oc>** tag
- An **<at>** tag with the `name="..."` argument

A dynamic entry has the most complex definition by default, allowing additional information about the source mapping:

- An **<entry>** tag with the `dn="..."` argument
- An **<oc>** tag
- An **<at>** tag with the `name="..."` argument
- A **<source>** entry with an optional `alias="..."` argument to give a nickname to the source and with optional tags for different LDAP operations.
- **<source-name>** subtag which identifies an existing Penrose Server source
- **<field>** subtags which define reverse mappings for the source attributes, and <*mapping_type*> tags defining the type of linking

Tag or Parameter	Description	Example
<entry>	Opens and closes the entire subtree entry. There is one parameter for the **<entry>** tag, *dn*.	
dn="..."	Gives the full distinguished name for the entry. For the base DN, this must be explicitly defined.	`dn="dc=example,dc=com"`
fetch="true|false"	Retrieves the entry from the sources before running any operations on the entry, such as **modify** or **add**. This can slow down operations.	`<entry dn="dc=example,dc=com" fetch="true">`
<oc>	Contains an object class defined for the base DN. There can be more than one **<oc>**, for each object class and superior object class defined for the entry.	`<oc>dcObject</oc>` `<oc>top</oc>`

<at>	Contains an attribute defined for the base DN. There can be more than one **<at>** line. An **<at>** tag can have two arguments, `name=` and `rdn=`. **<at>** also contains a sub-tag which defines the mapping type for the virtual attribute to the source attribute: constant, variable, or expression.	`<at name="dc" rdn="true">`
		`<constant>example</constant>`
		`</at>`

name="*attribute*"	Contains the LDAP attribute which is allowed for the entry.	`<at name="cn" rdn="true">`
rdn="true|false"	Identifies a naming attribute or *primary key*. If this argument is missing, the default value is false.	`<at name="cn" rdn="true">`
<*mapping_type*>	These are any of three sub-tags which are used with a **<at>** or **<field>** tag to define the method of mapping attributes. There are three options: • <constant> (specific text value) • <variable> (link to the source and attribute in the format	`<variable>LDAP1.cn</variable>`

Tag or Parameter	Description	Example
	source_name.attribute) • <expression> (BeanShell script to generate or extract the value)	

<aci>	Defines the access control instructions to the virtual directory subtree. By default, this contains a **<permission>** subtag, but the ACI can also define a target DN, target attributes, the scope, and whether to grant or deny access. This is explained more in *Section 8.6.4, "Setting Access Controls Manually"*.	`<aci>`
		`<permission>rs</permission>`
		`</aci>`

<source>	Contains the source entry definition for the directory.	
ldap_operation="..."	Optional argument for the **<source>** tag which indicates what authentication to require to connect to the source for different LDAP operations. The properties for a bind operation can be any of the following: • *required*, which means that the users must bind against this source to connect to the virtual directory. • *sufficient*, which means that the user can be authenticated against any of the allowed (**sufficient**) sources. • *ignore*, which means that Penrose Virtual Directory will not use that source for authentication. All other LDAP operations have the following options: • *required*, which means that the source must have the	`<source` **`bind="required"`**

Tag or Parameter	Description	Example
	identity present for the entry to exist in the virtual directory. This is the default. • *optional*, which means that Penrose Virtual Directory will create a virtual entry out of all available sources, even if the specific source does not have the identity. • *ignored*, which means that Penrose Virtual Directory will not used this source to perform that specific operation.	
alias="..."	Optional argument for the **<source>** tag which contains a nickname for the source.	`<source ali\ as="DBnickname">`
<source-name>	Contains the name of the source. This source must already be configured; for instructions, see *Chapter 7, Configuring Data Sources*.	`<source-name>LDAP1</source-name>`
<field>	Defines a reverse mapping from a source attribute to a virtual entry attribute. Like the **<at>** mapping, this has a name> parameter to identify the virtual entry's LDAP attribute and a sub-tag which defines the mapping from the source attribute to the virtual attribute: constant, variable, or expression.	`<field> <variable>cn</variable> </field>`

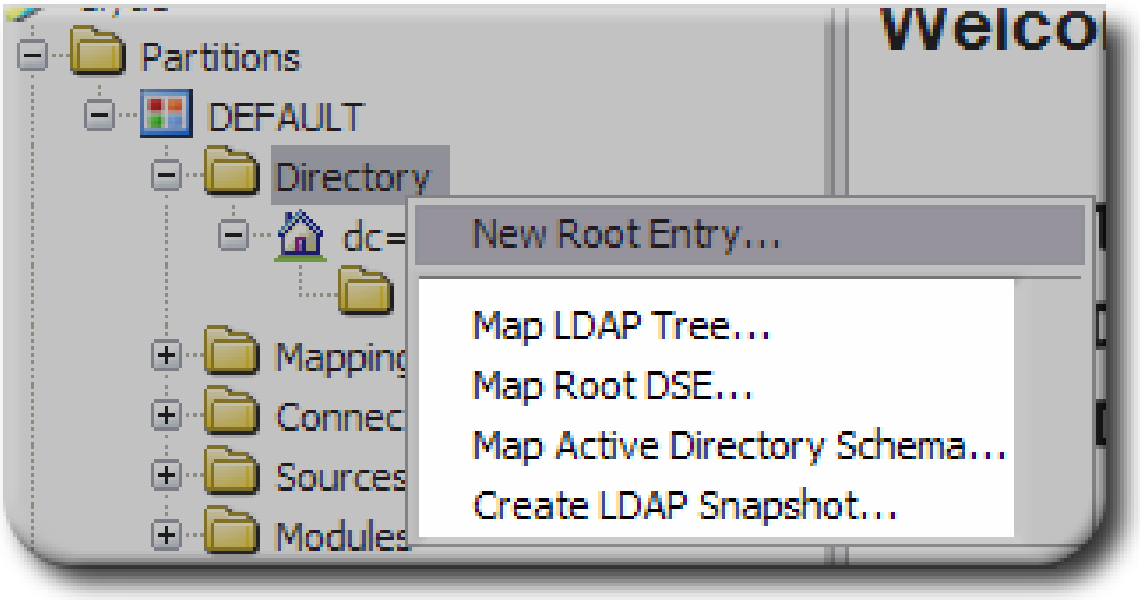
Table 8.1. Parameters for Subtree Entries

8.3. Creating Special Directory Entries

In some virtual directory configurations or for identity federation, it can be necessary to have operational entries defined for the directory, such as a root DSE or **cn=subschema**.

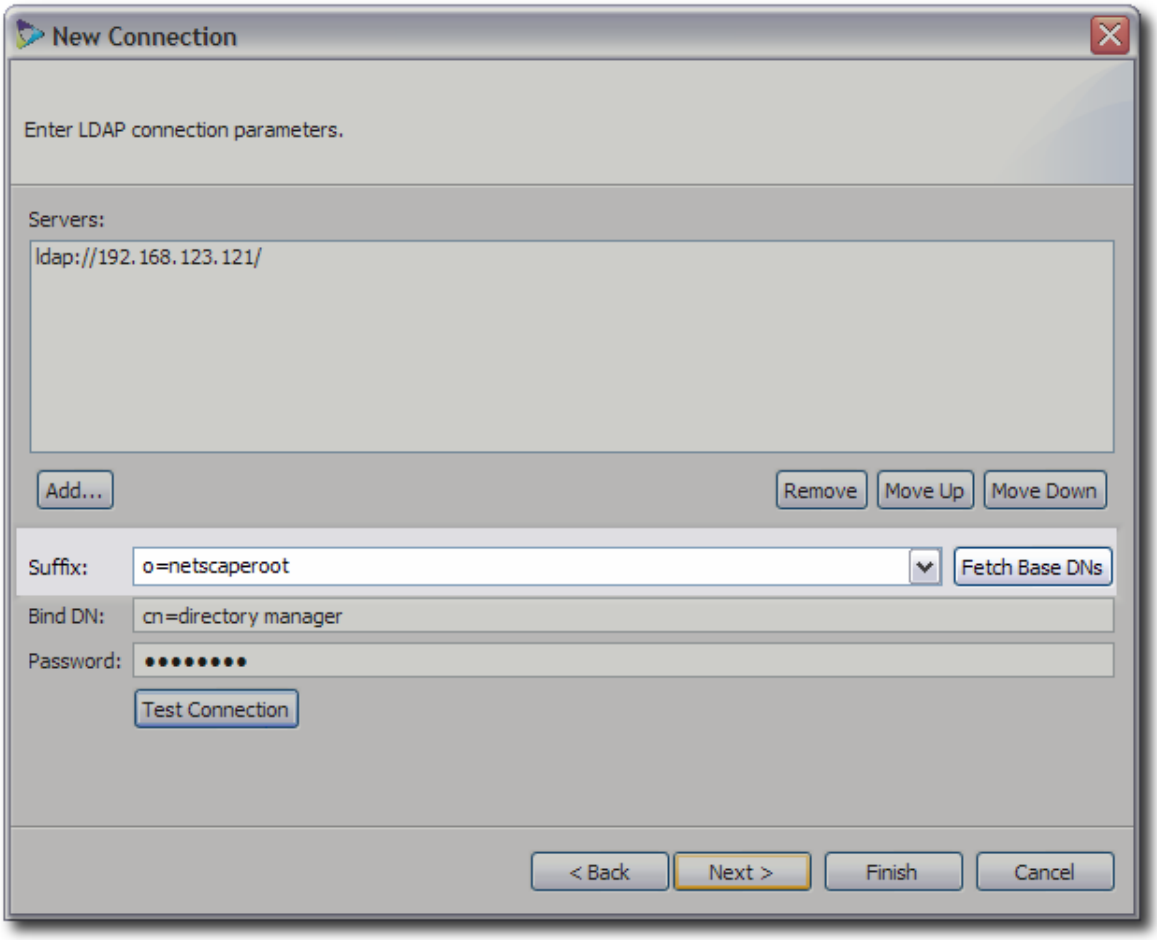
8.4. Duplicating Existing LDAP Servers

Penrose Virtual Directory provides several ways to duplicate, exactly, an existing LDAP directory source. This can be an effective way to proxy LDAP services or to mimic an existing LDAP structure in the virtual directory hierarchy.



Penrose Server can duplicate an existing directory's subtree or entire DIT. It is also possible to duplicate other definitions for the directory, such as its root directory entry (which identifies the instance) and its schema. Duplicating these special configuration entries makes the Penrose Virtual Directory appear to clients as the existing LDAP source.

Because these mappings duplicate the *configuration* of the source directory, they differ from other partitions or subtrees which connect with the user databases of the LDAP server. In this case, the mapping copies the configuration database, such as **`o=NetscapeRoot`** in Red Hat Directory Server, as the base DN.



8.4.1. Mapping an LDAP Tree

Mapping an LDAP tree copies the subtree or entire directory in the existing LDAP source and duplicates it in the virtua directory.

8.4.2. Mapping the Root DSE

The *root DSE* is a special entry which defines the directory server instance, hence the name *directory server entry*. This special entry defines the supported types of LDAP protocols, supported password and authentication schemes, the vendor name and version number, and other features of the instance. Duplicating the root DSE of an existing server means that LDAP clients will treat Penrose Virtual Directory as if they were talking to the original LDAP source directly. This is one way of using Penrose Virtual Directory as a proxy for the other server.

8.4.3. Mapping Active Directory Schema

By default, Penrose Virtual Directory's schema configuration is based on OpenDS. It is possible to reconfigure the schema configuration, stored in the **cn=subschema** entry, to match Active Directory's **cn=subschema** configuration.

8.5. Using Proxy Services

Penrose Virtual Directory can function as a proxy for an LDAP server. In this configuration, Penrose Virtual Directory flattens the namespaces between sources, but all operations are still transmitted to and carried out on the original source directly.

This chapter explains how to configure proxy entries, different proxy configurations, and authentication methods for users connecting to Penrose Virtual Directory as an LDAP proxy.

NOTE

Penrose Virtual Directory can be used as a proxy to a database or LDAP server. By setting the cache expiration to 0, no data will be cached by Penrose Virtual Directory, so all requests from the clients will be forwarded directly to the data sources.

Penrose Virtual Directory can be configured to work as a proxy to another LDAP server, allowing Penrose Virtual Directory to control the access to the LDAP server. There are two primary configurations for an LDAP proxy in Penrose Virtual Directory:

- In a *basic proxy*, only certain parts of the directory are exposed. A mapping is created for the subtree, such as **`ou=people,dc=example,dc=com`**, and the operation passes through Penrose Server and is performed on the source, **`ou=users,dc=source,dc=com`**.

- In an *aggregated proxy*, Penrose Virtual Directory serves as a proxy for multiple LDAP servers or sources. In this case, a separate subtree is created for each source, such as **`ou=people, ou=source1, dc=example,dc=com`** and **`ou=people, ou=source2, dc=example,dc=com`**. Searching the base DN, **`dc=example,dc=com`**, returns entries from both sources.

For any proxy configuration, Penrose Server performs two translations to the data:

1. The DN in the LDAP request is translated into the original name space.

2. The DNs of the search results are renamed into the proxy name space.

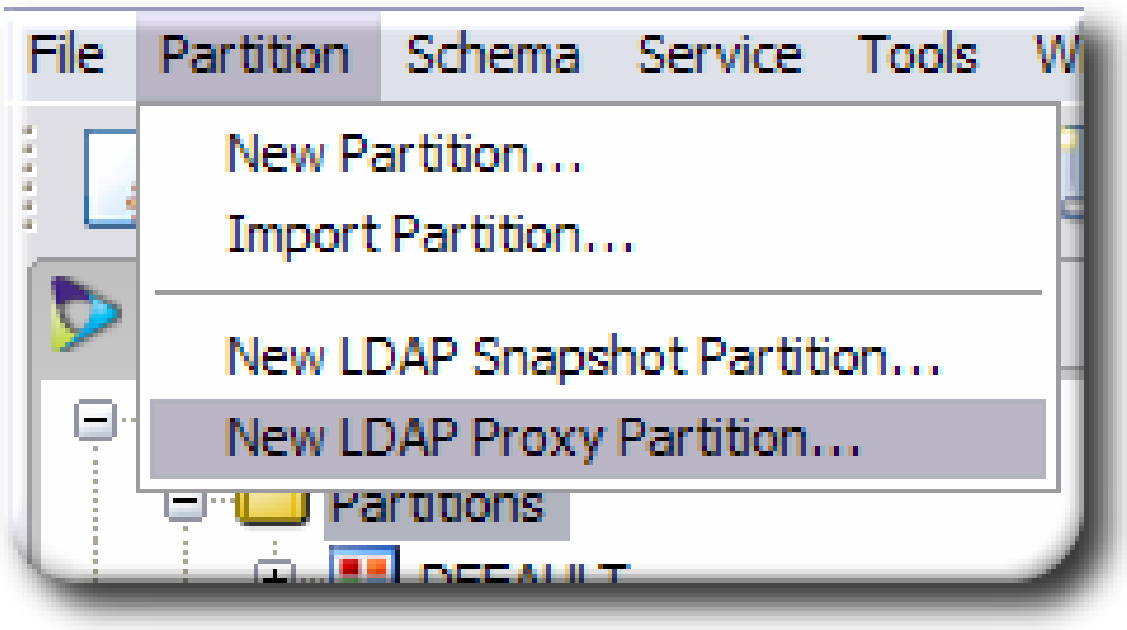
This data translation is an important function of proxy LDAP services in Penrose Server. All operations performed within an organization can use a standardized naming format without having to reconstruct existing servers. A standard namespace also makes it significantly easier and faster to find directory information or perform directory operations.

8.5.1. Creating an LDAP Proxy

Penrose Virtual Directory can be used as a proxy of an LDAP directory, including any LDAPv3-compliant directory and Active Directory. A proxy is configured through a simple wizard in Penrose Studio.

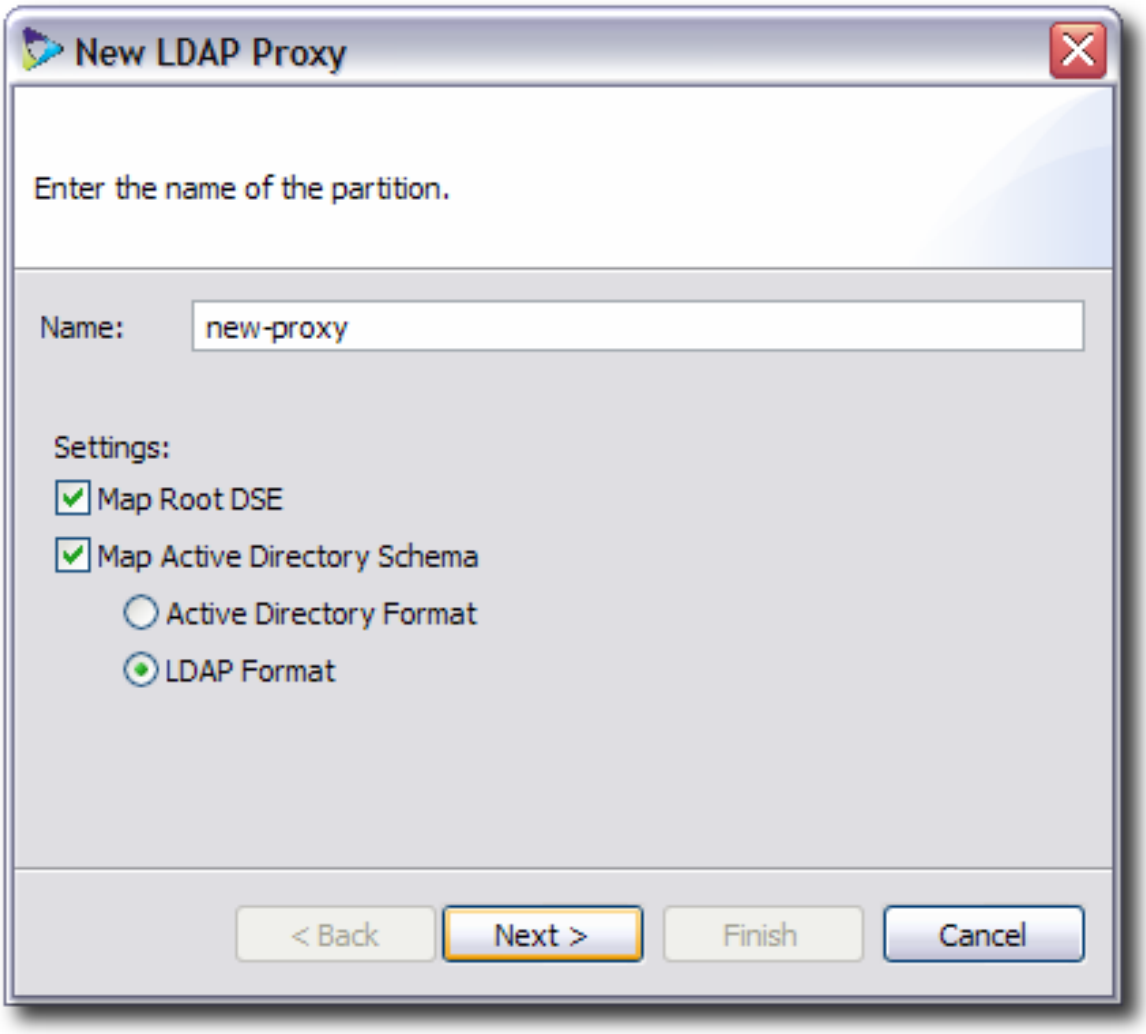
1. In Penrose Studio, open the **Partitions** in the top menu and select the **New LDAP Proxy Partition**

option.



Alternatively, open the server entry, and then right-click the **Partitions** folder, and select the **New LDAP Proxy Partition** option.

2. Enter the name for the proxy entry, and select whether to map (mirror) the directory hierarchy and, optionally, schema.

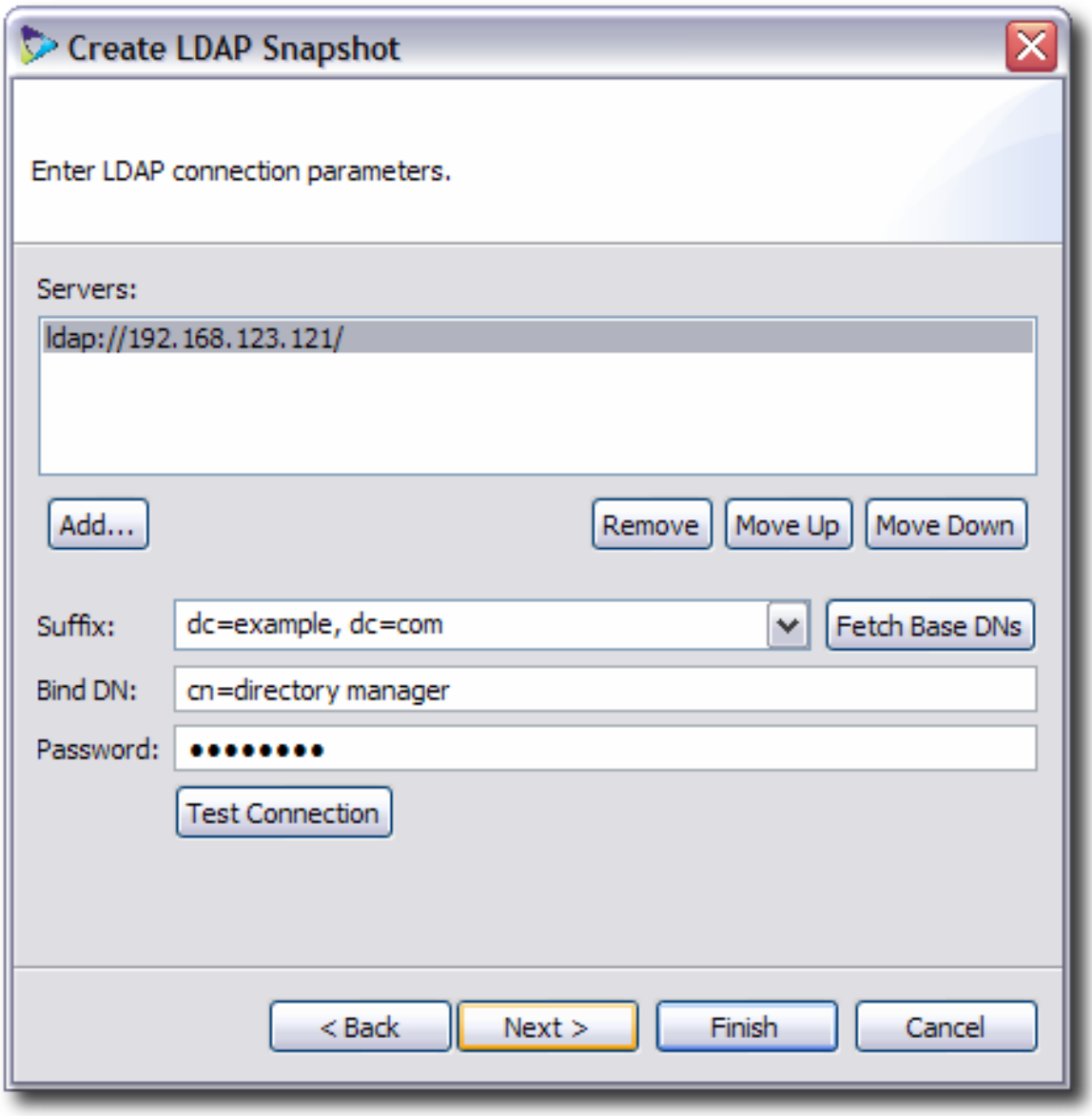


The root directory server entry (root DSE) is a vital resource for finding information about a directory server instance because the root DSE defines all of the characteristics of the directory itself. For mapping an Active Directory instance, start at the root DSE to identify the available naming context, supported controls, and locations for the domain, schema, and configuration entries.

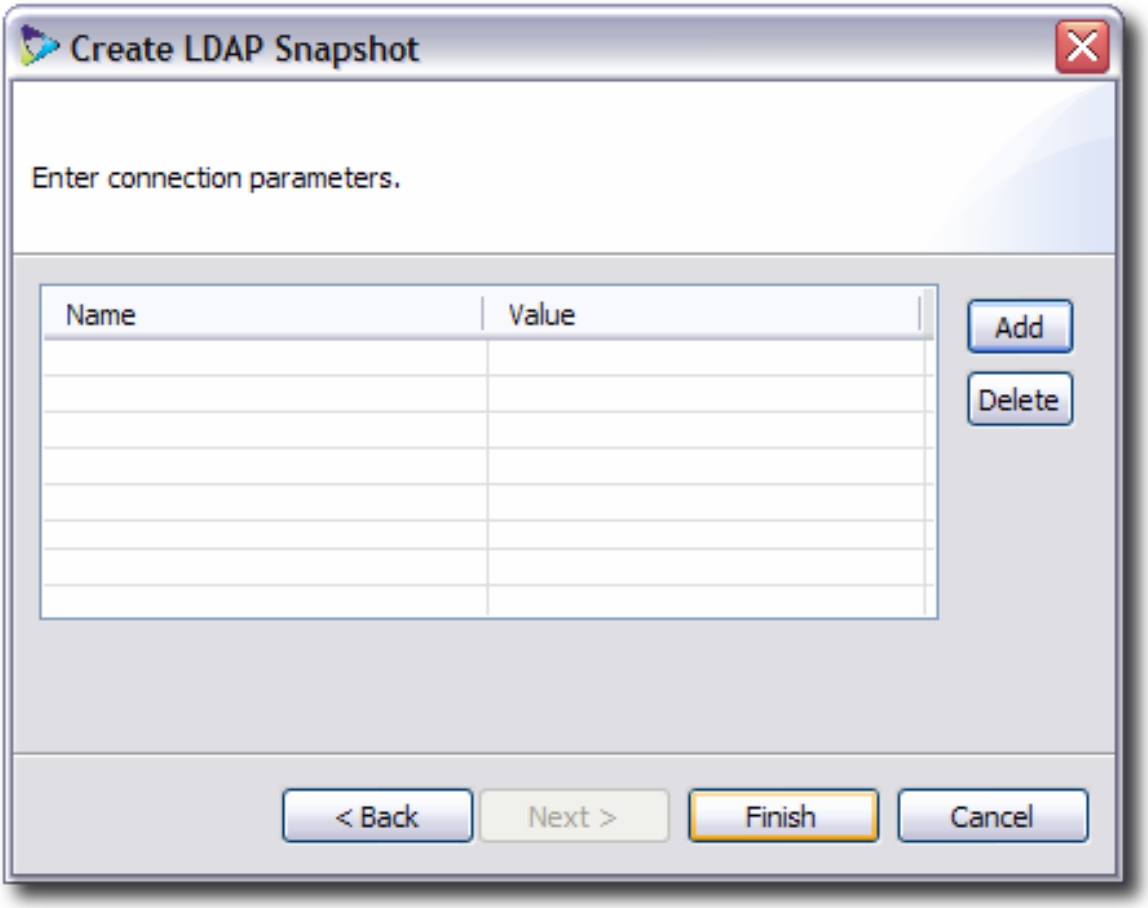
The schema mapping is another important consideration when creating a proxy to an Active Directory server. For the proxied virtual namespace, Penrose Server can use Active Directory schema to define entries or can map the Active Directory schema to LDAPv3 standard schema.

3. Click the **Add** button to add the LDAP server host and port.

4. Enter the connection information for the LDAP directory, including the server and port, log in name and password, and subtree or base DN to map.



5. Optionally, enter any additional configuration parameters, such as a Java class to extend the Penrose Server operations.



8.5.2. Configuring an LDAP Proxy Manually

A proxy entry essentially consists of a connection entry, a source entry, and a mapping entry. These entries can be configured manually by editing the appropriate configuration files for the partition.

1. Add the connection for the target LDAP server to **`connections.xml`**. Include the required connection information:

   - The URL to the server which is being proxied
   - The username and password to use to bind to the server

   For example:

```
<connections>

  <connection name="Proxy">
    <adapter-name>LDAP</adapter-name>
    <parameter>
```

```
      <param-name>java.naming.provider.url</param-name>
      <param-value>ldap://ldap.example.com</param-value>
    </parameter>
    <parameter>
      <param-name>java.naming.security.principal</param-name>
      <param-value>cn=proxy user,dc=example,dc=com</param-value>
    </parameter>
    <parameter>
      <param-name>java.naming.security.credentials</param-name>
      <param-value>password</param-value>
    </parameter>
  </connection>

</connections>
```

Creating connections is described in *Chapter 6, Configuring Connections*.

2. Create a source entry for the original subtree in the target server, which will be proxied in a virtual subtree on Penrose Server. Include the original subtree DN and, optionally, the authentication method allowed to that subtree. For example:

```
<sources>

  <source name="users">
    <connection-name>Proxy</connection-name>
    <parameter>
      <param-name>baseDn</param-name>
      <param-value>dc=source1,dc=com</param-value>
    </parameter>
    <parameter>
      <param-name>authentication</param-name>
      <param-value>full</param-value>
    </parameter>
  </source>

</sources>
```

Creating sources is described in *Chapter 7, Configuring Data Sources*.

3. For a proxy configuration, simply create a mapping entry in **`mapping.xml`** which points to the source and identifies it as a proxy. The DN specified is the virtual namespace, not the original namespace. For example:

```
<mapping>

  <entry dn="ou=proxy,dc=example,dc=com">
    <source name="users">
      <source-name>source1</source-name>
    </source>
    <handler>PROXY</handler>
  </entry>

</mapping>
```

4. Restart Penrose Server.

```
service vd-server restart
```

IMPORTANT

Always restart Penrose Server after editing the configuration file.

TIP

A sample proxy configuration is available in **`/opt/vd-server-2.0/samples/proxy`**. To create a new proxy partition quickly, copy the files from **`/opt/vd-server-2.0/samples/proxy/partition`** into a new proxy directory in the **`/opt/vd-server-2.0/partitions`** folder. Update the Penrose Server configuration files:

1. Edit the listed connections in proxy partition's **`connections.xml`** file.

2. Edit the server configuration file in **`/opt/vd-server-2.0/conf/server.xml`** to include the new proxy entry.

```
<server>
  <partition name="proxy" path="partitions/proxy"/>
</server>
```

3. Restart Penrose Server, and point the LDAP browser to the new proxy DN.

8.5.3. Configuring Authentication for Proxies

When Penrose Virtual Directory works as a proxy for an LDAP server, all operations for that server pass through Penrose Virtual Directory transparently and are sent to the LDAP directory. That means, in some way, Penrose Virtual Directory interacts with the target LDAP server as one of its users. Authentication between Penrose Virtual Directory and the LDAP server can be handled in three different ways:

- *Simple PTA.* The simplest method is for a user to bind to the virtual directory as a user in the proxy subtree, and that user and its credentials are used by Penrose Server to bind to the LDAP server. Because The user binding to the virtual directory passes through Penrose Server to bind as the same user on the LDAP directory, this is called *pass-through authentication*.

Only authentication is performed as the user; any other operation, such as search and modify operations, is performed as the Penrose Virtual Directory user which is configured in the **`connections.xml`** file.

This is the default.

- *Full PTA.* Like simple PTA, full PTA uses the bind credentials (username and password) sent to Penrose Virtual Directory to bind to the LDAP server. However, for full PTA, every operation — authentication, as well as searches, modifies, and deletes — is performed using the user's bind credentials.

- *Disabled.* When authentication is disabled, any attempts for a user to bind to the LDAP server is rejected. Only the Penrose Virtual Directory user configured in the **`connections.xml`** file is allowed to access the LDAP server.

To configure authentication for proxy access in Penrose Virtual Directory, add the *`authentication`* parameter with the appropriate value (`default`, `full`, or `disabled`) to the source entry in **`sources.xml`**. For example:

```
<source name="users">
    <connection-name>Proxy</connection-name>
    <parameter>
      <param-name>baseDn</param-name>
      <param-value>dc=source1,dc=com</param-value>
    </parameter>
    <parameter>
      <param-name>authentication</param-name>
      <param-value>full</param-value>
    </parameter>
  </source>
```

Creating sources is described in *Chapter 7, Configuring Data Sources*.

IMPORTANT

Always restart Penrose Server after editing the configuration file. For example:

```
service vd-server restart
```

8.6. Setting Access Controls on the Virtual Directory

Penrose Virtual Directory allows restrictions to be placed on what parts of the virtual directory can be accessed, by whom, and what operations they are allowed to perform. These *access control instructions* are enforced before any transaction is sent to the source, so these ACIs are enforced in addition to whatever ACIs are already set on the source.

8.6.1. Placing ACIs and ACI Inheritance

Penrose Virtual Directory also support *inherited ACIs*. If an ACI is set on the base DN or a subtree, then the ACI is automatically applied to every entry below that subtree in the virtual directory, unless the scope is limited specifically.

If an ACI is placed on an entry, it overrides any inherited ACIs which conflict with it, except for deny ACIs. Deny ACIs always supersede allow ACIs.

Those two ACI behaviors — inheritance and overrides — make organizing and placing ACIs very important. To be effective and manageable, plan out where ACIs should be placed carefully. Make ACIs at high levels in the directory tree as general as possible, and set very specific ACIs to the entry which will be modified.

8.6.2. Default ACIs

When a base DN entry is created in Penrose Studio, there is a single ACI set automatically which allows full read and search access to the virtual directory. This is inherited by any subtrees created beneath the base DN. This ACI configuration is shown in *Example 8.3, "Example Default ACI"*.

If a base DN entry is simply added to **`directory.xml`**, then no ACI is required. If no ACI is set for a subtree, then the default setting is to restrict all users but the superuser.

```
<entry dn="dc=example,dc=com">
  ...
  <aci>
    <permission>rs</permission>
  </aci>
</entry>
```

Example 8.3. Example Default ACI

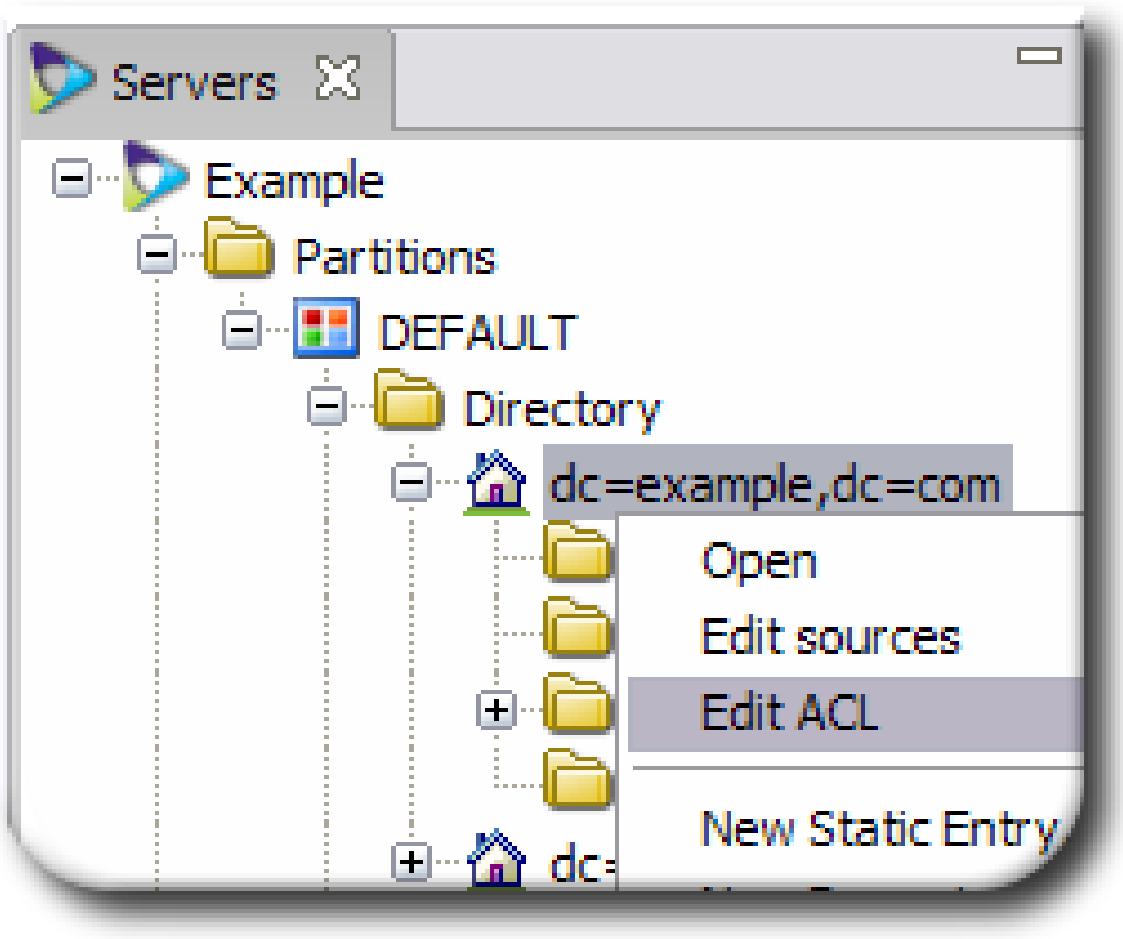
The only required parameter for the ACI entry is the **`<permission>`** line. In the default, the scope, subject, and propagation are assumed.

8.6.3. Setting Access Controls in Penrose Studio

1. Open the server entry in Penrose Studio, and expand the **Partitions** folder.

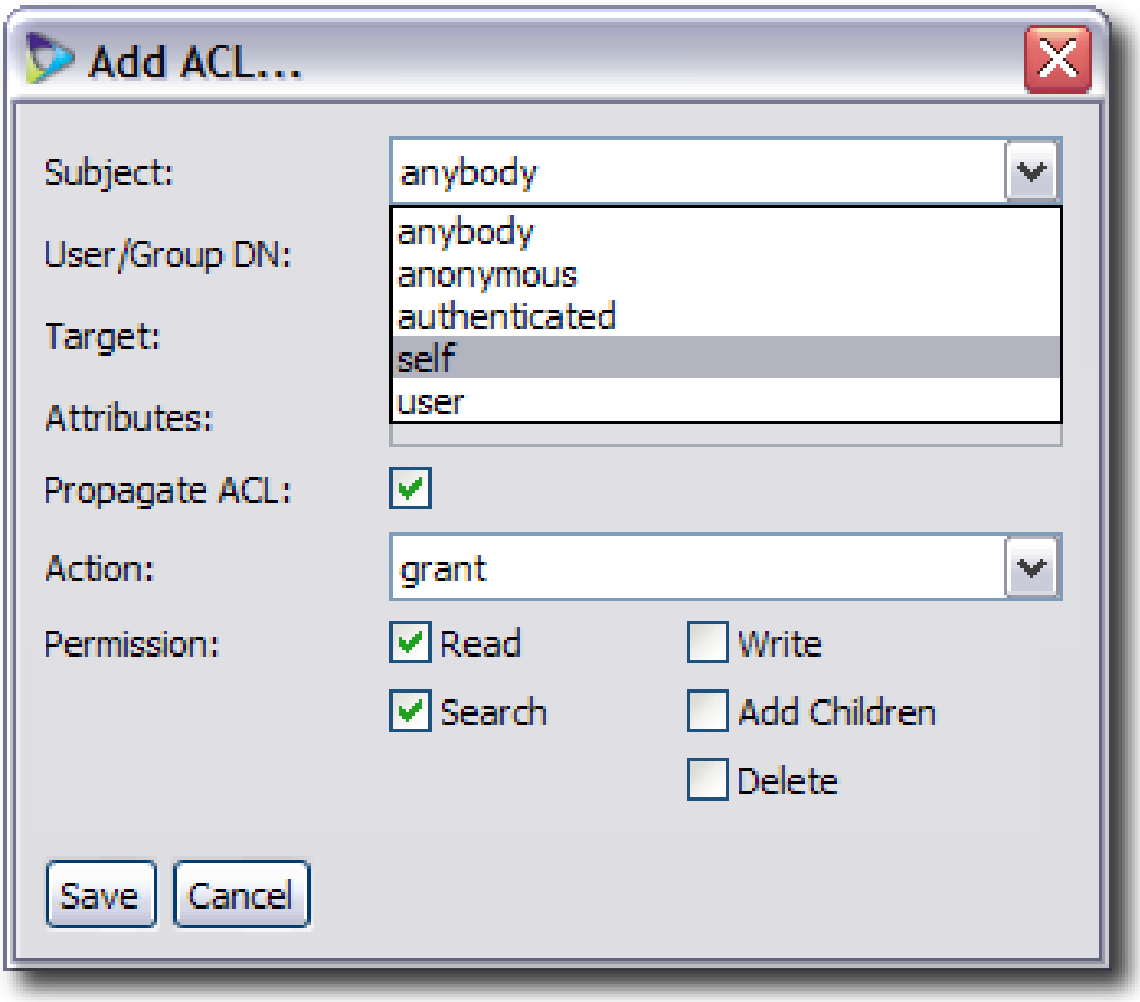
2. Open the **Directory** folder, and select the base DN or subtree to which to set the access control.

3. Right-click the subtree entry, and select **Edit ACLs** from the menu.



Alternatively, double-click the subtree, and, when the editor opens in the main window, click the **ACLs** table.

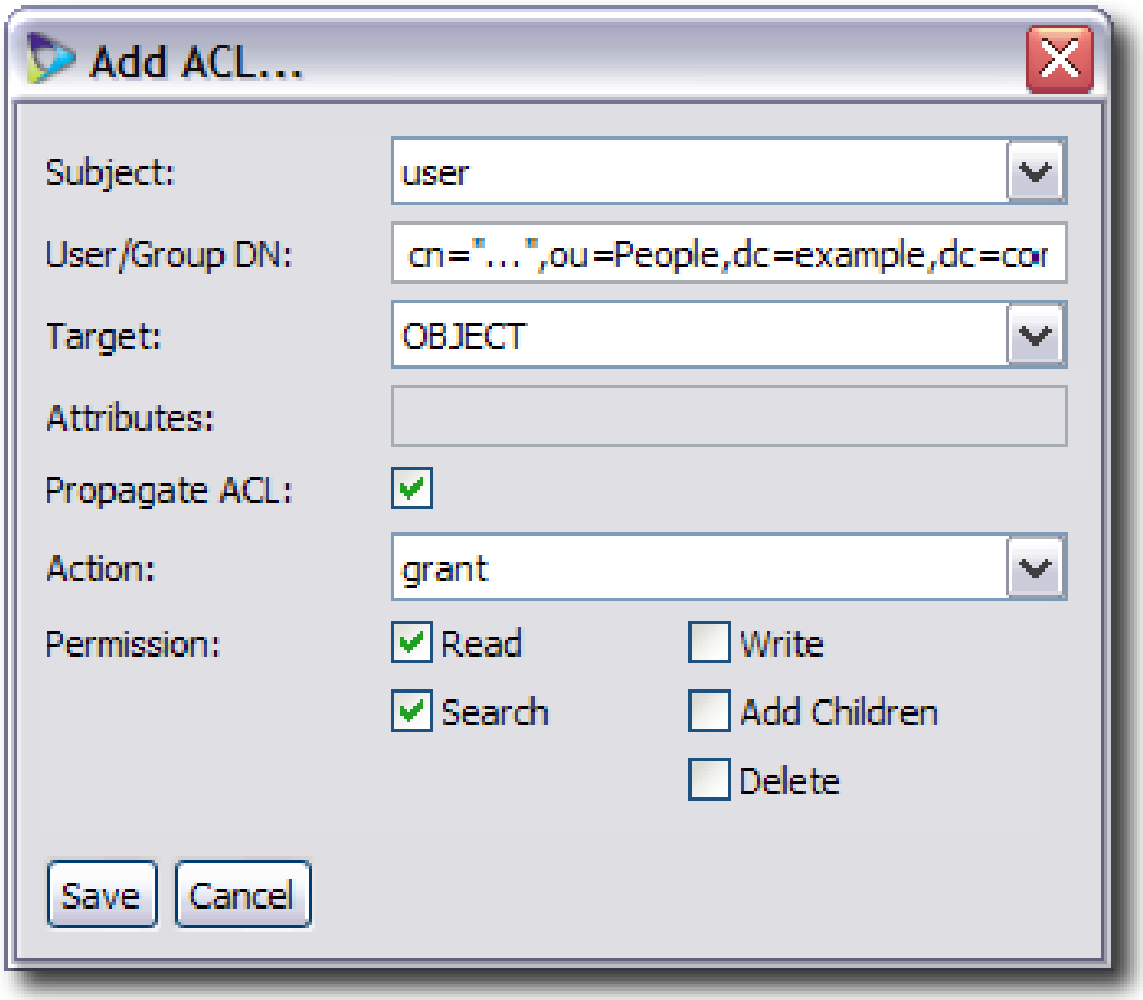
4. Set the **Subject** for the ACI. This is the kind of user to whom the ACI is applied.



