

Space Details

Key:	DIRxSRVx10
Name:	Apache Directory Server v1.0
Description:	Apache Directory Server v1.0
Creator (Creation Date):	hcunico (Sep 13, 2006)
Last Modifier (Mod. Date):	ersiner (Jan 27, 2007)

Available Pages

- ApacheDS v1.0 Basic User's Guide
 - About the sample configurations and sample directory data
 - Apache HTTP Server
 - Apache Tomcat
 - ApacheDS tools
 - Authentication options
 - Basic authorization
 - Basic configuration tasks
 - Command line tools
 - Connecting to ApacheDS with graphical tools (3rd party)
 - Connecting to ApacheDS with Java components
 - Custom application development
 - How to enable SSL
 - Installing and starting the server
 - LDAP Operations Overview
 - LDAP Operations, Modification
 - LDAP Operations, Searching
 - LDAP Studio
 - Mozilla Thunderbird
 - Other programming languages
 - Some Background. Directories, directory services and LDAP
 - What Apache Directory Server is

ApacheDS v1.0 Basic User's Guide

This page last changed on Feb 01, 2007 by [szoerner](#).

About this guide

Getting started. Learn how to download and install ApacheDS 1.0 on different platforms, connect to it with various clients (graphical tools among others), manipulate the data within your directory and integrate ApacheDS with other software. The same sample data, provided as a download, is used through the whole guide.

Audience

This guide is primarily for people new to ApacheDS. If you plan to use the server as is, maybe even for your first LDAP experience, you'll find all information necessary to do so. Only basic configuration tasks are covered. Learn more about the options you have (and many other things) in the "Advanced User's Guide".

Table of contents

1. How to begin
 - a. [What Apache Directory Server is](#)
 - b. [Some Background. Directories, directory services and LDAP](#)
 - c. [About the sample configurations and sample directory data](#)
 - d. [Installing and starting the server](#)
 - e. [Basic configuration tasks](#)
2. Handling of data within your directory
 - a. Connecting to a server with Apache Directory client tools
 - i. [LDAP Studio](#)
 - ii. [ApacheDS tools](#)
 - b. Connecting with other client tools
 - i. [Graphical tools](#)
 - ii. [Command line tools](#)
 - c. Custom application development
 - i. [Connecting to ApacheDS with Java components](#)
 - ii. [Other programming languages](#)
 - d. LDAP Operations
 - i. [Overview](#)
 - ii. [Searching](#)
 - iii. [Modification](#)
3. Basic Security
 - a. [Authentication options](#)
 - b. [Basic authorization](#)
 - c. [How to enable SSL](#)
4. Integrate ApacheDS in other programs
 - a. [Mozilla Thunderbird \(E-Mail Client\)](#)
 - b. [Apache Tomcat](#)
 - c. [Apache HTTP Server](#)

About the sample configurations and sample directory data

This page last changed on Jan 28, 2007 by [szoerner](#).

<< Previous: Some Background	ApacheDS v1.0 Basic User's Guide (TOC)	Next: Installing and starting the server >>
--	--	---

This section describes basic parameters used throughout the examples in this guide. It also introduces the sample directory "Sailors of the seven seas", and other requisites you need.

- [Basic server parameters](#)
- [LDAP Clients](#)
- [The sample data](#)

Basic server parameters

In the following sections we assume that you will install, configure and run Apache Directory Server on a host with the following host name using the parameters given in the following table:

Parameter name	Parameter value
Hostname	zanzibar
Port	10389
Suffix ("Base DN")	o=sevenSeas
Admin user DN	uid=admin,ou=system
Admin user password	secret

In order to increase recognition, all examples of the Basic User's Guide use these values. Adjust them to your needs (especially the password).

LDAP Clients

LDAP is a client/server protocol. Hence you need an LDAP client to connect remotely to the Apache Directory Server (or at least the directory part of it, to be precise). There are different options here. Because the protocol is standardized, you may use every LDAP compliant client. This is comparable to HTTP, where you can use each web browser to communicate with virtually each web server, and totally different to relational databases. The latter have a (more or less) standardized query language (SQL), but vendors tend to use individual network access protocols. In practice, the LDAP situation is even better than HTTP, because there were no LDAP browser wars ...

Many software components may act as an LDAP client. Normally they use LDAP libraries to connect. In the following sections you meet LDAP clients with GUI and LDAP command line tools. Some Java programming examples which takes advantage of JNDI are provided as well.

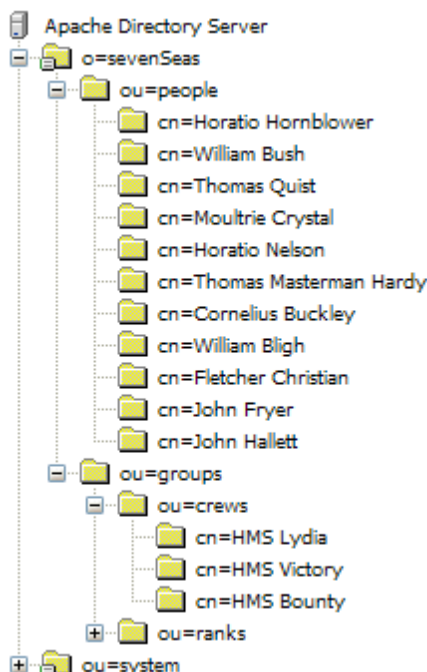
In other sections you will learn how E-Mail clients like Mozilla Thunderbird and application servers like Apache Tomcat connect to Apache Directory server, either to use the data stored in the directory (e.g. mail addresses) or to perform authentication and authorization.

Recapitulating the Basic User's Guide describes connecting to the server with tools provided by ApacheDS as well as third party products. In all cases the examples will use the connection data depicted above (*ldap://zanzibar:10389/o=sevenSeas*)

The sample data (Sailors of the seven seas)

The file [apache_ds_tutorial.ldif](#) contains some sample data, which is used in the following sections. It is a text file in the so called **LDIF** format. LDIF stands for LDAP Data Interchange Format. It is widely adopted in the LDAP world and standardized in [RFC 2849](#). Therefore you are able to import our sample data into other directory solutions as well, not only into Apache Directory Server.

The sample directory tree contains entries for persons and groups. These are structured in sub trees (*ou=people* and *ou=groups*), see image below. The person entries describe sailors (historic and fictional), the group entries bundle them. An example for a group is the ship crew of HMS Bounty.