There are several different options for the users for the ACI:

- *anybody* applies the ACI to all users
- *authenticated* applies the ACI to users who are logged into the source
- *anonymous* applies the ACI to users who are *not* logged into the source
- *self* applies the ACI to users who are accessing their personal entries
- *user* applies the ACI to whatever users are specified in the **User/Group DN** field

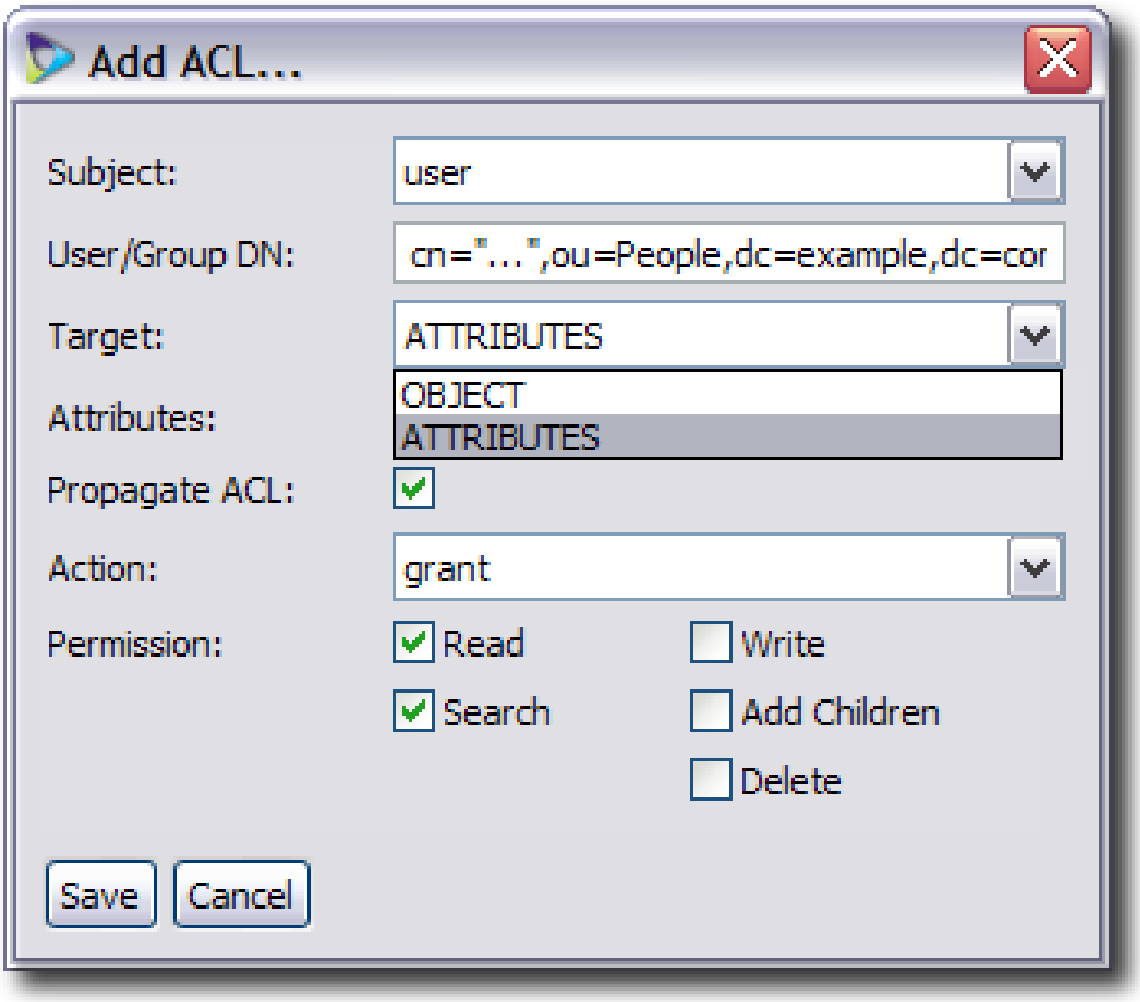
5. If the subject for the ACI is **`user`**, enter the DN of the user or users to target.



6. Set what part of the entry is affected by the ACI.

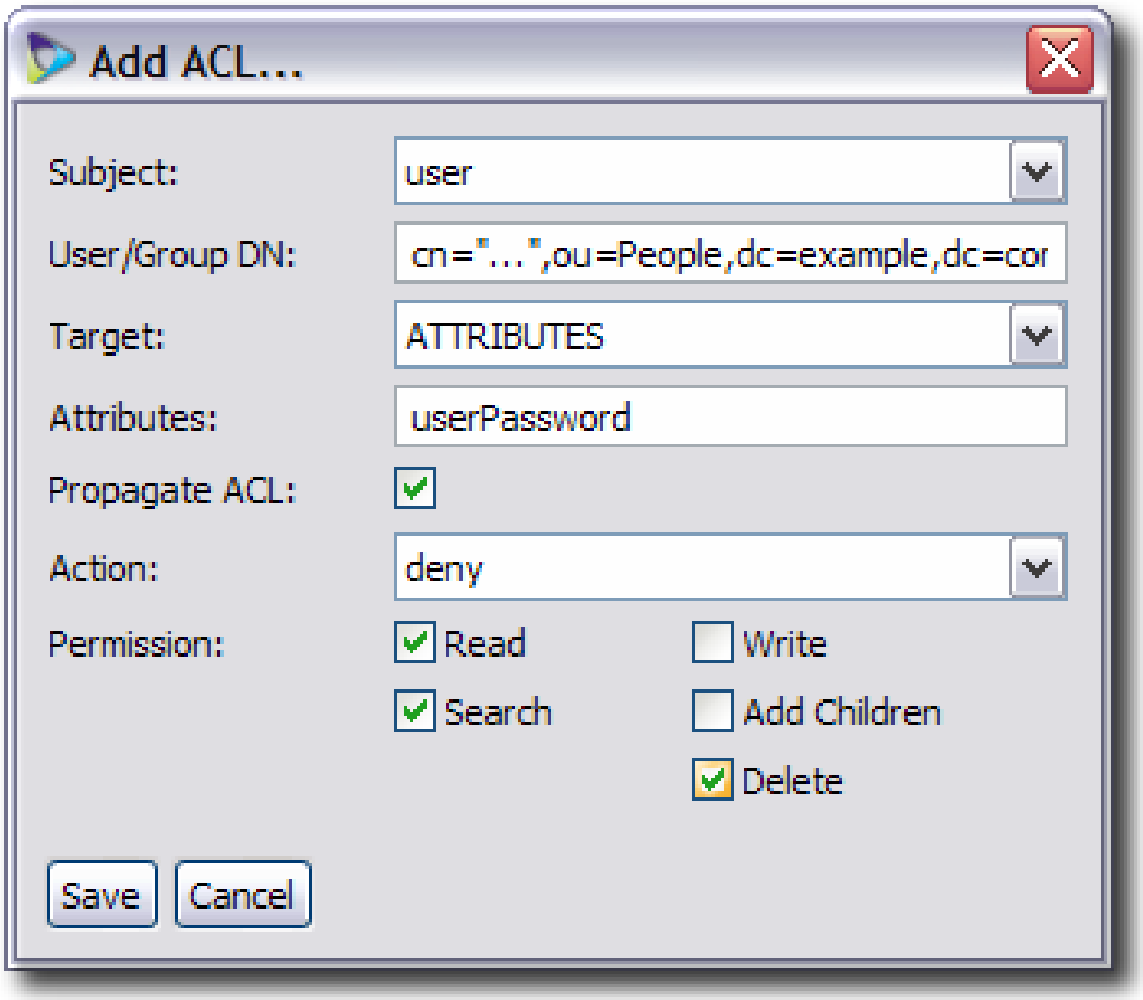
   - If the entire entry is the target, then set the target to **OBJECT**.

   - If only a subset of attributes in the entry are affected, then select the **ATTRIBUTE** menu item, and fill in a comma-separated list of LDAP attributes in the **Attributes** field.



7. Check the **Propagate ACL** checkbox if this ACL should be applied to entries below the subject DN in the virtual directory tree. If the ACI should only apply to the specified entry, then uncheck the box.

8. Set the action for the ACI. There are two options: to allow the specified actions or to deny the specified actions.

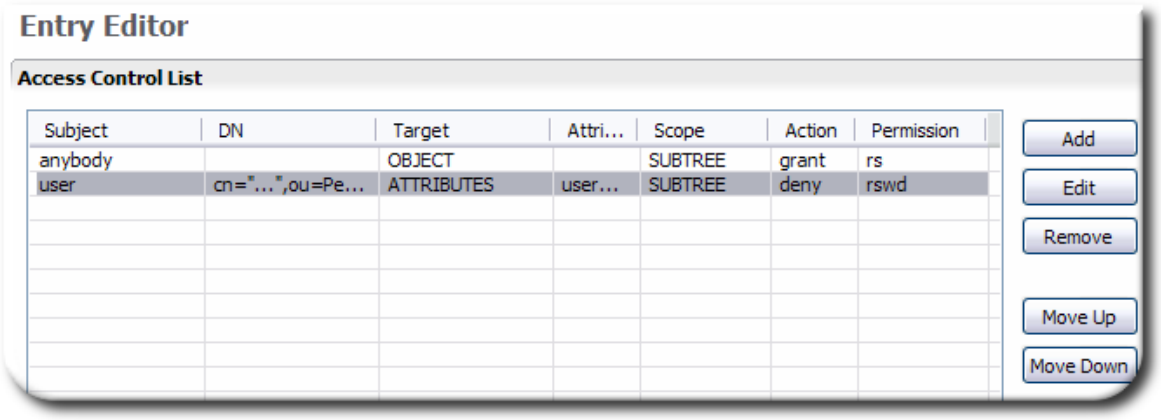


9. Select what actions to or on the source will be allowed or denied.

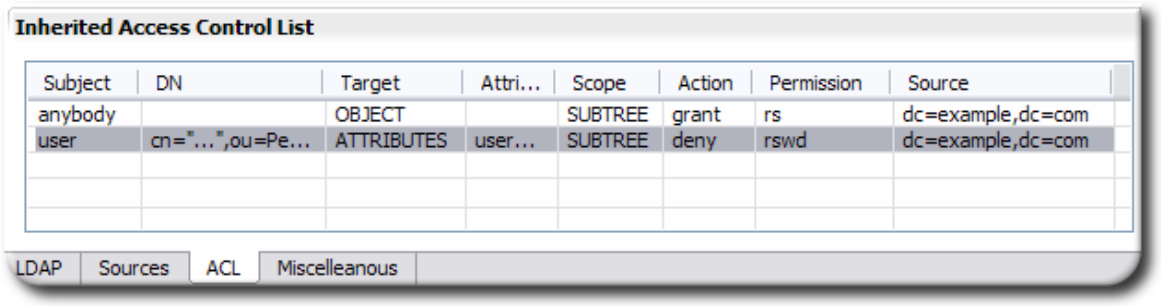
- *read* allows the user to view an entry or attribute; this is allowed by default
- *search* allows the user to search for an entry or attribute
- *write* allows the user to change the entry or attribute
- *delete* allows the user to remove the entry or attribute
- *add child* allows the user to add a new entry beneath this entry

10. Close the editor window, and save the changes when prompted.

The new ACI is listed in the **Access Control List** section for the entry.



If the **Propagate ACL** checkbox is selected, then the ACI is also listed in all of the child entries under the subtree.



8.6.4. Setting Access Controls Manually

Access controls are set on each individual subtree or base DN entry for the virtual directory in the **`directory.xml`** file. The **`directory.xml`** file is in **`/opt/vd-server-2.0/conf/`** for the default partition and in **`/opt/vd-server-2.0/partitions/`***partition_name*`/DIR-INF`** for any additional partition.

An ACI defines *what* operations can be done, *who* the permissions apply to, and *where* in the directory tree the permissions are set. Since the ACI is related to the virtual directory tree, this ACI entry is set in the **`directory.xml`** file in the subtree entry.

An ACI entry defines the following information:

- The *subject*, which is the virtual directory users to whom the ACI applies; for certain kinds of subjects, the *DN* is required to identify the entries to whom the ACI applies

- The *target*, which is the part of the entry which is controlled by the ACI, and, possibly, a list of *attributes* which are controlled by the ACI

- The *scope*, which sets whether the ACI applies only to the subject entry or to every entry beneath it

in the directory (meaning, whether it is inherited)

- The *action*, which is whether the operations listed are allowed or denied to the subject
- The *permission*, which is the operations, such as search and modify, which can be performed by the subject

The only required information is the permission.

IMPORTANT

Always restart Penrose Server after editing the configuration file. For example:

```
service vd-server restart
```

An example ACI is in *Example 8.4, "Annotated Example ACI"*. The different options and parameters for configuring the ACI are listed in *Table 8.2, "ACI Entry Tags and Parameters"*.

```
<entry dn="cn="...",ou=People,dc=example,dc=com">
...
   <aci subject="user"> defines the users affected by the ACI
      <dn>cn="...",ou=People,dc=example,dc=com</dn>   gives the DN of the
subject entry
      <target>ATTRIBUTES</target>  defines whether the ACI is for the en\
tire entry or an attribute
      <attributes>userPassword</attributes>  lists attributes controlled
by the ACI
      <action>deny</action> whether the action is allowed or denied for
the user
      <scope>OBJECT</scope> whether the ACI is for the entire subtree or
just the subject
      <permission>rswd</permission>  the operations which are affected b
the ACI
   </aci>
...
</entry>
```

Example 8.4. Annotated Example ACI

Tag or Parameter	Description	Default Value	Example
subject=	Defines to what users this ACI applies. This is an option set on the **<aci>** tab. There are five possible values:	anybody	<aci subject="user"

Tag or Parameter	Description	Default Value	Example
	anybody applies the ACI to all users *authenticated* applies the ACI to users who are logged into the source *anonymous* applies the ACI to users who are *not* logged into the source *self* applies the ACI to users who are accessing their personal entries *user* applies the ACI to whatever users are specified in the **User/Group DN** field		
<dn>	Gives the DN of the user to which the ACI is applied. *Only required with the argument* `subject=user`.		<dn>cn=*,ou=people,dc=example,dc=com</dn>
<target>	Sets whether the ACI applies to the entire entry or to a subset of attributes in the entry. There are two options: *OBJECT*, meaning the entire entry *ATTRIBUTES*, meaning a subset of attributes	OBJECT	<target>OBJECT</target>
<attributes>	Gives the attributes to which the ACI applies, in a comma-separate list. *Only required if the target is set to ATTRIBUTES.*		<attributes>userPassword, userCertificate</attributes>
<scope>	Determines whether the ACI is inherited by entries beneath this entry in the subtree or only applies to the entry on which it is set. There are two options: *SUBTREE* means the ACI is inherited by all entries beneath this	SUBTREE	<scope>OBJECT</scope>

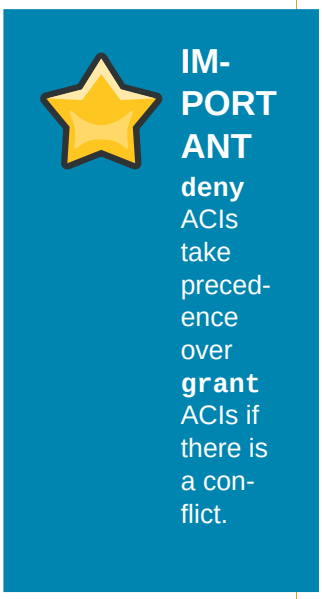
Tag or Parameter	Description	Default Value	Example
	entry *OBJECT* applies only to the entry		
<action>	Determines whether the ACI allows a certain action or prohibits a certain action. *grant* allows the operations *deny* restricts the operations  **IMPORTANT** **deny** ACIs take precedence over **grant** ACIs if there is a conflict.	grant	<action>deny</action>
<permission>	Defines what operations are being allowed or denied. There are five operations allowed through Penrose Virtual Directory: *r* means read (view) access *s* means search access *w* means write (modify) access *d* means an entry or attribute can be deleted *a* means a child entry can be added beneath this entry	n/a	<permission>rwsd</permission>

Table 8.2. ACI Entry Tags and Parameters

Mapping Entries and Attributes

Many forms of mapping described in *Chapter 8, Configuring the Virtual Directory* depend on having common attribute values which can be matched and mapped to define virtual entries. Some infrastructure situations do not have that consistency of data. In other situations, Penrose Virtual Directory is a bridge for migrating from one kind of data storage (NIS servers) to another (LDAP servers). In both of those scenarios, Penrose Virtual Directory provides an additional way to link entries, called *identity federation*.

9.1. Planning How to Map Entries

The next step is to define how the individual entries (leaves on the tree) are created. Again, there are many options, which can all be used depending on the type of information which is in the sources and the kind of information that should be included in the virtual directory entries.

Penrose Virtual Directory has several different mapping styles: basic, nested, joined, and identity linking, described in the sections below. These mappings are applied to the *attributes* for each identity, so any or all of the different mapping styles can be applied to any attribute, entry, or data source (though there can only be one mapping per attribute per source).

9.1.1. Basic Mapping

This simplest mapping is *basic mapping*, which is a one-to-one mapping of attributes. For example, if a users table in a database has an employee's username, first name, last name, and password, the mapping entry can simply convert the tuples to the `cn`, `givenName`, `sn`, and `userPassword` LDAP attributes.

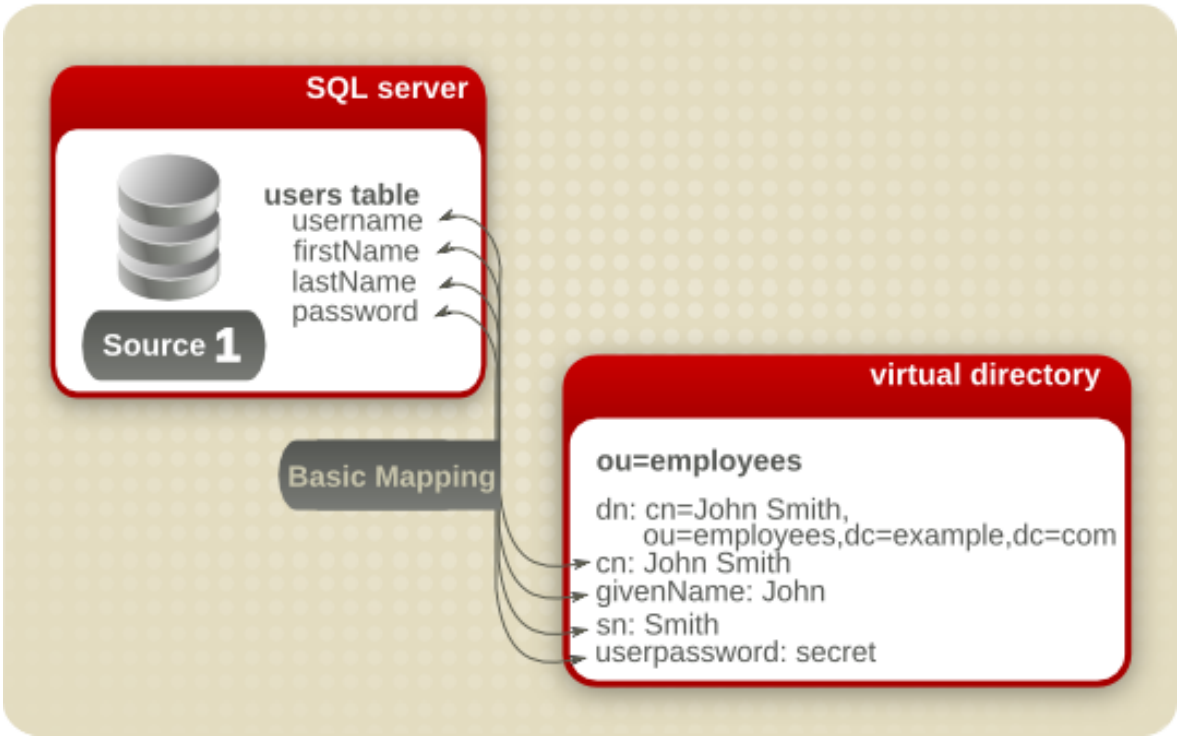


Figure 9.1. Basic Mapping

Basic mapping is configured in the **`directory.xml`** file for the partition, as described in *Section 9.3, "Configuring Basic Mapping"*.

9.1.2. Nested Mapping

A *nested mapping* is a dynamic entry created beneath another dynamic entry. In this case, the generated entry for the first entry is used as input for the second dynamic entry.

Each of the dynamic entries has its own source defined, and the content of these sources is joined when the entries are processed. The scope of the values for the second entry is restricted to areas where its source overlaps with the information in the first source.

For example, a parent entry is created which generates user entrys (**`uid=...`**). Its child entry generates a list of group entries (**`cn=...`**) which the user belongs to. The user entry is generated first, and the possible group entries under that user are limited to groups in the second source which that user belongs to.

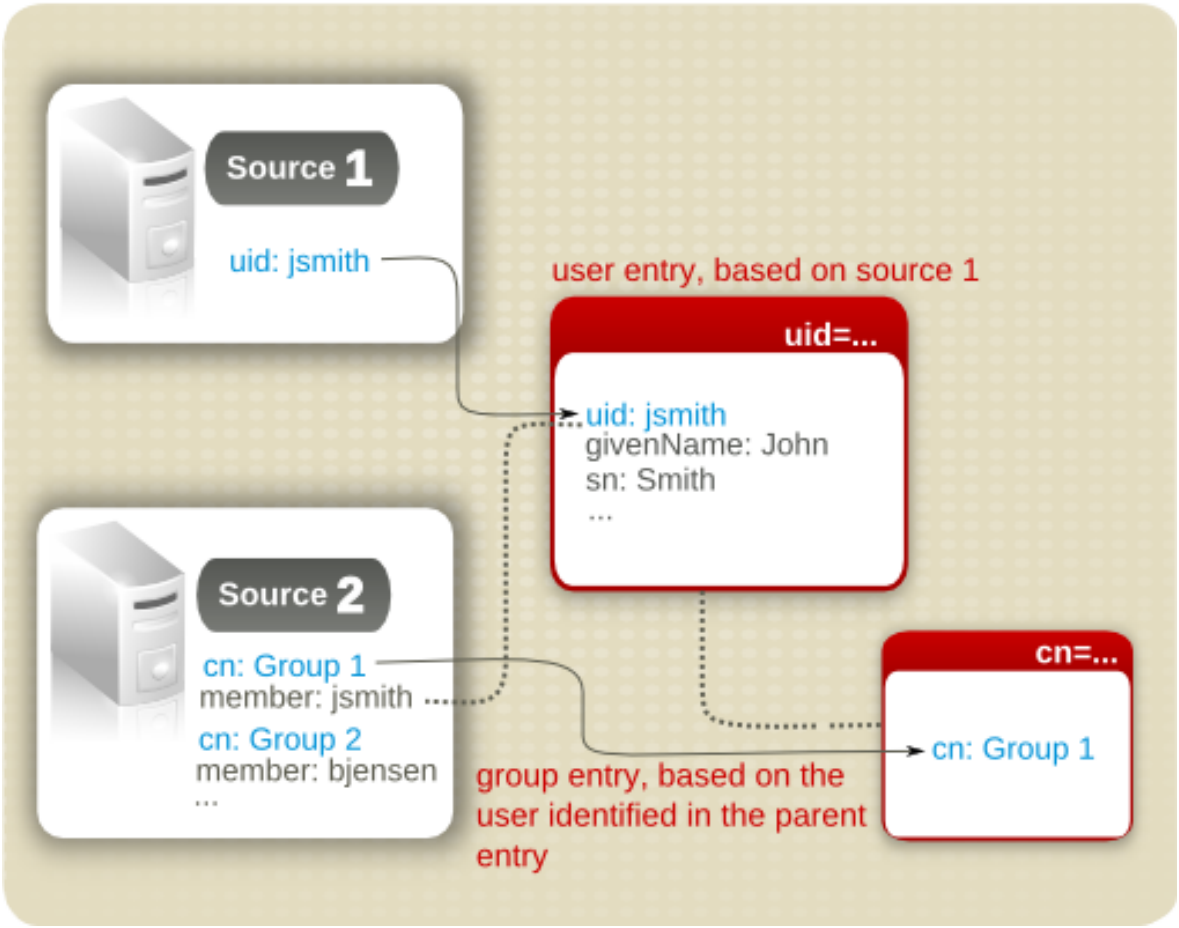


Figure 9.2. Nested Mapping

Nested mapping is configured in the **`directory.xml`** file for the partition, as described in *Section 9.4, "Configuring Nested Mapping"*.

9.1.3. Joined Mapping

Basic mapping is very simple correlation between attributes, but it doesn't handle correlations between entries. Many environments have multiple applications, and each entity has a different entry in each application. Penrose Virtual Directory can combine these separate entries into a single virtual directory entry.

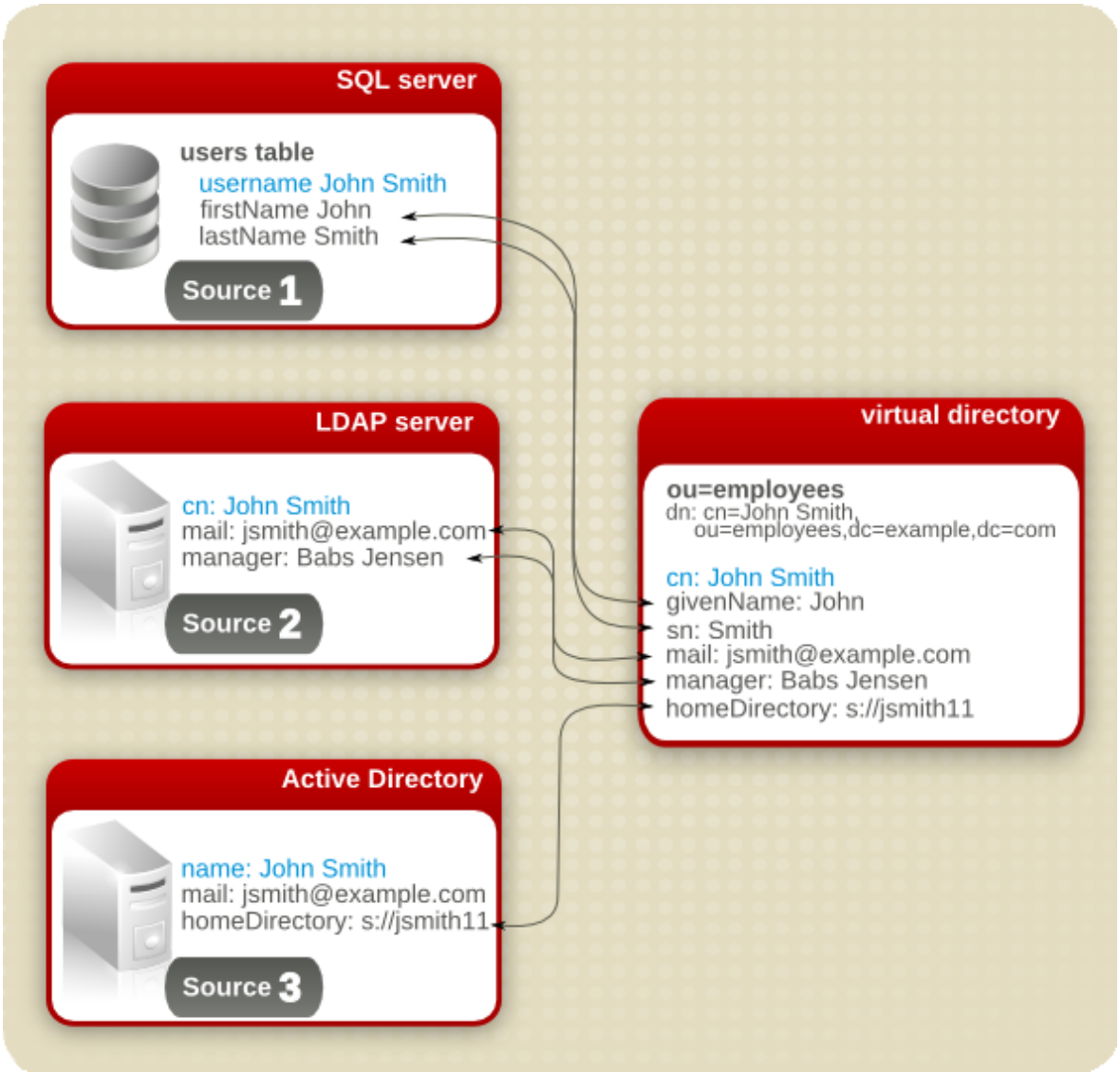


Figure 9.3. Join Mapping to Merge Entries

Join mapping can also be used to create new entries from two existing entries. For example, if one database holds user information (including group membership) and another holds group names and descriptions, Penrose Virtual Directory can create a new entry in the virtual directory.

For join mapping, a common attribute, a *key*, has to exist on all of the sources so that the entry in both sources can be linked from in one source to another. One of these sources is the *primary key*, which determines whether the entity exists on any of the other sources. The joined entry is dependent on the primary key, and if that primary key value does not exist in the primary source, then the entry will no show up in the virtual directory.

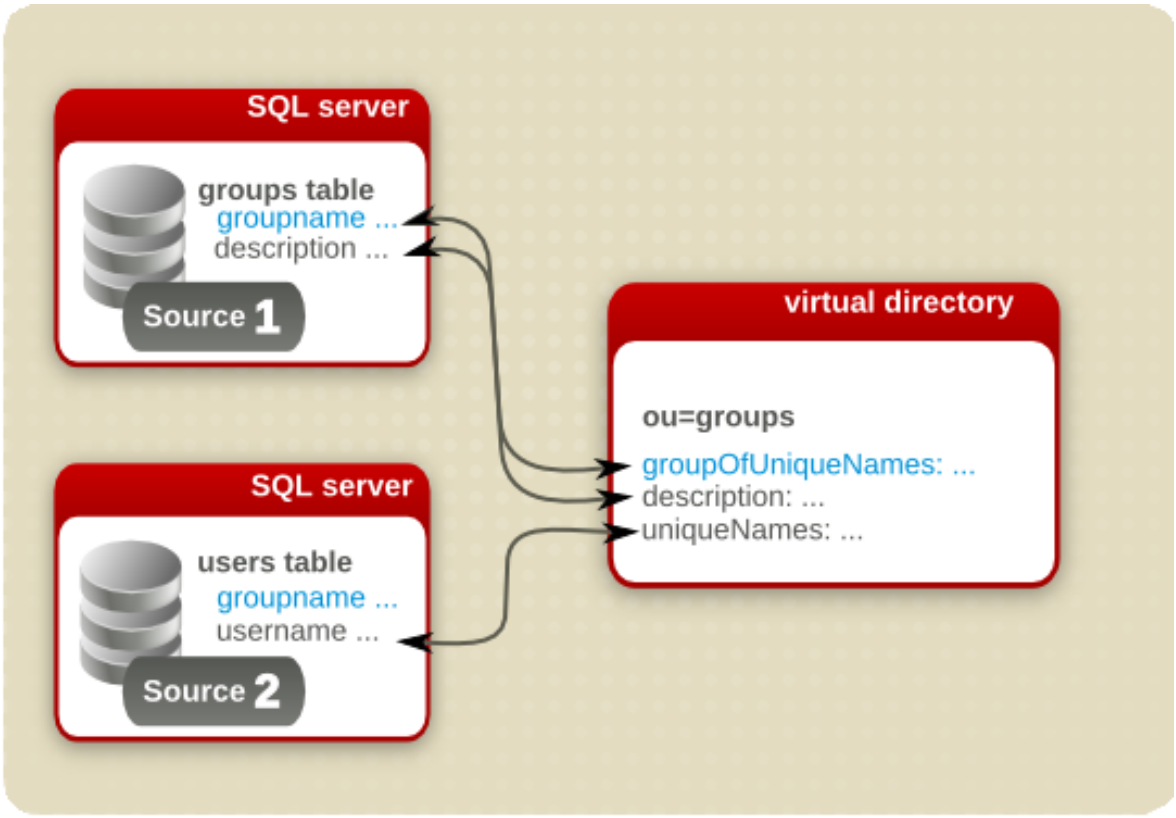


Figure 9.4. Join Mapping to Create New Entries

Join mapping is configured in the **`directory.xml`** in the directory entry, as described in *Section 9.5, "Joining Entities into a Single Virtual Entry"*.

9.2. Creating a Single Subtree from Multiple Sources

The same kind of information can be contained in multiple sources, such as having employee information spread out with authoritative sources at each branch office or distributing the load across several databases for high-traffic systems. It makes sense to design the virtual directory tree according to those data groups. For example, creating subtrees for all employee data, for all customer data, for servers, and for other kinds of infrastructure, such as buildings and conference rooms. If the same identities are not stored in different locations, only the same kind of entries, then the mapping is as simple as mapping each source to the same subtree.

IMPORTANT

A separate subtree entry needs to be created for every source which is merged into the subtree is because of how the attributes are mapped. Multi-valued attributes can be mapped to multiple sources; however, many required attributes, such as *`uid`*, are single-valued. This means that there has to be a separate entry to allow more than one source to supply the attribute.

In Penrose Studio, create a new subtree entry with the same name for each source, as shown in *Fig-*

ure 9.5, "Two Entry Mappings under ou=People". In this case, two dynamic entries are created for the **cn="..."** entry under the the **ou=People,dc=example,dc=com** subtree. Creating subrees is described in *Section 8.2.1, "Creating the Virtual Directory in Penrose Studio"*.

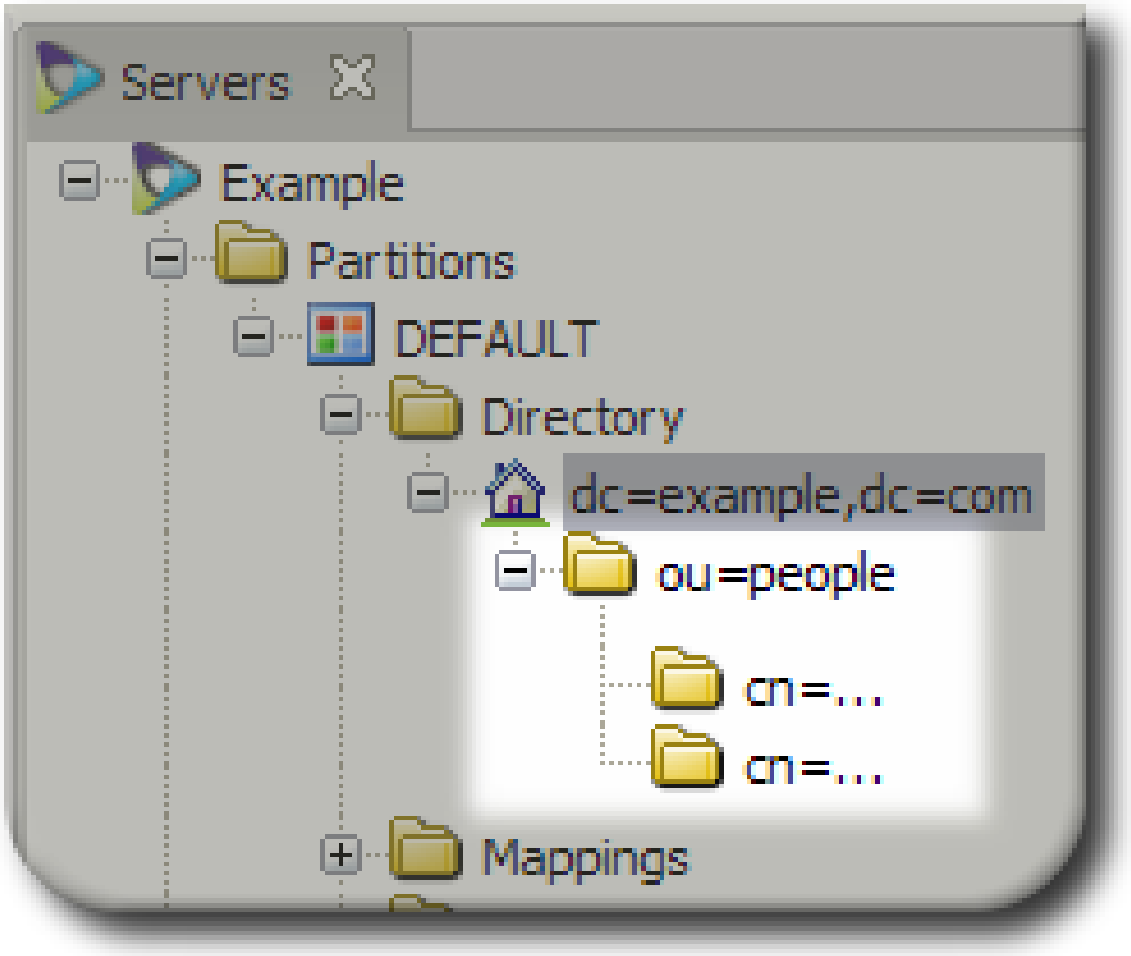


Figure 9.5. Two Entry Mappings under ou=People

To create the entries mapping manually (as in *Section 8.2.3, "Configuring the Virtual Directory Manually"*, add entries with the same **dn=** value (the same entry name) and a different **<source>** entry, pointing to each source being merged into the subtree. As a simplified example, *Example 9.1, "Merging User Database and LDAP Entries into a Virtual Directory"* combines employee entries from a database (**MySQL1**) and a Red Hat Directory Server instance (**LDAP1**) into the virtual **ou=People,dc=example,dc=com** subtree.

```
<entry dn="uid=...,ou=People,dc=example,dc=com">
    <oc>person</oc>
    <oc>organizationalPerson</oc>
    <oc>inetOrgPerson</oc>
 mapping
    <at name="uid" rdn="true">
        <variable>MySQL1.username</variable>
    </at>
 reverse mapping
```

```
    <source name="MySQL1">
        <source-name>MySQL1</source-name>
        <field name="username">
            <variable>uid</variable>
        </field>
    </source>
</entry>

<entry dn="uid=...,ou=Users,dc=Example,dc=com">
    <oc>person</oc>
    <oc>organizationalPerson</oc>
    <oc>inetOrgPerson</oc>
 mapping
    <at name="uid" rdn="true">
        <variable>LDAP1.uid</variable>
    </at>
 reverse mapping
    <source name="LDAP1">
        <source-name>LDAP1</source-name>
        <field name="uid">
            <variable>uid</variable>
        </field>
    </source>
</entry>
```

Example 9.1. Merging User Database and LDAP Entries into a Virtual Directory

IMPORTANT

Always restart Penrose Server after editing the configuration file. For example:

```
service vd-server restart
```

9.3. Configuring Basic Mapping

Basic mapping is a direct connection between one attribute in the virtual directory and one attribute on a source. There are two parts to any mapping: configuring the mapping between the virtual directory attribute and the source and then mapping back from the source to the virtual directory attribute.

NOTE

For multi-valued attributes, there can be more than one attribute mapping, but only one virtual attribute and one source attribute are referenced in a mapping.

TIP

Mappings for virtual directory attributes and reverse mappings for source attributes can be configured in the **`directory.xml`** file, but more complex mappings can be configured in the **`mappings.xml`**, which allows BeanShell scripts to be called before and after mapping attributes to perform futher transformations on the virtual entry data. For more information, see *Section 9.6, "Creating Advanced Mappings"*.

9.3.1. Configuring Basic Mapping in Penrose Studio

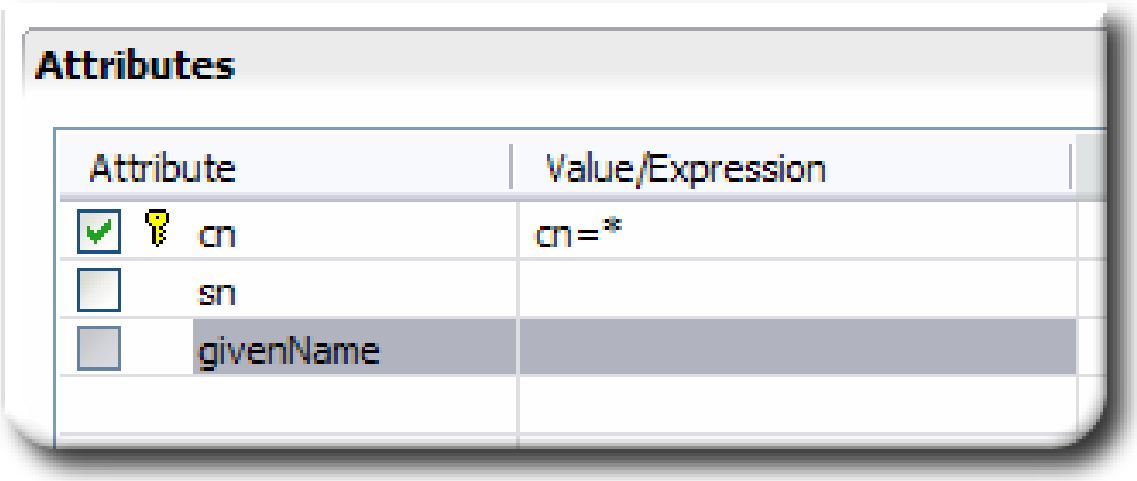
To create a basic mapping in the Penrose Studio:

1. Double-click the subtree entry to open the entry editor in the main window.

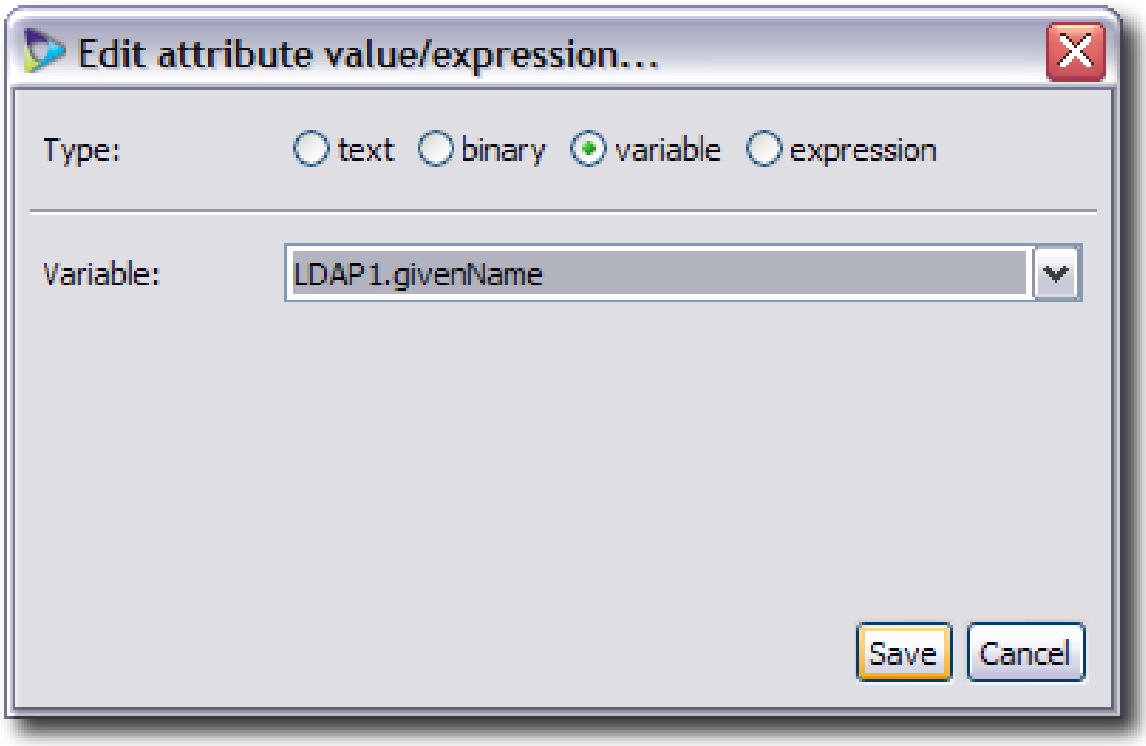
NOTE

Mapping can be configured when a directory subtree is created or can be added to a subtree entry.

2. Double-click the virtual attribute to map, listed in the **Attributes** section of the **LDAP** tab.

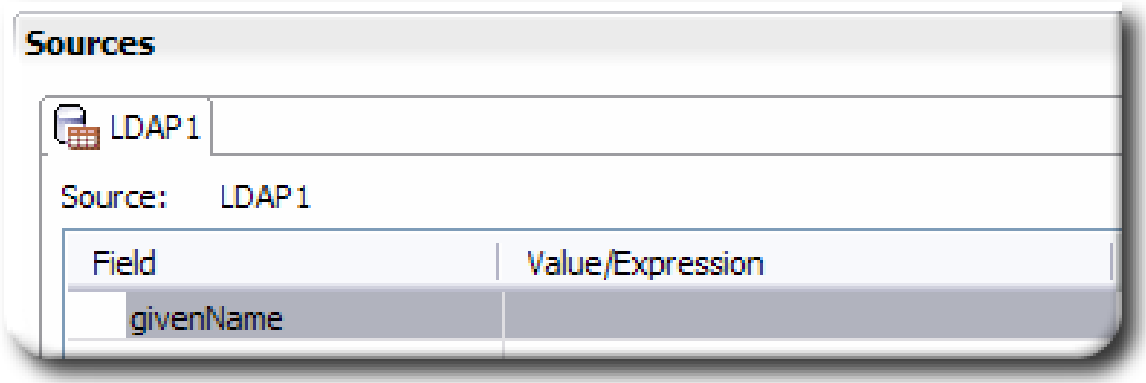


3. Select the radio button for how the value will be supplied. **text** sets an exact value, **variable** allows you to choose from all attributes configured for the virtual directory sources, and **expression** accepts BeanShell scripts.



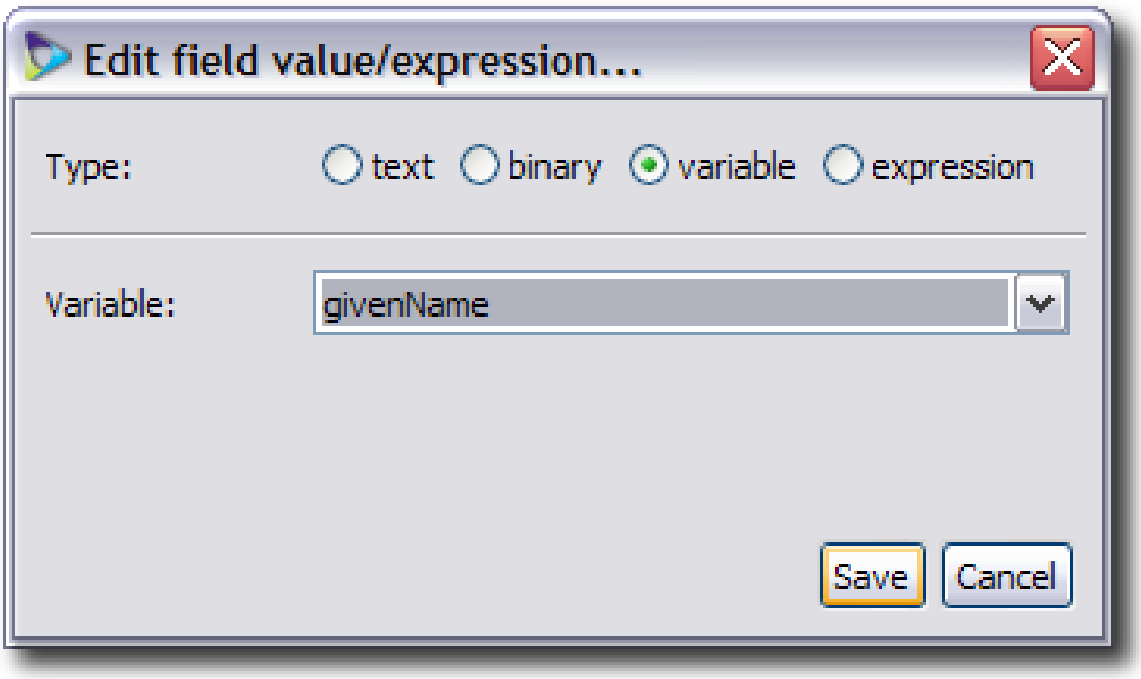
4. In the subtree editor window, open the **Sources** tab, and, if there is more than one source, select the correct source tab at the top of the section.

5. Double-click the corresponding source attribute in the list.



If the attribute is not listed, click the **Add field** link at the bottom, and add the attribute, then select it.

6. Select the radio button for how the value will be supplied. **text** sets an exact value, **variable** includes the attributes configured for all virtual directory sources as well as the virtual directory itself, and **expression** accepts BeanShell scripts. For both **variable** and the**expression** scripts, make sure that the attribute referenced is the virtual directory attribute.



7. Close the editor window and save the changes when prompted.

9.3.2. Configuring Basic Mapping Manually

Configuring basic mapping manually requires adding a pair of mapping entries for the attribute to the subtree entry in the **`directory.xml`** file.

- One entry adds the attribute and mapping in the subtree entry, linking back to the source

- The other entry adds a reverse mapping in the source description for the subtree, linking back to the virtual directory attribute

The first entry is an attribute entry (**`<at>`**) in the subtree definition. The attribute requires the `name` argument to identify the LDAP attribute and then a mapping entry to link it back to the source. The mapping tags and parameters are listed in *Table 8.1, "Parameters for Subtree Entries"*.

For example, to map the *`givenName`* attribute to the JDBC1 source attribute, add these lines to the subtree entry:

```
<entry dn="ou=people,dc=example,dc=com">
      <at>
           <variable>JDBC1.firstname</variable>
      </at>
...
```

The second entry is a reverse mapping (**<field>**) within the source definition in the subtree. Like the **<at>** attribute, the **<field>** is a mapping entry which uses the `name` argument to identify the source attribute and then a mapping entry linking it back to the virtual directory.

For example, to map the *`firstname`* attribute in the database back to the virtual directory's *`givenName`* attribute, add these lines to the subtree entry:

```
<source>
...
      <field name="firstname">
           <variable>givenName</variable>
      </field>
...
</source>
```

IMPORTANT

Always restart Penrose Server after editing the configuration file. For example:

```
service vd-server restart
```

9.4. Configuring Nested Mapping

Nested mapping is a way of embedding a dynamic subtree under another dynamic subtree. The child subtree can inherit entries from its parent, even though both subtrees can have different sources. This is one way to organize information from different sources. The DN of the nested mapping will have two dynamic components:

```
dn="uid=...,cn=...,ou=Locations,dc=Example,dc=com"
```

This can be useful to group entries and simplify the relationships between different sources. For example, Example Corp. has three database tables used by their resources group:

- *users* contains a complete list of user information, including their usernames, first names, and last names.
- *officepeople* contains a list of office locations and their members' usernames.
- *offices* contains a complete list of office details, including the locations and descriptions.

The virtual directory is organization to contain two subtrees: one for users and one for locations. The hierarchy can be arranged so that the users are grouped by their location, meaning that the users subtree is a child of the location subtree.

The key to configuring the nested mapping is in selecting the right attribute mapping and reverse mapping, as illustrated in *Figure 9.6, "Example Nested Mapping"*. The parent is the locations subtree. The office information is linked to the **officepeople** database through the location attribute, which both the **offices** and **officepeople** table share; the user information is pulled into the entry through the **username** attribute which both the **officepeople** and **users** tables share.

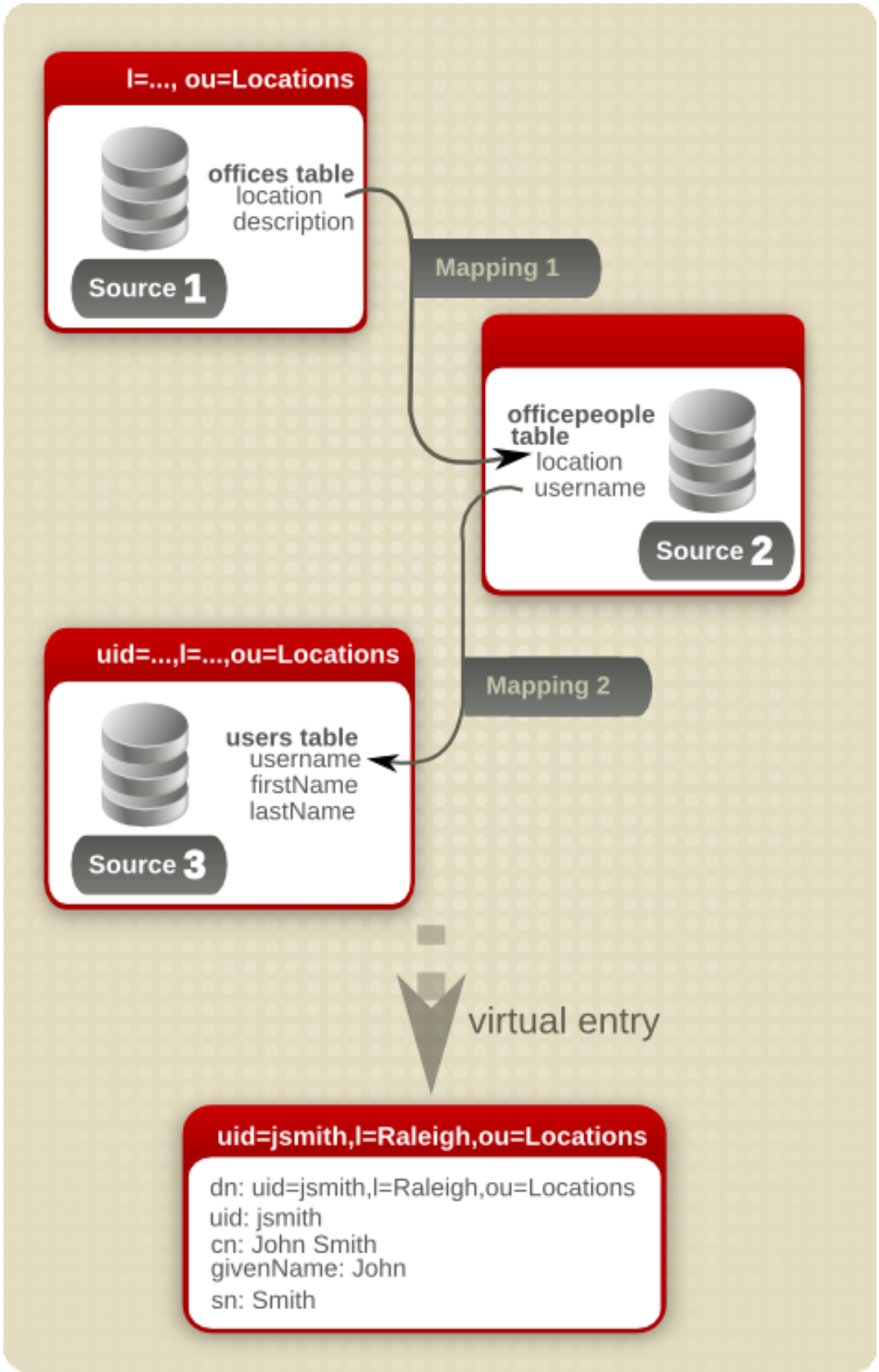


Figure 9.6. Example Nested Mapping

TIP

Mappings for virtual directory attributes and reverse mappings for source attributes can be configured in the **`directory.xml`** file, but more complex mappings can be configured in the **`mappings.xml`**, which allows BeanShell scripts to be called before and after mapping attributes to perform futher transformations on the virtual entry data. For more information, see *Section 9.6, "Creating Advanced Mappings"*.

9.4.1. Creating a Nested Mapping in Penrose Studio

For the nested mapping example, there are three entries which need defined:

- A directory entry, source entry, and mapping for the **`offices`** table which defines the location.
- A directory entry, source entry, and mapping for the **`users`** table which defines the username and other user attributes.
- A source entry and mapping for the **`officepeople`** table which links to the username in the **`users`** table and the location in the **`offices`** table.

To create a nested mapping:

1. Open the **Directory** folder.
2. Create a static entry for **`ou=Locations`**, as in *Section 8.2.1.2, "Adding a Static Directory Entry"*.
3. Create a dynamic entry for the location subtree, as in *Section 8.2.1.3, "Adding a Dynamic Directory Entry"*.
   - Add the **`office`** table as a source.
   - Set the naming attribute to the *`l`* and the object class to *`locality`*.
   - Map the the *`l`* attribute in the virtual entry to to the **`offices`** entry in the **`offices`** table.
   - Map the the *`description`* attribute in the virtual entry to to the **`description`** entry in the **`offices`** table.
4. Open the **Sources** folder, and double-click the **`offices`**'s source. Edit the source's fields to create reverse mappings on the source from the **`offices`** and **`description`** attributes back to the *`l`* and *`description`* attributes in the **`offices`** table, respectively.

For more information on creating reverse mappings on sources, see *Section 7.3, "Creating and Editing Sources Manually"*.

5. Right-click the dynamic **l=...** subtree in the virtual directory, and create a dynamic subtree underneath it for the user entries, as in *Section 8.2.1.3, "Adding a Dynamic Directory Entry"*.

   - Add the **users** table as a source.
   - Set the naming attribute to the *uid* and the object class to *inetorgperson*.
   - Map the *uid* attribute in the virtual entry to the **username** attribute in the **officepeople** table.
   - Create the *cn* attribute in the virtual entry from the **firstname** and **lastname** attributes in the **users** table.
   - Map the *surname* attribute in the virtual entry to the **lastname** attribute in the **users** table.

6. Open the **Sources** folder, and double-click the **users**'s source. Edit the fields to create reverse mappings from the table's **firstname** and **lastname** attributes back to the *cn*, *givenname*, and *surname* attributes, respectively.

   For more information on creating reverse mappings on sources, see *Section 7.3, "Creating and Editing Sources Manually"*.

7. In the **Sources** folder, double-click the **officepeople**'s source, and edit it to create reverse mappings from the table's **username** and and **offices** attributes to the *uid* and *l* attributes, respectively.

   For more information on creating reverse mappings on sources, see *Section 7.3, "Creating and Editing Sources Manually"*.

9.4.2. Creating a Nested Mapping Manually

To create a nested mapping, create two dynamic subtrees in the **directory.xml** file, one beneath the other in the directory hierarchy. The different tag and parameter values are listed in *Table 8.1, "Parameters for Subtree Entries"*.

1. Create a directory entry and source for the **offices** table. Two attributes, the **offices** and **description** attributes, are mapped to the *l* and *description* attributes in the virtual entry. A reverse mapping is given mapping the attributes back to the virtual directory.

```
<entry dn="l=...,ou=Locations,dc=Example,dc=com">

    <oc>locality</oc>
    <at name="l" rdn="true">
```

```
        <variable>offices.offices</variable>
    </at>
    <at name="description">
        <variable>offices.description</variable>
    </at>

    <source name="offices">
        <source-name>offices</source-name>
        <field name="offices">
            <expression>l</expression>
        </field>
        <field name="description">
            <expression>description</expression>
        </field>
    </source>

</entry>
```

This defines the entry beneath the **ou=Locations** subtree.

2. Create the directory entry and source for the users entries. This information is taken from the **users** table. The mapping takes the username from the **officepeople** table for the virtual entry's *uid* attribute and the first and last names from the user's entry in the **users** table.

   Also provide a reverse mapping from the **users** table attributes back to the **officepeople** table's **username** attribute and to the other user information in the **users** table.

```
<entry dn="uid=...,l=...,ou=Locations,dc=Example,dc=com">

    <oc>person</oc>
    <oc>inetOrgPerson</oc>
    <at name="uid" rdn="true">
        <variable>officepeople.username</variable>
    </at>
    <at name="cn">
        <expression>
if (users == void || users == null) return null;
return users.firstName+" "+users.lastName;
        </expression>
    </at>
    <at name="sn">
        <variable>users.lastName</variable>
    </at>

    <source name="users">
        <source-name>users</source-name>
        <field name="username">
            <variable>ug.username</variable>
        </field>
        <field name="firstName">
            <expression>
if (cn == void || cn == null) return null;
int i = cn.lastIndexOf(" ");
return cn.substring(0, i);
            </expression>
        </field>
        <field name="lastName">
```

```
            <expression>
if (cn == void || cn == null) return null;
int i = cn.lastIndexOf(" ");
return cn.substring(i+1);
            </expression>
        </field>
    </source>

</entry>
```

3. Lastly, define a reverse mapping for the **`officepeople`** table which links it back to the `l` attribute value drawn from the **`offices`** table.

```
<entry dn="uid=...,l=...,ou=Locations,dc=Example,dc=com">

    <source name="officepeople">
        <source-name>officepeople</source-name>
        <field name="l">
            <variable>offices.l</variable>
        </field>
        <field name="username">
            <variable>uid</variable>
        </field>
    </source>
</entry>
```

4. Restart Penrose Server. For example:

```
service vd-server restart
```

IMPORTANT

Always restart Penrose Server after editing the configuration file.

9.5. Joining Entities into a Single Virtual Entry

The same identity can have entries in multiple sources. Join mapping combines all of those separate entries into a single virtual entry. Basic mappings relate attributes to attributes; a join mapping combines entries. This joining is possible by using a *primary key*; this is a shared attribute value which is matched among all of the sources. If a source entry has that key, then it is included in the joined virtual entry; if not, it is skipped.

TIP

Mappings for virtual directory attributes and reverse mappings for source attributes can be configured in the **`directory.xml`** file, but more complex mappings can be configured in the **`mappings.xml`**, which allows BeanShell scripts to be called before and after mapping attributes to perform futher transformations on the virtual entry data. For more information, see *Section 9.6, "Creating Advanced Mappings"*.

9.5.1. Setting up Join Mapping in Penrose Studio

Joining entries in Penrose Studio is the same process as creating a basic mapping in *Section 9.3, "Configuring Basic Mapping"*. The difference is from what sources the attributes are pulled. The different mappings, however, reflect different attributes from different sources, and all of the sources need a reverse mapping back to the entry. Another important distinction: the reverse mappings on some of the sources map back to the *primary source*, not to the virtual entry, in order to map the primary keys.

For example, Example Company has two databases which contain group information. The **`groups`** database contains the name of the group (**`groupname`**) and the description (**`description`**). The **`users`** database contains all of the user information, including the username (**`username`**) and a list of each group to which the user belongs (**`groupname`**).

In this case, the *primary key* is the **`groupname`** attribute, which is present in both databases. The *primary source* is whichever source is considered authoritative for the primary key. This is the source which all other sources reference in the reverse mapping back to the primary key. In this case, this is obviously the **`groups`** database. In other situations, this can be arbitrary, meaning whichever source is best suited as being a source. So, if an employee has entries in a database, NIS server, LDAP server, and Active Directory domain, any of those sources can be selected as the primary source.

To create the join mapping for the group in Penrose Studio:

1. Open the subtree entry editor. In this case, the subtree is a dynamic subtree for Example Company groups, **`cn=...,ou=Groups,dc=example,dc=com`**.

2. For Example Company's group entry, there are four LDAP attributes which will be mapped:

   - *`cn`* attribute for the for the **`groupname`** database attribute in the **`groups`** database, to supply the entry DN

   - *`description`* attribute for the group description from the **`groups`** database

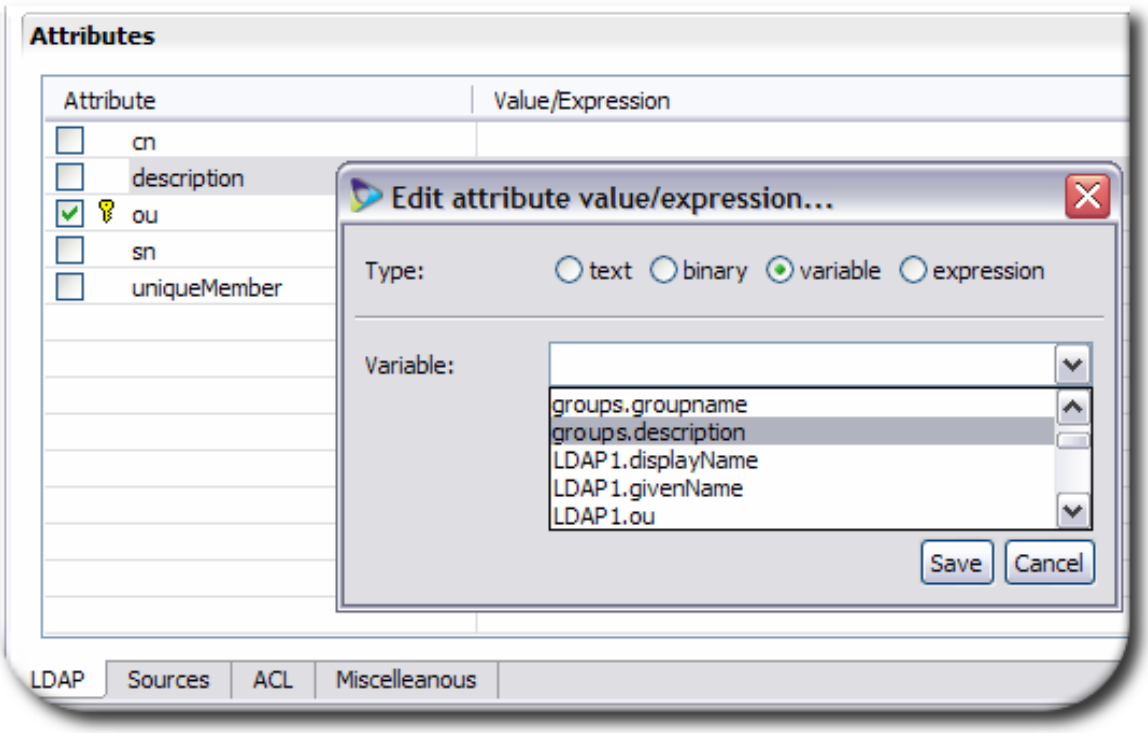
   - *`uniqueMember`* attribute for the list of members contained in the **`groupname`** attribute in the **`users`** database

   - *`uid`* attribute for the **`username`** database attribute in the **`users`** database

NOTE

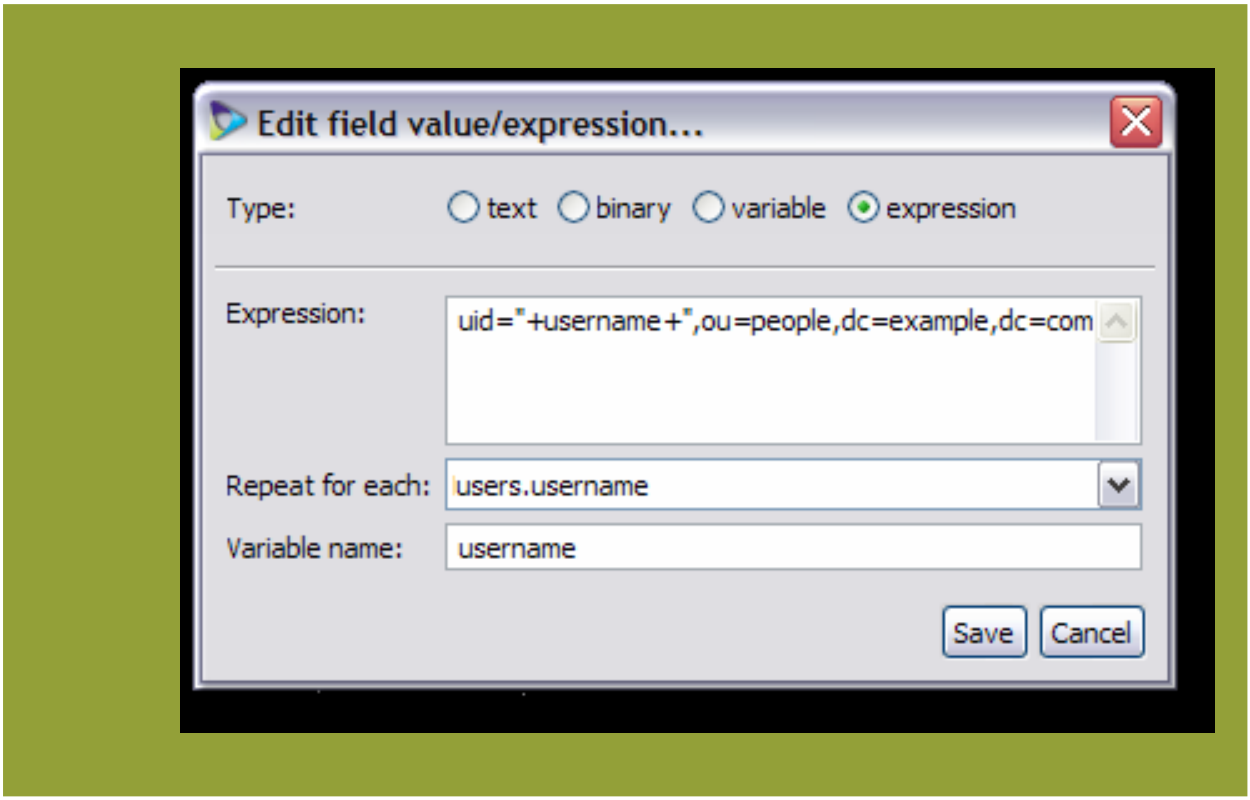
The sources must all already be configured for the directory subtree. To configure sources, see *Chapter 7, Configuring Data Sources*.

3. First, configure the basic mapping between the virtual directory attributes and the sources. In the **Attributes** tab, select the attributes and assign them to the appropriate database attribute.



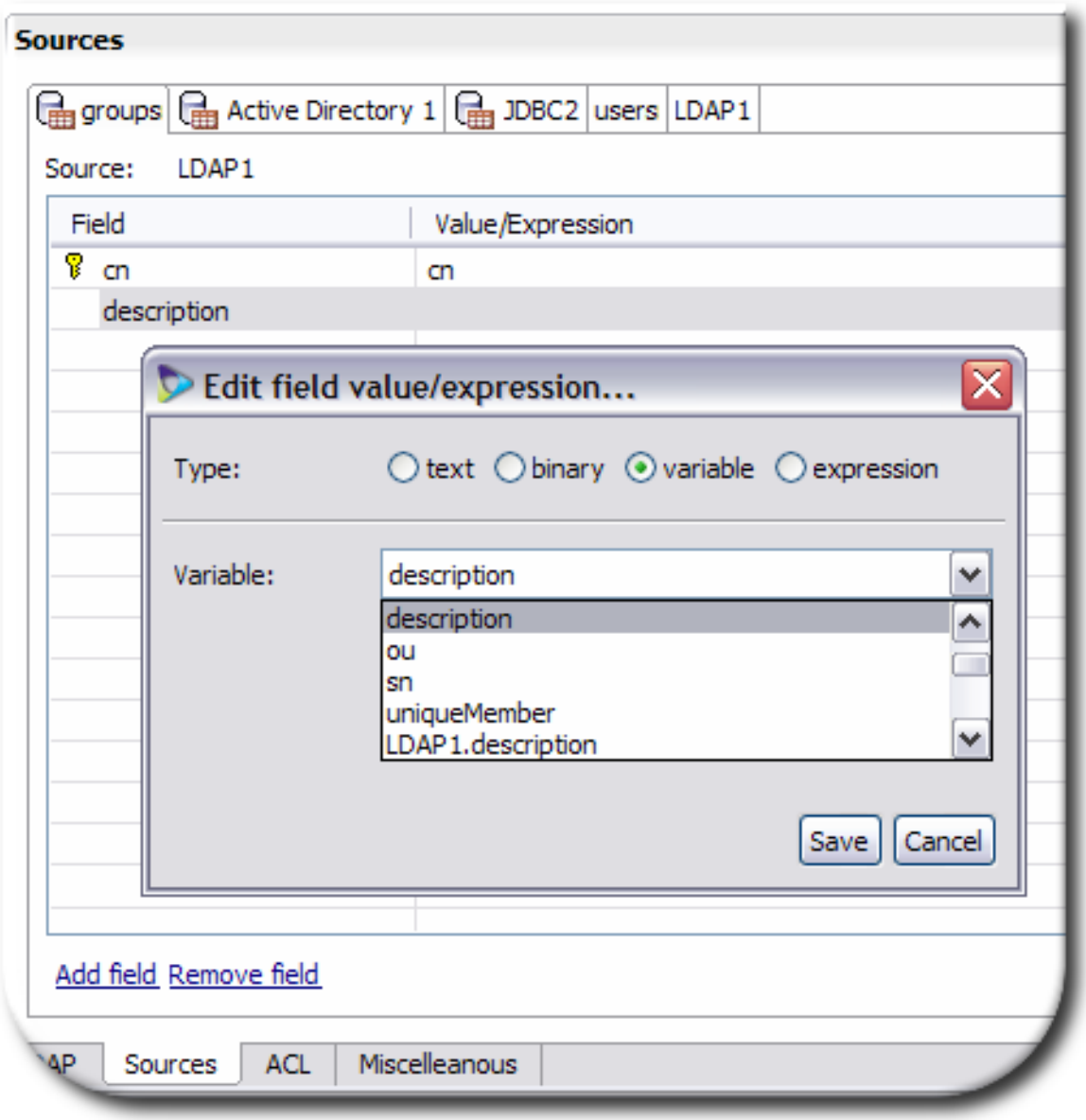
TIP

The format of Example Company's *uniqueName* values is **uid=***username*, **ou=people,dc=example,dc=com**. To configure that, the *uniqueName* value is an expression which builds the DN from the database's **username**:



4. Open the **Sources** tab for the subtree entry.

5. Configure the reverse mapping for the primary source. In this case, the primary source is the **`groups`** database.



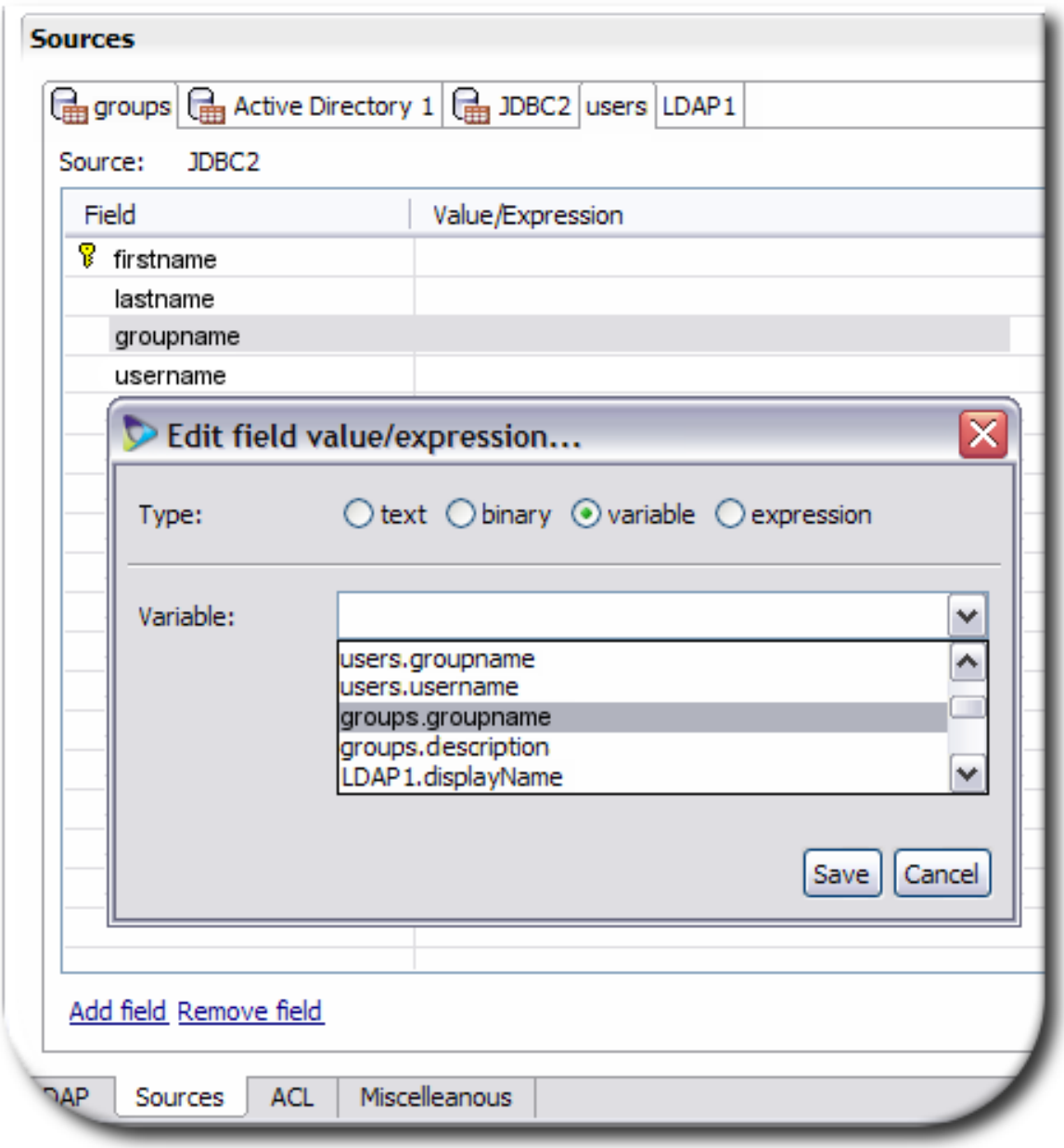
- The primary key is the **`groupname`** attribute. Map the primary key back to the corresponding virtual directory attribute, which is *cn*.

- Map the **`description`** attribute back to the virtual directory attribute, *`description`*.

NOTE

Variables have the format *source_name.attribute*. If there is no source given, the attribute belongs to the virtual directory.

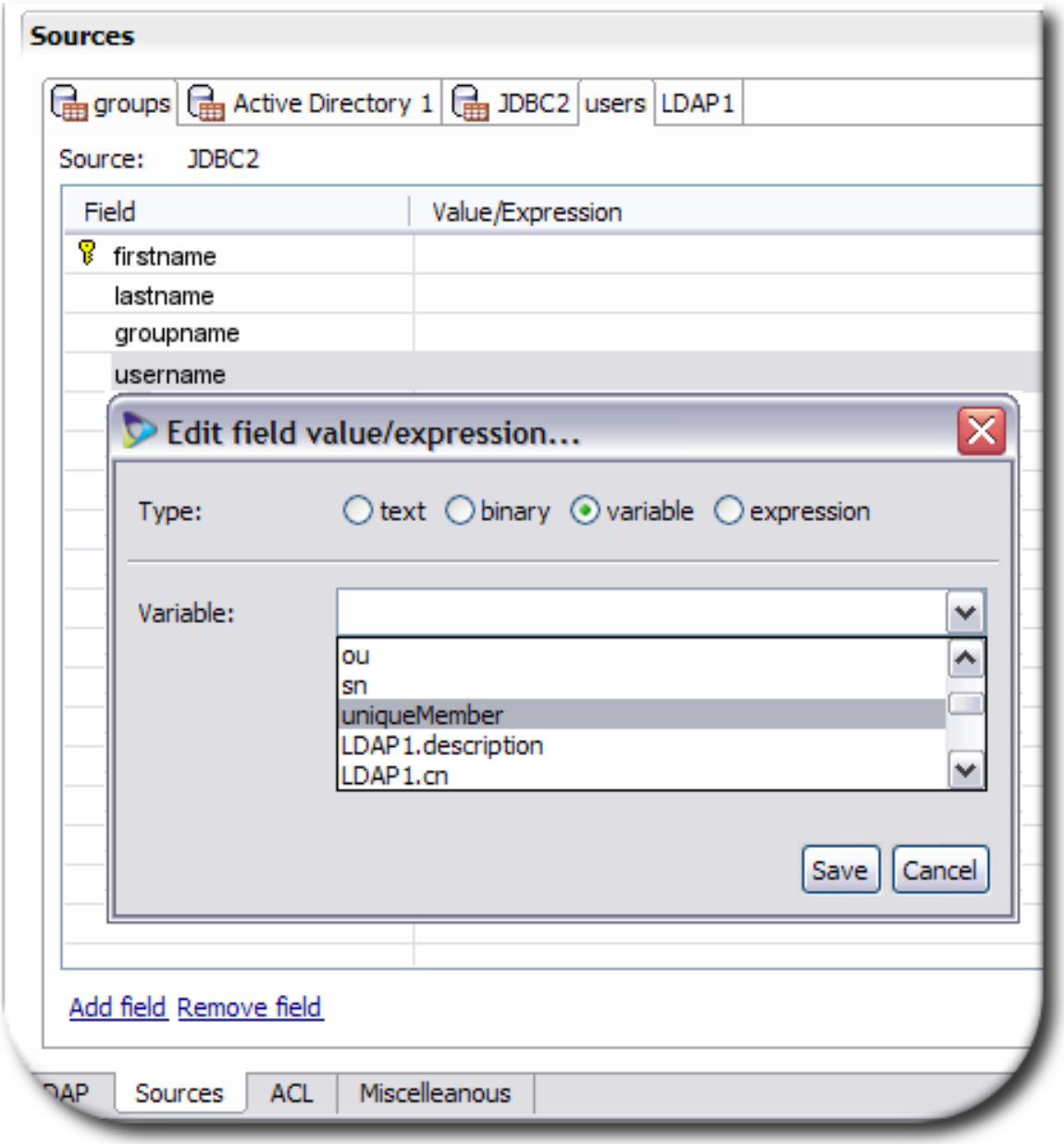
6. On the **users** database source, map the existing **groupname** database attribute to the primary key attribute on the **groups** source.



IMPORTANT

Map the primary key to the primary source for every source being joined.

7. On the **users** database source, configure the basic mapping for the **username** database attribute to the *uniqueMember* attribute.



8. Close the editor window and save the changes when prompted.

9.5.2. Manually Joining Entries

Join mapping is configured in the **`directory.xml`** file. For each attribute, a **`<at>`** expression is added to the subtree and then reverse mappings are added to every source which is involved in the joining. Additionally, reverse mappings are added to every source linking back to the *primary key* in the *primary source*. If an entry does not have an exact match to the primary key in the primary source, then that entry is not included in the joined entry.

Joining entries can consolidate scattered entries which refer to a single person or entity. Example Company, for instance, stores group names and descriptions in the **groups** database, and group members are determined by the group attribute on user entries in the **users** database. The common key in both of those databases is the **groupname** attribute; in the **groups** database, this identifies the group; in the **users** database, it shows group membership. Example Company wants to create a single group entry in the virtual directory which contains the group definition and the members list.

- Mapping the **groupname** attribute in the **groups** database to the *cn* attribute in the entry to generate the DN for the group

- Mapping the **description** attribute in the **groups** database to the *description* attribute in the virtual entry

- Mapping the **username** attribute in the **users** database to the *uniqueMember* attribute in the virtual entry to create the list of group members

- Mapping the **groupname** attribute for the **users** database source to the *groupname* attribute in the **groups** database source since this primary, shared key is how the entries are recognized, mapped, and merged into the virtual entry

To join entries manually, edit the subtree entry in the **directory.xml** file. The different tag and parameter values are listed in *Table 8.1, "Parameters for Subtree Entries"*.

1. Add the attributes to be mapped to the main entry. For Example Company's group mapping:

```
<entry dn="cn="...",dc=ou=Groups,dc=example,dc=com">
     <at name="cn" rdn="true">
          <variable>groups.groupname</variable>
     </at>
     <at name="description">
          <variable>groups.description</variable>
     </at>
     <at name="uniqueMember">
          <expression foreach="users.username" var="username">
               "uid="+username+",ou=Users,dc=Example,dc=com"
           </expression>
     </at>
```

2. In the source entry for the **groups** database, add the reverse mapping for the *cn* and *description* attributes:

```
<source>
     <source-name>groups</source-name>
...
     <field name="groupname">
          <variable>cn</variable>
     </field>

     <field name="description">
          <variable>description</variable>
```

```
     </field>
...
</source>
```

NOTE

Variables have the format *source_name.attribute*. If there is no source given, the attribute belongs to the virtual directory.

3. In the source entry for the **`users`** database, add the reverse mapping for the *`uniqueMember`* attributes:

```
<source>
     <source-name>users</source-name>
...
     <field name="username">
          <variable>uniqueMember</variable>
     </field>
...
</source>
```

4. Add the mapping from the **`users`** database to the primary key in the **`groups`** database, the **`groupname`** attributes. This is required for Penrose Virtual Directory to recognize which database entries are to be joined. Any user entry with a matching group entry will be mapped into the group members' list.

```
<source>
     <source-name>users</source-name>
...
     <field name="groupname">
          <variable>groups.groupname</variable>
     </field>
...
</source>
```

5. Restart the Penrose Server.

```
service vd-server restart
```

IMPORTANT

Always restart Penrose Server after editing the configuration file.

9.6. Creating Advanced Mappings

The format of advanced mappings is similar to the format of a source reverse mapping. The mapping is configured in *fields* which identify the attribute and a matching rule of some kind. Advanced mappings can also set whether the attribute is required to be present in the entry, conditions which must be met to evaluate the attribute, and scripts which can be run before or after the attributes are processed.

Mapping plays a very important role in identity federation because mapping is the way that the directory structure of the federated directory is defined, functioning as the object class and attribute declarations in the **`directory.xml`** configuration.

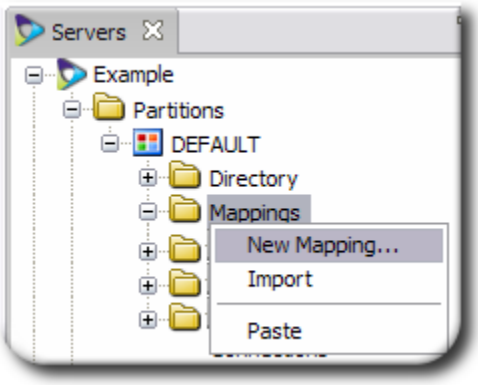
For identity federation, mappings are processed when local identities are imported into the global repository.

9.6.1. Mapping Attribute Fields

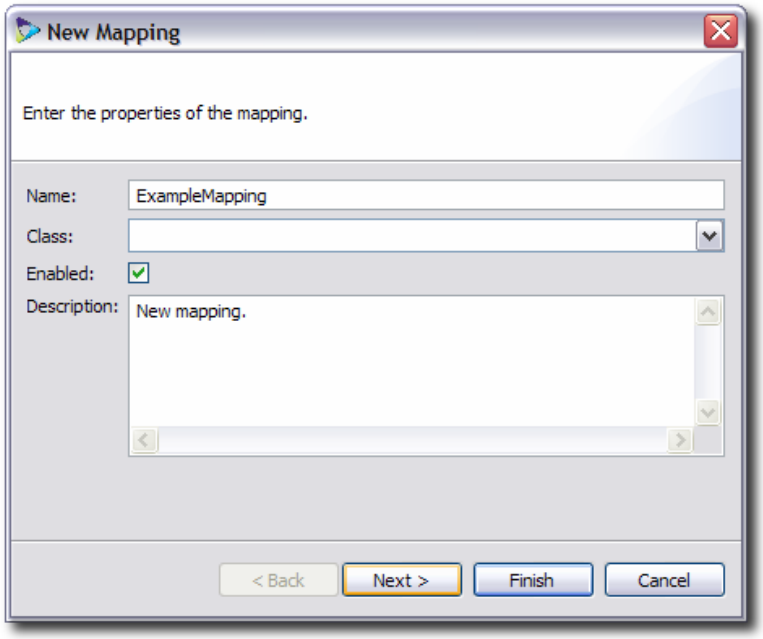
To create a new mapping in Penrose Studio:

1. Open the server entry in Penrose Studio, and expand the **Partitions** folder.

2. Right-click the **Mappings** folder, and select **New Mapping...**.



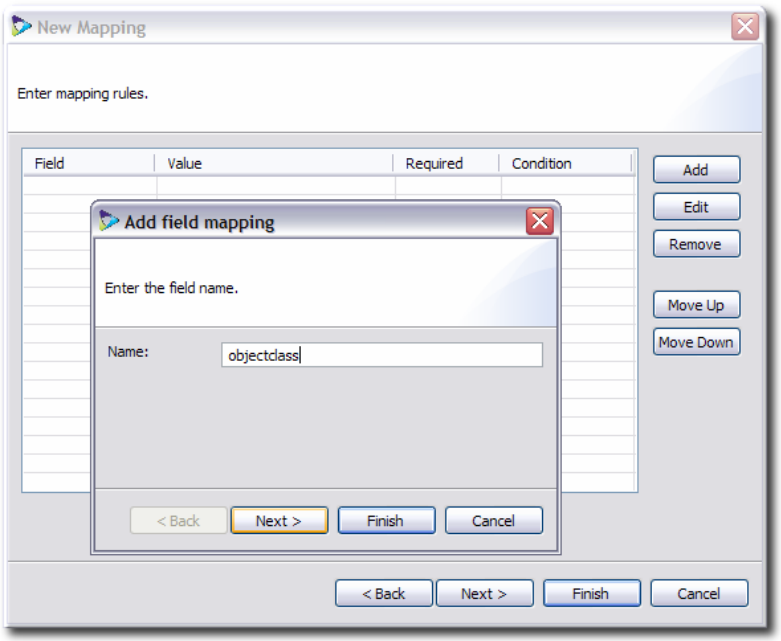
3. Name the mapping, and, optionally, give a description for it.