This snippet of the file represents a single entry, just to give you an impression of how LDIF files look like.

```
...
# Entry for Fletcher Christian
#
dn: cn=Fletcher Christian,ou=people,o=sevenSeas
cn: Fletcher Christian
objectClass: top
objectClass: person
objectClass: organizationalPerson
objectClass: inetOrgPerson
sn: Christian
givenName: Fletcher
description: Lieutenant Fletcher Christian
manager: cn=William Bligh,ou=people,o=sevenSeas
...
```

There are different ways to import the data. Generally perform the following steps:

- Download and install the server, described in [Installing and starting the server](#)
- Configure a partition for the sample data, described in [Basic configuration tasks](#)
- Import the data, for instance with the [ApacheDS tools](#)

Resources

- [RFC 2849](#) The LDAP Data Interchange Format (LDIF) – Technical Specification

Apache HTTP Server

This page last changed on Jan 01, 2007 by [ck](#).

<< Previous:Apache Tomcat	ApacheDS v1.0 Basic User's Guide (TOC)	
---	--	--

- [Description](#)
- [Apache HTTP Server 2.0.41 and above](#)
- [Apache HTTP Server 2.1 and above](#)
- [Resources](#)

Description

The Apache HTTP Server contains modules, that allow the authentication of users against an LDAP directory server. These modules vary between the different versions of the HTTP Server. For Apache HTTP 2.0.41 and above, an experimental module called `mod_auth_ldap` exists. For Apache HTTP 2.1 and above there is a module `mod_authnz_ldap`, which is no longer experimental, but a regular modul. In the following examples we use the sample data and configuration from the previous chapters, if you haven't already imported them you can find the data and a short description of the configuration [here](#).

Apache HTTP Server 2.0.41 and above

`mod_auth_ldap` is only a experimental module, therefore the documentation here is a bit sparse.

All Required Modules:

`mod_auth_ldap.so`
`mod_ldap.so`
`mod_auth.so`

Compiling:

```
./configure --with-ldap --with-ldap-lib=<PATH_TO_YOUR_LIBS>  
--with-ldap-include=<PATH_TO_YOUR_INCLUDE> \  
--enable-ldap --enable-auth-ldap
```

Simple Example:

Simple example with anonymous bind:

```
<Location "/secure">  
  AuthType Basic  
  AuthName "Seven Seas Area"
```

```
AuthLDAPEnabled on
AuthLDAPUrl "ldap://zanzibar:10389/ou=people,o=sevenSeas?uid"
Require valid-user
</Location>
```

Apache HTTP Server 2.1 and above

All Required Modules:

```
mod_auth_basic.so
mod_authz_user.so
mod_authnz_ldap.so
mod_ldap.so
```

Compiling:

If you build the server on your own, you have to call configure with the following flags to make sure that all required modules are included: (The HTTP Server need some external LDAP libs, for example the OpenLDAP SDK, for compiling the modules. Look at the HTTP Server documentation for details.)

```
./configure --with-ldap --with-ldap-lib=<PATH_TO_YOUR_LIBS>
--with-ldap-include=<PATH_TO_YOUR_INCLUDE> \
--enable-ldap --enable-authnz-ldap
```

Simple Example:

This is a simple configuration example with an anonymous bind to the LDAP Server. All users with a valid LDAP entry can get access to the protected resources. User credentials are the uid attribute and the userPassword attribute (this are the default values, so they doesn't need to be specified here) and every user with valid credentials can access the secured area.

```
<Location "/secure">
AuthType Basic
AuthName "Seven Seas Area"
AuthBasicProvider ldap
AuthLDAPUrl ldap://zanzibar:10389/ou=people,o=sevenSeas
AuthzLDAPAuthoritative OFF
Require valid-user
</Location>
```

AuthType: This directive selects the protocol for the transport of the authentication credentials. This can be digest oder basic.

AuthName: This directive provides the name of the authorization realm for a directory.

AuthBasicProvider: This directive specifies the use of the ldap provider for authentication.

AuthLDAPUrl: This directive specifies the URL of the LDAP server and the LDAP search parameters. A detailed description of this directive follows in the next section.

AuthzLDAPAuthoritative: Prevent other authentication modules from authenticating the user if LDAP authentication fails

Require valid-user: The require directive specifies who is grant authorization to access the protected directory. In this case this authorization is grant to all valid users, but there are more granular rules as you will see in the following examples.

AuthLDAPURL directive:

The general format for this url is:


```
protocol://host:port/basedn?attribute?scope?filter
```

protocol: ldap or ldaps (when using SSL)

host: Hostname of the LDAP Server

port: Port of the LDAP Server(optional; default ist 389)

basedn: a branch of the LDAP tree where the search for entries should begin

attributes: The LDAP attribute which contains the user name for authentication (optional; default is uid)

scope: (optional; default ist sub) The scope specifies how deep you want to search the branch specified by basedn for entries. The two possible values for this parameter are "one " and "sub". "one" indicates that only the direct children of the search base should be considered. "sub" indicates that you want to be searched the whole subtree from the search base down.

filter: (optional; default is objectClass=*)

Second Example:

This is a configuration example with two redundant LDAP servers (zanzibar & cyprus). It uses a filter expression so that only users with a cn attribute that starts with "John" can log in. Furthermore the require directive limits access to members of the crew of HMS Bounty.

```
<Location "/secure">
  AuthType Basic
  AuthName "Seven Seas Area"
  AuthBasicProvider ldap
  AuthLDAPUrl "ldap://zanzibar:10389 cyprus:10389/ou=people,o=sevenSeas?cn?sub?(cn=John*)"
  AuthLDAPBindDN uid=admin,ou=system
  AuthLDAPBindPassword secret
  AuthzLDAPAuthoritative OFF
  Require ldap-group cn=HMS Bounty, ou=crews, ou=groups, o=sevenSeas
</Location>
```

Require directive:

The Require directive can be used to configure granular rules for user authorization.

- valid-user: grants access for any user with valid LDAP credentials
- ldap-user: specifies a list of usernames which are allowed to access
- ldap-group: access is granted if the user is member of a specific group
- ldap-dn: grants access based on fully distinguished names
- ldap-attribute: only authenticated user with specific attributes are allowed to access
- ldap-filter: allows to grant access based on a complex LDAP search filter

More Details and Examples can be found [here](#)

Caching:

A sophisticated caching strategy helps the Apache HTTP Server to improve performance and to minimize the requests to the LDAP server. Therefore three caches are created for each LDAP-server: One cache for credentials and two for operations.

The credential cache caches successful bind operations to the server, it stores the user name, dn, password and a timestamp. At a new connection to the server with a username already stored in the cache, the password is validated against the cache entry. If this is successful and the entry has not expired another bind to the LDAP server is skipped. The behavior of this cache can be configured using the directives **LDAPCacheEntries** (number of entries in the cache) and the **LDAPCacheTTL** (Time To Live - period of time an entry is cached).

The operation caches are used to cache compare operations, which occurs when examining group membership and DN.