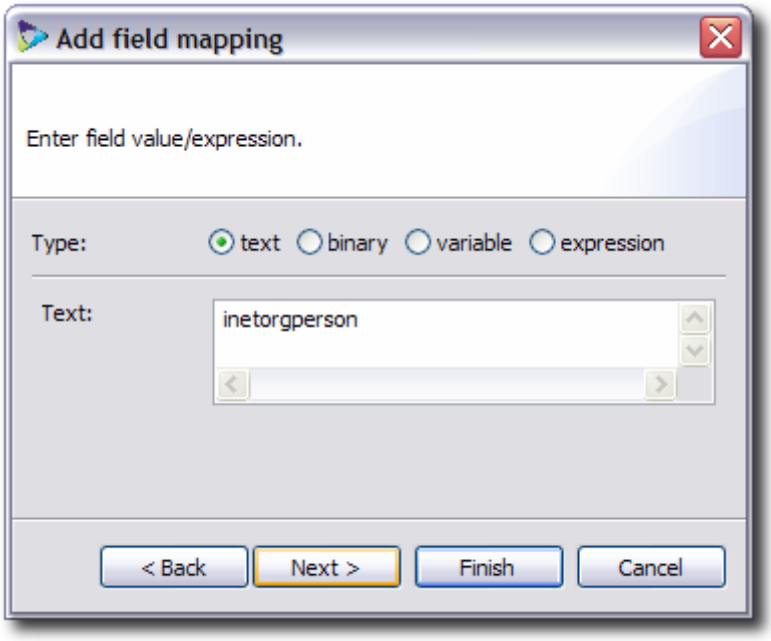
4. Create a mapping for each object class which will be available to the global entries.

   a. Click the **Add** button.

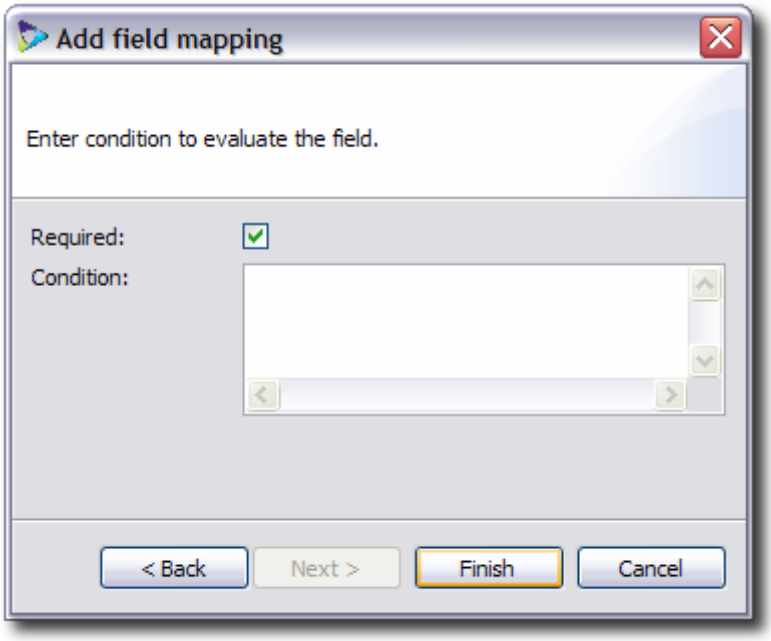
   b. For an object class, enter the attribute name *`objectclass`*.



   c. For the object classes, select the **text** radio button to set an exact value, and enter the name of the object class.



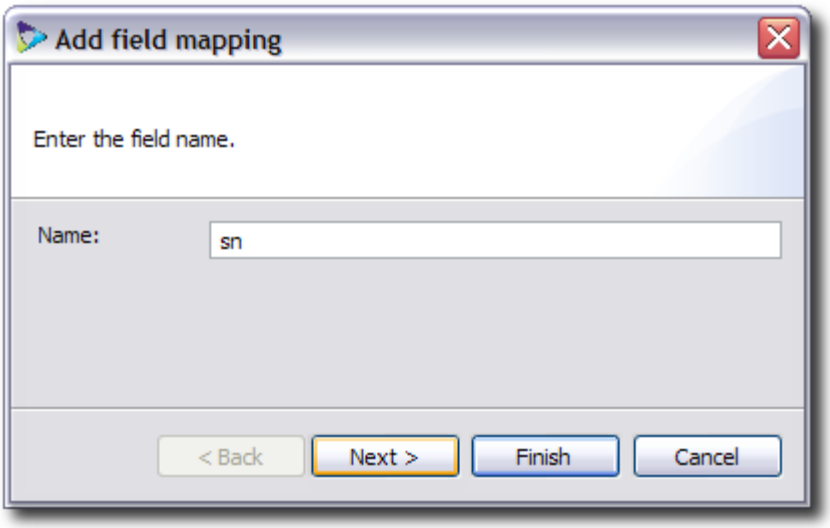
d. Select the **Required** checkbox so that object class is required for the global entry, and click finish.



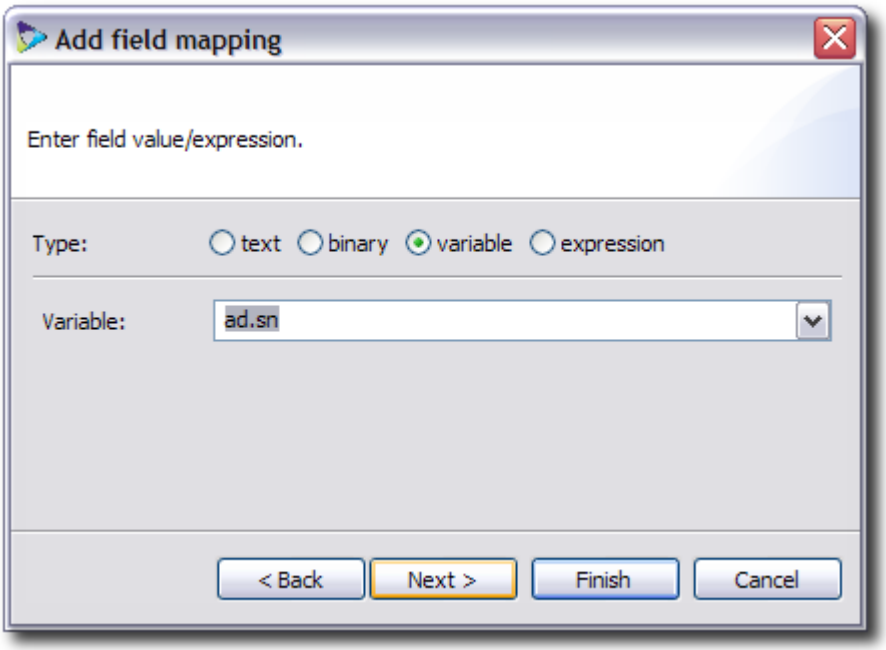
5. Create a mapping for each attribute which will be available to the global entries.

   a. Click the **Add** button.

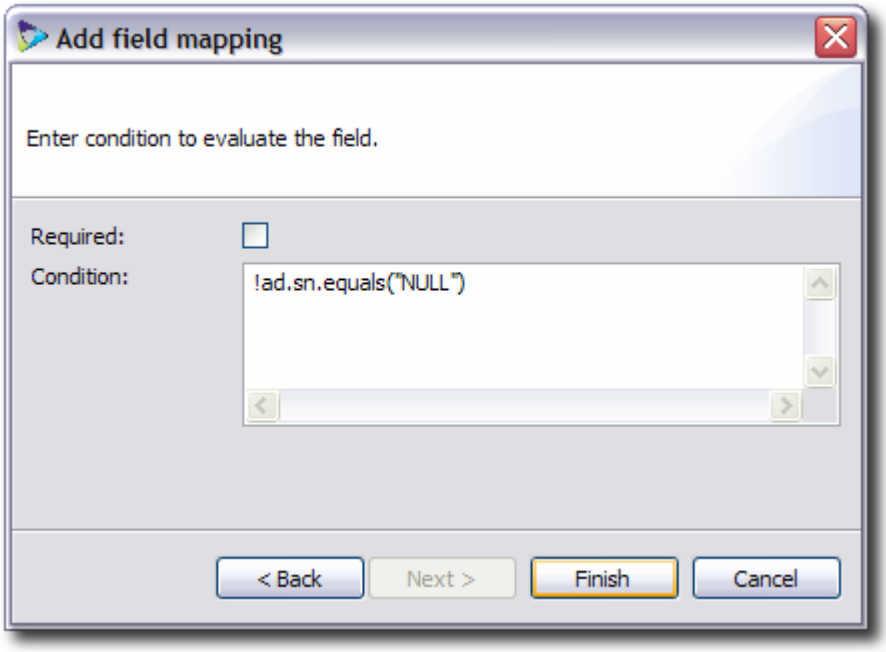
b. Enter the name of the attribute.



c. For the attributes, select the **variable** radio button to use a value from one of the repositories, and enter the value in the form *repo_name.attribute*.



d. Un-check the **Required** box and enter the conditions where the attribute is to be processed. For example, to control the behavior of Penrose Virtual Directory if an attribute is empty, use a condition such as **`!ad.sn.equals("NULL")`** to skip the *sn* attribute in the repository named **ad** if there is no value set.



NOTE

There can be more than one mapping configured for any attribute. The order that the attributes are listed is the order which they are processed; if one attribute mapping is skipped, the value is taken from the next attribute listed.

This *attribute preference* makes it possible to configure mapping even for single-valued attributes to every source. For example, the first instance can be taken from the global repository; if the attribute does not exist there, it can be taken from the Active Directory local repository, then from the Red Hat Directory Server repository, then from the NIS repository, until an attribute value is found.

6. Save the new mapping entry.

9.6.2. Using Scripts

Identity federation mappings allow scripts to be run before and after attributes are processed. These scripts can be used to transform, manipulate, or evaluate data in the repositories before being accessed. The only scripting language allowed is BeanShell, which cleanly handles Java objects, including the MBeans that make up Penrose Virtual Directory objects.

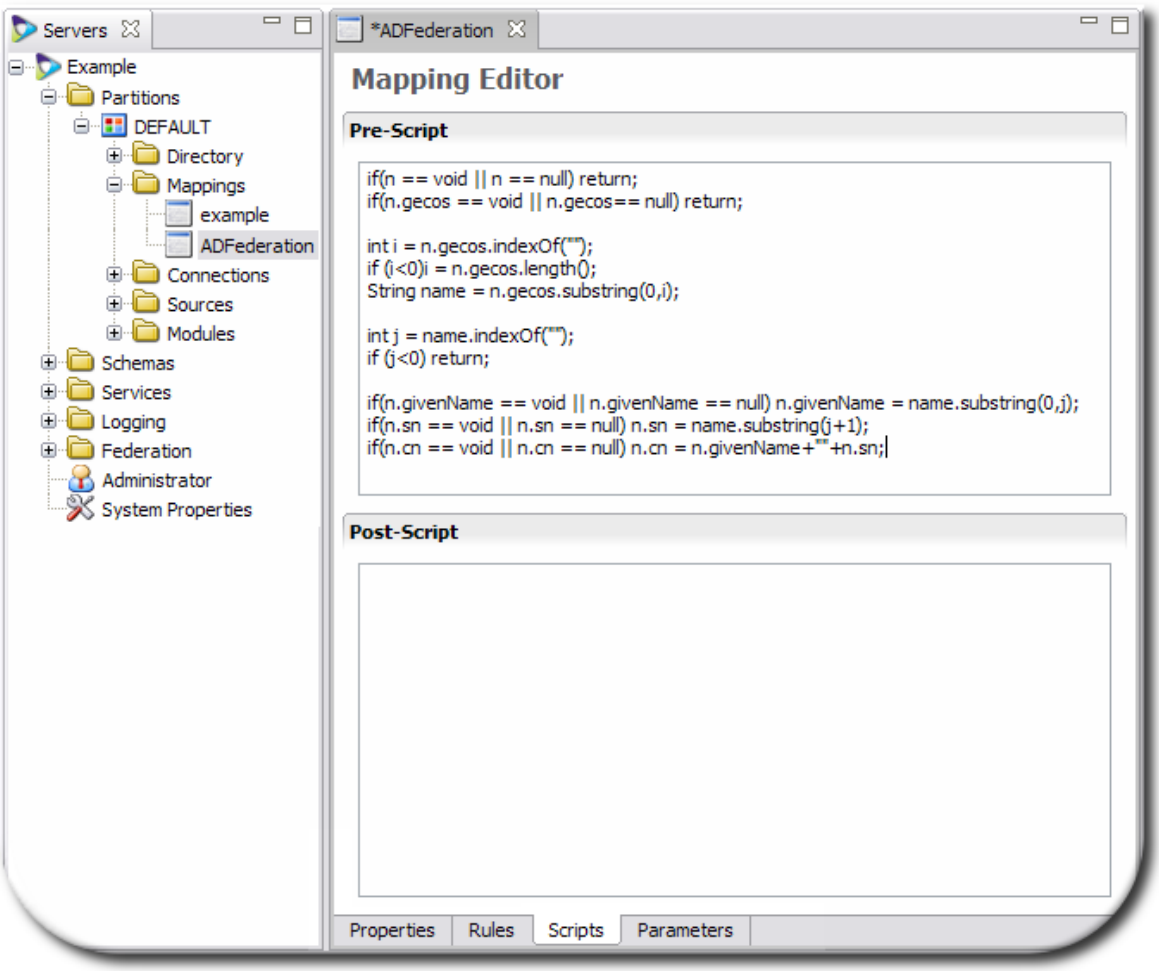
To add a script to a mapping:

1. Open the **Mappings** folder under the **Partitions** folder.

2. Double-click the mapping entry to open the entry editor.

3. Click the **Scripts** tag.

4. Paste in the BeanShell script in the **Pre-Script** or **Post-Script** text area.



5. Close the editor, and save the changes when prompted.

For more information on using and writing scripts in BeanShell, see the BeanShell user documentation at *http://1www.beanshell.org/1manual/1bshmanual.html*.

9.6.3. Mapping Attribute Fields and Scripts Manually

To configure a similar mapping manually, add a mapping entry for each object class and a mapping entry for each attribute to the **`mappings.xml`** file. The mapping file, like a source entry for a virtual directory configuration, provides mappings between the global and local repositories by creating **`<field>`** statements. The **`<field>`**s are processed in order, from top to bottom. A mapping is illus-

trated in *Example 9.2, "Annotated Mapping Entry"*.

IMPORTANT

Always restart Penrose Server after editing the configuration file. For example:

```
service vd-server restart
```

```
<mappings>  the main tag for the entire file
...
  <mapping name="..."> the mapping entry
      <field name="..." required="..."> an attribute entry with the LDAP
name
          <mapping_type>...</mapping_type>  the mapping definition for
the attribute
           <condition>...</condition> a BeanShell script that sets rules
on processing attributes
      </field>
      <pre>  an optional BeanShell script, run before attributes are pro\
cessed
           ....
      </pre>
      <post> an optional BeanShell script, run after attributes are pro\
cessed
           ....
      </post>
  </mapping>
...
</mappings>
```

Example 9.2. Annotated Mapping Entry

For example, to create a mapping for the *inetorgperson* object class in the global repository and the *sn* attribute in an Active Directory local repository, two fields are added to the mapping. The **<pre>** script extracts the *cn*, *givenName*, and *sn* attributes from the *gecos* attribute in the NIS server before processing the attribute fields.

```
<mappings>
...
     <mapping name="example">
           <field name="objectclass" required="true">
                  <constant>inetorgperson</constant>
           </field>
           <field name="sn">
                  <variable>ad.sn</variable>
                  <condition>!ad.sn.equals("NULL")</condition>
           </field>
           <pre>
                   if(n == void || n == null) return;
                   if(n.gecos == void || nis.gecos== null) return;
```

```
                    int i = nis.gecos.indexOf("");
                    if (i<0)i = nis.gecos.length();
                    String name = nis.gecos.substring(0,i);

                    int j = name.indexOf("");
                    if (j<0) return;

                    if(n.givenName == void || nis.givenName == null)
nis.givenName = name.substring(0,j);
                    if(n.sn == void || nis.sn == null) nis.sn =
name.substring(j+1);
                    if(n.cn == void || nis.cn == null) nis.cn =
nis.givenName+""+n.sn;
          </pre>
     </mapping>
...
</mappings>
```

The parameters and tags for the **`mappings.xml`** file are listed in *Table 9.1, "Parameters for Mapping Entries"*.

Tag or Parameter	Description	Example
<mapping>	Opens and closes the entire mapping entry.	

<field>	Defines a reverse mapping from a source attribute to a virtual entry attribute. Like the **<at>** mapping, this has a name> parameter to identify the virtual entry's LDAP attribute and a sub-tag which defines the mapping from the source attribute to the virtual attribute: constant, variable, or expression.	`<field name="cn">`
		`<variable>global.cn</variable>`
		`</field>`

name="*attribute*"	With the **<mapping>** tag, this contains the name of the mapping entry. With the **<field>** tag, this contains the LDAP attribute which is being set for the mapping field.	`<field name="cn">`
<*mapping_type*>	These are any of three sub-tags which are used with a **<field>** tag to define the method of mapping attributes: • <constant> (specific text value) • <variable> (link to the repository and attribute in the format	`<variable>LDAP1.cn</variable>`

Tag or Parameter	Description	Example
	repo_name.attribute) • <expression> (BeanShell script to generate or extract the value)	
<condition>	Defines any conditions which must be met to process the mapping, such as how to handle null attributes.	`<condition>!ad.sn.equals("NULL")</condition>`

<pre>	Contains a BeanShell script to run before the attribute mappings are processed.	`<pre>`
		`if(n.givenName == void || nis.givenName == null)`
		`nis.givenName = name.substring(0,j);`
		`if(n.sn == void || nis.sn == null)`
		`nis.sn = name.substring(j+1);`
		`if(n.cn == void || nis.cn == null)`
		`nis.cn = nis.givenName+""+n.sn;`
		`</pre>`

<post>	Contains a BeanShell script to run after the attribute mappings are processed.	
<description>	Contains a text description of the mapping.	`<description>for generating cn values</description>`
<mapping-class>	Gives a Java class to use for the mapping.	

Table 9.1. Parameters for Mapping Entries

Configuring Identity Federation

This chapter describes how to configure identity federation for NIS, LDAP, and Active Directory sources.

10.1. About Identity Federation

Identity federation or *identity linking* is a method to unite entries from Active Directory, LDAPv3, or NIS servers. This method can be used to synchronize these types of servers or to simplify a NIS migration.

Identity federation is similar to mapping, which takes source entries to create a virtual directory entry, but is for different source environments. Identity linking, like join mapping, takes entities from different sources and combines them to create a single global directory entry. (Unlike join mapping, identity linking does not depend on having a single common attribute shared among the other source entries.)

Not every infrastructure can use a purely virtual directory because of requirements for migrations, synchronization, or other network changes. In these cases, Penrose Virtual Directory provides identity linking. Identity linking (or federation) creates a hybrid virtual directory and meta directory by using a global repository to combine and synchronize entries from the different sources. Local repositories (analogous to sources in the virtual directory configuration) are linked to the global repository. Using a mapping tool, the local identities are then matched to entries in the global repository.

Identity linking works with a smaller number of source types than virtual directory configuration. These can be any of three things:

- LDAPv3 servers, such as Red Hat Directory Server
- Active Directory
- NIS servers

For example, in *Figure 10.1, “Merging Entries through Identity Linking”*, there is no common attribute among the NIS, LDAP, and Active Directory identities. With identity linking, there is a *global repository*, a separate and authoritative LDAP source for entries. In this figure, a NIS source is used to create the initial global entry. After that global repository is defined, entries from other sources (*local repositories*) are linked to the global entry either manually or through matching rules.

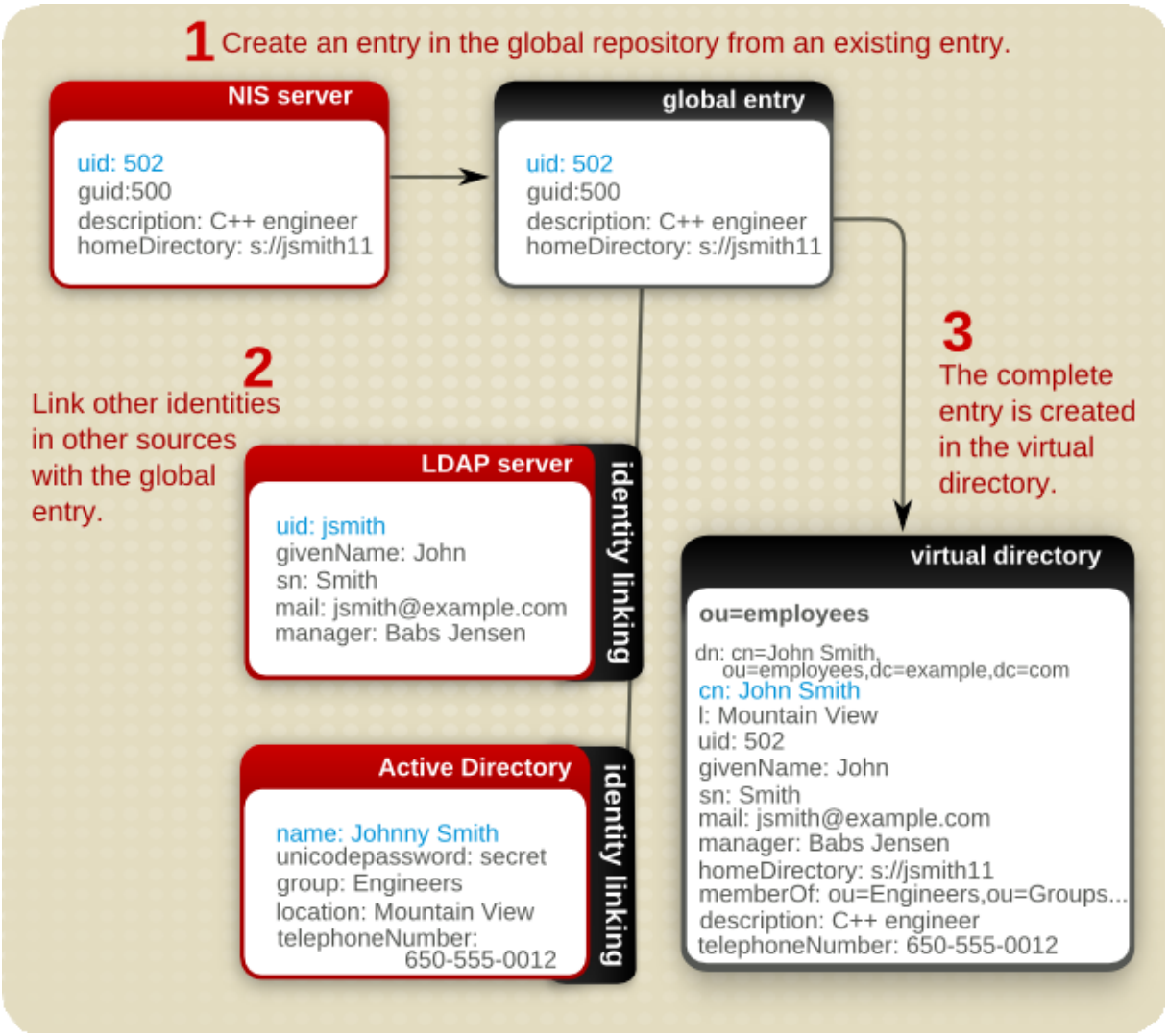


Figure 10.1. Merging Entries through Identity Linking

There are several important differences between join mapping and identity linking:

- Identity linking can combine entries which do not share a common attribute, using a tool to identify and link entries between sources, as described in *Section 10.6, "Linking Identities"*.

- Identity linking is configured through a plug-in which works with NIS, LDAP, and Active Directory sources.

Identity federation uses a centralized, specialized partition to create a *federation domain*. Each source repository is configured as its own sub-partition, which both defines its own connections, sources and mappings and also points back to the federation domain.

Federating or linking identities has the following process:

1. Create the federation domain (*Section 10.2, "Creating the Federation Domain"*).

2. Create templates for the different repository types (*Section 10.3, "Creating Templates"*).

   Alternatively, it is also possible to add individual local repositories through Studio (*Section 10.4, "Adding LDAP Local Repositories through Penrose Studio"* and *Section 10.5, "Adding NIS Local Repositories Using Penrose Studio"*).

3. Configure attribute mapping (*Section 9.6, "Creating Advanced Mappings"*).

4. Link local identities to global identities (*Section 10.6, "Linking Identities"*).

5. *Optionally.* Migrate and synchronize NIS information (*Section 10.9, "Synchronizing (or Migrating) NIS Data to LDAP"*).

6. Resolve any conflicts between UID and GID numbers among global entries (*Section 10.7, "Resolving UID and GID Conflicts"*).

10.2. Creating the Federation Domain

Identity federation works by creating relationships between different repositories of information. The primary organization for identity federation is a domain; each repository of information belongs to that domain. The domain and repositories themselves are essentially partition entries. The federation

The first step for configuring identity federation is creating the federation domain.

NOTE

The federation domain must be created manually.

1. Open the partitions directory.

```
cd /opt/vd-server-2.0/partitions
```

2. Create a new directory; the name of the directory is the name of the new federation domain. For example:

```
mkdir ExDomain/
```

3. Open the new federation directory.

```
cd ExDomain/
```

4. Create a **DIR-INF/** directory.

```
mkdir DIR-INF/
```

5. In the **DIR-INF/** directory, copy over the regular partition files, **partition.xml**, **connections.xml**, **sources.xml**, **modules.xml**, and **directory.xml**. For example:

```
cd DIR-INF/
cp /opt/vd-server-2.0/conf/*.xml .
```

6. Edit the **partition.xml**; this file is blank.

```
vim partition.xml

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE partition PUBLIC "-//Penrose/DTD Partition 2.0//EN" "ht\
tp://penrose.safehaus.org/dtd/partition.dtd">

<partition />
```

7. Create the **federation.xml** file. This file will eventually hold the entries for the global and local repositories.

   For example:

```
vim federation.xml

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE directory PUBLIC "-//Penrose/DTD Directory 2.0//EN" "ht\
tp://penrose.safehaus.org/dtd/directory.dtd">

<federation>

</federation>
```

8. Edit the **modules.xml** file to use the federation modules. There are several different modules to load for federation, including the main federation module, modules for the NIS and LDAP repositories, and plug-ins to manage user and group conflicts.

```
vim modules.xml

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE modules PUBLIC "-//Penrose/DTD Modules 2.0//EN" "ht\
tp://penrose.safehaus.org/dtd/modules.dtd">

<modules>
  <module name="Federation">
```

```
<module-class>org.safehaus.penrose.federation.module.FederationModule</mo
dule-class>
    <parameter>
      <param-name>config</param-name>
      <param-value>federation.xml</param-value>
    </parameter>
    <parameter>
      <param-name>repositoryTypes</param-name>
      <param-value>LDAP,NIS</param-value>
    </parameter>
    <parameter>
      <param-name>conflictDetections</param-name>
      <param-value>Users,Groups</param-value>
    </parameter>
  </module>

  <module name="LDAP">
<module-class>org.safehaus.penrose.federation.module.LDAPRepositoryModule
</module-class>
  </module>

  <module name="NIS">
<module-class>org.safehaus.penrose.federation.module.NISRepositoryModule<
/module-class>
  </module>

  <module name="Users">
<module-class>org.safehaus.penrose.federation.module.ConflictDetectionMod
ule</module-class>
    <parameter>
      <param-name>objectClass</param-name>
      <param-value>person</param-value>
    </parameter>
    <parameter>
      <param-name>attribute</param-name>
      <param-value>uidNumber</param-value>
    </parameter>
  </module>

  <module name="Groups">
<module-class>org.safehaus.penrose.federation.module.ConflictDetectionMod
ule</module-class>
    <parameter>
      <param-name>objectClass</param-name>
      <param-value>posixGroup</param-value>
    </parameter>
    <parameter>
      <param-name>attribute</param-name>
      <param-value>gidNumber</param-value>
    </parameter>
  </module>

</modules>
```

9. Restart the server.

```
service vd-server start
```

When the server is restarted, the new federation domain shows up as a partition under the **Partitions** folder and is listed under the **Federation** folder, with **NIS** and **LDAP** subfolders for the repositories.

10. Configure the **connections.xml** file to point to the global repository, which is an LDAPv3 server such as Red Hat Directory Server or Active Directory. For example:

```
vim connections.xml

<connections>
  <connection name="LDAP">
    <adapter-name>LDAP</adapter-name>
    <parameter>
      <param-name>java.naming.provider.url</param-name>
      <param-value>ldap://server1.example.com:1389/
ldap://server2.example.com:1389/</param-value>
    </parameter>
    <parameter>
      <param-name>java.naming.security.principal</param-name>
      <param-value>cn=Directory Manager</param-value>
    </parameter>
    <parameter>
      <param-name>java.naming.security.credentials</param-name>
      <param-value>secret</param-value>
    </parameter>
    <parameter>
      <param-name>java.naming.ldap.attributes.binary</param-name>
      <param-value>SeeAlso</param-value>
    </parameter>
  </connection>
</connections>
```

The possible parameters to configure connections are listed in *Table 6.1, "Connection Configuration Values"*.

11. Edit the **directory.xml** file. Only two virtual directory entries are necessary: one for the base DN (**dc=example,dc=com**) and one for the user-facing entry. Users will always access this subtree. On the backend, either the global repository or the local repository can be chained to that subtree. During a NIS migration, for example, this subtree can be chained to **ou=nis**, the NIS repository. After migration is complete, the subtree can be chained to **ou=global**. Users and LDAP clients, however, only have to access one subtree.

NOTE

The configuration in the **directory.xml** file depends on the formatting of the virtual directory tree and your infrastructure.

For example:

```
vim directory.xml
```

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE directory PUBLIC "-//Penrose/DTD Directory 2.0//EN" "ht\
tp://penrose.safehaus.org/dtd/directory.dtd">

<directory>
  <entry dn="dc=example,dc=com">
    <oc>dcObject</oc>
    <at name="dc" rdn="true">
      <constant>example</constant>
    </at>
    <oc>o</oc>
    <at name="o" rdn="true">
      <constant>example</constant>
    </at>
    <aci>
      <permission>rs</permission>
    </aci>
  </entry>

  <entry dn="ou=federation,dc=example,dc=com">
    <oc>organizationalUnit</oc>
    <at name="ou" rdn="true">
      <constant>federation</constant>
    </at>
  </entry>
</directory>
```

Once the federation domain entries are created, the federation configuration, including modules and containers for sources, directories, and mappings, is available in Penrose Studio.

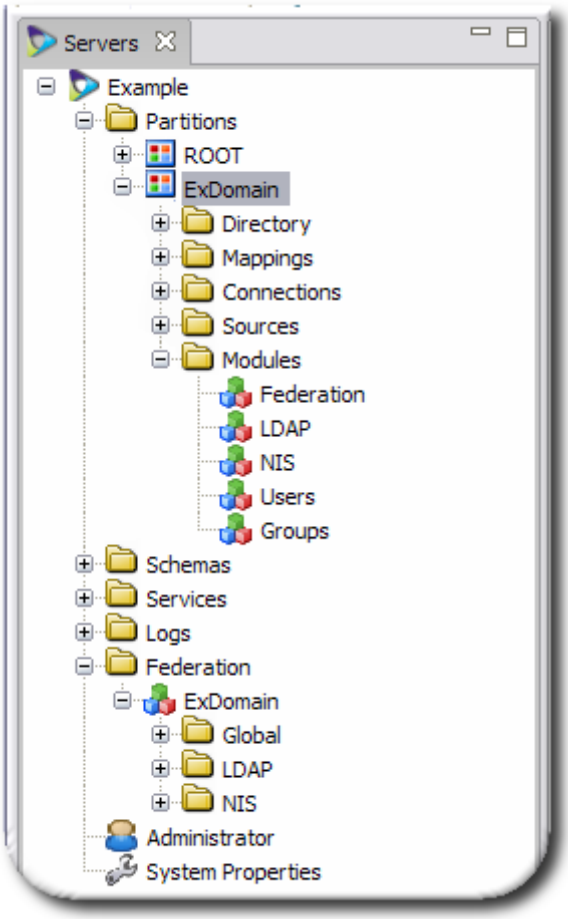


Figure 10.2. Federation Domain Entries

10.3. Creating Templates

Penrose Server can use templates to automatically generate repositories that have consistent mappings, directory structures and modules.

10.3.1. Creating Global Templates

1. Open the federation domain partition directory.

```
cd /opt/vd-server-2.0/partitions/ExDomain/
```

2. Create a templates directory.

```
mkdir templates/
```

3. Open the new templates directory.

```
cd templates/
```

4. Inside the templates directory, create a new partition-style directory for the global repository. For example:

```
mkdir global/
mkdir global/DIR-INF/
```

5. Open the **DIR-INF** directory.

```
cd global/DIR-INF/
```

6. Create a new **partition.xml** file that points to the federation domain and an entry for the global entry in the **federation.xml** file.

```
vim partition.xml

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE partition PUBLIC "-//Penrose/DTD Partition 2.0//EN" "ht\
tp://penrose.safehaus.org/dtd/partition.dtd">

<partition depends="ExDomain,ex_global">

</partition>
```

7. Create a new **directory.xml** file that defines the directory tree and entries for the global subtree.

NOTE

The configuration in the **directory.xml** file depends on the formatting of the virtual directory tree and your infrastructure.

For example:

```
vim directory.xml

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE directory PUBLIC "-//Penrose/DTD Directory 2.0//EN" "ht\
tp://penrose.safehaus.org/dtd/directory.dtd">

<directory>
```

```
  <entry dn="ou=global,dc=example,dc=com">
    <oc>organizationalUnit</oc>
    <at name="ou" rdn="true">
      <constant>global</constant>
    </at>
    <aci subject="self">
      <permission>rws</permission>
    </aci>
    <aci>
      <target>ATTRIBUTES</target>
      <attributes>userPassword</attributes>
      <action>deny</action>
      <permission>rs</permission>
    </aci>
    <aci>
      <permission>rs</permission>
    </aci>
  </entry>

  <entry dn="ou=Users,ou=global,dc=example,dc=com">
    <oc>organizationalUnit</oc>
    <at name="ou" rdn="true">
      <constant>Users</constant>
    </at>
  </entry>

  <entry dn="uid=...,ou=Users,ou=global,dc=example,dc=com">
<entry-class>com.analog.penrose.directory.GlobalUserEntry</entry-class>
    <mapping-name>virtual_users</mapping-name>
    <at name="uid" rdn="true">
      <variable>g.uid</variable>
    </at>
    <source alias="g" bind="required">
      <partition-name>example</partition-name>
      <source-name>users</source-name>
      <mapping-name>virtual_users_to_global_users</mapping-name>
      <field name="uid" primaryKey="true">
        <variable>rdn.uid</variable>
      </field>
    </source>
    <source alias="a" bind="required">
      <partition-name>ex_global</partition-name>
      <source-name>users</source-name>
      <mapping-name>virtual_users_to_ad_users</mapping-name>
      <field name="objectGUID">
        <variable>g.adiSeeAlsoObjectGUID</variable>
      </field>
    </source>
    <parameter>
      <param-name>checkAccountDisabled</param-name>
      <param-value>true</param-value>
    </parameter>
  </entry>

  <entry dn="ou=Groups,ou=global,dc=example,dc=com">
    <oc>organizationalUnit</oc>
    <at name="ou" rdn="true">
      <constant>Groups</constant>
    </at>
```

```
  </entry>

  <entry dn="cn=...,ou=Groups,ou=global,dc=example,dc=com">
<entry-class>org.safehaus.penrose.directory.DynamicEntry</entry-class>
    <mapping-name>virtual_groups</mapping-name>
    <at name="cn" rdn="true">
      <variable>g.primaryKey.cn</variable>
    </at>
    <source alias="g">
      <partition-name>example</partition-name>
      <source-name>groups</source-name>
      <mapping-name>virtual_groups_to_global_groups</mapping-name>
      <field name="cn" primaryKey="true">
        <variable>rdn.cn</variable>
      </field>
    </source>
    <source alias="a">
      <partition-name>ex_global</partition-name>
      <source-name>groups</source-name>
      <mapping-name>virtual_groups_to_ad_groups</mapping-name>
      <field name="objectGuid">
        <variable>g.adiSeeAlsoObjectGUID</variable>
      </field>
    </source>
  </entry>

</directory>
```

8. Create the mappings for the global directory. Mappings are described in *Section 9.6, "Creating Advanced Mappings"*, and the number, types, and values of the mappings, as well as the source which supplies the values, varies for each deployment.

```
<mappings>

  <mapping name="virtual_users">

    <rule name="objectClass">
      <constant>person</constant>
    </rule>
    <rule name="sn" required="false">
      <condition>!a.sn.equals("NULL")</condition>
      <variable>a.sn</variable>
    </rule>
    <rule name="sn" required="false">
      <condition>!g.sn.equals("NULL")</condition>
      <variable>g.sn</variable>
    </rule>
    <rule name="cn" required="false">
      <condition>!a.name.equals("NULL")</condition>
      <variable>a.name</variable>
    </rule>
    <rule name="cn" required="false">
      <condition>!g.cn.equals("NULL")</condition>
      <variable>g.cn</variable>
    </rule>

</mapping>
```

9. Configure the modules for the repository, and map them to the subtree defined for the global repository.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE modules PUBLIC "-//Penrose/DTD Modules 2.0//EN" "ht\
tp://penrose.safehaus.org/dtd/modules.dtd">

<modules>

  <module name="Cache">
<module-class>org.safehaus.penrose.cache.module.CacheModule<mod/ule-class
>
    <parameter>
      <param-name>querySize</param-name>
      <param-value>100</param-value>
    </parameter>
    <parameter>
      <param-name>resultSize</param-name>
      <param-value>100</param-value>
    </parameter>
    <parameter>
      <param-name>expiration</param-name>
      <param-value>5</param-value>
    </parameter>
  </module>

  <module-mapping>
    <module-name>Cache<mod/ule-name>
    <base-dn>ou=global,dc=example,dc=com</base-dn>
  <mod/ule-mapping>

<mod/ules>
```

10. Add the global repository to the **federation.xml** file as a partition entry. The partition entry should point to the template for the global repository, and the name of the entry should be the same as the one given in the **partition.xml** file. For example:

```
vim federation.xml

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE directory PUBLIC "-//Penrose/DTD Directory 2.0//EN" "ht\
tp://penrose.safehaus.org/dtd/directory.dtd">

<federation>

  <partition name="ex_global">
       <template>partitions/ExDomain/templates/global</template>
  </partition>

</federation>
```

11. Restart the server. The global partition will be automatically created using the global template.

```
service vd-server restart
```

10.3.2. Creating LDAP Templates

The local repository template uses variables to generate the specific information to the server being automatically supplied when the new partition is generated.

1. Open the federation domain partition directory.

```
cd /opt/vd-server-2.0/partitions/ExDomain/
```

2. Create a templates directory, if it is not already created.

```
mkdir templates/
```

3. Open the templates directory.

```
cd templates/
```

4. Inside the templates directory, create a new partition-style directory for the LDAP repository. For example:

```
mkdir ldap/
mkdir ldap/DIR-INF/
```

5. Open the **`DIR-INF`** directory.

```
cd ldap/DIR-INF/
```

6. Create a new **`partition.xml`** file that points to the federation domain and an entry for the local LDAP repository entry in the **`federation.xml`** file.

```
vim partition.xml

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE partition PUBLIC "-//Penrose/DTD Partition 2.0//EN" "ht\
tp://penrose.safehaus.org/dtd/partition.dtd">

<partition depends="ExDomain,ldap1">
```

```
</partition>
```

7. Create a new **connections.xml** file that defines the directory tree and entries for the global subtree. The values for the connection parameters should be variables for the server-specific values. It is also possible to set connection configuration options, as listed in *Table 6.1, "Connection Configuration Values"*.

```
vim connections.xml

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE connections PUBLIC "-//Penrose/DTD Connections 2.0//EN" "ht\
tp://penrose.safehaus.org/dtd/connections.dtd">

<connections>

  <connection name="LDAP">
    <adapter-name>LDAP</adapter-name>
    <parameter>
      <param-name>java.naming.provider.url</param-name>
      <param-value>${ldap.url}</param-value>
    </parameter>
    <parameter>
      <param-name>java.naming.security.principal</param-name>
      <param-value>${ldap.user}</param-value>
    </parameter>
    <parameter>
      <param-name>java.naming.security.credentials</param-name>
      <param-value>${ldap.password}</param-value>
    </parameter>

  </connection>

</connections>
```

8. Create the mappings for the local repository. Mappings are described in *Section 9.6, "Creating Advanced Mappings"*, and the number, types, and values of the mappings, as well as the source which supplies the values, varies for each deployment. This is an abbreviated example:

```
vim mappings.xml

<mappings>

  <mapping name="import_users">

    <rule name="objectClass">
      <constant>person</constant>
    </rule>
    <rule name="sn" required="false">
      <condition>!a.sn.equals("NULL")</condition>
      <variable>a.sn</variable>
    </rule>
    <rule name="cn" required="false">
```

```
      <condition>!a.name.equals("NULL")</condition>
      <variable>a.name</variable>
    </rule>

</mapping>
```

NOTE

These same mappings are also specified in the global repository. Here, source alias for the LDAP template is **a**, so all of the mappings here are prefaced with **a**. In the global template, there are mappings for both **a** (LDAP) and **g** (global) attributes, and there are usually two mappings per attribute.

9. Configure the identity linking modules. The linking modules define relationships between mappings, source keys, source aliass, and other attributes used by the linking service. For example:

```
vim modules.xml

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE modules PUBLIC "-//Penrose/DTD Modules 2.0//EN" "ht\
tp://penrose.safehaus.org/dtd/modules.dtd">

<modules>

  <module name="IdentityLinking" enabled="true">
    <module-
class>org.safehaus.penrose.federation.module.IdentityLinkingModule</modul
e-class>
    <parameter>
      <param-name>source</param-name>
      <param-value>LDAP</param-value>
    </parameter>
    <parameter>
      <param-name>target</param-name>
      <param-value>ExDomain.Global</param-value>
    </parameter>
    <parameter>
      <param-name>sourceKey</param-name>
      <param-value>objectGUID</param-value>
    </parameter>
    <parameter>
    <param-name>targetAttribute</param-name>
      <param-value>seeAlso</param-value>
    </parameter>
    <parameter>
      <param-name>mapping</param-name>
      <param-value>import_users</param-value>
    </parameter>
    <parameter>
      <param-name>mappingPrefix</param-name>
      <param-value>a</param-value>
    </parameter>
    <parameter>
```

```
      <param-name>guidAttributes</param-name>
      <param-value>objectGUID</param-value>
    </parameter>
    <parameter>
      <param-name>sidAttributes</param-name>
      <param-value>objectSid</param-value>
    </parameter>
  </module>

</modules>
```

10. Edit the **`sources.xml`** file to define the LDAP sources. As with the **`connections.xml`** file, this allows variables to be inserted into the parameter values, which will be automatically generated when the partition is created.

    This example mapping defines sources and fields for both user and group entries. For example:

```
vim sources.xml

<sources>

  <source name="LDAP">

    <connection-name>LDAP</connection-name>

    <parameter>
      <param-name>baseDn</param-name>
      <param-value>${ldap.suffix}</param-value>
    </parameter>

  </source>

  <source name="users">

<source-class>org.safehaus.penrose.ldap.source.ADUserSource</source-class
>

    <connection-name>LDAP</connection-name>

    <field name="dn" primaryKey="true"/>
    <field name="sn"/>
    <parameter>
      <param-name>objectClasses</param-name>
      <param-value>user</param-value>
    </parameter>
    <parameter>
      <param-name>filter</param-name>
      <param-value>(objectClass=user)</param-value>
    </parameter>
    <parameter>
      <param-name>baseDn</param-name>
      <param-value>${ldap.suffix}</param-value>
    </parameter>
    <parameter>
      <param-name>checkPassword</param-name>
      <param-value>false</param-value>
    </parameter>
    <parameter>
```

```
      <param-name>checkAccountDisabled</param-name>
      <param-value>true</param-value>
    </parameter>

  </source>

 <source name="groups">

    <connection-name>LDAP</connection-name>

    <field name="dn" primaryKey="true"/>
    <field name="objectGUID"/>

    <parameter>
      <param-name>objectClasses</param-name>
      <param-value>group</param-value>
    </parameter>
    <parameter>
      <param-name>filter</param-name>
      <param-value>(objectClass=group)</param-value>
    </parameter>
    <parameter>
      <param-name>baseDn</param-name>
      <param-value>${ldap.suffix}</param-value>
    </parameter>

  </source>

</sources>
```

1 Add an LDAP repository entry (or multiple entries) to the **federation.xml** file as a
1. **<repository>** entry. The **<repository>** entry should point to the template for the LDAP repository, the LDAP server's URL and suffix, and give the username and password to access that server. The values for those parameters are applied to the template to generate the new partition. For example:

```
vim federation.xml

<federation>

  <repository name="ldap1" type="LDAP">
    <parameter>
      <param-name>url</param-name>
      <param-value>ldap://ldap.example.com/</param-value>
    </parameter>
    <parameter>
      <param-name>suffix</param-name>
      <param-value>DC=ldap,DC=server,DC=com</param-value>
    </parameter>
    <parameter>
      <param-name>user</param-name>
      <param-value>cn=Directory Manager</param-value>
    </parameter>
    <parameter>
      <param-name>password</param-name>
      <param-value>secret</param-value>
    </parameter>
```

```
  </repository>
...
</federation>
```

1. Restart the server. The newly-defined LDAP partition will be automatically created using the local
2. LDAP template and the values in the repository entry.

```
service vd-server restart
```

10.3.3. Creating NIS-Related Templates

The local repository template uses variables to generate the specific information to the server being automatically supplied when the new partition is generated. For NIS services, there are actually three templates to be created for NSS, NIS, and YP services. Any or all of these repositories can be created, depending on the NIS configuration.

10.3.3.1. Creating a NIS Template

NOTE

The **`directory.xml`** files for the NIS template is empty.

1. Open the federation domain partition directory.

```
cd /opt/vd-server-2.0/partitions/ExDomain/
```

2. Create a templates directory, if it is not already created.

```
mkdir templates/
```

3. Open the templates directory.

```
cd templates/
```

4. Inside the templates directory, create a new partition-style directory for the NIS repository. For example:

```
mkdir nis/
mkdir nis/DIR-INF/
```

5. Open the **DIR-INF** directory.

```
cd nis/DIR-INF/
```

6. Create a new **partition.xml** file that points to the federation domain.

```
vim partition.xml

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE partition PUBLIC "-//Penrose/DTD Partition 2.0//EN" "ht\
tp://penrose.safehaus.org/dtd/partition.dtd">

<partition depends="ExDomain">

</partition>
```

7. Create a new **connections.xml** file that points to the JDBC server used by NIS. As with the LDAP configuration, this allows a variable for the NIS server name, which is automatically filled when the new partition is generated. It is also possible to set connection configuration options, as listed in *Table 6.1, "Connection Configuration Values"*.

   For example:

```
vim connections.xml

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE connections PUBLIC "-//Penrose/DTD Connections 2.0//EN" "ht\
tp://penrose.safehaus.org/dtd/connections.dtd">

<connections>

  <connection name="Database">
    <adapter-name>JDBC</adapter-name>
    <parameter>
      <param-name>driver</param-name>
      <param-value>org.hsqldb.jdbcDriver</param-value>
    </parameter>
    <parameter>
      <param-name>url</param-name>
      <param-value>jdbc:hsqldb:file:db/${nis.name}</param-value>
    </parameter>
    <parameter>
      <param-name>user</param-name>
      <param-value>admin</param-value>
    </parameter>
    <parameter>
      <param-name>password</param-name>
```

```
      <param-value></param-value>
    </parameter>
    <parameter>
      <param-name>shutdown</param-name>
      <param-value>true</param-value>
    </parameter>

  </connection>

</connections>
```

8. Create the mappings for the local repository. Mappings are described in *Section 9.6, "Creating Advanced Mappings"*, and the number, types, and values of the mappings, as well as the source which supplies the values, varies for each deployment. This file defines the mappings to use for entries being moved from the NIS domain to the global repository. This is an abbreviated example:

```
vim mappings.xml

<mappings>

  <mapping name="import_users">

    <rule name="objectClass">
      <constant>person</constant>
    </rule>
    <rule name="sn" required="false">
      <condition>!n.sn.equals("NULL")</condition>
      <variable>n.sn</variable>
    </rule>
    <rule name="cn" required="false">
      <condition>!n.name.equals("NULL")</condition>
      <variable>n.name</variable>
    </rule>

</mapping>
```

9. Configure the identity linking modules. The linking modules define relationships between mappings, source keys, source aliass, and other attributes used by the linking service. For NIS servers, there are also modules for performing synchronization and for realigning file permissions after resolving UID or GID conflicts. The Ownership Alignment Module can be applied twice, once for users and again for groups. For example:

```
vim modules.xml

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE modules PUBLIC "-//Penrose/DTD Modules 2.0//EN" "ht\
tp://penrose.safehaus.org/dtd/modules.dtd">

<modules>

  <module name="Synchronization" enabled="true">
org.safehaus.penrose.nis.module.NISSynchronizationModul
<module-class>e</module-class>
    <parameter>
```

```
      <param-name>source</param-name>
      <param-value>ex_${nis.name}_yp.LDAP</param-value>
    </parameter>
    <parameter>
      <param-name>target</param-name>
      <param-value>LDAP</param-value>
    </parameter>
    <parameter>
      <param-name>errors</param-name>
      <param-value>errors</param-value>
    </parameter>
    <parameter>
      <param-name>ignoredObjectClasses</param-name>
      <param-value>nisNoSync</param-value>
    </parameter>
    <parameter>
      <param-name>ignoredAttributes</param-name>
      <param-value>seeAlso</param-value>
    </parameter>
    <parameter>
      <param-name>nisMap.aliases</param-name>
      <param-value>ou=Aliases</param-value>
    </parameter>

... there can be multiple NIS maps and subtrees specific, depending on
the NIS domains ...

  </module>

  <module name="IdentityLinking" enabled="true">
org.safehaus.penrose.federation.module.IdentityLinkingM
<module-class>odule</module-class>
    <parameter>
      <param-name>source</param-name>
      <param-value>LDAP</param-value>
    </parameter>
    <parameter>
      <param-name>target</param-name>
      <param-value>ExDomain.Global</param-value>
    </parameter>
    <parameter>
      <param-name>sourceAttribute</param-name>
      <param-value>seeAlso</param-value>
    </parameter>
    <parameter>
      <param-name>sourceKey</param-name>
      <param-value>dn</param-value>
    </parameter>
    <parameter>
      <param-name>targetAttribute</param-name>
      <param-value>seeAlso</param-value>
    </parameter>
    <parameter>
      <param-name>targetKey</param-name>
      <param-value>dn</param-value>
    </parameter>
    <parameter>
      <param-name>mapping</param-name>
      <param-value>import_users</param-value>
    </parameter>
    <parameter>
```

```
      <param-name>mappingPrefix</param-name>
      <param-value>n</param-value>
    </parameter>
  </module>

  <module name="Users">
org.safehaus.penrose.federation.module.OwnershipAlignme
<module-class>ntModule</module-class>
    <parameter>
      <param-name>objectClass</param-name>
      <param-value>posixAccount</param-value>
    </parameter>
    <parameter>
      <param-name>rdnAttribute</param-name>
      <param-value>uid</param-value>
    </parameter>
    <parameter>
      <param-name>sourceAttribute</param-name>
      <param-value>uidNumber</param-value>
    </parameter>
    <parameter>
      <param-name>targetAttribute</param-name>
      <param-value>uidNumber</param-value>
    </parameter>
    <parameter>
      <param-name>linkingAttribute</param-name>
      <param-value>seeAlso</param-value>
    </parameter>
    <parameter>
      <param-name>linkingKey</param-name>
      <param-value>dn</param-value>
    </parameter>
  </module>

...  the OwnershipAlignmentModule can be invoked again for groups ...
</modules>
```

10. Edit the **sources.xml** file to define the LDAP sources. As with the **connections.xml** file, this allows variables to be inserted into the parameter values, which will be automatically generated when the partition is created.

    This example mapping defines sources and fields for both user and group entries. For example:

```
vim sources.xml

<sources>

  <source name="LDAP">

    <connection-name>LDAP</connection-name>

    <parameter>
      <param-name>baseDn</param-name>
      <param-value>${ldap.suffix}</param-value>
    </parameter>

  </source>
```

```
  <source name="users">

<source-class>org.safehaus.penrose.ldap.source.ADUserSource</source-class
>

    <connection-name>LDAP</connection-name>

    <field name="dn" primaryKey="true"/>
    <field name="sn"/>
    <parameter>
      <param-name>objectClasses</param-name>
      <param-value>user</param-value>
    </parameter>
    <parameter>
      <param-name>filter</param-name>
      <param-value>(objectClass=user)</param-value>
    </parameter>
    <parameter>
      <param-name>baseDn</param-name>
      <param-value>${ldap.suffix}</param-value>
    </parameter>
    <parameter>
      <param-name>checkPassword</param-name>
      <param-value>false</param-value>
    </parameter>
    <parameter>
      <param-name>checkAccountDisabled</param-name>
      <param-value>true</param-value>
    </parameter>

  </source>

 <source name="groups">

    <connection-name>LDAP</connection-name>

    <field name="dn" primaryKey="true"/>
    <field name="objectGUID"/>

    <parameter>
      <param-name>objectClasses</param-name>
      <param-value>group</param-value>
    </parameter>
    <parameter>
      <param-name>filter</param-name>
      <param-value>(objectClass=group)</param-value>
    </parameter>
    <parameter>
      <param-name>baseDn</param-name>
      <param-value>${ldap.suffix}</param-value>
    </parameter>

  </source>

</sources>
```

1 Add an NIS repository entry (or multiple entries) to the **`federation.xml`** file as a **`<repository>`**
1. entry. The **`<repository>`** entry should point to the template for the NIS repository, the NIS domain, and the NIS server. The values for those parameters are applied to the template to generate

the new partition.

Then add a **<partition>** entry for the NIS server, pointing to the NIS template directory.

```
vim federation.xml

<federation>

  <repository name="nis1" type="NIS">
    <parameter>
      <param-name>server</param-name>
      <param-value>nis1.example.com/</param-value>
    </parameter>
    <parameter>
      <param-name>domain</param-name>
      <param-value>nis.example.com/</param-value>
    </parameter>
  </repository>

...

  <partition name="ex_nis">
    <template>partitions/ExDomain/templates/nis</template>
    <repository-ref name="nis" repository="ex_nis"/>
  </partition>


</federation>
```

1 Restart the server. The newly-defined LDAP partition will be automatically created using the local
2. LDAP template and the values in the repository entry.

```
service vd-server restart
```

10.3.3.2. Creating an NSS Template

1. Open the federation domain partition directory.

```
cd /opt/vd-server-2.0/partitions/ExDomain/
```

2. Create a templates directory, if it is not already created.

```
mkdir templates/
```

3. Open the templates directory.

```
cd templates/
```

4. Inside the templates directory, create a new partition-style directory for the NSS repository. For example:

```
mkdir nss/
mkdir nss/DIR-INF/
```

5. Open the **DIR-INF** directory.

```
cd nss/DIR-INF/
```

6. Create a new **partition.xml** file that points to the federation domain, the NIS repository, and the LDAP repository entry in the **federation.xml** file.

```
vim partition.xml

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE partition PUBLIC "-//Penrose/DTD Partition 2.0//EN" "ht\
tp://penrose.safehaus.org/dtd/partition.dtd">

<partition depends="ExDomain,ex_${nis.name},ldap1">

</partition>
```

7. Add a source for each NIS service, such as services, protocols, hosts, and automounts.

   The source is the LDAP repository, and the base DN for the subtree is mapped between the NIS subtree and the NSS subtree.

   A variable can be used for the NIS server's name, which is filled in automatically when the partition is created.

```
vim sources.xml

<sources>

  <source name="hosts">
    <partition-name>ExDomain</partition-name>
    <connection-name>LDAP</connection-name>
    <parameter>
      <param-name>baseDn</param-name>
dc=analog,dc=com</param
<param-value>ou=Hosts,ou=${nis.name},ou=nis,-value>
    </parameter>
    <parameter>
      <param-name>newBaseDn</param-name>
```

```
dc=analog,dc=com</param
<param-value>ou=Hosts,ou=${nis.name},ou=nss,-value>
    </parameter>
  </source>

</sources>
```

8. Define the directory subtrees for the NSS services in the **directory.xml** file. This directory configuration is going to be different for every network. The directory entries can use variables to identify the NIS domain, which are automatically supplied when the real partition is generated.

   First, define the subtree.

```
vim directory.xml

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE directory PUBLIC "-//Penrose/DTD Directory 2.0//EN" "ht\
tp://penrose.safehaus.org/dtd/directory.dtd">

<directory>

  <entry dn="ou=${nis.name},ou=nss,dc=analog,dc=com">
    <oc>organizationalUnit</oc>
    <oc>nisDomainObject</oc>
    <at name="ou" rdn="true">
      <constant>${nis.name}</constant>
    </at>
    <at name="nisDomain">
      <constant>${nis.domain}</constant>
    </at>
    <aci subject="self">
      <permission>rws</permission>
    </aci>
    <aci>
      <target>ATTRIBUTES</target>
      <attributes>userPassword</attributes>
      <action>deny</action>
      <permission>rs</permission>
    </aci>
    <aci>
      <permission>rs</permission>
    </aci>
  </entry>
```

   Next, define a users subtree and groups subtree. (Only the users subtree is shown in this example.) Notice that this entry points to several mappings.

```
  <entry dn="uid=...,ou=Users,ou=${nis.name},ou=nss,dc=analog,dc=com">
    <entry-class>com.analog.penrose.directory.NSSUserEntry</entry-class>
    <mapping-name>virtual_users</mapping-name>
    <at name="uid" rdn="true">
      <variable>n.uid</variable>
    </at>
    <source alias="n" bindOrder="1" bind="sufficient">
      <partition-name>analog_${nis.name}</partition-name>
```

```
      <source-name>users</source-name>
      <mapping-name>virtual_users_to_nis_users</mapping-name>
      <field name="uid" primaryKey="true">
        <variable>rdn.uid</variable>
      </field>
    </source>
    <source alias="g" bindOrder="0" bind="sufficient">
      <partition-name>analog</partition-name>
      <source-name>users</source-name>
      <mapping-name>virtual_users_to_global_users</mapping-name>
      <field name="dn">
        <variable>n.seeAlso</variable>
      </field>
    </source>
    <source alias="a" bindOrder="2" search="ignore" bind="required">
      <partition-name>analog_ad</partition-name>
      <source-name>users</source-name>
      <field name="objectGUID">
        <variable>g.adiSeeAlsoObjectGUID</variable>
      </field>
    </source>
    <parameter>
      <param-name>checkAccountDisabled</param-name>
      <param-value>true</param-value>
    </parameter>
  </entry>
```

Last, create proxy entries in the subtree for each NIS service, like aliases, automounts, protocols, hosts, or bootparams. These proxy entries all correspond to the different sources defined in the **`sources.xml`** file and to the new base DN set in the source parameters. For example:

```
  <entry dn="ou=Hosts,ou=${nis.name},ou=nss,dc=analog,dc=com">
    <entry-class>org.safehaus.penrose.directory.ProxyEntry</entry-class>
    <source>
      <source-name>hosts</source-name>
    </source>
  </entry>
```

9. Create the mappings for the local repository. Mappings are described in *Section 9.6, "Creating Advanced Mappings"*, and the number, types, and values of the mappings, as well as the source which supplies the values, varies for each deployment. As with the global template, there are two mappings for each attribute, pointing to the global attribute (in the **`g.`***attributes*) and the NIS attributes (in the **`n.`***attributes*). This is an abbreviated example:

```
vim mappings.xml

<mappings>

  <mapping name="virtual_users">

    <rule name="objectClass">
      <constant>person</constant>
    </rule>
    <rule name="sn" required="false">
      <condition>!g.sn.equals("NULL")</condition>
```

```
      <variable>g.sn</variable>
    </rule>
    <rule name="sn" required="false">
      <condition>!n.sn.equals("NULL")</condition>
      <variable>n.sn</variable>
    </rule>
    <rule name="cn" required="false">
      <condition>!g.cn.equals("NULL")</condition>
      <variable>g.cn</variable>
    </rule>

</mapping>
```

10. For the NIS services, the only module to use is the cache module, mapped to the NIS server's sub-tree under the NSS's subtree. For example:

```
vim modules.xml

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE modules PUBLIC "-//Penrose/DTD Modules 2.0//EN" "ht\
tp://penrose.safehaus.org/dtd/modules.dtd">

<modules>

  <module name="Cache">
<module-class>org.safehaus.penrose.cache.module.CacheModule</module-class
>
    <parameter>
      <param-name>querySize</param-name>
      <param-value>100</param-value>
    </parameter>
    <parameter>
      <param-name>resultSize</param-name>
      <param-value>100</param-value>
    </parameter>
    <parameter>
      <param-name>expiration</param-name>
      <param-value>5</param-value>
    </parameter>
  </module>

  <module-mapping>
    <module-name>Cache</module-name>
    <base-dn>ou=${nis.name},ou=nss,dc=analog,dc=com</base-dn>
  </module-mapping>

</modules>
```

11. The NIS repository should already be added to the **federation.xml** file as a **<repository>** entry. Add the NSS as a partition entry which points to that NIS **<repository>** entry and to the NSS template entry. For example:

```
vim federation.xml
```

```
<federation>

...
  <partition name="ex_nis_nss">
    <template>partitions/ExDomain/templates/nss</template>
    <repository-ref name="nis" repository="ex_nis"/>
  </partition>
...

</federation>
```

1 Restart the server. The newly-defined NSS partition is automatically created using the local NSS
2. template and the values in the repository entry.

```
service vd-server restart
```

10.3.3.3. Creating a YP Template

NOTE

The **`partition.xml`** and **`modules.xml`** files are empty for the YP template. There is no **`mappings.xml`** file for the YP service.

1. Open the federation domain partition directory.

```
cd /opt/vd-server-2.0/partitions/ExDomain/
```

2. Create a templates directory, if it is not already created.

```
mkdir templates/
```

3. Open the templates directory.

```
cd templates/
```

4. Inside the templates directory, create a new partition-style directory for the YP repository. For example:

```
mkdir yp/
mkdir yp/DIR-INF/
```

5. Open the **DIR-INF** directory.

```
cd yp/DIR-INF/
```

6. Create a new **connections.xml** file that points to the YP server on the NIS server. The values for the connection parameters should be variables for the server-specific values. It is also possible to set connection configuration options, as listed in *Table 6.1, "Connection Configuration Values"*.

```
vim connections.xml

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE connections PUBLIC "-//Penrose/DTD Connections 2.0//EN" "ht\
tp://penrose.safehaus.org/dtd/connections.dtd">

<connections>

  <connection name="NIS">
    <adapter-name>NIS</adapter-name>
    <parameter>
      <param-name>java.naming.factory.initial</param-name>
      <param-value>com.sun.jndi.nis.NISCtxFactory</param-value>
    </parameter>
    <parameter>
      <param-name>java.naming.provider.url</param-name>
      <param-value>nis://${nis.server}/${nis.domain}</param-value>
    </parameter>
    <parameter>
      <param-name>com.sun.jndi.nis.mailaliases</param-name>
      <param-value>nonull</param-value>
    </parameter>
    <parameter>
      <param-name>method</param-name>
      <param-value>yp</param-value>
    </parameter>
  </connection>

</connections>
```

7. Edit the **sources.xml** file to define the LDAP source and the NIS sources for the YP service and to list the basic mappings for each source.

   For each NIS source — like hosts, services, protocols, automounts — specified in the NIS nad NSS configurations, there is a corresponding source in the YP **sources.xml** file. This example shows the **users** source and the **hosts** source.

   Variables can be inserted into the parameter values, which will be automatically generated when the partition is created.

```
vim sources.xml

<sources>

  <source name="LDAP">
<source-class>org.safehaus.penrose.partition.source.PartitionSource</sour
ce-class>
  </source>

  <source name="users">
    <connection-name>NIS</connection-name>
    <field name="uid" primaryKey="true"/>
    <field name="uidNumber"/>
    <field name="gidNumber"/>
    <field name="homeDirectory"/>
    <field name="userPassword"/>
    <field name="loginShell"/>
    <field name="gecos"/>
    <field name="description"/>
    <parameter>
      <param-name>objectClasses</param-name>
      <param-value>posixAccount</param-value>
    </parameter>
    <parameter>
      <param-name>base</param-name>
      <param-value>passwd</param-value>
    </parameter>
  </source>

  <source name="hosts">
    <connection-name>NIS</connection-name>
    <field name="cn"/>
    <field name="ipHostNumber" primaryKey="true"/>
    <parameter>
      <param-name>objectClasses</param-name>
      <param-value>ipHost</param-value>
    </parameter>
    <parameter>
      <param-name>base</param-name>
      <param-value>hosts</param-value>
    </parameter>
  </source>

</sources>
```

8. Define the directory structure for the YP services entries. This example points to the user subtree and the hosts subtree, with basic mappings for the entry attributes.

   Different variables can be used to supply the NIS server and domain information, and the values are automatically supplied when the new partition is generated.

   First create an entry that points to the top NIS subtree.

```
vim directory.xml

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE directory PUBLIC "-//Penrose/DTD Directory 2.0//EN" "ht\
```

```
tp://penrose.safehaus.org/dtd/directory.dtd">

<directory>

  <entry dn="ou=${nis.name},ou=nss,dc=analog,dc=com">
    <oc>organizationalUnit</oc>
    <oc>nisDomainObject</oc>
    <at name="ou" rdn="true">
      <constant>${nis.name}</constant>
    </at>
    <at name="nisDomain">
      <constant>${nis.domain}</constant>
    </at>
    <aci subject="self">
      <permission>rws</permission>
    </aci>
    <aci>
      <target>ATTRIBUTES</target>
      <attributes>userPassword</attributes>
      <action>deny</action>
      <permission>rs</permission>
    </aci>
    <aci>
      <permission>rs</permission>
    </aci>
  </entry>
```

Then, create the users and groups subtrees.

```
  <entry dn="ou=Users,ou=${nis.name},ou=nss,dc=analog,dc=com">
    <oc>organizationalUnit</oc>
    <at name="ou" rdn="true">
      <constant>Users</constant>
    </at>
  </entry>

  <entry dn="uid=...,ou=Users,ou=${nis.name},ou=nss,dc=analog,dc=com">
    <entry-class>com.analog.penrose.directory.NSSUserEntry</entry-class>
    <mapping-name>virtual_users</mapping-name>
    <at name="uid" rdn="true">
      <variable>n.uid</variable>
    </at>
    <source alias="n" bindOrder="1" bind="sufficient">
      <partition-name>analog_${nis.name}</partition-name>
      <source-name>users</source-name>
      <mapping-name>virtual_users_to_nis_users</mapping-name>
      <field name="uid" primaryKey="true">
        <variable>rdn.uid</variable>
      </field>
    </source>
    <source alias="g" bindOrder="0" bind="sufficient">
      <partition-name>analog</partition-name>
      <source-name>users</source-name>
      <mapping-name>virtual_users_to_global_users</mapping-name>
      <field name="dn">
        <variable>n.seeAlso</variable>
      </field>
    </source>
```

```
    <source alias="a" bindOrder="2" search="ignore" bind="required">
      <partition-name>analog_ad</partition-name>
      <source-name>users</source-name>
      <field name="objectGUID">
        <variable>g.adiSeeAlsoObjectGUID</variable>
      </field>
    </source>
    <parameter>
      <param-name>checkAccountDisabled</param-name>
      <param-value>true</param-value>
    </parameter>
  </entry>
```

Last, create proxy entries for every source for the NIS domain, like hosts, protocols, or automounts.

```
  <entry dn="ou=Hosts,ou=${nis.name},ou=nss,dc=analog,dc=com">
    <entry-class>org.safehaus.penrose.directory.ProxyEntry</entry-class>
    <source>
      <source-name>hosts</source-name>
    </source>
  </entry>
```

9. The NIS repository should already be added to the **`federation.xml`** file as a **`<repository>`** entry. Add the YP service as a partition entry which points to that NIS **`<repository>`** entry and to the YP template entry. For example:

```
vim federation.xml

<federation>

...
  <partition name="ex_nis_nss">
    <template>partitions/ExDomain/templates/yp</template>
    <repository-ref name="nis" repository="ex_nis"/>
  </partition>
...

</federation>
```

10. Restart the server. The newly-defined LDAP partition will be automatically created using the local LDAP template and the values in the repository entry.

```
service vd-server restart
```

10.4. Adding LDAP Local Repositories through Penrose

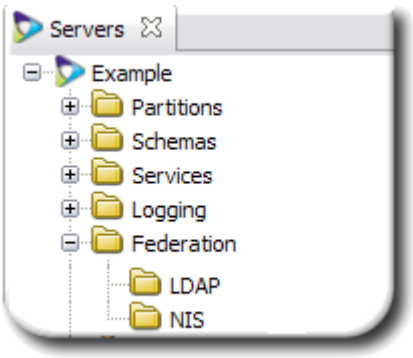
Studio

A local repository is a source for the data in the centralized, or *federated*, global repository. The LDAP local repository can be used for identity federation or as part of a migration from NIS to LDAP, between different kinds of LDAP servers, or migrations from LDAP to Red Hat IPA. An LDAP local repository can be any LDAP server, including Active Directory, Red Hat Directory Server, and OpenLDAP.

LDAP repositories can be created individually in Penrose Studio, specifying the templates created in *Section 10.3.2, "Creating LDAP Templates"*.

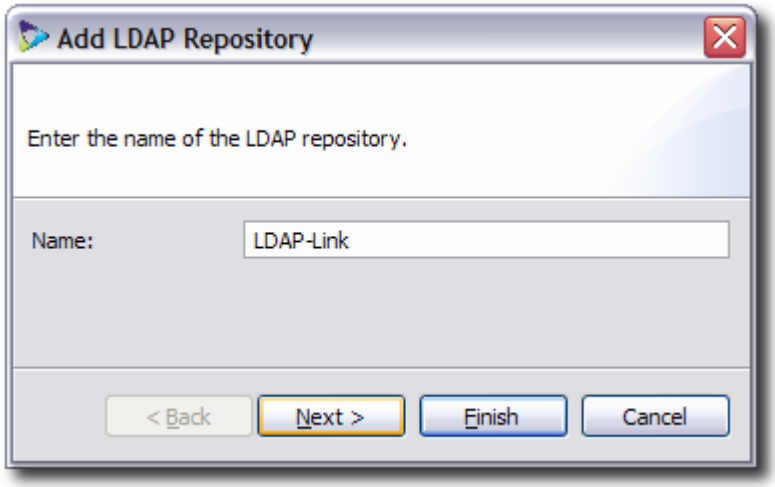
1. Open the **Federation** folder.

2. Double-click the **LDAP** link.

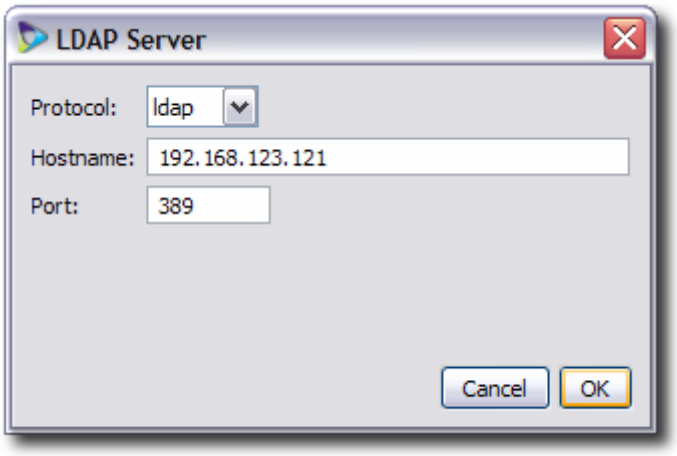


3. Click the **Add** button to add the LDAP server for the local repository.

4. Enter a name for the LDAP repository.

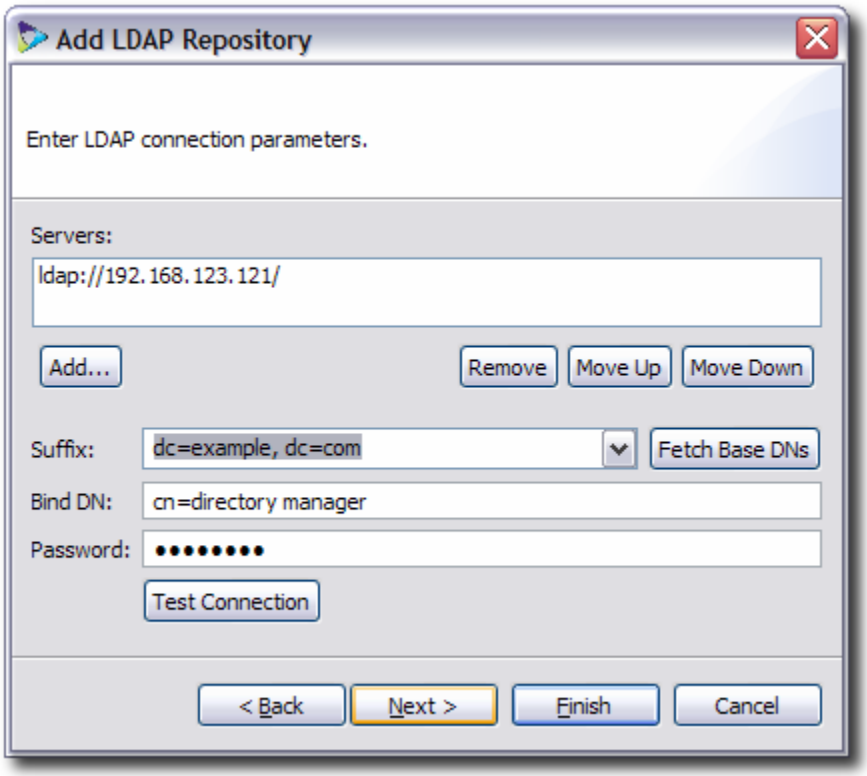


5. Click **Add**, and enter the LDAP server information.



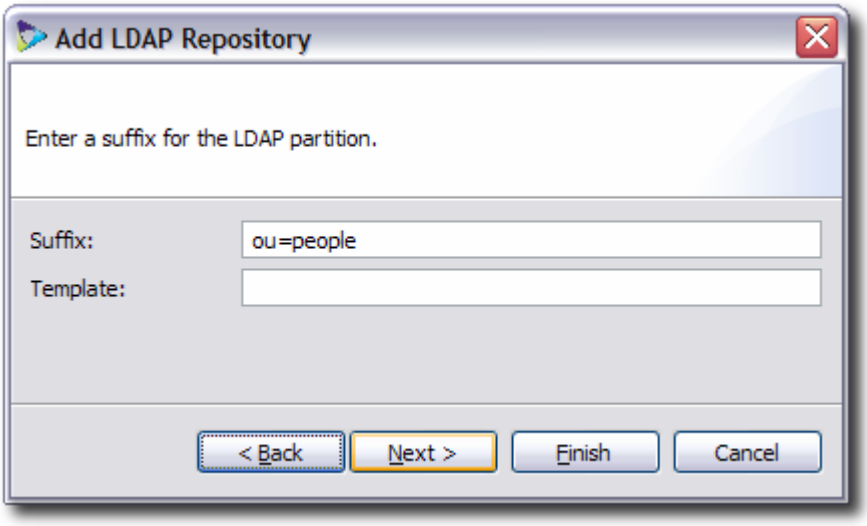
- Connection protocol, either standard LDAP or secure LDAPS
- The hostname and port

6. Click **Add**, and enter the LDAP directory connection information.

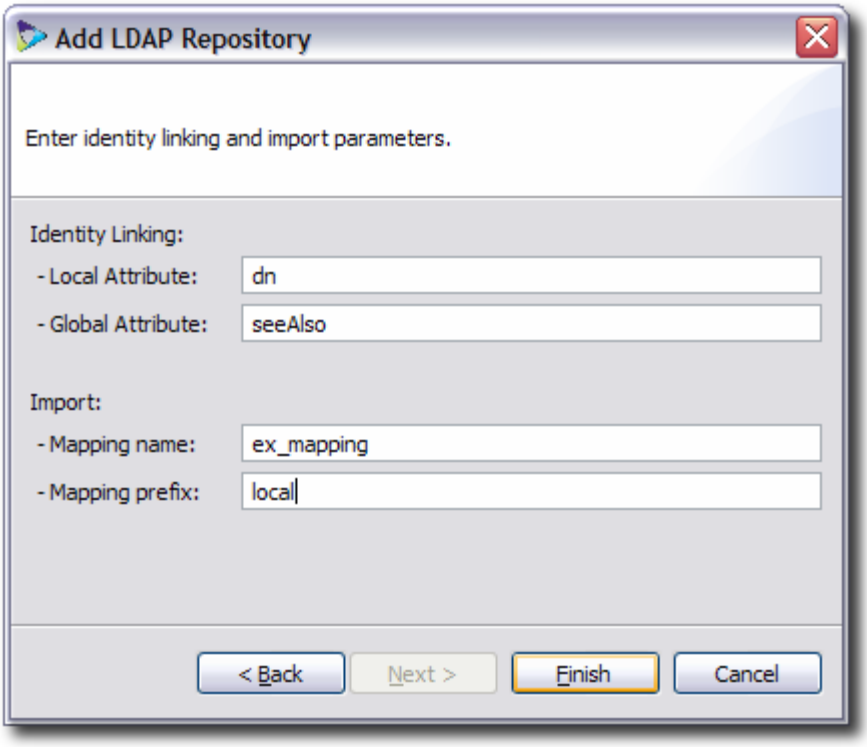


- The directory subtree (suffix) to access
- The bind DN and password to use

7. Give the name of the suffix and template to use to create a partition entry for the LDAP repository. Creating templates is described in *Section 10.3.2, "Creating LDAP Templates"*. When the template is given, the corresponding partition is automatically created.



8. Enter the attributes to use for identity linking.



- The *local attribute* is the LDAP attribute to store in the global entry; this is a unique identifying attribute, such as the DN or UID.

- The *global attribute* is the attribute in the global entry that stores the local attribute; this is usually a reference attribute, such as `seeAlso`.

9. Fill in the mapping entry to use to import local entries into global entries.

10. Fill in the prefix to use to identify local attributes during import. Mapping entries have the form *repo_name.attribute* to identify the source of an entry value. This field is the *repo_name*.

11. Click **Finish**.

12. Close Penrose Studio, and restart Penrose Server.

```
service vd-server restart
```

A new entry for the LDAP local repository is listed under the **LDAP** folder in the **Federation** area and in the **Partitions** area.

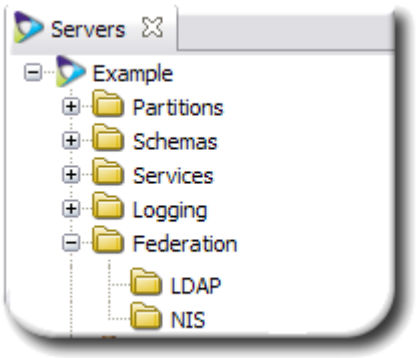
10.5. Adding NIS Local Repositories Using Penrose Studio

A local repository is a source for the data in the centralized, or *federated*, global repository. The NIS local repository can be used for identity federation or as part of a migration from NIS to LDAP.

NIS repositories can be created individually in Penrose Studio, specifying the templates created in *Section 10.3.3, "Creating NIS-Related Templates"*.

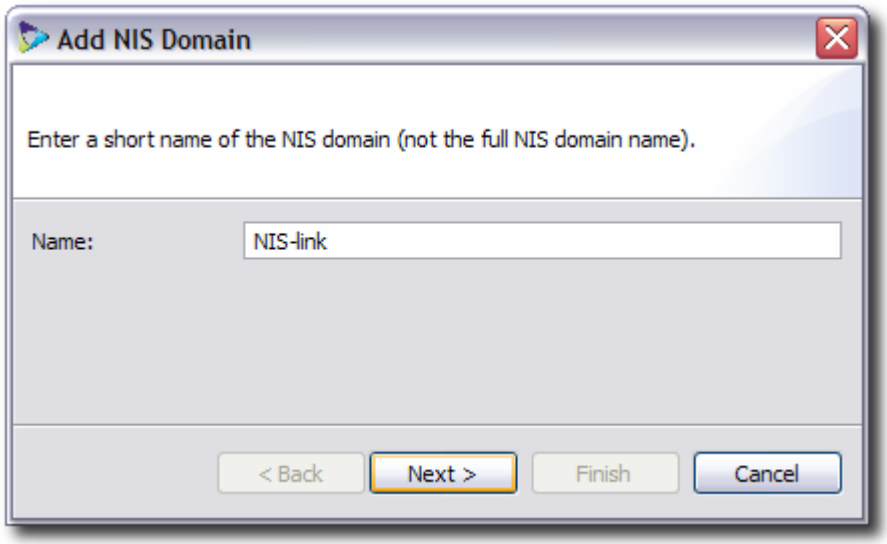
1. Open the **Federation** folder.

2. Double-click the **NIS** link.

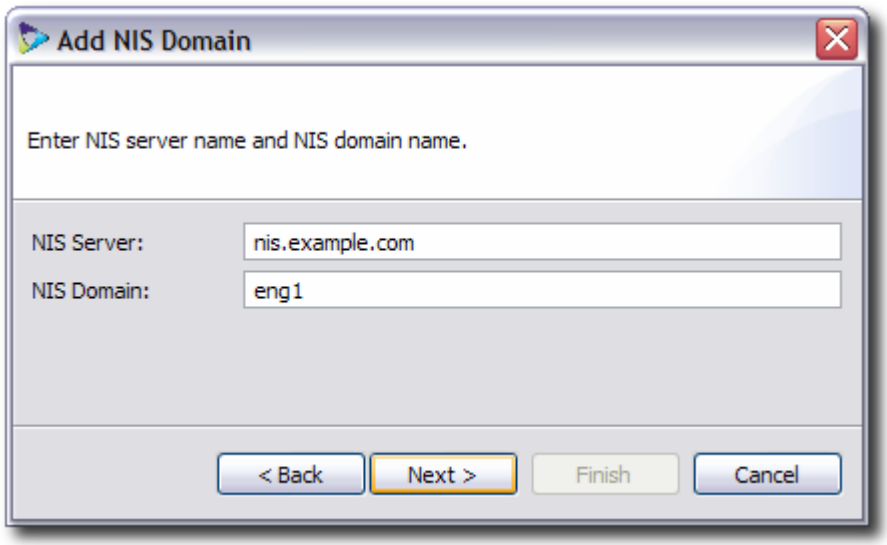


3. Click the **Add** button to add the NIS server for the local repository.

4. Enter a name for the NIS repository entry.

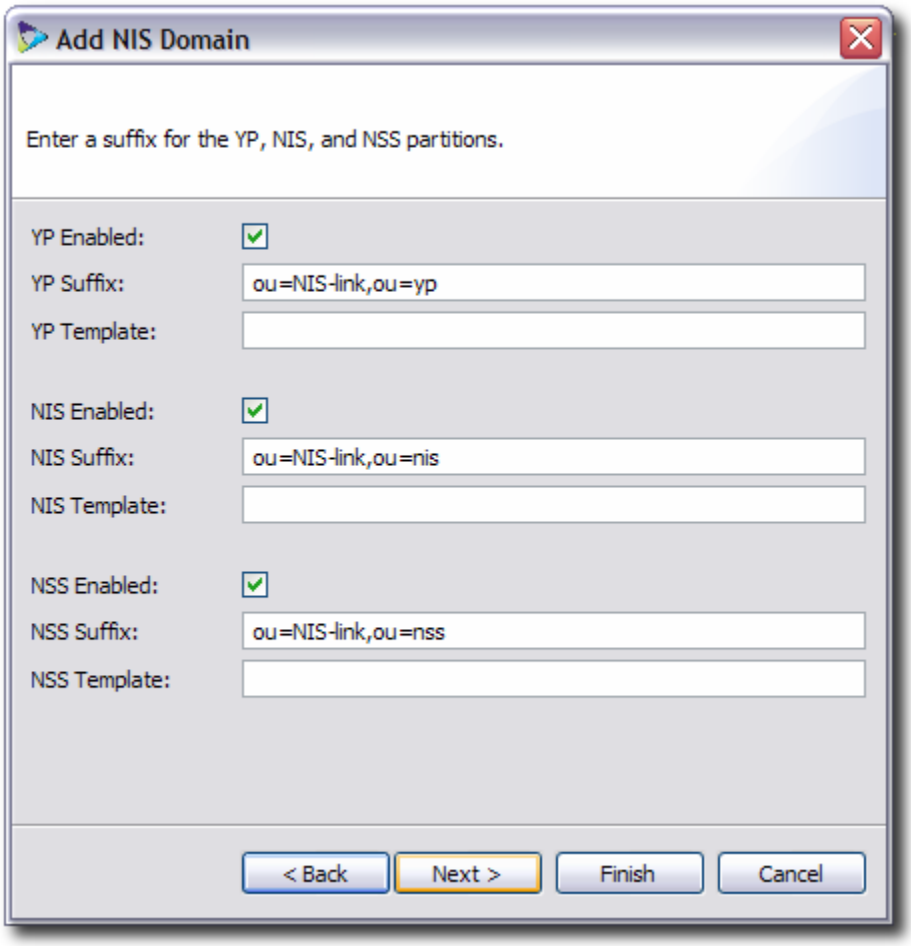


5. Enter the NIS server information.



- The fully-qualified domain name for the NIS server
- The name of the NIS domain

6. Each NIS domain can have NIS, NSS, or YP services running. Select whether each service is enabled, and then enter the suffix and templates to use to create the NIS, NSS, and YP partitions, as were created in *Section 10.3.3, "Creating NIS-Related Templates"*. The template path is relative to the Penrose Virtual Directory installation directory.



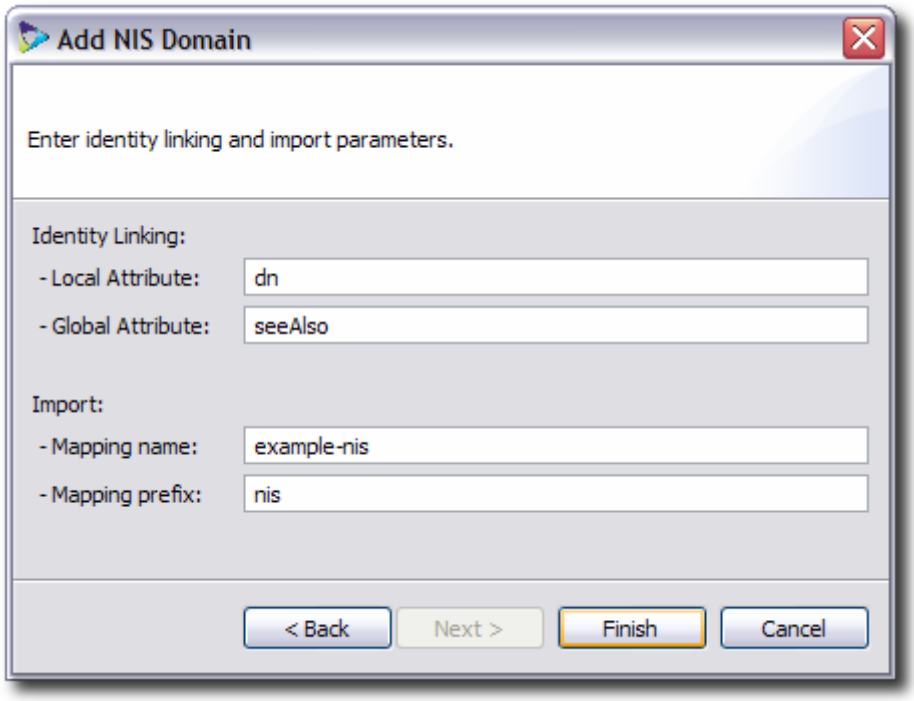
- The NIS partition provides a proxy to the NIS data stored in the Red Hat Directory Server instance used in NIS migration.