The behavior of this cache can be configured using the directives **LDAPOpCacheEntries** (number of

entries in the cache) and **LDAPOpCacheTTL** (period of time an entry is cached).
The LDAP Cache Status Monitor can be used to monitor the loading of the ldap caches with a web browser. The following is a simple configuration example which demonstrates this function:

```
<Location /cache-stats>
  SetHandler ldap-status
  Order deny,allow
  Allow from all
</Location>
```

Cache Name	Entries	Avg. Chain Len.	Hits	Ins/Rem	Purges	Avg Purge Time
LDAP URL Cache	1 (0% full)	1.0	60/61 98%	1/0 (none)		0ms
ldap://zanzibar:10389 cyprus:10389/ou=people,o=sevenSeas?cn?sub?(cn=John*) (Searches)	1 (0% full)	1.0	19/22 86%	1/0 (none)		0ms
ldap://zanzibar:10389 cyprus:10389/ou=people,o=sevenSeas?cn?sub?(cn=John*) (Compares)	2 (0% full)	1.0	38/42 90%	2/0 (none)		0ms
ldap://zanzibar:10389 cyprus:10389/ou=people,o=sevenSeas?cn?sub?(cn=John*) (DNCompares)	0 (0% full)	0.0	0/0 100%	0/0 (none)		0ms

Done

Resources

Apache HTTP Server 2.0:

[Authentication, Authorization and Access Control](#)
[mod_auth_ldap](#)

Apache HTTP Server 2.2:

[Authentication, Authorization and Access Control](#)
[mod_authnz_ldap](#)
[mod_ldap](#)
[Using LDAP Authentication in Apache 2.2, talk by Brad Nicholes at the ApacheCon US 2006 \(PPT\)](#)

[<< Previous:Apache Tomcat](#)

[ApacheDS v1.0 Basic User's Guide \(TOC\)](#)

Apache Tomcat

This page last changed on Jan 27, 2007 by [ck](#).

<< Previous:Mozilla Thunderbird	ApacheDS v1.0 Basic User's Guide (TOC)	Next:Apache HTTP Server>>
---	--	---

- [Description](#)
- [Example](#)
- [Resources](#)

Description

A common task when developing a web application is user authentication and authorization - parts of the application should only be seen by the users which you want to see them. Three things are required for realizing this, a mechanism for authentication which checks the credentials provided by the user in the login form. A mechanism for authorization which decides about user privileges and a data store where user information & credentials are stored.

A perfect choice for the data store is ApacheDS. LDAP is a widely adopted standard so you can reuse your user data also for other systems.

For authentication and authorization J2EE provides a few standard mechanisms. The most popular mechanism for authentication, and the one used in the example, is **form based authentication**, where you can create your own **customized JSP login form**. There exist three further authentication mechanisms which are not discussed here, so look at the resources if you are interested in.

So let's talk about authorization. In a J2EE environment it's possible to assign one or more roles to each user. A role is a logical grouping of users, for example you could have different roles for employees, customers and guests. Basing on these roles you can grant different rights on what you allow your users to see and to do.

The following example shows the building of a simple web application where you can Login using username and password and afterwards receive a page confirming your successful login and presenting details about it.

Example

Prerequisites:

- **Tomcat 5.5** (this example was tested with 5.5.20 but should work with every 5.5.x version)
- **ApacheDS**

Configuration used in this example:

Tomcat	
Host	madagaskar
Port	8080
ApacheDS	
Host	zanzibar
Port	10389
Suffix	o=sevenSeas
Admin user dn	uid=admin,ou=system
Admin pass	secret

Data

First import the example data in an own partition of your directory server, you can find them [here](#), if you haven't already done this. How to create a partition is described [here](#), how to import data into the directory is described [here](#). This should result in the following directory structure:

[Download](#) the archive with the JSP files for this example, (which are taken from an tomcat example) and extract it to the webapps folder of your tomcat installation.

Deployment Descriptor

The archive doesn't contain a deployment descriptor (DD) for Tomcat, hence we have to create one for our application. The DD defines which part of our application we wish to protect, whom we want grant access to it, which authentication method should be used and other things, but it does not define the details for connecting to ApacheDS. In our example, we want to grant only the crew of "HMS Bounty" access to our application. The DD is stored at **/security/WEB-INF/web.xml**.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
    http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd"
  version="2.4">

  <security-constraint>
    <web-resource-collection>
      <web-resource-name>Protected Area</web-resource-name>
      <!-- Define the context-relative URL(s) to be protected -->
      <url-pattern>/protected/*</url-pattern>
    </web-resource-collection>
    <auth-constraint>
      <!-- Anyone with one of the listed roles may access this area -->
      <role-name>HMS Bounty</role-name>
    </auth-constraint>
  </security-constraint>

  <!-- Default login configuration uses form-based authentication -->
  <login-config>
    <auth-method>FORM</auth-method>
    <form-login-config>
      <form-login-page>/protected/login.jsp</form-login-page>
      <form-error-page>/protected/error.jsp</form-error-page>
    </form-login-config>
  </login-config>

  <!-- Security roles referenced by this web application -->
  <security-role>
    <role-name>HMS Bounty</role-name>
  </security-role>
```

```
</web-app>
```

Let's have a closer look at it.

```
<url-pattern>/protected/*</url-pattern>
```

This describes which url should be protected, which means, that the url's defined here can only be requested by a user who has successfully been authenticated (and has an appropriate role assigned). You can provide multiple patterns here. It's also possible to restrict access only for specific HTTP Methods. Which roles are allowed to request these restricted URL's is defined in the following lines. You can specify either some concrete role names here, in our example "HMS Bounty", or just write "*" to allow access for all registered users.

```
<role-name>HMS Bounty</role-name>
```

The login-config element provides details about the authentication method you want to use. This example uses form-based authentication, so we need to define the authentication method "FORM" and the location for our login and error pages.

```
<login-config>
  <auth-method>FORM</auth-method>
  ...
</login-config>
```

The security-role element lists all roles available.

JNDI Realm

The details for the connection to ApacheDS are defined using a Tomcat JNDI Realm. Save the following in a file at **/security/protected/META-INF/context.xml**

```
<?xml version="1.0" encoding="UTF-8"?>
<Context path="/jsp-examples/*" >
  <Realm className="org.apache.catalina.realm.JNDIRealm"
    debug="99"
    connectionName="uid=admin,ou=system"
    connectionPassword="secret"
    connectionURL="ldap://zanzibar:10389"
    roleBase="ou=crews,ou=groups,o=sevenSeas"
    roleName="cn"
    roleSearch="(uniqueMember={0})"
    roleSubtree="false"
    userPassword="userPassword"
    userPattern="cn={0},ou=people,o=sevenSeas"
  />
</Context>
```

Let's have a closer look at it. The following specifies the details for the user connecting to ApacheDS. Make sure that this user has the appropriate rights for reading all required entries and attributes. We have used the admin credentials here, but in general it's not a good idea to provide your admin credentials everywhere in your config files. We have done this here because it's only a very simple example.