- The NSS partition allows stacking authentication, as described in *Section 1.3.7, "Planning Authentication"*.

- The YP partition provides a proxy to the NIS data stored in the NIS server.

The YP and NIS partitions are used to synchronize NIS data into Red Hat Directory Server. The NSS partition is used by client machines listed in the **`/etc/nsswitch.conf`**.

7. Enter the attributes to use for identity linking.



- The *local attribute* is the LDAP attribute to store in the global entry; this is a unique identifying attribute, such as the DN or UID.

- The *global attribute* is the attribute in the global entry that stores the local attribute; this is usually a reference attribute, such as `seeAlso`.

8. Fill in the mapping entry to use to import local entries into global entries.

9. Fill in the prefix to use to identify local attributes during import. Mapping entries have the form *repo_name.attribute* to identify the source of an entry value. This field is the *repo_name*.

10. Click **Finish**.

11. Close Penrose Studio, and restart Penrose Server.

```
service vd-server restart
```

A new entry for the NIS local repository is listed under the **NIS** folder in the **Federation** area and in the **Partitions** area.

10.6. Linking Identities

After the global and local repositories have been created, then the identities in those repositories need to be linked. Linking the entries is used for authentication and other operations performed through Penrose Virtual Directory against the repositories.

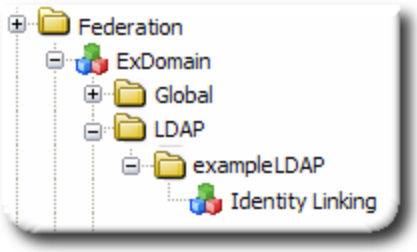
Both LDAP and NIS repositories have identity linking tools.

NOTE

NIS entries must be migrated into a Red Hat Directory Server instance before linking can be configured.

1. Open the **Federation** folder.

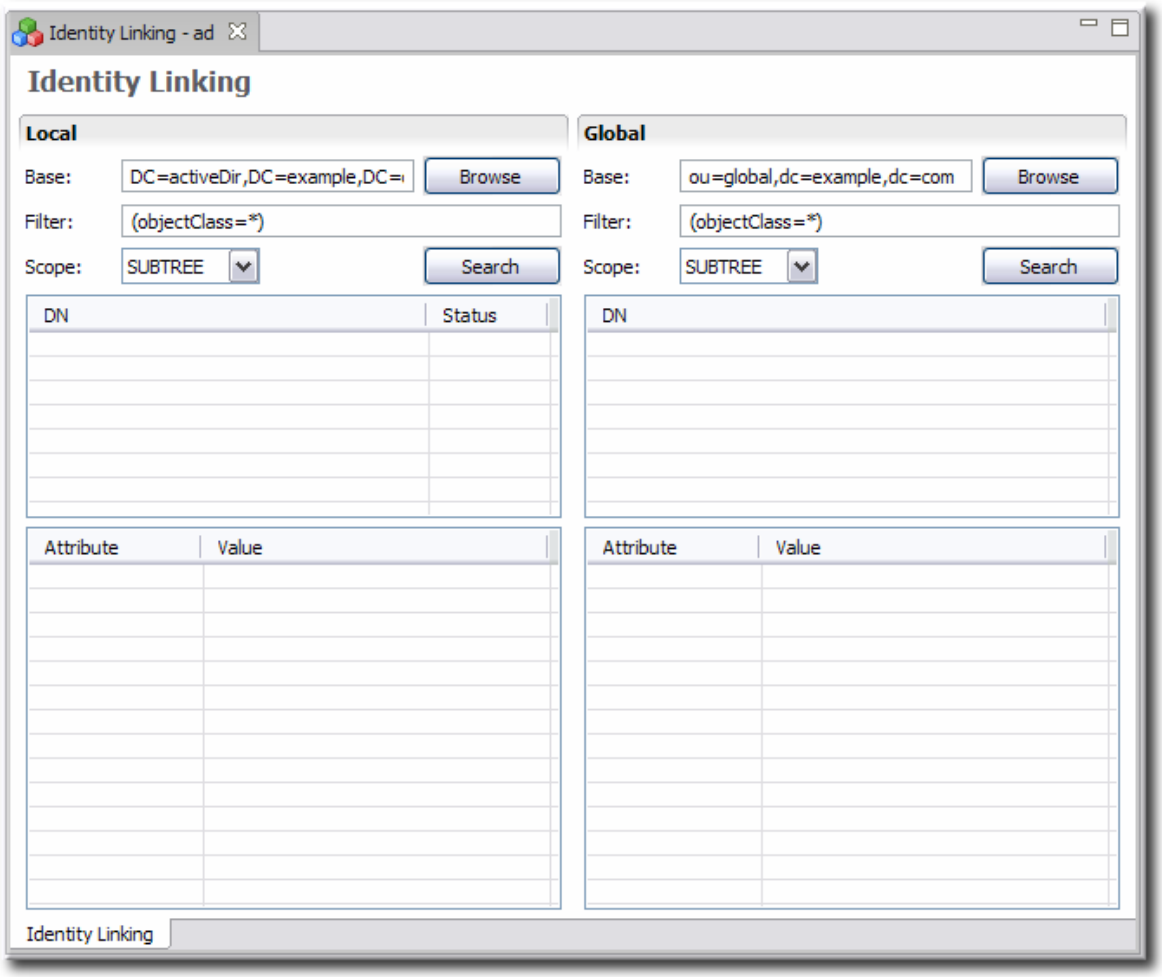
2. Click one of the local repository folders, **NIS** or **LDAP**, and open the specific repository to link.



3. Double-click the **Identity Linking** link to open the linking editor in the main window.

   The left side of the window shows the local identity, and the right side the global identity.

4. Enter the search parameters to find the local identities which will be linked.



- The *base* is the subtree to search.
- The *filter* is the object class or attribute to search for, such as **`(objectclass=inetorgperson)`**
- The *scope* is how far down the subtree to search.

5. Click **Search**.
6. Next, search for matching global identities to link to the local entries. Again, enter the search parameters to search for entries in the global repository.
   - The *base* is the subtree to search.
   - The *filter* is the object class or attribute to search for, such as **`(uid=${uid})`**

TIP

The expression **`(uid=${uid})`** means that Penrose Virtual Directory searches for entries in the global repository which match *`uid`*s of the returned entries for the local repository.

- The *scope* is how far down the subtree to search.

7. Click **Search**.

8. Check the search results.

   - If there are matching global entities, verify that the local identity and the global identity reference the same person. If they are the same, then right-click on the *global identity*, and select **Link**.

     Multiple local identities can be selected at the same time to perform the same operation at once.

     The status for the local identity is now **`Linked`**.

   - If there is no matching global identity, either change the search parameters and search again or import the local identity as a new global identity. Right-click the local identity and select **Import**. The new global entry attributes are mapped using the mapping configured for the local repository.

     After it is imported, the new global entry is automatically linked back to the local identity.

The linking information is stored in the global identity in the *`seeAlso`* attribute. If there is more than one local identity linked to the same global identity, then the global entry has multiple *`seeAlso`* attributes.

To unlink identities, right-click either the local or global identity, and select **Unlink**. To delete a global identity, right-click it, and select **Delete**; this removes the global identity and all linking information, even to other local identities.

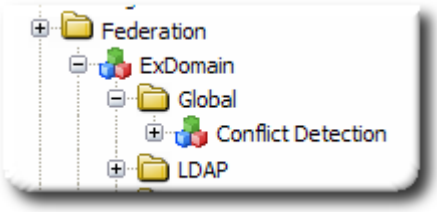
10.7. Resolving UID and GID Conflicts

Because identity federation combines entries from many different sources, there can be conflicts between the user and group identification numbers used by the systems. The *Ownership Alignment Tool* in Penrose Virtual Directory processes any conflicts in the UID or GID values assigned to global identities and allows you to reassign UIDs and GIDs to resolve conflicts.

1. Open the **Federation** folder.

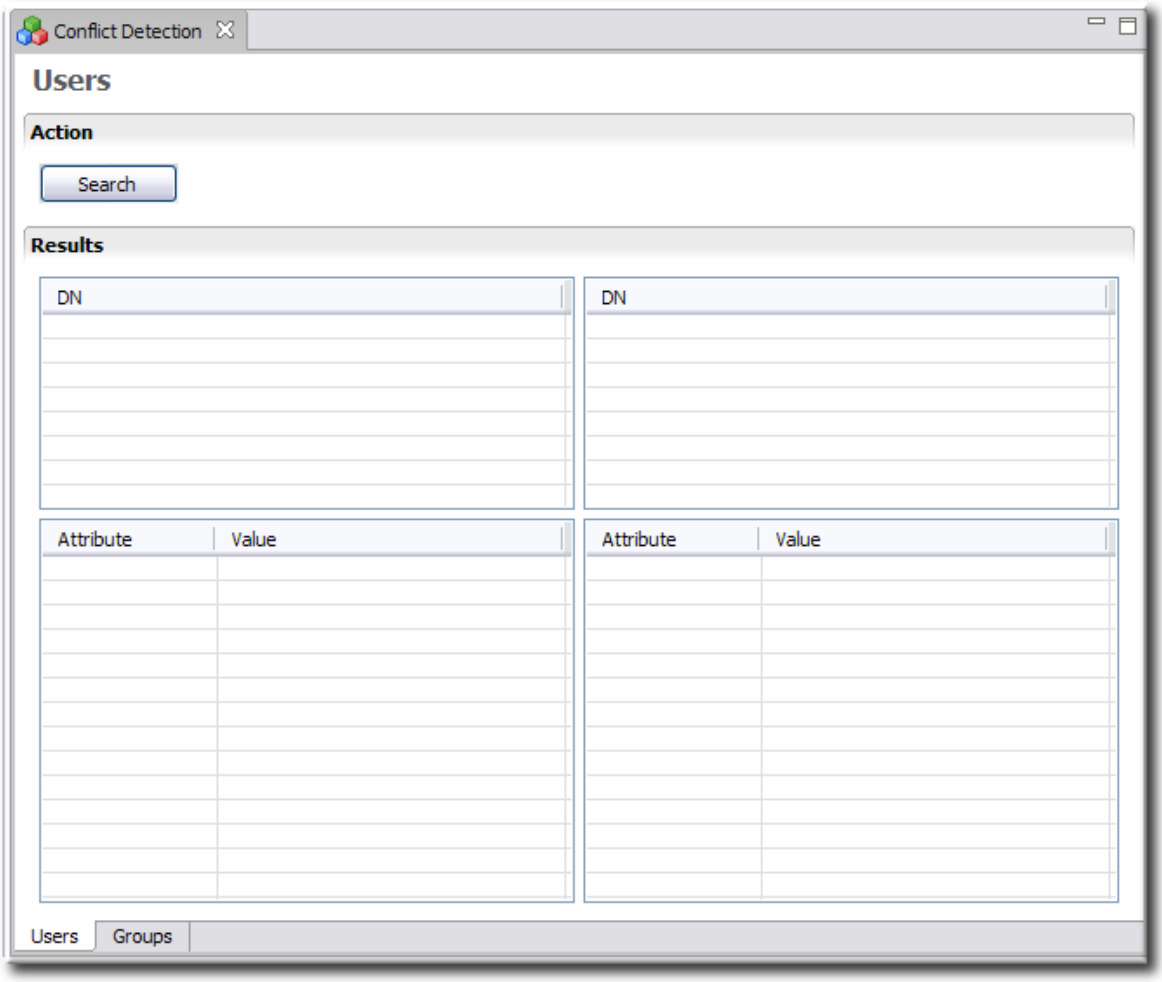
2. Open the **Global** folder.

3. Double-click the **Conflict Detection** folder.



4. Click the **Users** tab to check for user conflicts, or click the **Groups** tab to check for group conflicts.

5. Click **Search** to search for conflicts.



6. In the results list on the left, click the line of an identity with conflicts. The entries with conflicting UIDs will be listed in the top table on the right. Matching entries for that identity are listed in the bottom table on the right.

7. Resolve the conflict. For example, change the UID of one of the conflicting entries by selecting the identity and clicking **Edit**.

8. Save the changes.

9. Click **Search** again to make sure that the conflicts were resolved.

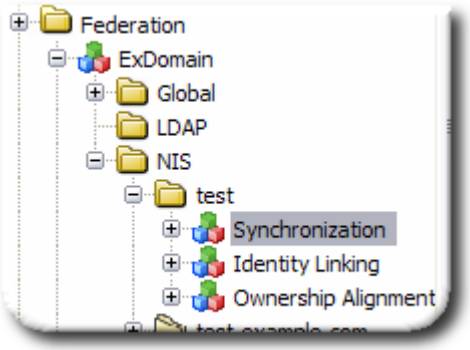
10.8. Checking File Ownership with the Ownership Alignment Tool

The Conflict Detection Tool for the global repository detects and helps resolve collisions in user or group ID numbers when the different repositories are unified. Once those numbers are resolved, however, the permissions on files and directories can be wrong. The Ownership Alignment Tool identifies the list of all files and directories which have permissions that need updated, and the UID with which they are associated.

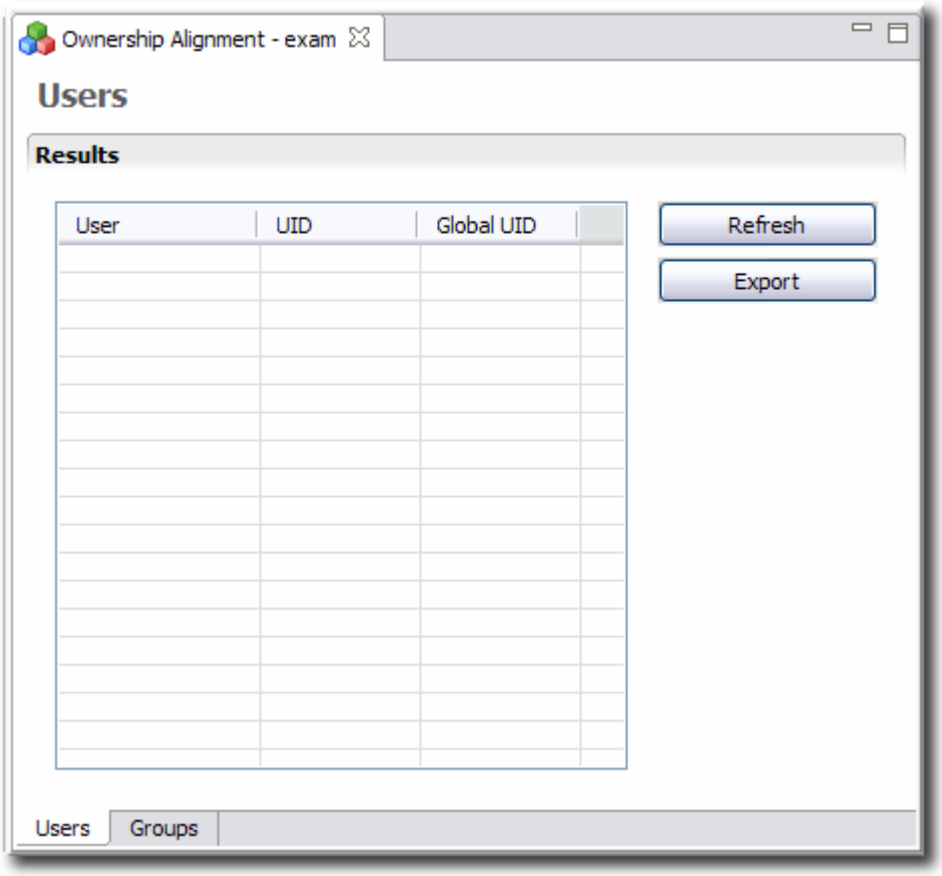
1. Open the **Federation** folder.

2. Open the **NIS** folder.

3. Double-click the **Ownership Alignment** folder.



4. Click **Refresh** to create the list of changed file permissions.



5. If there is a list of affected files, then export the list by clicking the **Export** button. The exported file can be used with a script or as a reference to update the file permissions.

10.9. Synchronizing (or Migrating) NIS Data to LDAP

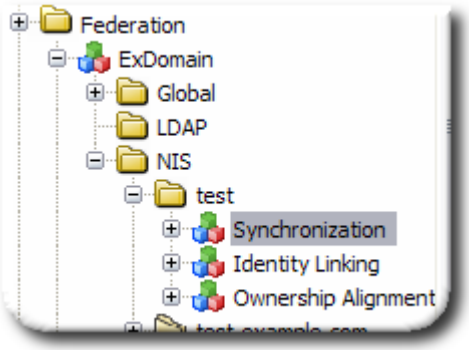
Data can be imported from a NIS domain into Red Hat Directory Server.

NOTE

Migrating data is not required for identity federation, but the identity federation tools and configuration can be used to facilitate a NIS to LDAP migration.

1. Open the **Federation** folder.

2. Open the **NIS** folder. The NIS domains configured in Penrose Virtual Directory are listed below it; open the specific NIS instance.



3. Double-click **Synchronize** to start importing the NIS data into Red Hat Directory Server.

When the information is migrated, Apache Directory Studio can be used to view the NIS data in Red Hat Directory Server.

When the information is changed on the NIS server, synchronize the changes to Red Hat Directory Server. There are two ways to synchronize NIS data:

- To synchronize manually, simply click the **Synchronize** button in Penrose Server. To synchronize through the command line, use the **`nis.sh`** tool:

```
/opt/vd-server-2.0/bin/nis.sh -D uid=admin,ou=system -w secret synchron\
ize domain   map
```

`-D` is the username to bind to Penrose Virtual Directory, and `-w` is the password.

- To synchronize the NIS domain and Red Hat Directory Server periodically, set up a cron job for the **`nis.sh`** script. For information on setting up cron jobs, see *http://1www.hmug.org/1man/15/1crontab.php*.

Configuring Modules

Modules are Java classes which extend the Red Hat Penrose Server functionality. One module is included and configured by default, the Federation Module used for identity federation. Custom modules can be added and supported for Penrose Server.

This chapter explains how to add and enable modules in Penrose Virtual Directory and how to map modules to virtual directory subtrees and entries.

11.1. Adding Modules

Modules are available to a partition, so they are configured with other partition entries.

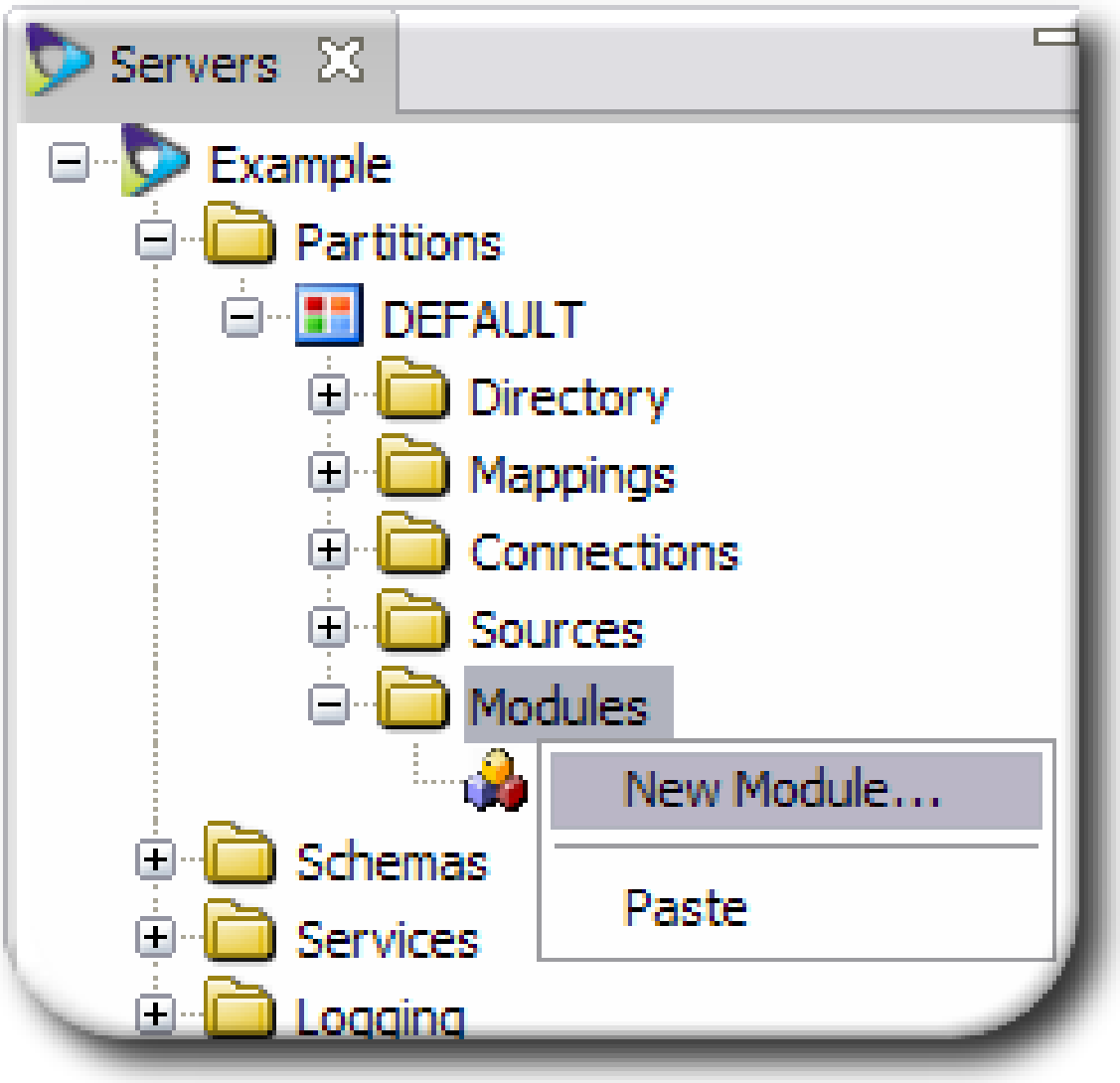
11.1.1. Adding Modules in Penrose Studio

1. For custom modules, copy the **`.jar`** into the **`/opt/vd-server-2.0/lib`** directory so that Penrose Server can access it.

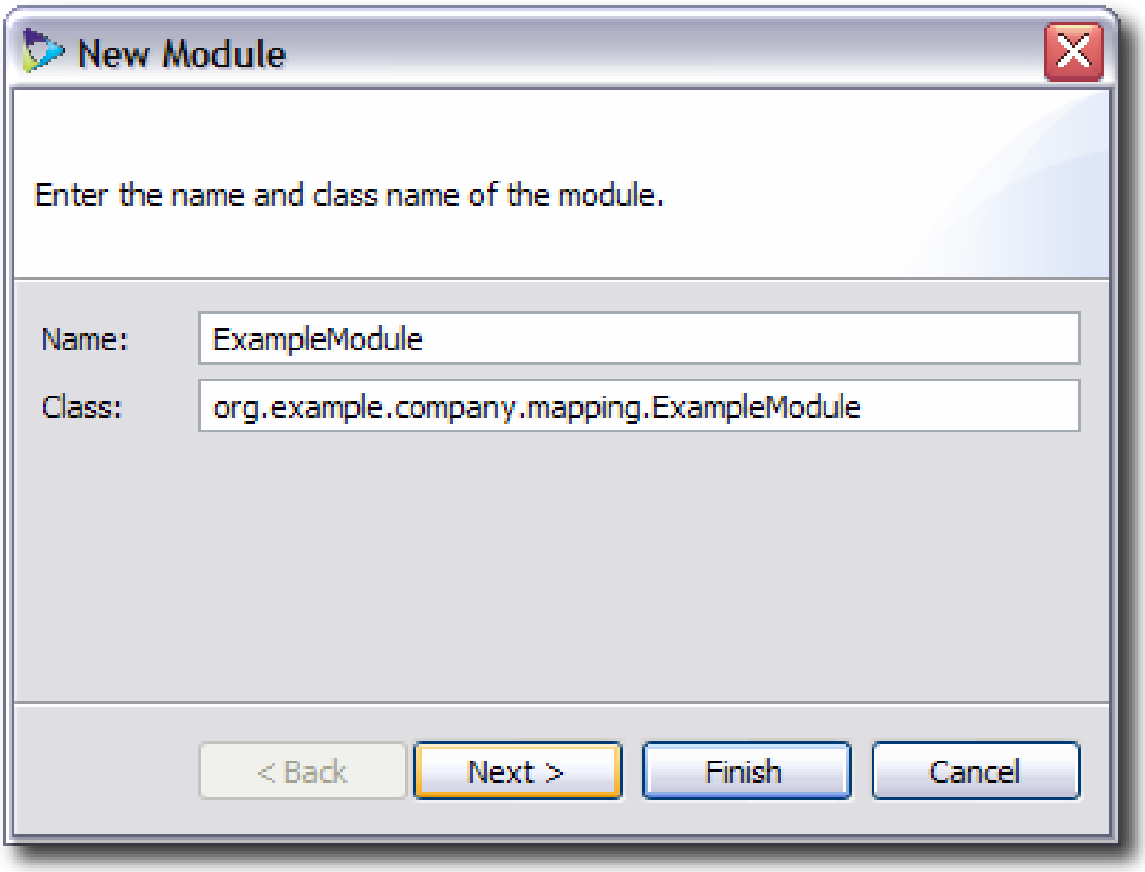
2. In Penrose Server, open the server, and expand the **Partitions** folder.

3. Open the partition to which to add the module.

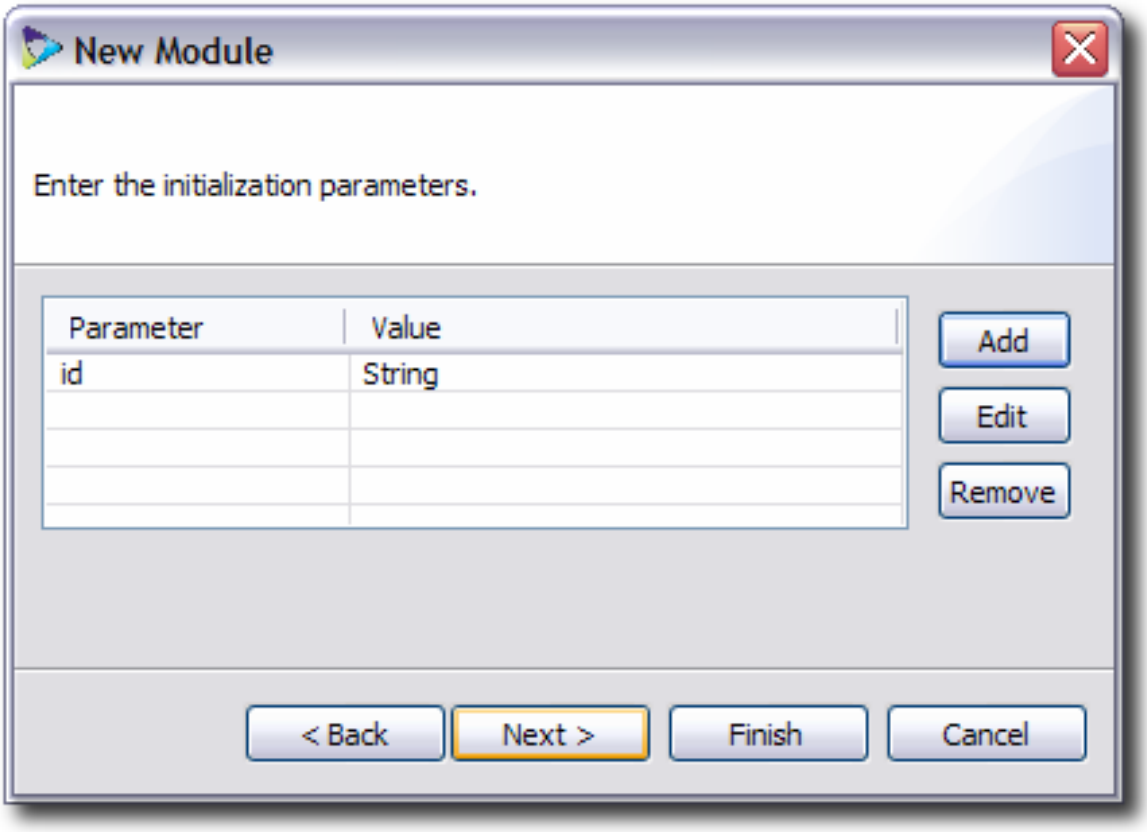
4. Right-click the **Modules** folder, and select **New Module...**.



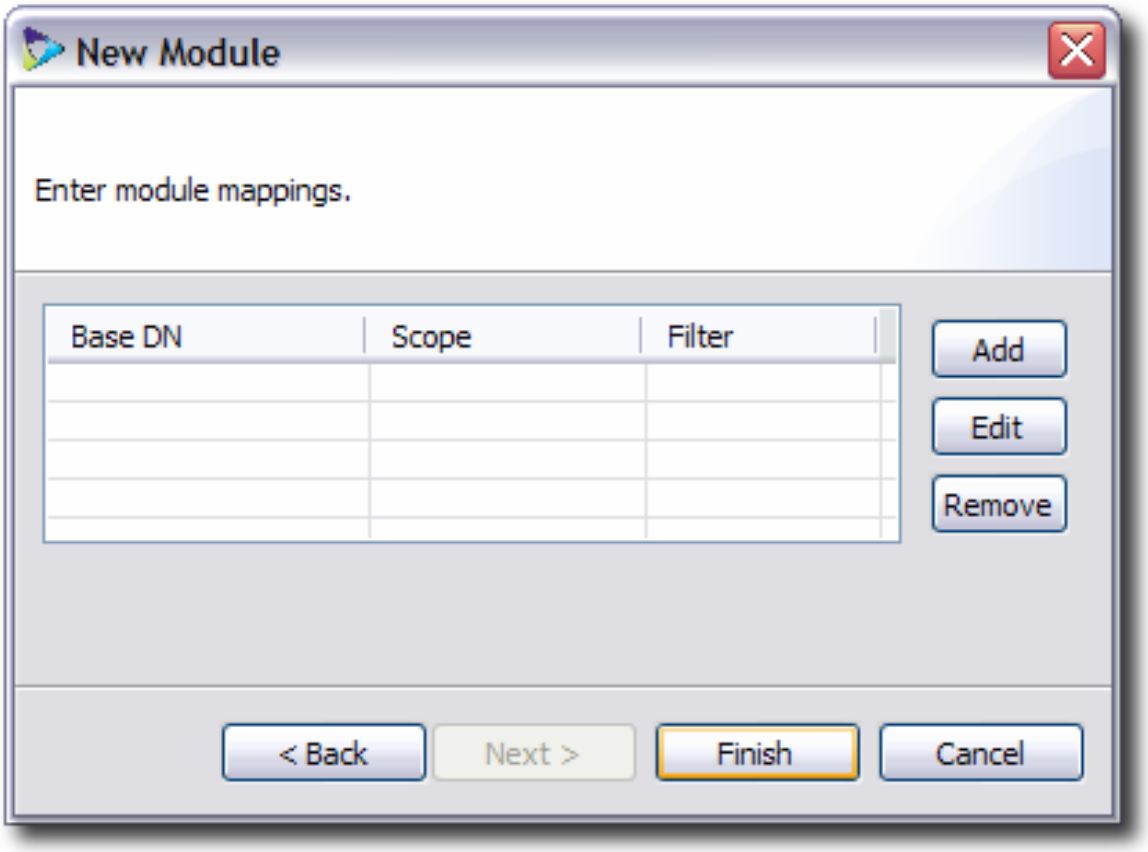
5. Enter the name of the module entry and the name of the class to use for the module. This information is required.



6. Enter any parameters required for the Java class.



7. Optionally, map the module to any virtual directory subtree entries. The information for the module mapping is used only to identify the virtual entry and requires the parameters of an LDAP search query: the base DN to search, the object class or attribute search filter, and the scope of the search.



11.1.2. Adding Modules Manually

The main configuration file for modules is **`modules.xml`** in **`/opt/vd-server-2.0/conf`** or **`/opt/vd-server-2.0/partitions/`*`partition_name`*`/DIR-INF`**. The actual Java classes for the module must be in **`/opt/vd-server-2.0/lib`**.

A complete module configuration has two separate entries in the **`modules.xml`** file. The first defines the module, including the entry name, Java class, and any parameters. The second is an optional mapping entry which maps the module to a virtual directory root DSE, base DN, or subtree entry.

Example 11.1, "Annotated Module Entry" shows a complete **`modules.xml`** file.

```
<modules> the main file tag

  <module name="..."> the module entry

    <module-class>...</module-class> the Java class name of the module

    <parameter>   optional parameters or settings for the module
        <param-name>...</param-name>
        <param-value>...</param-value>
    </parameter>
```

```
  </module>

  <module-mapping> a mapping entry

    <module-name>...</module-name> the name of the module to map
    <base-dn>...</base-dn>  the virtual directory DN to use for the mod\
ule
    <filter>...</filter>  LDAP search filter to search for entries in the
virtual directory
    <scope>...</scope>  scope for the LDAP search

  </module-mapping>

</modules>
```

Example 11.1. Annotated Module Entry

The only required information to configure a module is the name for the entry and the name of the Java class. For example:

```
<modules>
  <module name="FederationModule">
<module-class>org.safehaus.penrose.federation.module.FederationModule</mo
dule-class>
  </module>
</modules>
```

IMPORTANT

Always restart Penrose Server after editing the configuration file. For example:

```
service vd-server restart
```

The parameters for the **`modules.xml`** file are listed in *Table 11.1, "Module Entry Parameters"*.

Tag or Parameter	Description	Example
<module>	The entry tag for a module entry.	
name="..."	Gives the name of the module.	`<module name="ExampleModule">`
enables="true|false"	Sets whether the module is enabled. If this argument is not present, then the default value is **`true`**.	`<module name="ExampleModule" enabled="true">`

Tag or Parameter	Description	Example
<module-class>	Contains the Java class for the module; this library must be in **`/opt/vd-server-2.0/lib`**.	`<module-class>org.example.company.mapping.ExampleModule`
<parameter>	A container entry for any initialization parameters required or used by the module.	
<param-name>	The name of the initialization parameter.	
<param-value>	The value or setting of the initialization parameter.	

Table 11.1. Module Entry Parameters

11.2. Mapping Modules to Data Entries

The final step when configuring a module in Penrose Studio is creating optional mappings from the module to virtual directory subtrees. Many modules perform background tasks for other Penrose Virtual Directory services, such as modules for synchronization, which apply to an *operation*. However, it can be useful in some sitiations to have a module which performs tasks within a certain *area* of the directory. This module mapping configuration matches the module to specific parts of the tree only; this can be useful, for example, for applying a custom caching module to a part of the tree with frequent search operations.

Multiple subtrees can be mapped to a module. In Penrose Studio, this is done by editing the module and adding information for the LDAP search to match the subtree: the base DN, object class or attribute filter, and search scope. As many mapping entries can be added through the editor as required, and these are automatically applied to the module.

In the **`modules.xml`** file, this mapping is done by adding a new **`<module-mapping>`** entry for every module mapping. The same information as in the Penrose Studio editor — the name of the module, base DN, object class or attribute filter, and search scope — are added as parameters to the entry. For example:

```
<module-mapping>
  <module-name>ExampleModule</module-name>
  <base-dn>ou=people,dc=example,dc=com</base-dn>
  <filter>(objectclass="inetorgperson")</filter>
  <scope>subtree</scope>
</module-mapping>
```

The parameters for the module mapping entry are listed in *Table 11.2, "Module Mapping Parameters"*.

IMPORTANT

Always restart Penrose Server after editing the configuration file. For example:

```
service vd-server restart
```

Tag or Parameter	Description	Example
<module-mapping>	Identifies a mapping entry.	
<module-name>	Gives the name of the module which is being mapped.	`<module-name>ExampleModule</module-name>`
<base-dn>	Contains the distinguished name of the root entry or sub-tree entry in the virtual directory being mapped to the module.	`<base-dn>ou=people,dc=example,dc=com</base-dn>`
<filter>	Contains the LDAP filter to search for the object class or attribute in the mapped subtree entry.	`<filter>(objectclass=organization\ alUnit)</filter>`
<scope>	Defines the scope of the search, meaning how many levels down from the base DN for Penrose Server to search. There are three options: subtree (all the way down), onelevel (only direct children of the entry), and base (only the base DN).	`<scope>onelevel</scope>`

Table 11.2. Module Mapping Parameters

11.3. Enabling and Disabling Modules

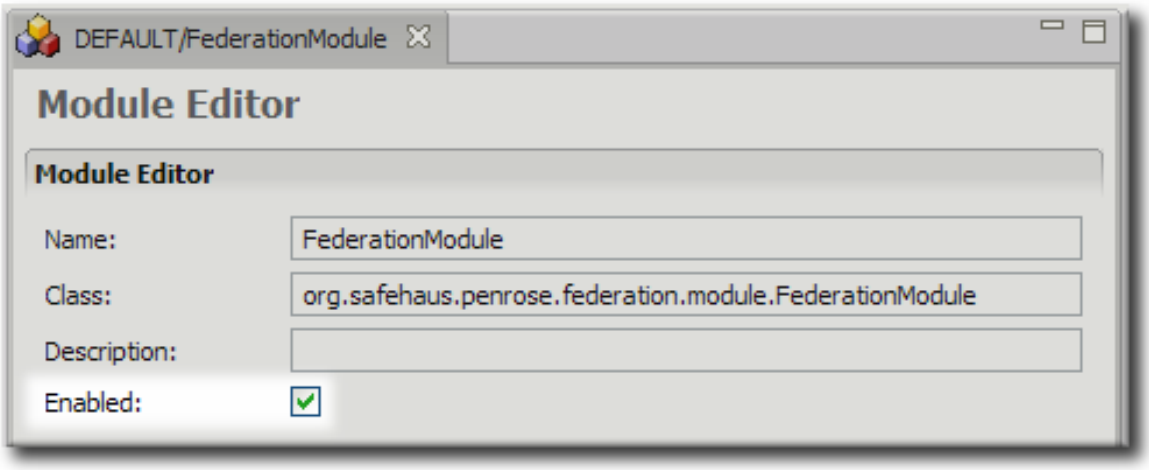
Modules can be enabled and disabled quickly in Penrose Studio.

1. Open the server entry, and expand the **Partitions** folder.

2. Open the partition where the module belongs, and then open the **Modules** folder.

3. Double-click the module entry to open the entry editor.

4. Uncheck the **Enable** checkbox to disable the module or check the box to enable it.



Using Services with Penrose Virtual Directory

Services are front-end components of Penrose Virtual Directory which communicate directly with client applications. This chapter gives an overview of what services are and how they are used, as well as instructions for creating custom services to use in addition to the default Penrose Virtual Directory services.

12.1. About Services in Penrose Virtual Directory

Services provide a communication avenue between clients and Penrose Server. By default, Penrose Virtual Directory provides services that communicate with LDAP servers (ApacheDS, OpenDS, and MINA) and Java clients (JMX).

Penrose Virtual Directory provides three different LDAP services, ApacheDS, OpenDS, and MINA. By default, only OpenDS is enabled.

NOTE

All three LDAP services can be configured to run for Penrose Virtual Directory, as long as they are listening on different ports.

The JMX service connects to any client which uses the JMX protocol, including Penrose Studio.

Each service configured in Penrose Virtual Directory is located in **`/opt/vd-server-2.0/services`**; similarly to the partition configuration, each service has a folder named for the service and a subfolder called **`SERVICE-INF`**. There can also be additional folders for libraries, databases, LDIF files, configuration files, and other files used by the specific service. The files used by the default services are listed in *Table 12.1, "Default Service Configuration Files"*.

Service	Folder	Description
ApacheDS	/opt/vd-server-2.0/services/ApacheDS	The main service directory.
ApacheDS	/opt/vd-server-2.0/services/ApacheDS/bin	Contains the **`schema.sh`** and **`schema.bat`** files to compile custom schema to allow it to be used by ApacheDS.
ApacheDS	/opt/vd-server-2.0/services/ApacheDS/conf	Contains a **`server.xml`** file which defines the MBeans object used to run the service.
ApacheDS	/opt/vd-server-	Contains the **`service.xml`** file, which defines the service for Penrose Server, and a **`lib/`** directory which contains the lib-

Service	Folder	Description
	2.0/services/ApacheDS/SERVICE-INF	raries used by the LDAP service.
ApacheDS	/opt/vd-server-2.0/services/ApacheDS/SERVICE-INF/lib	Contains the libraries used by the LDAP service.
JMX	/opt/vd-server-2.0/services/JMX	The main service directory.
JMX	/opt/vd-server-2.0/services/JMX/conf	Contains a **log4j.xml** file which defines logging for the JMX service.
JMX	/opt/vd-server-2.0/services/JMX/SERVICE-INF	Contains the **service.xml** file, which defines the service for Penrose Server, and a **lib/** directory which contains the libraries used by the JMX service.
JMX	/opt/vd-server-2.0/services/JMX/SERVICE-INF/lib	Contains the libraries used by the JMX service.
MINA	/opt/vd-server-2.0/services/LDAP	The main service directory.
MINA	/opt/vd-server-2.0/services/LDAP/SERVICE-INF	Contains the **service.xml** file, which defines the service for Penrose Server, and a **lib/** directory which contains the libraries used by the MINA LDAP service.
MINA	/opt/vd-server-2.0/services/LDAP/SERVICE-INF/lib	Contains the libraries used by the MINA LDAP service.
OpenDS	/opt/vd-server-2.0/services/OpenDS	The main service directory.
OpenDS	/opt/vd-server-2.0/services/OpenDS/config	Contains LDIF files, scripts, schema, and templates used by the OpenDS service.
OpenDS	/opt/vd-serv-	Contains database files used by the OpenDS service.

Service	Folder	Description
	er-2.0/services/OpenDS/db	
OpenDS	/opt/vd-server-2.0/services/OpenDS/locks	Contains lock files used by different components of the OpenDS service.
OpenDS	/opt/vd-server-2.0/services/OpenDS/logs	Contains access, error, and replication logs generated by the OpenDS service.
OpenDS	/opt/vd-server-2.0/services/OpenDS/SERVICE-INF	Contains the **`service.xml`** file, which defines the service for Penrose Server, and a **`lib/`** directory which contains the libraries used by the OpenDS LDAP service.
OpenDS	/opt/vd-server-2.0/services/OpenDS/SERVICE-INF/lib	Contains the libraries used by the OpenDS LDAP service.

Table 12.1. Default Service Configuration Files

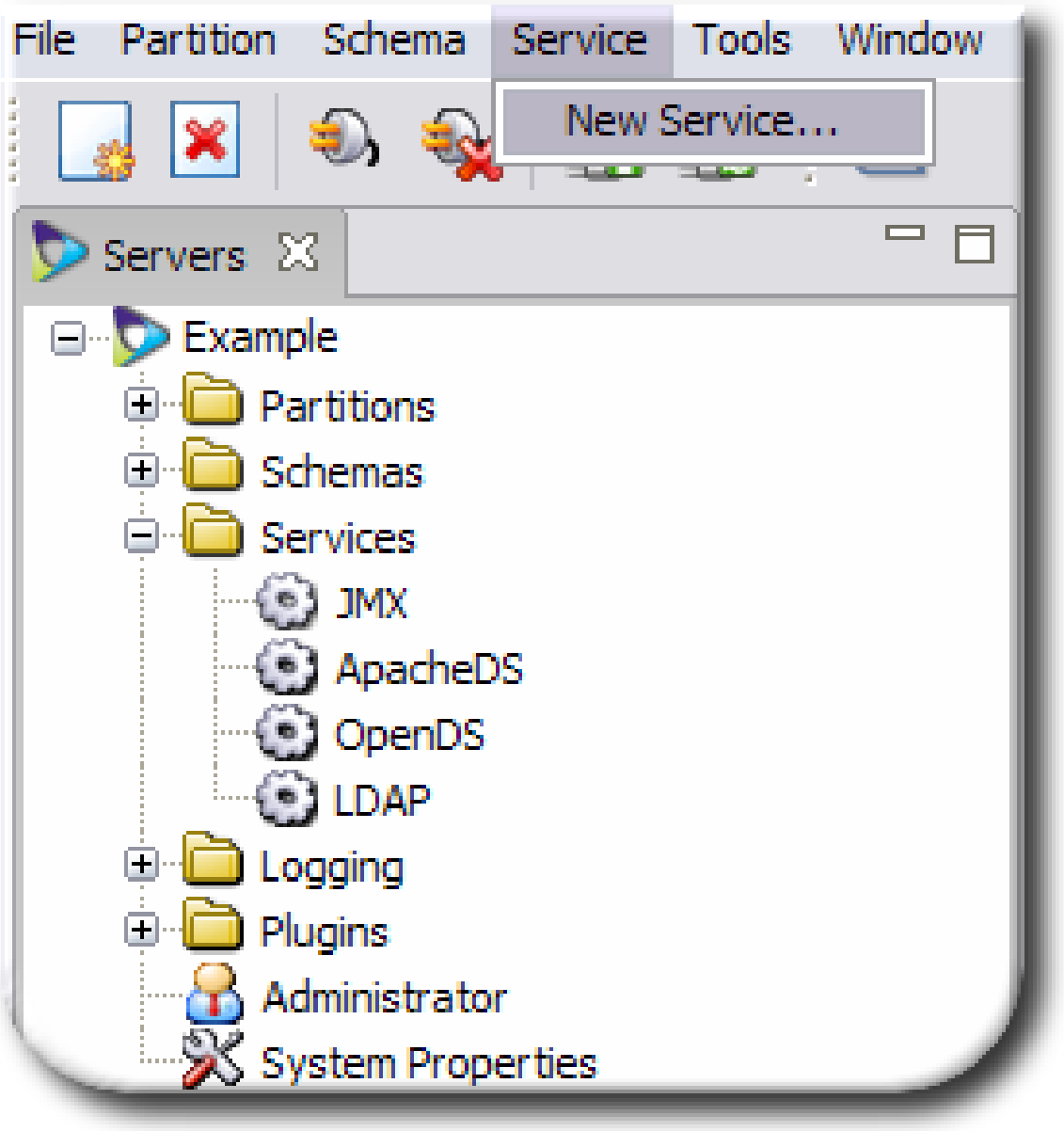
Creating a custom service requires creating a new service directory in **`/opt/vd-server-2.0/services`** such as **`/opt/vd-server-2.0/services/`***service_name*. The service is defined in the **`SERVICE-INF/service.xml`** file. Put any custom libraries in the **`SERVICE-INF/lib`** directory. The default class, if none are specified, is **`org.safehaus.penrose.service.Service`**.

12.2. Configuring Additional Services

Other services can be added to operate with Penrose Server, giving a new or extended interface to communicate with server clients.

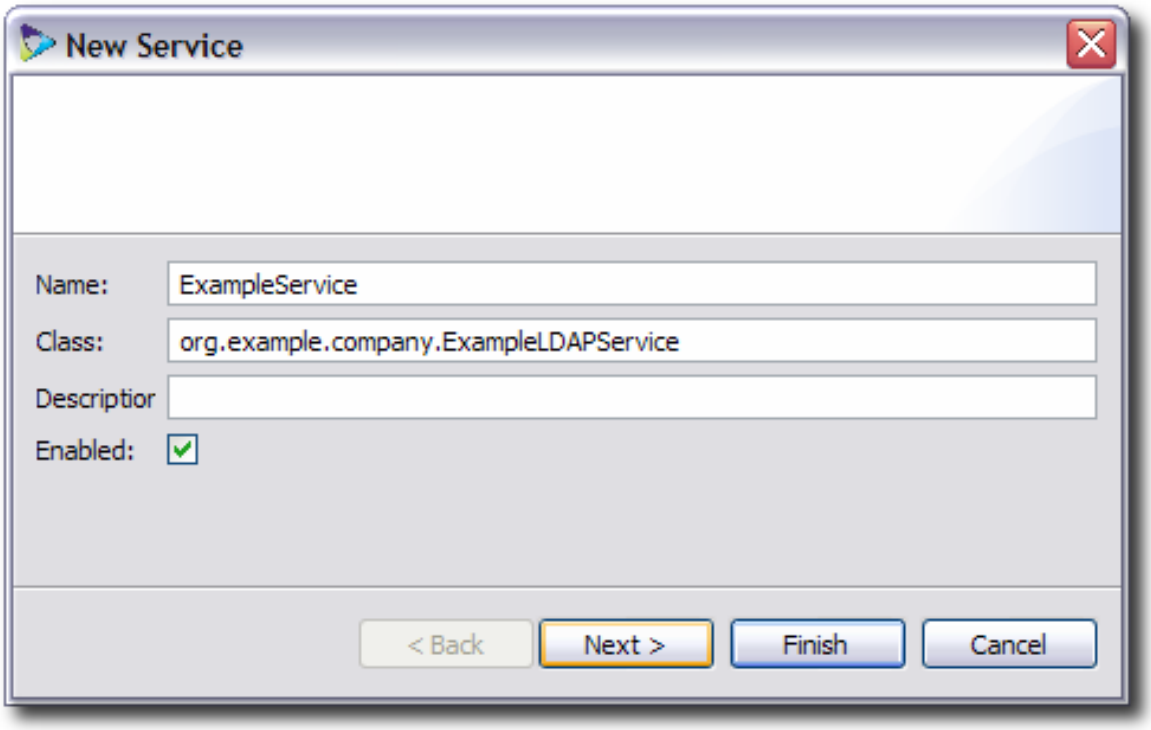
12.2.1. Adding Services in Penrose Studio

1. In the top menu, click the **Services** menu, and select **New Service...**.



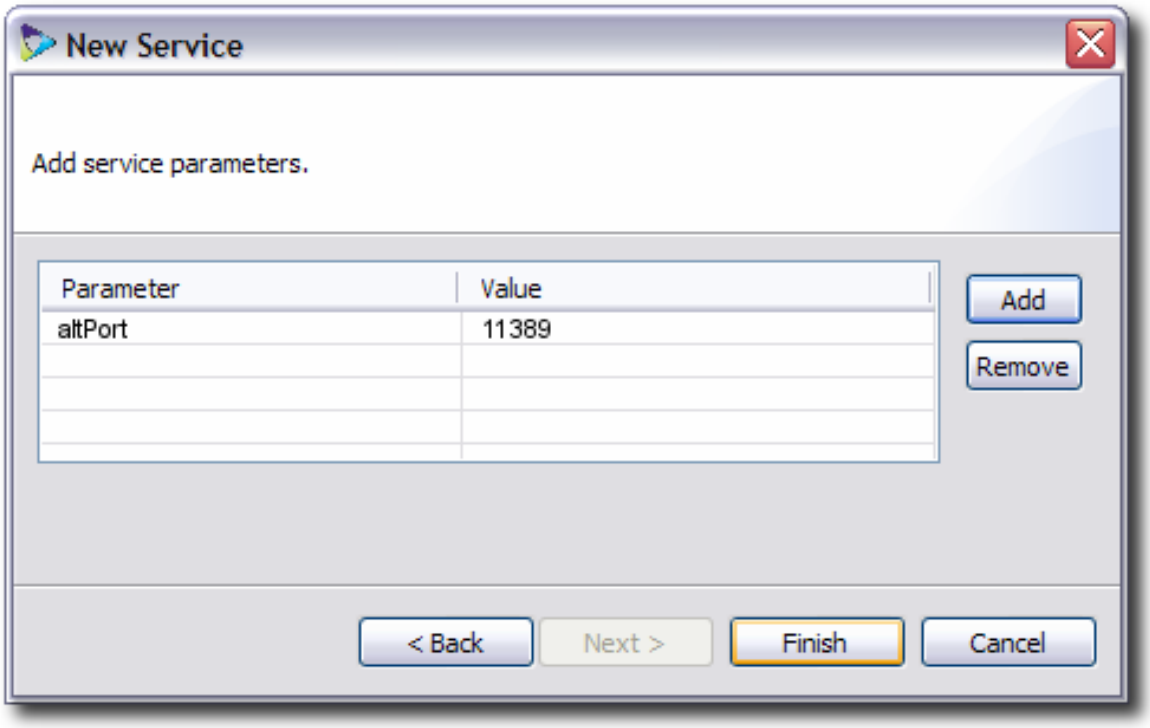
Alternatively, open the server, and right-click the **Services** folder, then select **New Service...**.

2. Fill in the name of the service, the service's class, and whether the service is enabled.



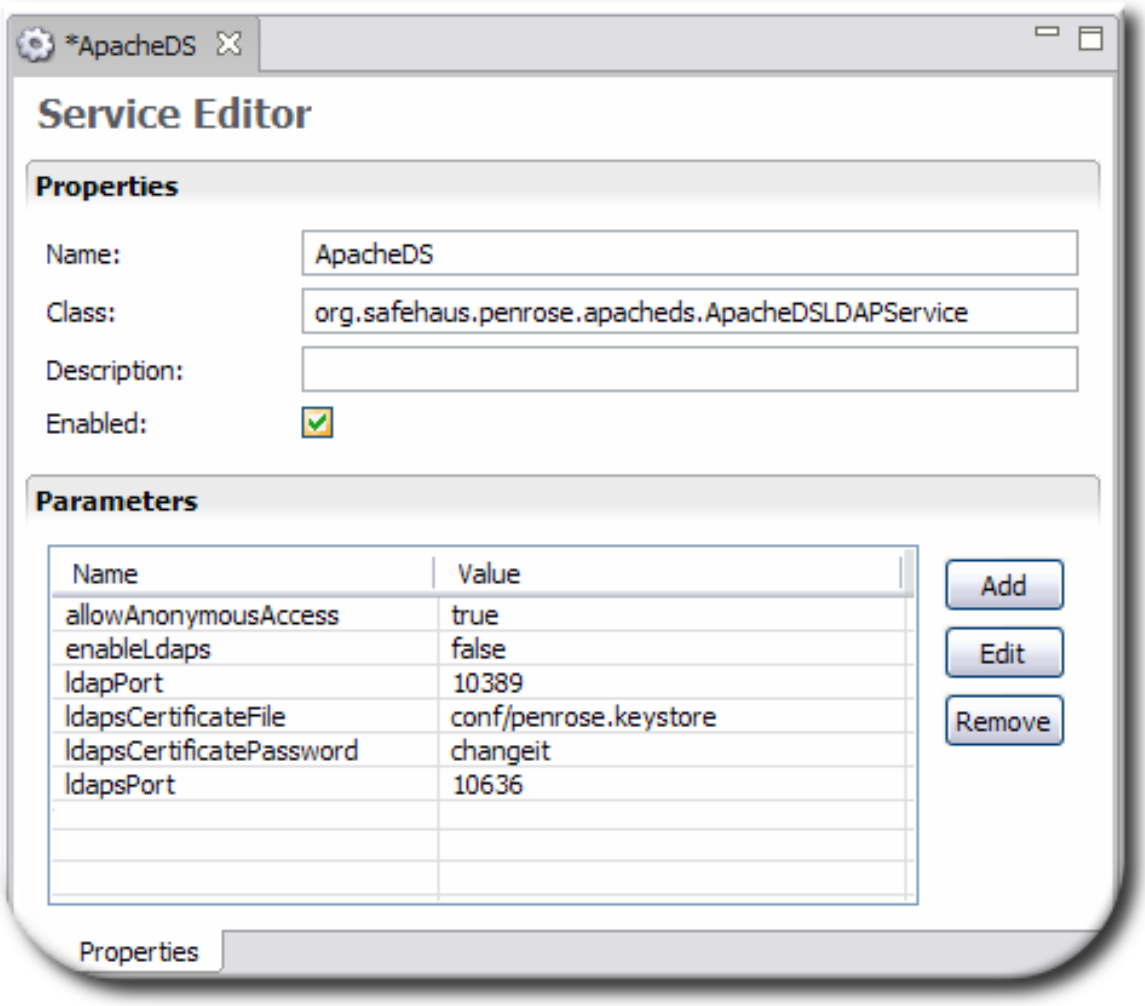
Any custom class must be listed in the service's configuration folder in the proper locate for all classes used by the service, **`/opt/vd-server-2.0/services/`***service_name*`/SERVICE-INF/lib`**. The default class, if none are specified, is **`org.safehaus.penrose.service.Service`**.

3. Fill in any configuration parameters that will be used by clients to connect to the service, such as port numbers or bind credentials.



4. Click **Finish**.

To edit a service's configuration, double-click the service under the **Services** folder to open the entry editor, and then edit or add configuration settings as desired.



12.2.2. Adding and Editing Services Manually

All of the service configuration, for a basic configuration, is stored in a subdirectory under **`/opt/vd-server-2.0/services/`**. This subdirectory is named whatever the Penrose Server service name is. For example, a JBoss service may be defined in **`/opt/vd-server-2.0/services/JBoss/`**; in Penrose Studio, this service is displayed as **JBoss**.

Within the *service_name* folder is, at least, a **`SERVICE-INF/`** folder which contains a **`service.xml`** file to define the service and a **`lib/`** directory which holds all of the JAR files used by the service.

NOTE

Depending on the type of service created, there can be additional database, configuration, tool, or log folders. The service can be as complex as required.

The **`service.xml`** file contains three pieces of information:

- Whether the service is enabled
- The class name used by the service
- Optional parameters to define connections to the service

This is illustrated in *Example 12.1, "Annotated Example service.xml File"*.

```
<service enabled="true"> whether the service is active

<service-class>org.safehaus.penrose.mina.MinaLDAPService</service-class>
class used by the service

  <parameter>  optional connection or configuration parameters
    <param-name>ldapPort</param-name>
    <param-value>10389</param-value>
  </parameter>

</service>
```

Example 12.1. Annotated Example service.xml File

IMPORTANT

Always restart Penrose Server after editing the configuration file. For example:

```
service vd-server restart
```

The configuration options for the **`service.xml`** file are listed in *Table 12.2, "Tags and Parameters for service.xml"*.

Source Parameters	Description	Example
<service>	The main file tag.	
enabled="..."	Sets whether the service is available to communicate with clients.	`<service en\ abled="true">`
<parameter>	Contains an attribuate-value pair of tags.	`<parameter> <param-name>ldapPort< /param-name> <param-valuee>10389</ param-valuee> </arameter>`

Source Parameters	Description	Example

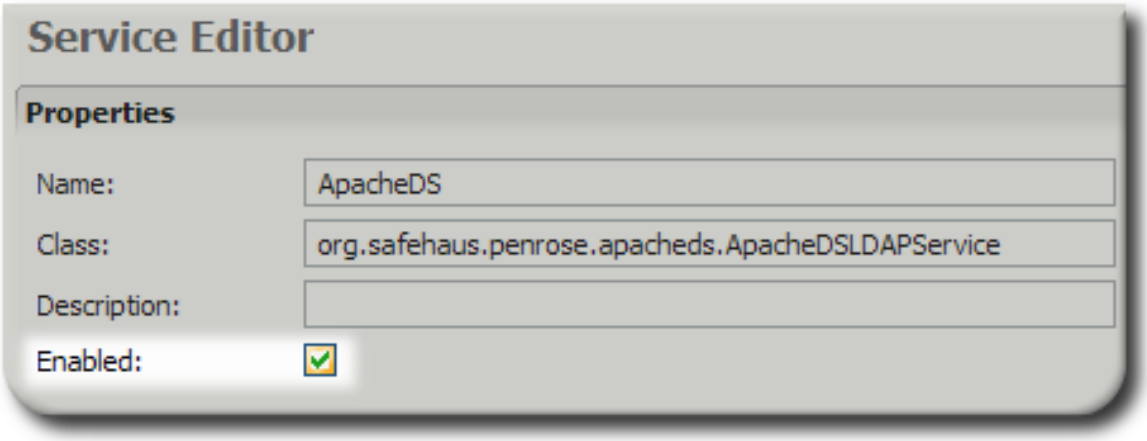
<param-name>	Contains the name of the attribute or configuration parameter.	`<param-name>`
		`  ldapPort`
		`</param-name>`

<param-value>	Contains the value of the attribute or configuration parameter specified in the **<param-name>** tag.	`<param-value>`
		`  10389`
		`</param-value>`

Table 12.2. Tags and Parameters for service.xml

12.3. Enabling and Disabling Services

A service must be enabled for a client to be able to use it to communicate with Penrose Server. To enable or disable services in Penrose Studio, open the properties editor for the service and check the **Enable** checkbox.



To enable or disable a service manually, edit its **service.xml** file and change the *enabled*= to **true** (enabled) or **false** (disabled).

```
<service enabled="true">
```

IMPORTANT

Always restart Penrose Server after editing the configuration file. For example:

```
service vd-server restart
```

Customizing Schema

This chapter describes the default schema included with Penrose Virtual Directory and explains how to create and load custom schema.

13.1. About Directory Schema

Schema is the way the LDAP directories like Red Hat Directory Server and Active Directory describe entries. The schema defines the type of entry in general through the *object classes*. Each object class has certain *attributes*, something which describes a part of the entry. For example, a **`person`** object class is for a general entry about a person. This has attributes for a person's first and last names and telephone number, all normal information to describe about a person.

TIP

For more detailed information on schema, check out *Understanding and Deploying LDAP Directory Services* by T. Howes, M. Smith, and G. Good and the IETF definitions at *http://1www.ietf.org/1rfc/1rfc2252.txt*.

The schema elements (object classes and attributes) which are available for Penrose Virtual Directory entries are loaded in schema files.

An attribute entry must define the following:

- An object identifier (OID)
- The syntax of the attribute, meaning the kind of value that is allowed, like an integer or string (see *Table 13.2, "Common LDAP Syntaxes"*)
- Whether there can only be a single value or whether there can be multiple uses of this attribute in the same entry
- Indexing information

For example, this is the Penrose Virtual Directory schema entry for the *`displayName`* attribute:

```
# displayName
attributetype ( 2.16.840.1.113730.3.1.241
        NAME 'displayName'
        DESC 'RFC2798: preferred name to be used when displaying entries'
        EQUALITY caseIgnoreMatch
        SUBSTR caseIgnoreSubstringsMatch
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.15
        SINGLE-VALUE )
```

An object class entry must define the following:

- An object identifier (OID)
- The list of required (**MUST**) and allowed (**MAY**) attributes
- Any superior object classes; a superior object class is an object class from which this object class inherits required and allowed attributes, without having to include those attributes in this object class definition
- The function of the object class. **STRUCTURAL** defines the type of entry being created and there can be only one per entry. **AUXILIARY** is an object class which contains attributes that can apply to many different kinds of directory entries. An **ABSTRACT** object class is not used directly by an LDAP entry but is referenced as a subclass by a **STRUCTURAL** or **AUXILIARY** object class.

For example, this is the Penrose Virtual Directory schema entry for the **inetorgperson** object class:

```
# inetOrgPerson
objectclass     ( 2.16.840.1.113730.3.2.2
    NAME 'inetOrgPerson'
    DESC 'RFC2798: Internet Organizational Person'
    SUP organizationalPerson
    STRUCTURAL
        MAY (
                audio $ businessCategory $ carLicense $ departmentNumber
$
                displayName $ employeeNumber $ employeeType $ givenName $
                homePhone $ homePostalAddress $ initials $ jpegPhoto $
                labeledURI $ mail $ manager $ mobile $ o $ pager $
                photo $ roomNumber $ secretary $ uid $ userCertificate $
                x500uniqueIdentifier $ preferredLanguage $
                userSMIMECertificate $ userPKCS12 )
        )
```

Penrose Virtual Directory schema files use the OpenLDAP schema formatting. This means that each schema file has the extension **.schema** and uses some of the default OpenLDAP schema files. For more information on OpenLDAP schema elements and files, see *http://1www.openldap.org/1doc/1admin24/1schema.html*.

13.2. Default Schema Elements and Files

A variety of applciations use schemas to describe their information; some of these schemas conform to international standards, and some are specific to the application. Penrose Virtual Directory has schema definitions for a variety of common directory applications loaded to define a variety of information in the virtual directory; these are listed in *Table 13.1, "Default Virtual Directory Schema Files"*.

In Penrose Virtual Directory, all of the schema files are located in the **/opt/vd-server-2.0/schema**.

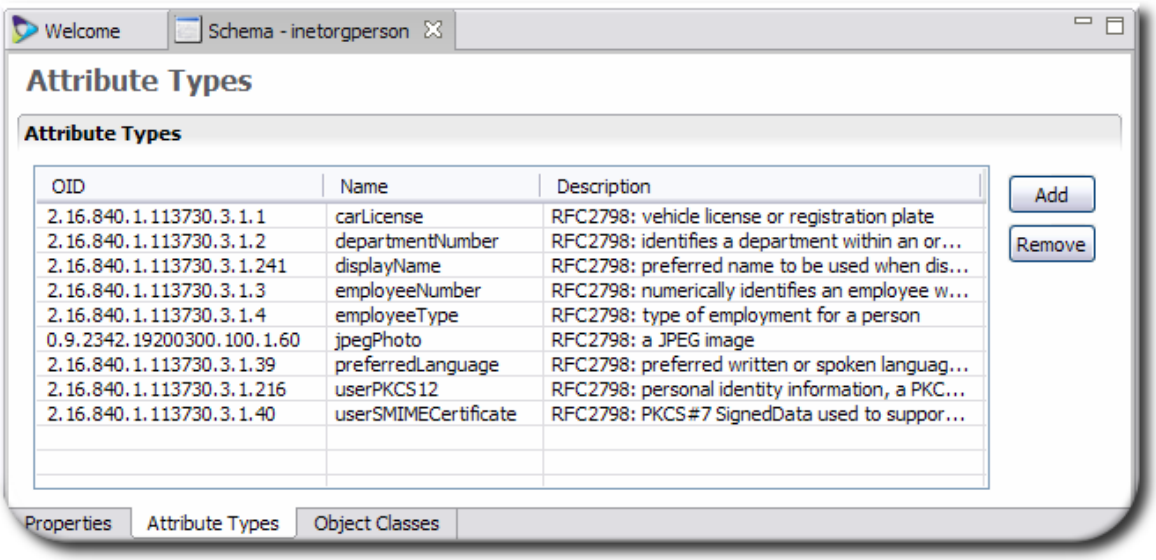
Schema File	Description
apache.schema	Contains object classes and attributes required to define Apache LDAP directory server entries.
apachedns.schema	Contains object classes and attributes required to define DNS records.
autofs.schema	Contains object classes and attributes required to define NFS mount shares in an NIS entry.
changelog.schema	Contains object classes and attributes required to define a changelog entry to track changes in data masters.
collective.schema	Contains X.500 elements for collective attributes for LDAP entries.
corba.schema	Contains schema elements to represent CORBA objects in an LDAP directory.
core.schema	Contains the core OpenLDAP schema elements.
cosine.schema	Contains LDAP schema derived from X.500 COSINE pilot schema.
dhcp.schema	Contains object classes and attributes to define DHCP server information.
extension.schema	Contains object classes and attributes to extend LDAP support in applications like Outlook and PAM.
inetorgperson.schema	Contains attributes to define **`inetOrgPerson`** entries for people within an organization.
java.schema	Contains object classes and attributes to define Java objects in an LDAP directory.
krb5kdc.schema	Contains object classes and attributes to define Kerberos configuration entries.
misc.schema	Contains various object classes and attributes to define mail entries, including as mail accounts and mail servers. **WARNING** Several of these attributes are obsolete, not fully supported, or depend on expired standards. These attributes should not be used in a full production environment.
nis.schema	Contains object classes and attributes to define NIS servers and domains in an LDAP directory.
samba.schema	Contains object classes and attributes for storing Samba user accounts and group maps in LDAP entries.
solaris.schema	Contains object classes and attributes required

Schema File	Description
	to configure a NIS entry in a Red Hat Directory Server-style LDAP directory on Solaris.
system.schema	Contains required and administrative object classes and attributes to configure an LDAP directory, including root DSEs, administrative subschema, referrals, and general entry information.

Table 13.1. Default Virtual Directory Schema Files

These files are listed in Penrose Studio in the **Built-in Schema** folder.

The full configuration for each schema file is viewable in Penrose Studio and can be edited by double-clicking the attribute or object class to edit its setup, but adding additional schema elements, or by removing schema elements.



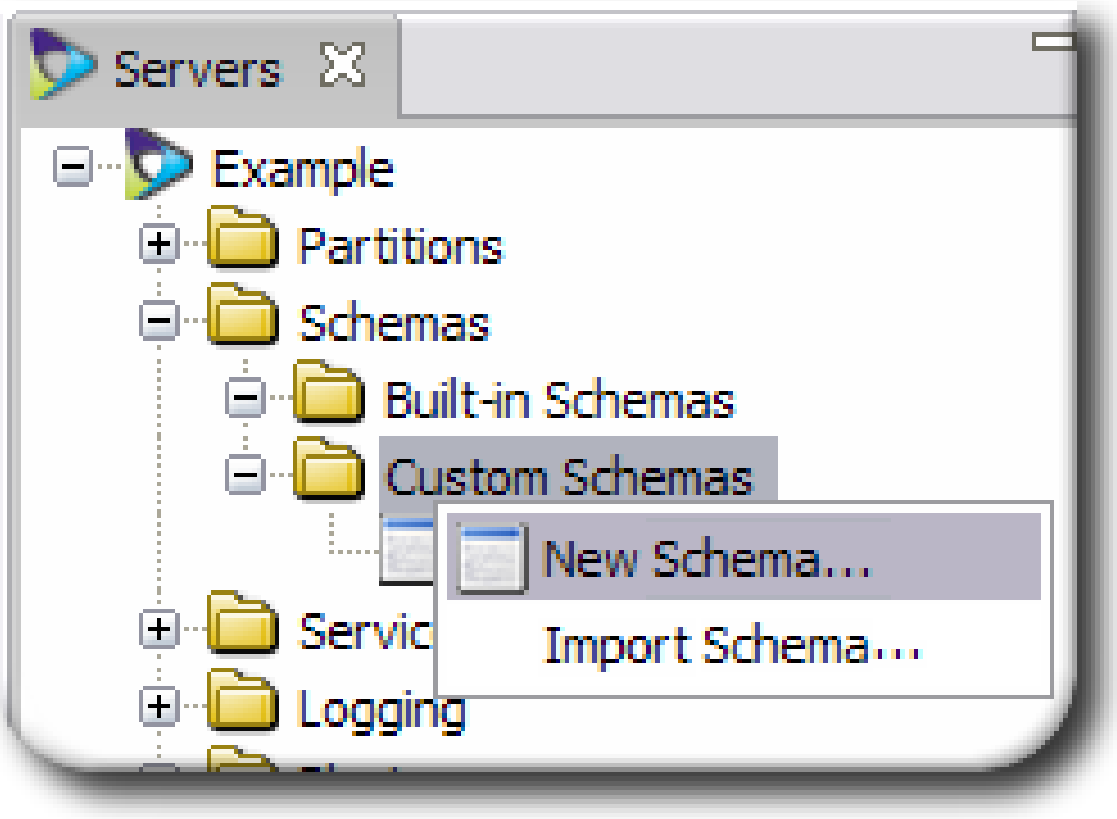
13.3. Creating Custom Schema

Schema files can be created through Penrose Studio. The new schema is then processed through Penrose Studio and sent to the Penrose Server, which writes it to a new file in the server configuration.

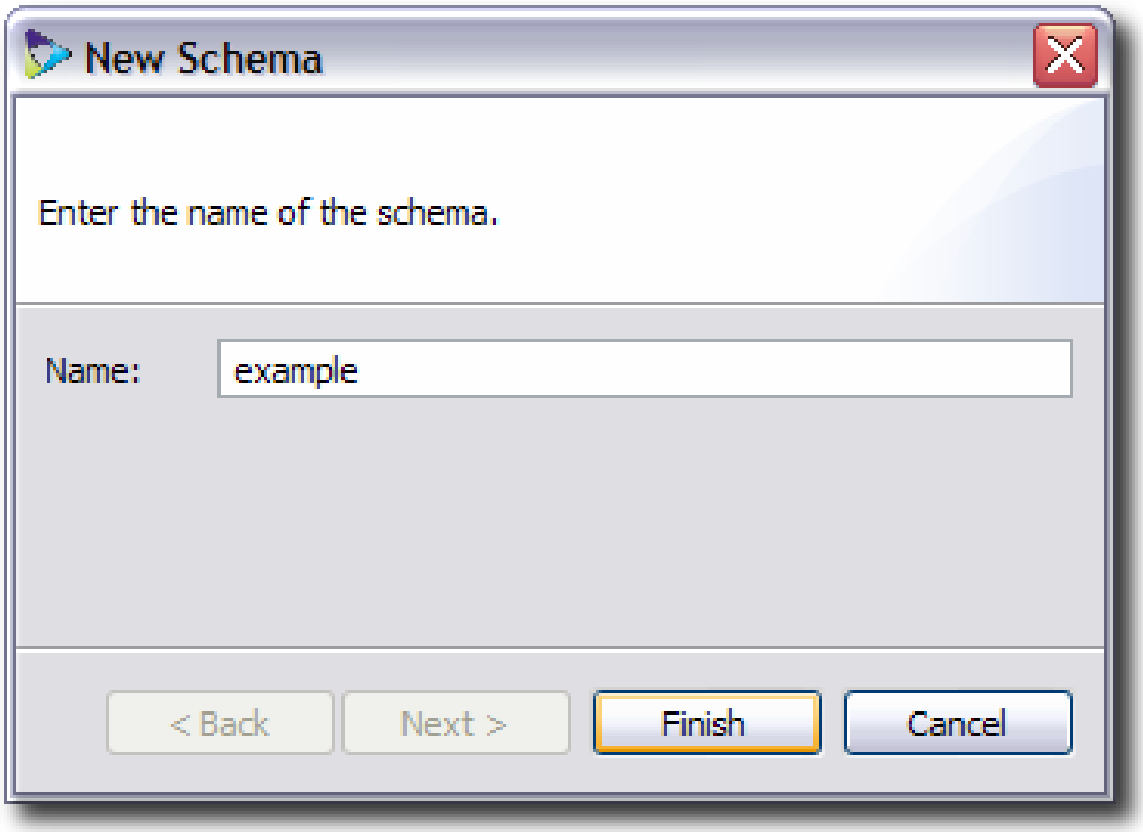
To create custom schema in Penrose Studio:

1. In the server entry, expand the **Schema** folder.

2. Right-click the **Custom Schemas** folder, and select **New Schema...**.



3. Name the new schema, and click **Finish**.



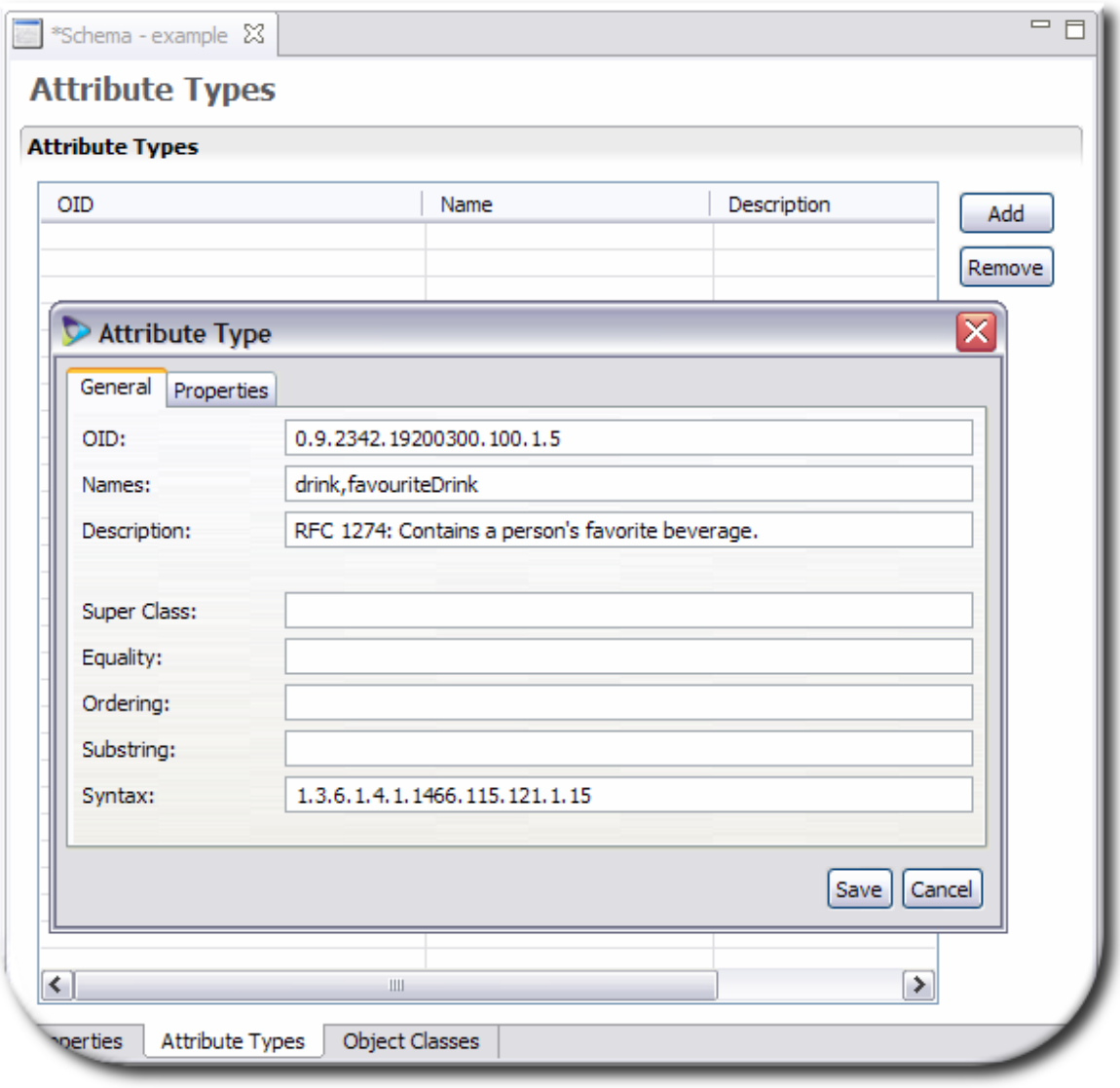
4. Double-click the new schema entry to open the entry editor.

5. Click the **Attributes** tab at the bottom of the editor.

6. Click the **Add** button.

7. Fill in the new attribute informaton.

   - The OID (required)
   - The name (required)
   - The OID of the syntax, which is the format of the attribute value (required)
   - A description (optional)
   - The matching rules for equality, substring, and ordering indexes, such as **`caseIgnoreMatch`** (optional)



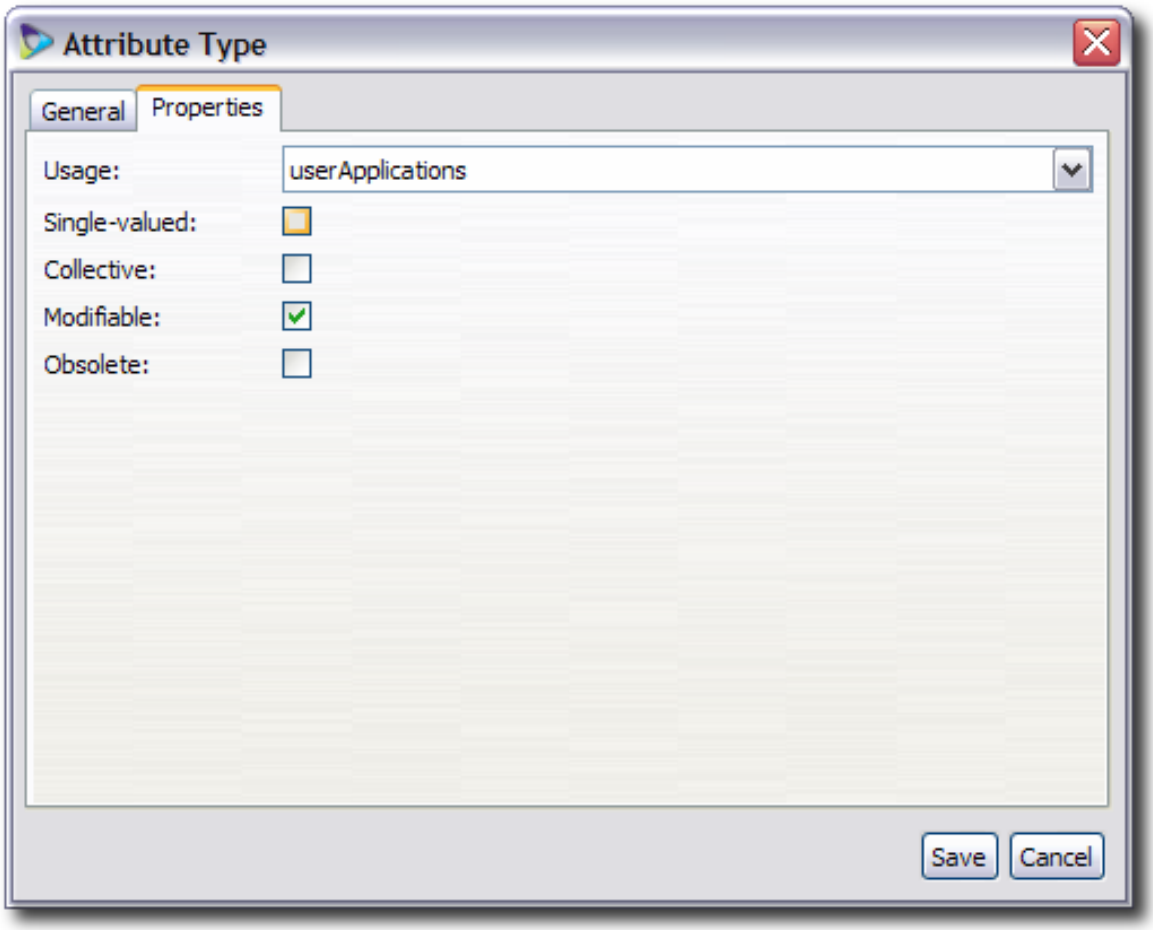
The OID can be any attribute defined by IANA or can be custom. The possible syntax OIDs are listed in *Table 13.2, "Common LDAP Syntaxes"*.

OID	Name
1.3.6.1.4.1.1466.115.121.1.4	Audio
1.3.6.1.4.1.1466.115.121.1.5	Binary
1.3.6.1.4.1.1466.115.121.1.7	Boolean
1.3.6.1.4.1.1466.115.121.1.12	DN
1.3.6.1.4.1.1466.115.121.1.15	Directory String
1.3.6.1.4.1.1466.115.121.1.22	Telephone Number
1.3.6.1.4.1.1466.115.121.1.24	Generalized Time
1.3.6.1.4.1.1466.115.121.1.26	IA5 String
1.3.6.1.4.1.1466.115.121.1.27	Integer
1.3.6.1.4.1.1466.115.121.1.28	JPEG

OID	Name
1.3.6.1.4.1.1466.115.121.1.36	Numeric String
1.3.6.1.4.1.1466.115.121.1.40	Octet String
1.3.6.1.4.1.1466.115.121.1.38	OID
1.3.6.1.4.1.1466.115.121.1.41	Postal Address

Table 13.2. Common LDAP Syntaxes

8. Fill in additional properties for the attribute.



- The type of attribute, in the **Usage** drop-down menu, can be **`userApplication`** (used by a client independent of the LDAP directory), **`directoryOperation`** (used by the LDAP directory), **`distributedOperation`** (shared between directory server instances), and **`dSAOperation`** (unique to each individual directory server instance).

- Whether the attribute is single- or multi-valued

- Whether the attribute is shared among multiple types of entries (collective)

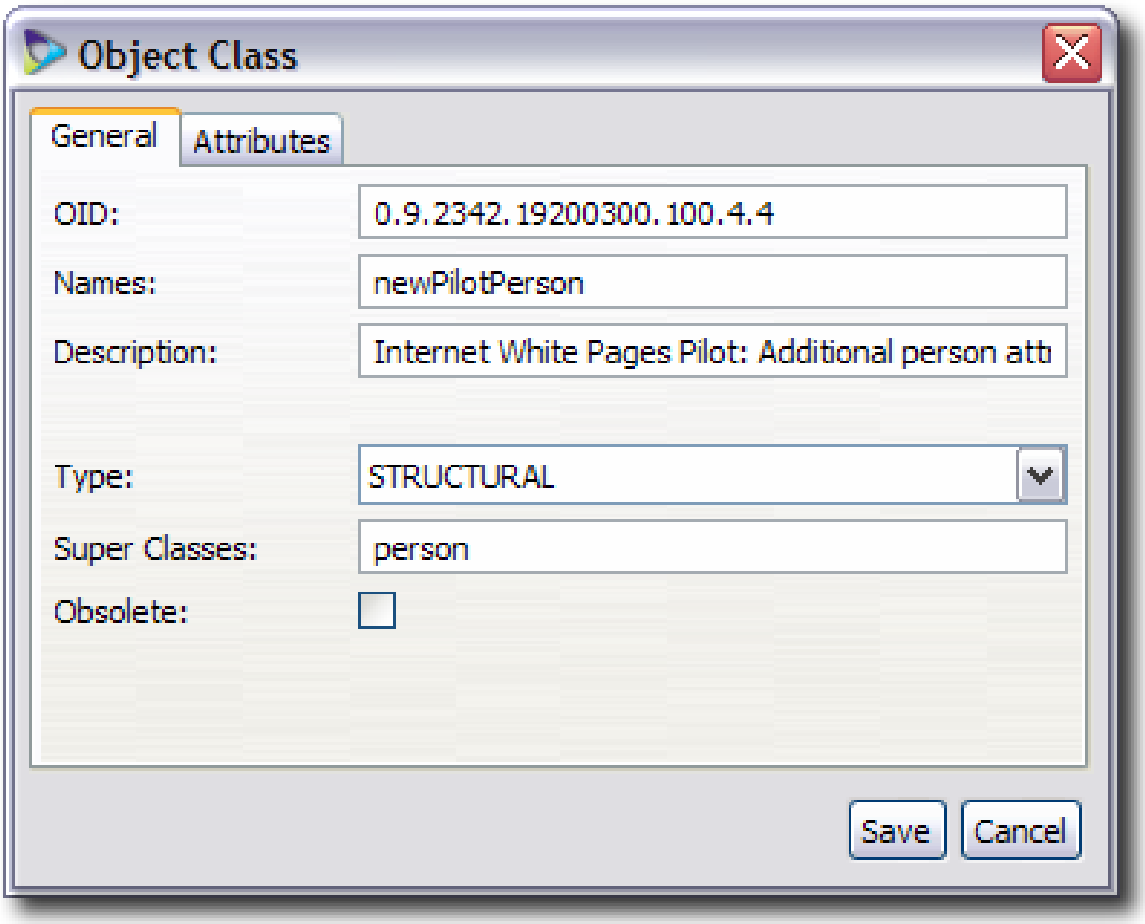
- Whether the attribute definition can be modified
- Whether the attribute is obsolete, which means it may not be widely supported

9. Add an attribute entry for every attribute which will be used by the object classes in the schema.

10. Click the **Attributes** tab at the bottom of the editor.

11. Click the **Add** button.

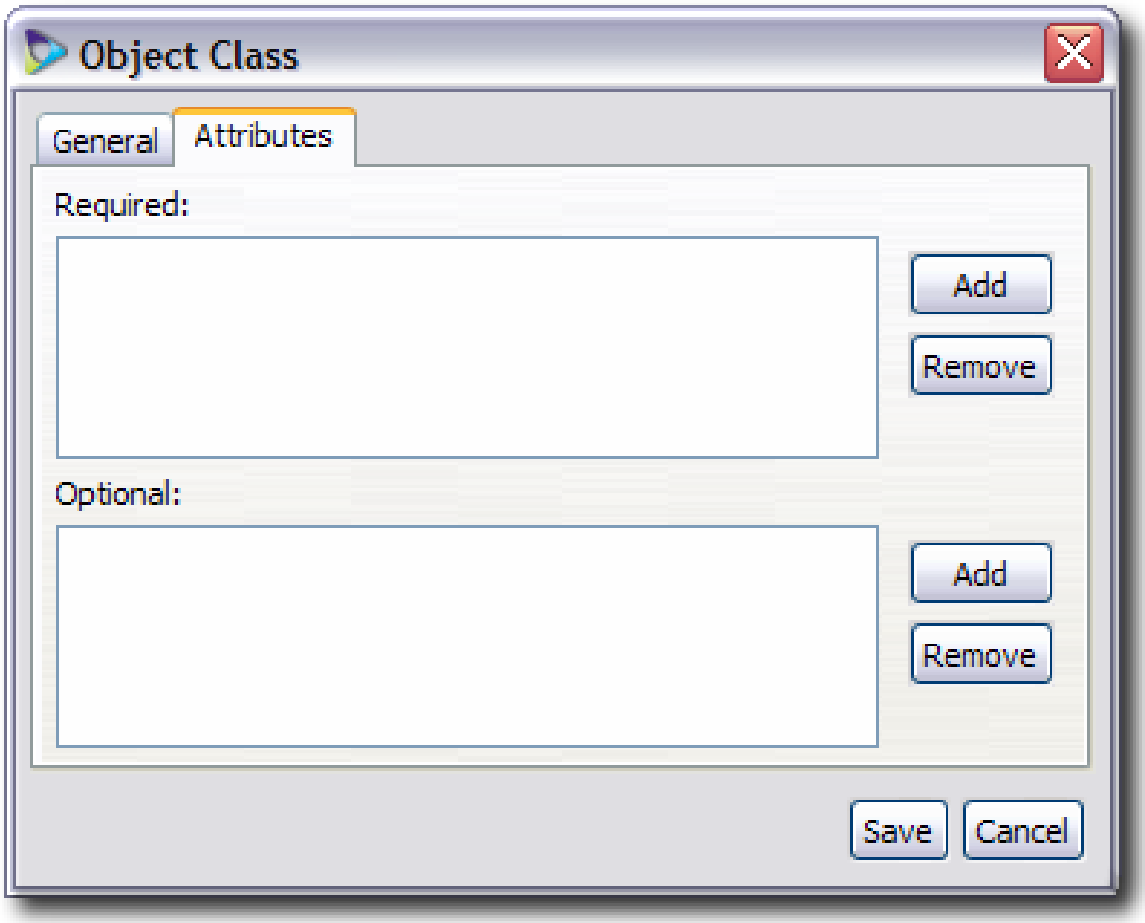
12. Fill in the object class information.