```
connectionName="uid=admin,ou=system"
connectionPassword="secret"
connectionURL="ldap://zanzibar:10389"
```

But the question is: How does Tomcat figure out where the entries for the user's you want to allow to login are stored?

```
userPassword="userPassword"
userPattern="cn={0},ou=people,o=sevenSeas"
```

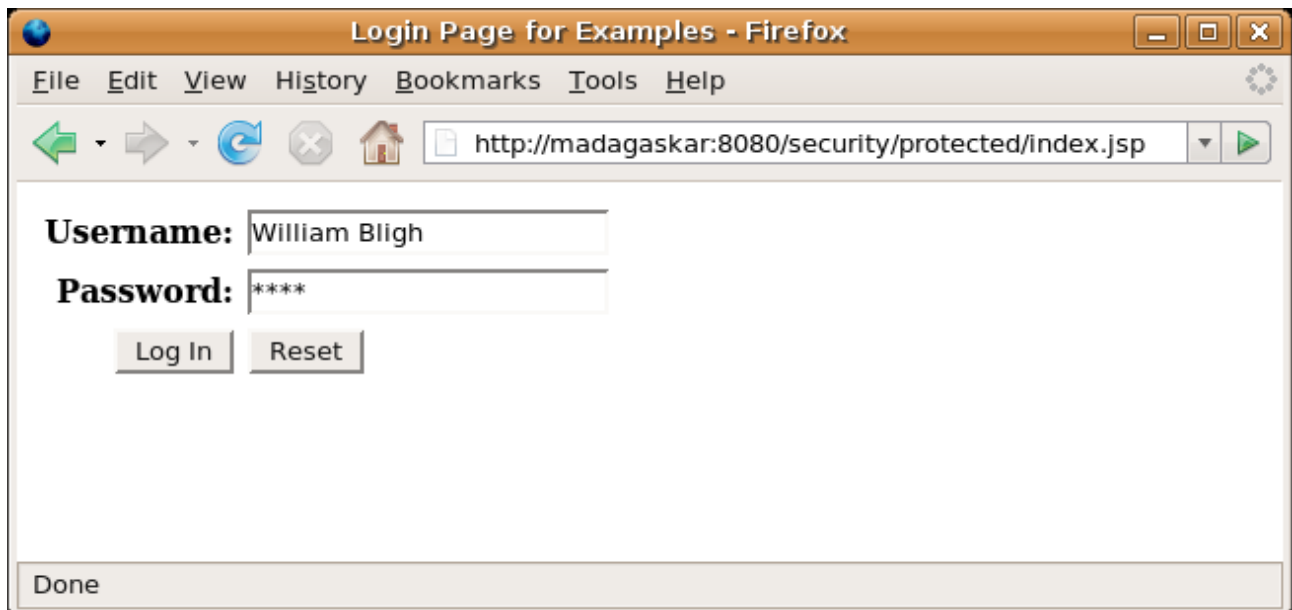
The attribute "userPassword" contains the name of the attribute, which contains the password of the user. "userPattern" specifies where to search for user entries. The expression "{0}" is a placeholder for the value of the username entered by the user into the login form.

Finally you have to tell Tomcat how he can determine if someone is member of the crew from HMS Bounty. All members of the crew are stored in a groupOfUniqueNames entry below "ou=crews,ou=groups,o=sevenSeas".

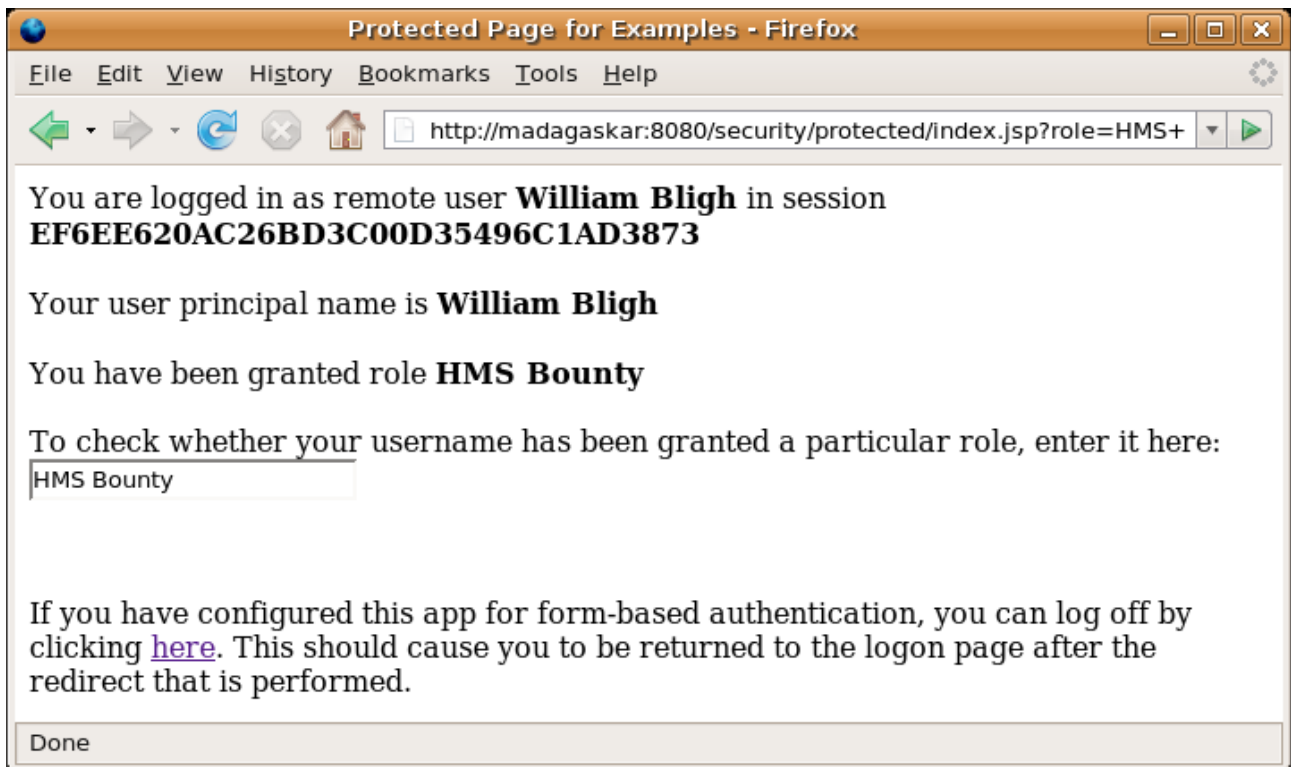
```
roleBase="ou=crews,ou=groups,o=sevenSeas"
roleName="cn"
roleSearch="(uniqueMember={0})"
roleSubtree="false"
```

Starting the application

Now (re)start Tomcat and go to <http://madagaskar:8080/security/protected/index.jsp>. You should see the following Login Form. Enter name and password (which is simply "pass" in our example) of a member of the crew of HMS Bounty.



After successful login you can test if your user belongs to a specific role.



Login Form

If you have a closer look at the source of the login form (/security/protected/login.jsp) you'll see the following

```
<form method="POST" action="j_security_check" >
...
<input type="text" name="j_username">
...
<input type="password" name="j_password">
...
</form>
```

It's always crucial that the Login Form has the action "j_security_check" and that the fields "j_username" and "j_password" are present, otherwise Form-Based authentication won't work properly. This and further details are explained in the J2EE Tutorial and servlet specification.

Resources

[Tomcat's Homepage](#)

[Realms in Tomcat](#)

[The J2EE 1.4 Tutorial](#) (Chapter 32 contains more information about J2EE security)

[Servlet Specification 2.4](#) (the one which belongs to J2EE 1.4)

[<< Previous:Mozilla Thunderbird](#)

[ApacheDS v1.0 Basic User's Guide \(TOC\)](#)

[Next:Apache HTTP Server>>](#)

ApacheDS tools

This page last changed on Jan 13, 2007 by [ck](#).

<< Previous:LDAP Studio	ApacheDS v1.0 Basic User's Guide (TOC)	Next:Graphical tools>>
---	--	--

Apache DS tools are simple operations that help user to interact with the server. The following operations are available:

- **Import:** A command to import data into a server
- **Dump:** Simple tool used to dump the contents of a jdbm based partition
- **Diagnostic:** A command to send an extened request which launches a diagnostic UI on the server's console
- **Disconnect Notification:** Responds to unsolicited notifications by launching an external process
- **Graceful Shutdown:** A command used to send a graceful disconnect to established clients while allowing them time to complete operations already in progress
- **Capacity Test:** A command which will generate bogus user entries and add them under a base DN. It will output a table of values mapping the capacity of the partition to the time it took to add an entry to it.
- **Index:** A command which adds attribute indices to an existing partition

Each one of these commands does have a set of options. They will be described below.

Launching a command

Using those command is pretty simple. Jump to the **bin** directory of your installation and simply type the following command line :

```
java -jar apacheds-tools.jar <command> [options]
```

Import

The **import** comand allows to import entries into a Ldap server.

Available options are :

Option	description	default value
-a	Authentication type	simple
-e	Continue to process the file even if errors are encountered	false
-f	The file to be imported	

-h	The server host	localhost
-p	The server port	10389
-u	The user	uid=admin, ou=system
-w	The administrator password	secret

example :

```
java -jar apacheds-tools.jar import -e -f newUsers.ldif
```

Imports new users in the default server.

Dump

Allows you to dump the contents of a jdbm based partition.

Available options are:

Option	description	default value	required
-e	the attributes to exclude		false
-f	file to output the dump to		false
-p	the partitions to dump		true
-o	include operational attributes	false	false
-i	path to apacheds installation directory		true

example :

```
java -jar apacheds-tools.jar dump -i C:\Programme\apacheds-1.0.0 -p system
```

Dumps the content of the system partition.

Diagnostic

Option	description	default value	required
-h	server host	localhost	false
-p	server port	10389 or server.xml specified port	false
-w	the apacheds administrator's password	secret	false

-i	path to apacheds installation directory		false
----	---	--	-------

Disconnect Notification

Option	description	default value	required
-h	server host	localhost	false
-p	server port	10389 or server.xml specified port	false
-w	the apacheds administrator's password	secret	false
-u	an apacheds user's dn	uid=admin,ou=system	false
-i	path to apacheds installation directory		false

Graceful Shutdown

Available options are:

Option	description	default value	required
-h	server host	localhost	false
-p	server port	10389 or server.xml specified port	false
-e	delay (seconds) before shutdown	0	false
-w	the apacheds administrator's password	secret	false
-t	server offline time (minutes)	0 (indefinite)	false
-i	path to apacheds installation directory		false

example :

```
java -jar apacheds-tools.jar graceful
```

Capacity Test

Option	description	default value	required
-f	file to output the stats to	console	false

-i	path to apacheds installation directory		true
-h	server host	localhost	false
-p	server port	10389 or server.xml specified port	false
-w	the apacheds administrator's password	secret	false
-s	start on id: number to start on (user.start)	0	false
-e	end on id: number to end on (user.end)	2 31 -1 (Integer.MAX_VALUE)	false

Adding an index

Option	description	default value	required
-p	the partitions to add the attribute indices to		true
-a	the attribute to index		true
-i	path to apacheds installation directory		true

example :

```
java -jar apacheds-tools index -i /usr/local/apacheds-1.0 -p examplePartition -a postalCode
```

Authentication options

This page last changed on Jan 14, 2007 by [szoerner](#).

	ApacheDS v1.0 Basic User's Guide (TOC)	Next:Basic authorization>>
--	--	--

This section describes the authentication options of ApacheDS 1.0. Currently, only anonymous and simple binds are supported; configuring and using them is described below with the help of examples.

- [What is authentication?](#)
- [Simple binds](#)
- [Passwords stored one-way encrypted](#)
- [Anonymous binds](#)
- [How to authenticate a user by uid and password?](#)
- [Resources](#)

What is authentication?

Authentication is the process of determining whether someone (or something) in fact is what he/she/it asserts to be.

Within ApacheDS you will likely want to authenticate clients in order to check whether they are allowed to read, add or manipulate certain data stored within the directory. The latter, i.e. whether an authenticated client is permitted to do something, is deduced during **authorization**.

Quite often, the process of authentication is delegated to a directory service by other software components. Because in doing so, authentication data (e.g. username, password) and authorization data (e.g. group relationships) are stored and managed centrally in the directory, and all connected software solutions benefit from it. The integration sections of this guide provide examples for Apache Tomcat, Apache HTTP servers, and others.

ApacheDS 1.0 supports only simple authentication and anonymous binds while storing passwords within userPassword attributes in user entries. Passwords can be stored in clear text or one-way encrypted with a hash algorithm like MD5 or SHA1. We start with anonymous binds.

Simple binds

Authentication via simple bind is widely used. The method is supported by ApacheDS 1.0 for all person entries stored within any partition, if they contain a password attribute. How does it work? An LDAP client provides the DN of a user entry and a password to the server, the parameters of the bind operation.

ApacheDS checks whether the given password is the same as the one stored in the *userpassword* attribute of the given entry. If not, the bind operation fails (LDAP error code 49, LDAP_INVALID_CREDENTIALS), and the user is not authenticated.

Using command line tools

Assume this entry from the Seven Seas partition is stored within the directory (only a fragment with the relevant attributes is shown).