- The OID (required), which can be defined by IANA or custom
- The name (required)
- The type (required)

- Any superior object class (optional)
- A description (optional)
- Whether the object class is obsolete (optional)

1 Click the **Attributes** tab at the top of the object class editor, and click **Add** to add any required or al-
3. lowed attributes for the object class.



NOTE

Penrose Server does not validate the schema, so it is possible to create schema entries which violate the LDAP standards.

1 Click **Finish**.
4.

15. Close Penrose Studio, and restart Penrose Server. When Penrose Studio is opened again, the new imported schema is listed under the **Custom Schema** folder.

13.4. Exporting Schema

It is possible to export the schema configured for a connection so it can be used in other areas of Penrose Server or with other applications.

1. Open the server entry.

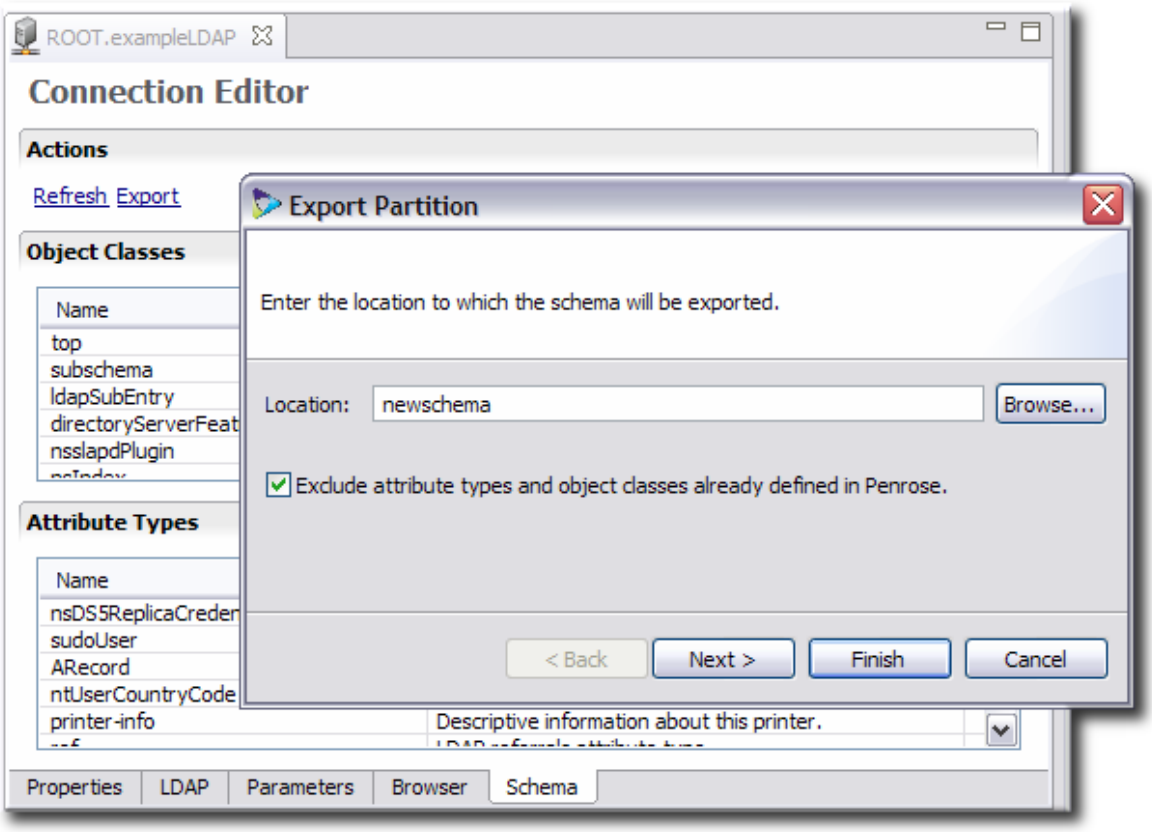
2. In the top menu, expand the **Partitions** menu item, and select the **Connections** folder.

3. Open the **Connections** folder, and double-click the connection.

4. Click the **Schema** tab at the bottom of the window.

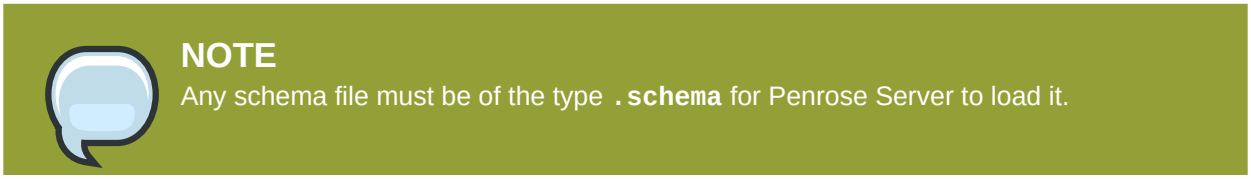
5. Click the **Export** link.

6. Select the location to save the schema file to.



13.5. Importing Schema Files

Importing a schema file uses Penrose Studio to load an existing schema file into Penrose Server. As with creating a new schema file, the local file is imported into the Penrose Studio configuration, serialized to Penrose Server, and written into the server configuration.

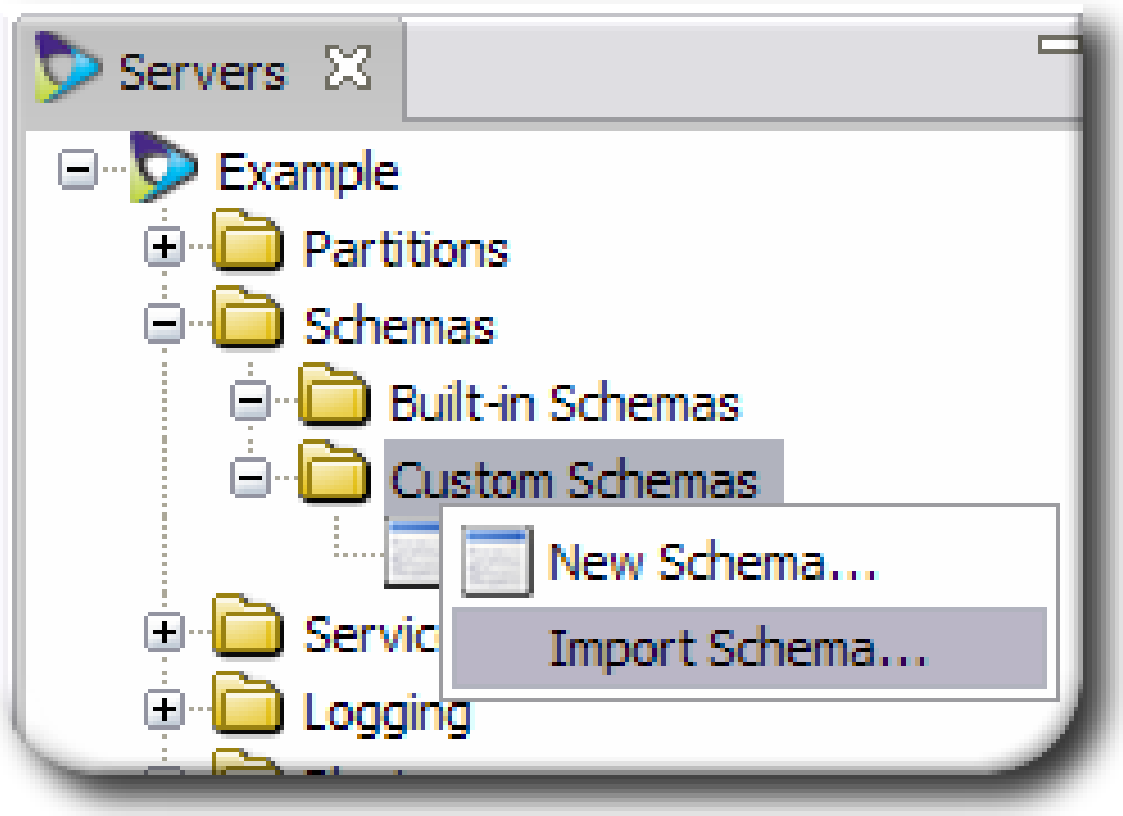


NOTE

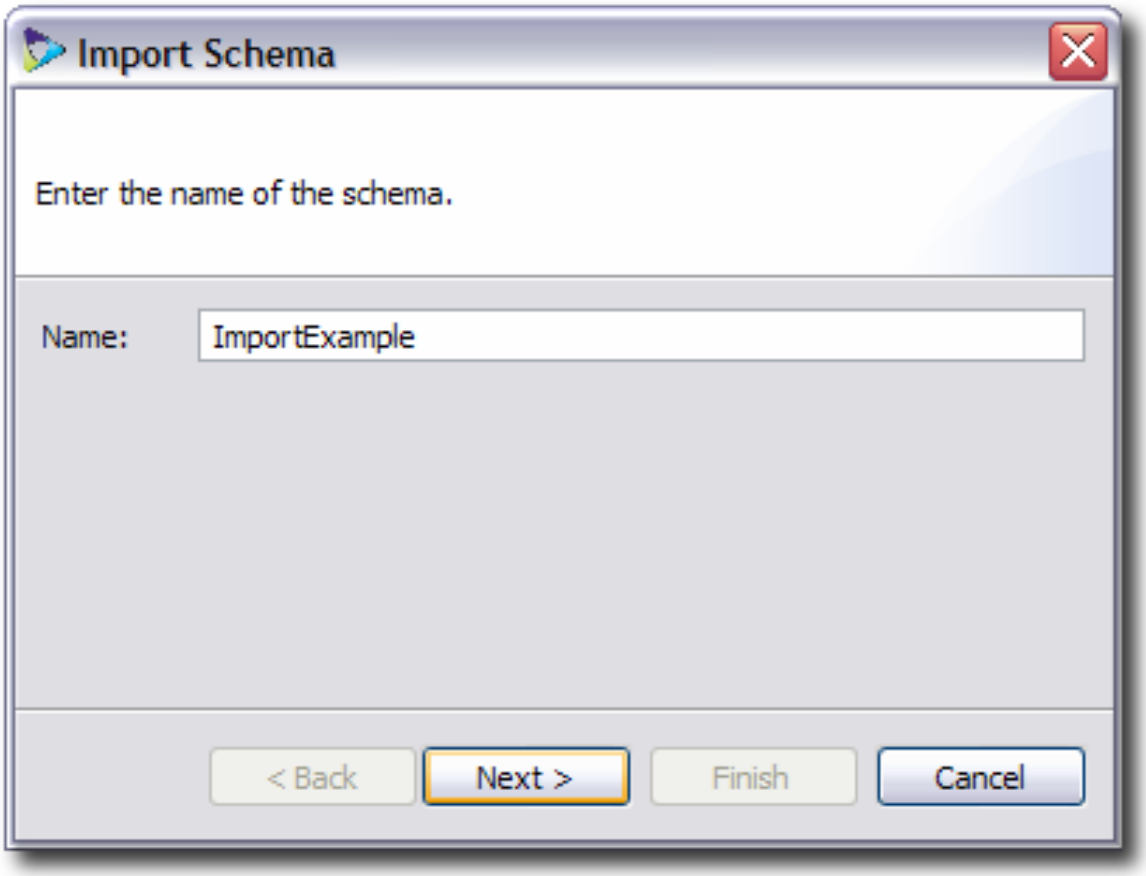
Any schema file must be of the type **`.schema`** for Penrose Server to load it.

1. In the server entry, expand the **Schema** folder.

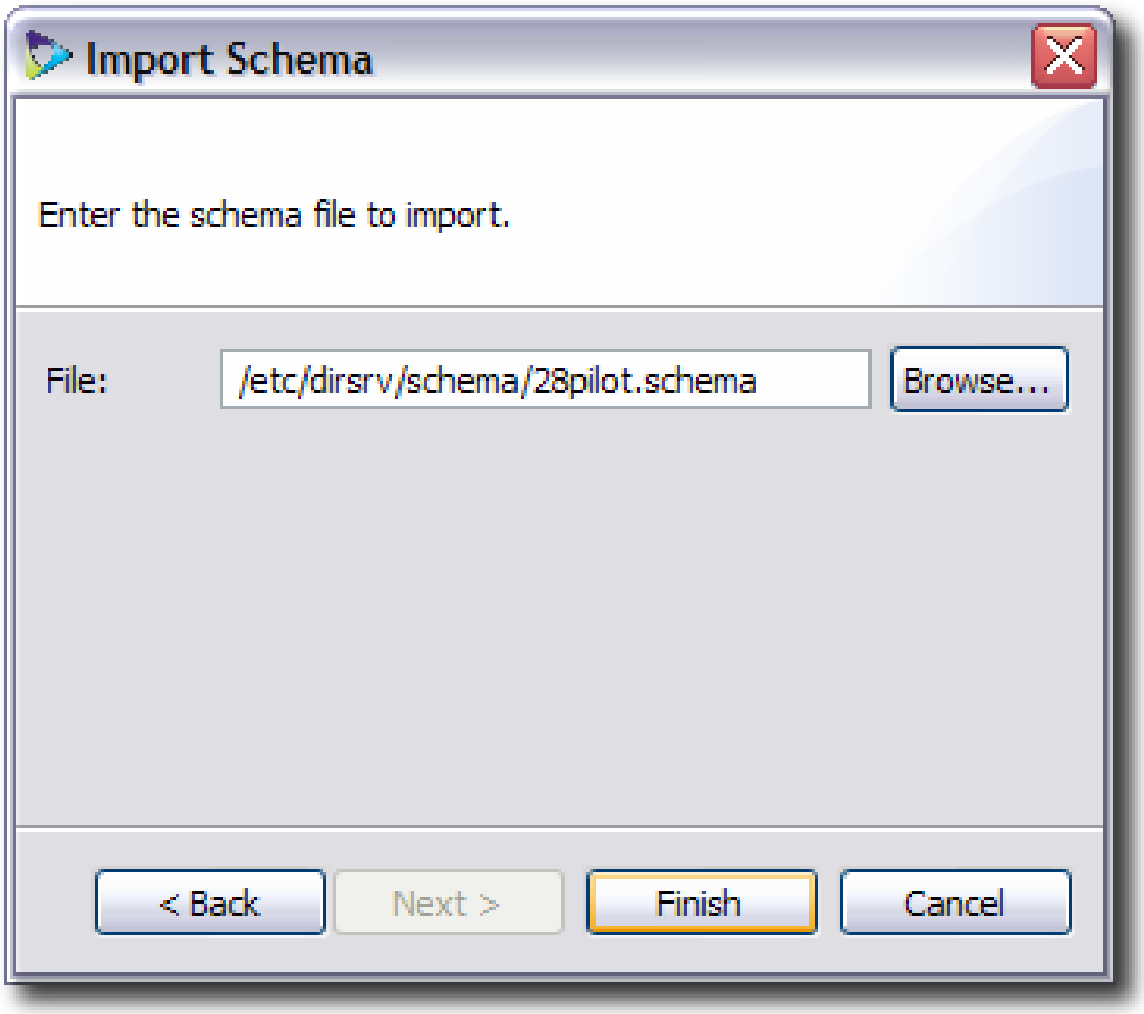
2. Right-click the **Custom Schemas** folder, and select **Import Schema**.



3. Name the schema entry.



4. Enter the absolute path to the schema file.



5. Click **Finish**.

6. Close Penrose Studio, and restart Penrose Server. When Penrose Studio is opened again, the new imported schema is listed under the **Custom Schema** folder.

7. All custom schema files must also be loaded into Penrose Virtual Directory's OpenDS service. The formats used for Penrose Virtual Directory and OpenDS schema are the same, but the extensions are different. Schema files in Penrose Virtual Directory have a **`.schema`** extension and in OpenDS have a **`.ldif`** extension.

```
cp custom.schema
/opt/vd-server-2.0/services/OpenDS/config/schema/custom.ldif
```

   If necessary, convert the custom schema file to an OpenDS format, as described in *Section 13.7, "Converting the Schema Formatting from OpenLDAP to OpenDS"*.

13.6. Loading a Schema File Manually

Any schema file can be loaded into Penrose Server to be available for the virtual directory entries. For Penrose Server to recognize the schema file, it must have a **.schema** extension.

To load the schema into Penrose Server manually:

1. Create the schema file; this is described somewhat in *Section 13.1, "About Directory Schema"*.

NOTE
The schema file must have a **.schema** extension.

2. Copy the schema file into the **/opt/vd-server-2.0/schema** directory.

3. All custom schema files must also be loaded into Penrose Virtual Directory's OpenDS service. The formats used for Penrose Virtual Directory and OpenDS schema are the same, but the extensions are different. Schema files in Penrose Virtual Directory have a **.schema** extension and in OpenDS have a **.ldif** extension.

```
cp custom.schema
/opt/vd-server-2.0/services/OpenDS/config/schema/custom.ldif
```

   If necessary, convert the custom schema file to an OpenDS format, as described in *Section 13.7, "Converting the Schema Formatting from OpenLDAP to OpenDS"*.

4. Restart Penrose Server.

```
service vd-server restart
```

IMPORTANT
Always restart Penrose Server after editing the configuration file.

13.7. Converting the Schema Formatting from OpenLDAP to OpenDS

Penrose Virtual Directory uses OpenDS-style schema files for some internal operations and Open-LDAP-style schema files for others. Penrose Virtual Directory has a tool, **schema.sh**, which can convert the OpenLDAP schema files to OpenDS schema files so that they can be loaded into Penrose

Server.

Any custom schema file used with Penrose Virtual Directory has to be loaded into its OpenDS service as well as Penrose Server.

1. Open the OpenDS directory for the integrated OpenDS services.

```
cd /opt/vd-server-2.0/services/OpenDS/bin/
```

2. Run the **schema.sh** script to convert the specified schema file to the appropriate format.

```
schema.sh /path/to/file.schema custom.ldif
```

3. Copy the converted schema file into the OpenDS directory.

```
cp custom.ldif /
opt/vd-server-2.0/services/OpenDS/config/schema/custom.ldif
```

4. Restart Penrose Server.

```
service vd-server restart
```

Configuring Cache

Penrose Virtual Directory can configure cache settings on data sources and for individual entries. Both caches can be configured to use persistent or in-memory cache or be disabled. The settings for both the source and entry cache affect the Penrose Server performance.

14.1. About Penrose Cache Types

Penrose Studio uses cache at two levels, for the Penrose Server source and another for the Penrose Server entry. The source cache caches the data read from the data sources. The entry cache caches the LDAP entries generated by the mappings. The cache is invalidated when it expires or an update operation is performed against it.

Each source and entry mapping has its own cache which is configured separately, described in *Chapter 5, Managing Partitions* and *Chapter 5, Managing Partitions*, respectively, for the cache size and expiration.

The type of cache, however, is configured globally, and Penrose Virtual Directory can configure its cache in two ways:

- *In-memory cache*. Storing the cache data in local memroy is fast because it does not require initialization and stores only a partial copy of the data from the source. However, there is a risk since all data are lost when the server is restarted. In-memory cache is recommended for databases that are small or rarely change.

- *No cache (the default)*. Penrose Server has cache disabled by default, so all operations are run directly against the underlying sources. Disabling cache ensures that the information displayed in Penrose is the same as the data stored in the source. Disabling cach is recommended for using real-time mapping.

There are some performance differences between in-memory cache, persistent cache, and no cache, as shown in *Table 14.1, "Performance Measurements for Different Cache Configurations"*.

Type of Cache	Number of Database	Cache Hit Times (in seconds)		Cache Miss Times (in seconds)	
		First Entry	Last Entry	First Entry	Last Entry
No Cache[1]	100			0s	2s
	1000			1s	6s
	5000			3s	27s
	10000			5s	55s
In-Memory Cache	100	0s	2s	0s	2s
	1000	0s	6s	1s	6s
	5000	1s	25s	3s	27s
	10000	8s	60s	8s	60s

[1] If cache is disabled, then there are no entries in the cache. Any search requests, then, appear as a cache miss.

Table 14.1. Performance Measurements for Different Cache Configurations

14.2. Using In-Memory Cache

In-memory cache is configured through a specific module, the **CacheModule**. This module must be specified in the **modules.xml** file (either in **/opt/vd-server-2.0/conf** for the default partition or in **/opt/vd-server-2.0/partitions/*partition_name*/DIR-INF** for additional partitions). If the module is not listed in **modules.xml**, then Penrose Virtual Directory does not use in-memory cache for virtual entries.

1. Open the **modules.xml** file.

```
cd /opt/vd-server-2.0/conf

vi modules.xml
```

2. Configure the cache settings. There are three possible paramters: *querySize*, which sets the maximum number of unique queries to the cache; *resultSize*, which sets the maximum number of entries which are returned; and *expiration*, which sets when the cache expires (in minutes).

```
<modules>

  <module name="CacheModule">
<module-class>org.safehaus.penrose.cache.module.CacheModule</module-class
>
    <parameter>
      <param-name>querySize</param-name>
      <param-value>250</param-value>
    </parameter>
    <parameter>
      <param-name>resultSize</param-name>
      <param-value>100</param-value>
    </parameter>
    <parameter>
      <param-name>expiration</param-name>
      <param-value>3</param-value>
    </parameter>
  </module>

</modules>
```

Parameter	Default Value	Value Range
querySize	10	0 (unlimited) to any integer
resultSize	100	0 (unlimited) to any integer
expiration	5	0 (never expires) to any integer

3. As with any module, it is possible to map the **CacheModule** to a specific subtree in the virtual directory. Add a module mapping entry, specifying the **CacheModule**, to the **modules.xml** file. For example:

```
  <module-mapping>
    <module-name>CacheModule</module-name>
```

```
    <base-dn>ou=people,dc=example,dc=com</base-dn>
  </module-mapping>
```

4. Restart Penrose Server.

```
service vd-server restart
```

IMPORTANT
Always restart Penrose Server after editing the configuration file.

14.3. Disabling Cache

The cache is disabled by setting the `enabled=` argument for the entry.

1. Open the **`modules.xml`** file.

2. Add the `enabled=` argument to the cache's entry, and set its value to **`false`**.

```
<module name="CacheModule" enabled=false>
```

3. Restart Penrose Server.

```
service vd-server restart
```

IMPORTANT
Always restart Penrose Server after editing the configuration files.

Appendix A. Using Penrose Virtual Directory Command-Line Tools

Several scripts are included with Penrose Virtual Directory to configure and control Penrose Server operations. The various script provide a simple way to perform Penrose Server management operations (like configuring partitions) through the command line; each script has a corollary to a Penrose Studio action. These scripts' options and usage information are described in more detail in this chapter.

A.1. Tool Locations

These tools are all contained in the **`/opt/vd-server-2.0/bin`** directory and must be run from that directory.

A.2. cache.sh

This script can clear the in-memory cache.

Options

cache.sh [*options*] *commands* [*partition* [*source*]]

Options	Description
-?, - -help	Prints the help for the script.
-d	Runs the script in debug mode.
-v	Runs the script in verbose mode.

Table A.1. cache.sh Options

Commands	Description
clean	Empties the tree nodes and tables for the cache.
drop	Deletes the tree notes and tables for the cache.
invoke *application_command*	Manually runs the specified command for the cache. For example, this is another way of having the cache module automatically run the **`clear`** method.
partition	Optionally, specifies the name of the partition being modified.
source	Optionally, specifies the data source for the partition. This can only be used when a *partition* is specified.

Table A.2. cache.sh Commands

A.3. connection.sh

The **`connection.sh`** script manages connections within the partition by importing or exporting, starting and stopping, and editing the connections.

Options

connection.sh [*options*] *commands* connection *partition*

Options	Description
-?, - -help	Prints the help for the script.
-d	Runs the script in debug mode.
-D *rootDN*	Gives the root user for Penrose Server; the default username is **`uid=admin`**, **`ou=system`**.
-h *hostName*	Gives the hostname for the machine on which the service is running.
-p *port*	Gives the port number for the service to which Penrose Server is connecting.
-P *protocol*	Gives the protocol for Penrose Server to communicate with the MBeans object; the only supported JMX protocol is **`rmi`**.
-w *password*	Gives the password for the Penrose Server root user; the default is **`secret`**.
-v	Runs the script in verbose mode.

Table A.3. connection.sh Options

Commands	Description
show	Lists the connections configured for the given partition.
start	Starts the connection for that partition.
stop	Stops the connection for that partition.
restart	Restarts the connection for that partition.
import connection *path*	Imports the connection configuration from the specified directory into the given partition.
export connection *path*	Exports the connection configuration for the named partition to the specified directory.
rename connection *name1 name2*	Renames the connection from *name1* to *name2*.
remove connection *name*	Deletes the specified connection.
partition	Specifies the name of the partition being modified.
invoke *application_command*	Manually runs the specified command for the connection server.

Table A.4. connection.sh Commands

A.4. module.sh

The **`module.sh`** script manages modules used by the partition. The script can be used to import or export, start and stop, modify, and remove modules associated with any partition.

Options

module.sh [*options*] *commands* module *partition*

Options	Description
-?, - -help	Prints the help for the script.
-d	Runs the script in debug mode.
-D *rootDN*	Gives the root user for Penrose Server; the default username is **`uid=admin`**, **`ou=system`**.
-h *hostName*	Gives the hostname for the machine on which the service is running.
-p *port*	Gives the port number for the service to which Penrose Server is connecting.
-P *protocol*	Gives the protocol for Penrose Server to communicate with the MBeans object; the only supported JMX protocol is **`rmi`**.
-w *password*	Gives the password for the Penrose Server root user; the default is **`secret`**.
-v	Runs the script in verbose mode.

Table A.5. module.sh Options

Commands	Description
show	Lists the modules configured for the given partition.
start	Starts the module for that partition.
stop	Stops the module for that partition.
restart	Restarts the module for that partition.
import module *path*	Imports the module configuration from the specified directory into the given partition.
export module *path*	Exports the module configuration for the named partition to the specified directory.
rename module *name1 name2*	Renames the module from *name1* to *name2*.
remove module *name*	Deletes the specified module.
partition	Specifies the name of the partition being modified.
invoke *application_command*	Manually runs the specified command for the cache. For example, this is another way of having the cache module automatically run the **`clear`** method.

Table A.6. module.sh Commands

A.5. monitor.sh

Penrose Server has an integrated monitoring feature which tracks the Penrose Server process and will send a notification, either an email or an SNMP trap, if the server stops.

Options

monitor.sh [*options*] [start|stop]

NOTE

All of the options can only be run when the **`monitor.sh`** script is run in the foreground.

Options	Description
-?, - -help	Prints the help for the script.
-d	Runs the script in debug mode.
-v	Runs the script in verbose mode.
--version	Returns the version number of the current Penrose Server.

Table A.7. monitor.sh Options

NOTE

Using either script command causes the **`monitor.sh`** script to run in the background.

Commands	Description
start	Starts the monitor service for Penrose Server.
stop	Stops the monitor service for Penrose Server.

Table A.8. monitor.sh Commands

A.6. nis.sh

Used to synchronize a NIS server, meaning this script copies the entries in the NIS server and any changes over to the global repository.

Options

nis.sh [*options*] synchronize *`Federation domain`* [*`NIS domain`* [*`NIS maps...`*]]

Options	Description
-?, - -help	Prints the help for the script.
-d	Runs the script in debug mode.
-D *rootDN*	Gives the root user for Penrose Server; the default username is **uid=admin**, **ou=system**.
-h *hostName*	Gives the hostname for the machine on which the service is running.
-p *port*	Gives the port number for the service to which Penrose Server is connecting.
-P *protocol*	Gives the protocol for Penrose Server to communicate with the MBeans object; the only supported JMX protocol is **rmi**.
-w *password*	Gives the password for the Penrose Server root user; the default is **secret**.
-v	Runs the script in verbose mode.
--version	Returns the version number of the current Penrose Server.

Table A.9. nis.sh Options

Commands	Description
synchronize	Sycnhronizes the NIS server with the global repository. Depending on the options used with it, this can synchronize a specific NIS domain or NIS maps with a specific federation domain.

Table A.10. nis.sh Commands

A.7. partition.sh

Partitions are the primary entry in Penrose Server, which contain all of the data sources, connections, modules, and other virtual directory configuration. Partitions, as with the other configuration entries, can be managed through the command line,

Options

partition.sh [*options*] *commands partition*

Options	Description
-?, - -help	Prints the help for the script.
-d	Runs the script in debug mode.
-D *rootDN*	Gives the root user for Penrose Server; the default username is **uid=admin**, **ou=system**.
-h *hostName*	Gives the hostname for the machine on which the service is running.

Options	Description
-p *port*	Gives the port number for the service to which Penrose Server is connecting.
-P *protocol*	Gives the protocol for Penrose Server to communicate with the MBeans object; the only supported JMX protocol is **`rmi`**.
-w *password*	Gives the password for the Penrose Server root user; the default is **`secret`**.
-v	Runs the script in verbose mode.

Table A.11. partition.sh Options

Commands	Description
show	Lists the partitions configured for the Penrose Server.
start	Starts the partition.
stop	Stops the partition.
restart	Restarts the partition.
invoke *application_command*	Manually runs the specified command for the partition (the virtual directory). For example, **`invoke`** can be used to run LDAP commands such as **`ldapmodify`** and **`ldapsearch`** against the partition's virtual directory.
import *path*	Imports the partition configuration from the specified directory.
export *path*	Exports the partition configuration to the specified directory.
rename *name1 name2*	Renames the partition from *name1* to *name2*.
remove *name*	Deletes the specified partition.
partition	Specifies the name of the partition being modified.

Table A.12. partition.sh Commands

A.8. schema.sh

There are two **`schema.sh`** scripts with Penrose Virtual Directory, both to use for the integrated LDAP services to communicate with clients. One is for compiling schema to work with ApacheDS. The other is for converting OpenLDAP formatted schema files into OpenDS formatted schema files.

These **`schema.sh`** scripts are in two different locations:

- The ApacheDS tool is in **`/opt/vd-server-2.0/services/ApacheDS/bin/`**.

- The ApacheDS tool is in **`/opt/vd-server-2.0/services/OpenDS/bin/`**.

Options

schema.sh [*options*] *commands partition*

Options	Description
/path/to/file	The absolute path to the schema file which is being compiled or formatted. For schema files to be loaded by Penrose Server, they should be located in the **`/opt/vd-server-2.0/schema`** directory and have the extension **`.schema`**.

Table A.13. schema.sh Options

A.9. source.sh

The **`source.sh`** script manages data sources used by the partition. The script can be used to import or export, start and stop, modify, and remove sources associated with any partition.

Options

source.sh [*options*] *commands* source *partition*

Options	Description
-?, - -help	Prints the help for the script.
-d	Runs the script in debug mode.
-D *rootDN*	Gives the root user for Penrose Server; the default username is **`uid=admin`**, **`ou=system`**.
-h *hostName*	Gives the hostname for the machine on which the service is running.
-p *port*	Gives the port number for the service to which Penrose Server is connecting.
-P *protocol*	Gives the protocol for Penrose Server to communicate with the MBeans object; the only supported JMX protocol is **`rmi`**.
-w *password*	Gives the password for the Penrose Server root user; the default is **`secret`**.
-v	Runs the script in verbose mode.

Table A.14. source.sh Options

Commands	Description
show	Lists the sources configured for the given partition.
start	Starts the source for that partition.
stop	Stops the source for that partition.

Commands	Description
restart	Restarts the source for that partition.
invoke *application_command*	Manually runs the specified command for the source. For example, **`invoke`** can be used to run LDAP commands such as **`ldapmodify`** and **`ldapsearch`** against the source LDAP directory.
import source *path*	Imports the source configuration from the specified directory into the given partition.
export source *path*	Exports the source configuration for the named partition to the specified directory.
rename source *name1 name2*	Renames the source from *name1* to *name2*.
remove source *name*	Deletes the specified source.
partition	Specifies the name of the partition being modified.

Table A.15. source.sh Commands

Glossary

A

Access control
Restrictions and rules which control the areas of the virtual directory which people can view and the operations which they can perform. In Penrose Virtual Directory, the access control set on the virtual directory is enforced apart from any access control set on the data source.

Adapters
An abstraction layer which interacts between Penrose Server and the data source; works on the server frontend. There are three adapters in Penrose Virtual Directory for LDAP, JDBC, and NIS sources.

Attribute
A single descriptive element for an entry, defined in the schema for a server.

Authentication
The process of binding, or logging in, to a server and providing a set of credentials which must be verified to allow the login.

B

Base DN
The distinguished name against which an LDAP operation is carried out. A search, for example, begins searching for entries at the base DN and then goes down the directory tree through all of its children subtrees and entries.

Basic mapping
A simple, direct association where one attribute is specifically matched to corresponding attributes in the virtual directory, such as mapping the **`firstname`** row in a database table to the *`givenName`* LDAP attribute in the virtual directory.

C

Cache
Temporary storage used by Penrose Virtual Directory to contain virtual directory entries. Penrose Virtual Directory supports two kinds of cache, persistent and in-memory.

Connections
The server or host information which is used to connect to data sources.

Constant
Specific text used to define a static mapping rule, field, or Penrose Virtual Directory other entry element. In the configuration files, this is

indicated by **<constant>** tags; in Penrose Studio, this is sometimes labeled *text*.

D

Data source

Any server or application which contains information that is used by Penrose Virtual Directory to create a virtual entry.

Directory hierarchy

See *Directory tree (DIT)*.

Directory migration

The process of copying information from one data source configured in Penrose Virtual Directory to another data source. This is usually configured using identity federation.

See Also *Lazy migration*.

Directory tree (DIT)

The organization of the entries and subentries contained in a directory; the visual layout of the directory is similar to a filesystem hierarchy. This is also called a directory information tree (DIT) or directory hierarchy.

Distinguished name (DN)

The full representation of an entry's name in an LDAP directory. This includes a naming attribute (the leftmost element in the name) and the position in the directory tree by including its parent entries.

Dynamic subtree

A type of virtual subtree in Penrose Virtual Directory where the entries within the subtree are generated according to a filter rather than being explicitly defined.

E

Entries

The identity in the virtual directory or in one of its sources. This can also refer to a configuration record in the Penrose Virtual Directory configuration.

Entry attributes

An attribute for a virtual directory entry. This is defined with the corresponding LDAP attribute name and an expression to supply the value, either dynamically through a Java expression or variable pointing to a source value or explicitly through a constant. Optionally, there is also a key to indicate whether the attribute is used to name an entry.

Expressions

BeanShell expressions used to map attributes in the virtual directory to attributes in the sources. BeanShell expressions are described in more detail at the BeanShell site, *http://1www.beanshell.org*.

F

Fields

An attribute within a source entry which can be referenced for mapping into the virtual directory.

Flattened namespace

A unified naming organization for entries in the virtual directory. When entries are stored in multiple sources, particularly different kinds of sources, the way that entries are named and the organization of the directory structure can be very different. Virtual directories "flatten" or unify those different namespaces into a single namespace with a consistent rule for naming entries.

G

Global repository

The centralized directory used to combine entries for identity federation; this is usually an LDAP server, such as Red Hat Directory Server or Active Directory.

I

Identities

The person, server, or other entity which is described by an entry. Frequently, there can be more than one entry referring to the same person.

Identity federation

The configuration concept which combines entries from different applications or sources which all describe the same person into a single entry in a centralized global repository. This is most commonly used for LDAP and NIS environments or to facilitate a migration between NIS and LDAP, LDAP to Red Hat IPA, or different kinds of LDAP servers.

Identity linking

The process of matching entries in a local repository or source to the corresponding entry in the centralized global repository for identity federation.

Interpreter

The parts of the Penrose Virtual Directory code which translates the mappings configured in the virtual directory to generate the virtual entries.

J

Join mapping

A method of mapping which takes attributes from multiple source

	entries, which all represent a single identity, and use them to build a virtual entry. For join mapping, the different source entries must have at least one value in common in order to identify and combine properly the entries.

L

Lazy migration	The process of copying information from one data source configured in Penrose Virtual Directory to another data source. This is usually configured using identity federation.
Local repository	A source used to supply information for identity federation. This is the original source for information; the data are migrated over to the global repository.

M

Mapping	Any association which is defined to supply an attribute and value in the virtual directory, either by pulling the attribute value from a source entry or by explicitly defining it.
Mapping rules	A single definition for an attribute mapping. A mapping rule is set for the virtual directory, not the source.
MBeans	A managed bean in a Java environment. All of the partition-related operations for the virtual directory, including sources and connections, are MBeans.
Metadirectory	A kind of centralized directory which interacts with multiple kinds of directory and database services. A meta directory holds an independent copy of information; it collects information from its sources and can synchronize that information back to other sources, to help unify the entire environment. This is a similar function to a virtual directory, except virtual directories do not store information.
Modules	Specialized Java classes which extend or change the behavior of Penrose Virtual Directory components; works on the server backend.

N

Namespace	The organization, based on the name of the highest entry in the directory, which defines the relationship and names of entries.

Naming attribute

The attribute which supplies the relative distinguished name, the first element in an entry's name. For example, *uid* is the naming attribute in **uid=jsmith,ou=people,dc=example,dc=com**.

Nested mapping

A method of associating entry values where one attribute value is dynamically generated and those results are used to generate the next part of the mapped value.

O

Object class

An entry element which defines the type of entry and has defined rules on what attributes are required and allowed to describe the entry; for a directory, this is an LDAP schema element.

P

Partitions

A designated object within Penrose Virtual Directory which has a common set of Java classes and libraries to run the virtual directory and specific configuration defining the sources, connections, modules, mappings, and other virtual directory configurations.

Pass-through authentication

A form of binding to a directory service where the bind information, such as the username and password, are sent to one service (Penrose Virtual Directory) and are then sent to a source, unchanged, to allow the user to bind to the source. Penrose Virtual Directory supports pass-through authentication to services such as Active Directory and Red Hat Directory Server.

Primary key

An argument set for an attribute defined in the mapping which indicates whether the attribute is a naming attribute, meaning that is supplies the element on the far left of the entry's distinguished name. For example, if the *uid* parameter is the primary key, then that attribute value is used in the entry name, such as **uid=jsmith,ou=people,dc=example,dc=com**.

Proxy

A service, such as Penrose Virtual Directory, which is contacted in place of another service. Penrose Virtual Directory can function as an LDAP proxy for Red Hat Directory Server and Active Directory, for example, and route requests and operations to the appropriate LDAP server.

See Also *Pass-through authentication*.

R

Rules

A defined mapping association for a single attribute in the virtual directory.

Relative distinguished name

The first element of the distinguished name; for example, **`uid=jsmith`** in **`uid=jsmith,ou=people,dc=example,dc=com`**.

Repository

A source to store information in identity federation.

See Also *Local repository*, *Global repository*.

Root DSE

The entry in the directory tree configuration which defines the configuration of the directory service, such as its LDAP protocol, supported LDAP mechanisms and controls, and its server URLs. The root DSE for the virtual directory is configured in the partition and defines what the virtual directory supports.

S

Schema

A list of different elements which are used to describe entries in directory services. Attribute elements define what descriptive elements are available; object class elements identify the kind of entry being describe and the required or allowed attributes to describe it. The list of attributes and object classes, together, is the schema.

Scripts

In Penrose Virtual Directory, any code which is configured to run before or after mapping rule to manipulate the data as used in the virtual directory entry.

Security provider

Additional libraries which extend SSL/TLS support for Penrose Virtual Directory and additional algorithms, ciphers, and encryption-related information.

Services

In Penrose Virtual Directory, interfaces which are used by Penrose Virtual Directory to connect to LDAP or database sources. An additional JMX service connects to Penrose Studio and other Java-based clients.

SIE

The server instance entry, usually the name assigned to a directory server instance.

Single sign-on

A concept where users need only log into a single service and can have authenticated access to any service on the network. Penrose Virtual Directory, by creating a centralized location for nearly all dir-

ectory and database services, is part of a single sign-on solution because users can authenticate to Penrose Virtual Directory and be authenticated, through the Penrose Virtual Directory configuration, to all of the sources in the virtual directory.

Snapshot

A copy of the entire LDAP subtree or directory, with all entries, as it exists at that moment. Because this mirrors an existing directory on Penrose Virtual Directory, this is similar to an LDAP proxy.

See Also *Proxy*.

Sources

Any server or application which contains information that is used by Penrose Virtual Directory to create a virtual entry. Source entries are configured within a partition.

For synchronization or migration, the source is the repository which contains the original copy of the data being synchronized.

Static subtree

A type of virtual subtree in Penrose Virtual Directory where the entries within the subtree are explicitly defined. This is common for configuration subtrees which are directory or hierarchy container entries.

Studio

A Java-based user interface which manages, views, creates, and edits virtual directory entries for Penrose Virtual Directory.

Subtree

A branch point within the directory hierarchy. For example, organizational units (*ou* entries) are branch points because they indicate a subset of the overall directory entries.

Synchronization

The operation of copying data between two sources or repositories so that the same information is mirrored in two locations. Penrose Virtual Directory supports synchronization between NIS, LDAP, Active Directory, Red Hat IPA, and other sources.

T

Target

In migration or synchronization, the target server is the one *to* which information is copied. For example, in a migration from NIS to LDAP, the NIS server is the source repository and the LDAP server is the target.

U

UID/GID conflict

In identity federation, the potential conflict in assigned user ID or group ID numbers. Because there can be multiple sources combined

	in identity federation, the same ID numbers may be duplicated. These conflicts can be resolved by assigning new ID numbers as part of the identity federation process and any file ownerships reassigned.

V

Variable	In Penrose Virtual Directory, a source attribute which is used to supply an attribute value in a virtual directory mapping entry. These have the form *source_name.attribute_name*.
Virtual directory	A lightweight, centralized directory which interacts with multiple kinds of directory and database services. A virtual directory uses configuration files to identify data sources and rules for how to recognize and apply attributes to entries, as well as definitions for the centralized directory configuration. The actual data remain in the data sources; views of the directory and the information contained in the entries is located and displayed on the fly, rather than physically copying the information into a different directory, as in a meta directory.
Virtual Directory Server	The lightweight directory engine which creates the virtual directory, interacts with LDAP and database sources, and contains all configuration information.

Index

Symbols

A

B

C

D

E

M

N

O

P

R

S

U

V