```
dn: cn=Horatio Hornblower,ou=people,o=sevenSeas
objectclass: person
objectclass: organizationalPerson
cn: Horatio Hornblower
sn: Hornblower
userpassword: pass
...
```

In the following search command, a user tries to bind with the given DN (option -D) but a wrong password (option -w). The bind fails and the command terminates without performing the search.

```
$ ldapsearch -h zanzibar -p 10389 -D "cn=Horatio Hornblower,ou=people,o=sevenSeas" -w wrong -b
"ou=people,o=sevenSeas" -s base "(objectclass=*)"
ldap_simple_bind: Invalid credentials
ldap_simple_bind: additional info: Bind failed: null
```

If the user provides the correct password during the call of the `ldapsearch` command, the bind operation succeeds and the search operation is performed afterwards.

```
$ ldapsearch -h zanzibar -p 10389 -D "cn=Horatio Hornblower,ou=people,o=sevenSeas" -w pass -b
"ou=people,o=sevenSeas" -s base "(objectclass=*)"
version: 1
dn: ou=people,o=sevenSeas
ou: people
description: Contains entries which describe persons (seamen)
objectclass: organizationalUnit
objectclass: top
```

Binds from Java components using JNDI

Using JNDI, authentication via simple binds is accomplished by appropriate configuration. One option is to provide the parameters in a `Hashtable` object like this

```
import java.util.Hashtable;

import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NamingEnumeration;
import javax.naming.NamingException;

public class SimpleBindDemo {

    public static void main(String[] args) throws NamingException {

        if (args.length < 2) {
            System.err.println("Usage: java SimpleBindDemo <userDN> <password>");
            System.exit(1);
        }
    }
}
```

```

Hashtable env = new Hashtable();
env.put(Context.INITIAL_CONTEXT_FACTORY, "com.sun.jndi.ldap.LdapCtxFactory");
env.put(Context.PROVIDER_URL, "ldap://zanzibar:10389/o=sevenSeas");

env.put(Context.SECURITY_AUTHENTICATION, "simple");
env.put(Context.SECURITY_PRINCIPAL, args[0]);
env.put(Context.SECURITY_CREDENTIALS, args[1]);

try {
    Context ctx = new InitialContext(env);
    NamingEnumeration enm = ctx.list("");
    while (enm.hasMore()) {
        System.out.println(enm.next());
    }
    ctx.close();
} catch (NamingException e) {
    System.out.println(e.getMessage());
}
}

```

If the DN of a user entry and the fitting password are provided as command line arguments, the program binds successfully and performs a search:

```

$ java SimpleBindDemo "cn=Horatio Hornblower,ou=people,o=sevenSeas" pass
ou=people: javax.naming.directory.DirContext
ou=groups: javax.naming.directory.DirContext

```

On the other hand, providing an incorrect password results in a failed bind operation. JNDI maps it to a *NamingException*:

```

$ java SimpleBindDemo "cn=Horatio Hornblower,ou=people,o=sevenSeas" quatsch
[LDAP: error code 49 - Bind failed: null]

```

In real life, you obviously want to separate most of the configuration data from the source code, for instance with the help of the *jndi.properties* file.

Passwords stored one-way encrypted

If passwords are stored in the directory in clear like above, the administrator (*uid=admin,ou=system*) is able to read them. This holds true even if authorization is enabled. The passwords would also be visible in exported LDIF files. This is often unacceptable.



Not only the administrator will be able to read your password, or be visible in LDIF files, but if one does not use SSL, the the password is transmitted in clear text above the wire...

Passwords not stored in clear text

ApacheDS does also support simple binds, if user passwords are stored one-way encrypted. An LDAP client, which creates user entries, applies a hash-function (SHA for instance) to the user passwords beforehand, and stores the users with these fingerprints as *userpassword* values (instead of the clear text

values), for instance:

```
dn: cn=Horatio Hornblower,ou=people,o=sevenSeas
objectclass: person
objectclass: organizationalPerson
cn: Horatio Hornblower
sn: Hornblower
userpassword: {SHA}nU4eI71bcnBGqeO0t9tXvY1u5oQ=
...
```

The value "{SHA}nU4eI71bcnBGqeO0t9tXvY1u5oQ=" means that *SHA* (Secure Hash Algorithm) was applied to the password, and "nU4eI71bcnBGqeO0t9tXvY1u5oQ=" was the result (Base-64 encoded). Please note that it is not possible to calculate the source ("pass" in our case) back from the result. This is why it is called one-way encrypted – it is rather difficult to decrypt it. One may guess many times, calculate the hash values (the algorithms are public) and compare the result. But this would take a long time, especially if you choose a more complex password than we did ("pass").

But how to obtain the hash value for a password?

With some lines of code, it is quite easy to accomplish this task programmatically in Java:

```
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import sun.misc.BASE64Encoder;

public class DigestDemo {
    public static void main(String[] args) throws NoSuchAlgorithmException {
        String password = "pass";
        String algorithm = "SHA";

        // Calculate hash value
        MessageDigest md = MessageDigest.getInstance(algorithm);
        md.update(password.getBytes());
        byte[] bytes = md.digest();

        // Print out value in Base64 encoding
        BASE64Encoder base64encoder = new BASE64Encoder();
        String hash = base64encoder.encode(bytes);
        System.out.println('{'+algorithm+'}' + hash);
    }
}
```

The output is "{SHA}nU4eI71bcnBGqeO0t9tXvY1u5oQ=".

Another option is to use command line tools to calculate the hash value; the [OpenSSL](#) project provides such stuff. Furthermore many LDAP tools allow you to store passwords automatically encrypted with the hash algorithm of your choice. Below two examples ([Softerra LDAP Administrator](#) left, [JXplorer](#) right):

The image shows two side-by-side dialog boxes. The left dialog, titled 'Edit Password', has a 'Please enter your password:' label. It contains two text fields: 'Password:' and 'Confirm:', both filled with four dots. Below these is a checkbox 'Use an encrypted password (recommended)' which is checked, and an unchecked checkbox 'Edit hash manually'. Under 'Edit hash manually' is a 'Hash:' field containing the value '{SHA}nU4eI71bcnBGqe00t9tXvY1u5oQ='. Below the hash field are several radio button options: SHA (selected), SSHA, DES (Crypt), MD5, SMD5, DES (LanMan), MD4, and SMD4. There is also a 'Verify...' button. At the bottom are 'OK', 'Cancel', and 'Help' buttons. The right dialog, titled 'User Password Data', has an 'Enter Password:' field with four dots and a 'Re-enter Password:' field with four dots. Below these is a dropdown menu currently showing 'sha'. At the bottom are 'OK', 'Cancel', and 'Help' buttons.

From an LDAP client point of view

From an LDAP client point of view, the behaviour during authentication is the same as with passwords stored in clear. During a simple bind, a client sends DN and password (unencrypted, i.e. no hash algorithm applied) to the server. If ApacheDS detects, that the user password for the given DN is stored in the directory with a hash function applied, it calculates the hash value of the given password with the appropriate algorithm (this is why the algorithm is stored together with the hashed password). Afterwards it compares the result with the stored attribute value. In case of a match, the bind operation ends successfully:

```
$ ldapsearch -h zanzibar -p 10389 -D "cn=Horatio Hornblower,ou=people,o=sevenSeas" -w pass -b
"ou=people,o=sevenSeas" -s base "(objectclass=*)"
version: 1
dn: ou=people,o=sevenSeas
ou: people
description: Contains entries which describe persons (seamen)
objectclass: organizationalUnit
objectclass: top
```

Providing the hashed value of the userpassword attribute instead of the original value will be rejected by ApacheDS:

```
$ ldapsearch -h zanzibar -p 10389 -D "cn=Horatio Hornblower,ou=people,o=sevenSeas" \\\
-w "{SHA}nU4eI71bcnBGqe00t9tXvY1u5oQ=" -b "ou=people,o=sevenSeas" -s base "(objectclass=*)"
ldap_simple_bind: Invalid credentials
ldap_simple_bind: additional info: Bind failed: null
```

This is intended. If someone was able to catch this value (from an LDIF export for instance), s/he must still provide the password itself in order to get authenticated.



Be Warned: Limited security added

Please note that storing user passwords one-way encrypted only adds limited security. During the bind operation, the credentials are still transmitted unencrypted, if no SSL/TLS communication is used (thus you should definitely consider to do so).

Furthermore, if someone gets an LDIF file with userpassword values digested with SHA etc., s/he may be able to determine some of the passwords with brute force. Calculation of hash functions can be done very fast, and the attacker can attempt millions of values with ease, without you getting notice of it. Therefore protect your data, even if one-way encryption is applied to the passwords!

Anonymous binds

In some occasions it is appropriate to allow LDAP clients to permit operations without authentication. If data managed by the directory service is well known by all clients, it is not uncommon to allow search operations (not manipulation) within this data to all clients – without providing credentials. An example for this are enterprise wide telephone books, if clients access the directory service from the intranet.

Enable/disable anonymous binds

Anonymous access is disabled by default. Changing this is one of the basic configuration tasks. If you use the server standalone configured with a *server.xml* file, you can enable it by changing the value for property *allowAnonymousAccess* in the Spring bean definition for bean *configuration*, as depicted in the following fragment:

```
<bean id="configuration"
class="org.apache.directory.server.configuration.MutableServerStartupConfiguration">
    ...
    <property name="allowAnonymousAccess" value="true" />
    ...
</bean>
```

A restart of the server is necessary for this change to take effect.

Example: Server behaviour with anonymous binds disabled

Assume anonymous binds are disabled, as configured by default in ApacheDS 1.0, and our sample partition *Seven Seas* present in the server. Here is an example with a search operation performed by a command line tool as a client. It tries to connect anonymously (no DN and password given, i.e. options -D and -w missing) to the server. Afterwards the entry *ou=people,o=sevenSeas* should be displayed.

See the command and the resulting error message provided by the server below

```
$ ldapsearch -h zanzibar -p 10389 -b "ou=people,o=sevenSeas" -s one "(objectclass=*)"
ldap_search: Insufficient access
ldap_search: additional info: failed on search operation: Anonymous binds have been disabled!
```

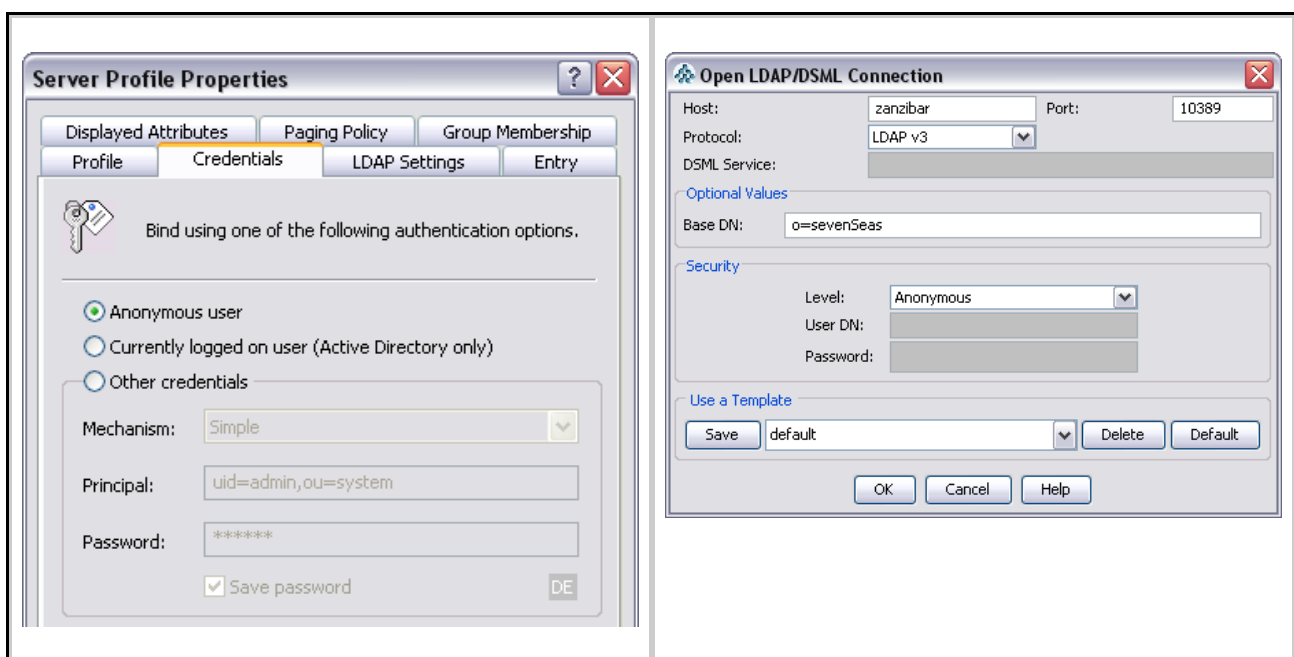
Example: Server behaviour with anonymous binds enabled

Now the same command performed against ApacheDS 1.0 with anonymous access enabled as described above. The behaviour is different – the entry is visible.

```
$ ldapsearch -h zanzibar -p 10389 -b "ou=people,o=sevenSeas" -s base "(objectclass=*)"
version: 1
dn: ou=people,o=sevenSeas
ou: people
description: Contains entries which describe persons (seamen)
objectclass: organizationalUnit
objectclass: top
```

Other clients

The examples above have used a command line tool. Of course graphical tools and programmatical access (JNDI etc.) allow anonymous binds as well. Here are screenshots from the configuration dialogs of [Softerra LDAP Administrator](#) (left) and [JXplorer](#) as examples. During configuration of the connection data, the option *Anonymous user* (*Anonymous* respectively) leads to anonymous binds.



Use this feature wisely

With anonymous access enabled it is not only possible to search the directory without providing username and password. With authorization disabled, anonymous users may also be able to modify data. It is therefore highly recommended to enable and configure the authorization subsystem as well. Learn more about authorization in the [Basic Authorization](#) section.

How to authenticate a user by uid and password?

If you want to use simple binds with user DN and password within a Java component, in order to authenticate users programatically, in practice one problem arises: Most users do not know their DN. Therefore they will not be able to enter it. And even if they know it, it would be frequently very laborious due to the length of the DN. It would be easier for a user if s/he only has to provide a short, unique *ID* and the password, like in this web form

Enter your account details below to login to Confluence.

Username:

Password:

☐ **Remember my login on this computer**

Log In

Usually the ID is an attribute within the user's entry. In our sample data (Seven Seas), each user entry contains the *uid* attribute, for instance uid=hhornblo for Captain Hornblower:

```
dn: cn=Horatio Hornblower,ou=people,o=sevenSeas
objectclass: person
objectclass: organizationalPerson
objectclass: inetOrgPerson
objectclass: top
cn: Horatio Hornblower
description: Capt. Horatio Hornblower, R.N
givenname: Horatio
sn: Hornblower
uid: hhornblo
mail: hhornblo@royalnavy.mod.uk
userpassword: {SHA}nU4eI7lbcnBGqe00t9tXvYlu5oQ=
```

But how to authenticate a user who provides "hhornblo"/"pass" instead of "cn=Horatio Hornblower,ou=people,o=sevenSeas"/"pass" with the help of ApacheDS?

An algorithm

In order to accomplish this task programmatically, one option is to perform the following steps

Arguments

- *uid* of a user (e.g. "hhornblo")
- *password* proclaimed to be correct for the user

Steps

1. Bind to ApacheDS anonymously, or with the DN of a technical user. In both cases it must be possible to search the directory afterwards (authorization has to be configured that way)
2. Perform a search operation with an appropriate filter to find the user entry for the given ID, in our case "(&(objectClass=inetorgperson)(uid=hhornblo))"
 - If the search result is empty, the user does not exist – terminate
 - If the search result contains more than one entry, the given ID is not unique, this is likely a data error within your directory
3. Bind to ApacheDS with the DN of the entry found in the previous search, and the *password* provided

as argument

- If the bind operation fails, the password is wrong, and the result is *false* (not authenticated)
- If the bind is successful, authenticate the user

Sample code with JNDI

The algorithm described above is implemented by many software solutions which are able to integrate LDAP directories. You will learn more about some of them and their configuration options within a later section of this guide.

For illustration purposes, here is a simple Java program which performs the steps with the help of JNDI. It uses anonymous bind for the first step, hence it must be enabled (replace with a technical user, if it better meets your requirements).

```
import java.util.Hashtable;
import javax.naming.Context;
import javax.naming.NamingEnumeration;
import javax.naming.NamingException;
import javax.naming.directory.DirContext;
import javax.naming.directory.InitialDirContext;
import javax.naming.directory.SearchControls;
import javax.naming.directory.SearchResult;

public class AdvancedBindDemo {

    public static void main(String[] args) throws NamingException {

        if (args.length < 2) {
            System.err.println("Usage: java AdvancedBindDemo <uid> <password>");
            System.exit(1);
        }

        Hashtable env = new Hashtable();
        env.put(Context.INITIAL_CONTEXT_FACTORY, "com.sun.jndi.ldap.LdapCtxFactory");
        env.put(Context.PROVIDER_URL, "ldap://zanzibar:10389/");
        env.put(Context.SECURITY_AUTHENTICATION, "simple");

        String uid = args[0];
        String password = args[1];

        DirContext ctx = null;
        try {
            // Step 1: Bind anonymously
            ctx = new InitialDirContext(env);

            // Step 2: Search the directory
            String base = "o=sevenSeas";
            String filter = "(&(objectClass=inetOrgPerson)(uid={0}))";
            SearchControls ctls = new SearchControls();
            ctls.setSearchScope(SearchControls.SUBTREE_SCOPE);
            ctls.setReturningAttributes(new String[0]);
            ctls.setReturningObjFlag(true);
            NamingEnumeration enm = ctx.search(base, filter, new String[] { uid }, ctls);

            String dn = null;
            if (enm.hasMore()) {
                SearchResult result = (SearchResult) enm.next();
                dn = result.getNameInNamespace();

                System.out.println("dn: "+dn);
            }

            if (dn == null || enm.hasMore()) {
                // uid not found or not unique
                throw new NamingException("Authentication failed");
            }

            // Step 3: Bind with found DN and given password
```

```

        ctx.addToEnvironment(Context.SECURITY_PRINCIPAL, dn);
        ctx.addToEnvironment(Context.SECURITY_CREDENTIALS, password);
        // Perform a lookup in order to force a bind operation with JNDI
        ctx.lookup(dn);
        System.out.println("Authentication successful");

    } catch (NamingException e) {
        System.out.println(e.getMessage());
    } finally {
        ctx.close();
    }
}

```

Some example calls:

```

$ java AdvancedBindDemo unknown sailor
Authentication failed

$ java AdvancedBindDemo hornblo pass
dn: cn=Horatio Hornblower,ou=people,o=sevenSeas
Authentication successful

$ java AdvancedBindDemo hornblo quatsch
dn: cn=Horatio Hornblower,ou=people,o=sevenSeas
[LDAP: error code 49 - Bind failed: null]

```

The examples consist of an unknown user (an *inetOrgPerson* entry with uid=unknown does not exist), a successful authentication, and an attempt with an existing uid but a wrong password.

Resources

- [RFC 2829](#) Authentication Methods for LDAP
- [The Secure Hash Algorithm Directory](#) MD5, SHA-1 and HMAC Resources

Basic authorization

This page last changed on Jan 14, 2007 by [szoerner](#).

<< Previous:Authentication options	ApacheDS v1.0 Basic User's Guide (TOC)	Next:How to enable SSL>>
--	--	--

This section describes the default authorization functionality of ApacheDS 1.0, which is very simple. On the other hand, it is unadequate for most serious deployments. Therefore a basic example to the "real" authorization subsystem is provided as well.

- [What is authorization?](#)
- [Default authorization behavior for directory operations](#)
- [Simple example for the ACI subsystem](#)
- [Verification, that it works](#)
- [Resources](#)

What is authorization?

After authentication of a user or an application (or more generally an LDAP client) against the directory server (or attaining anonymous access respectively), certain LDAP operations will be granted or rejected, according to configuration and certain rules. This process of granting access is called authorization.

Authorization for directory operations is not strictly standardized in the LDAP world, [RFC 2829](#) describes various scenarios and concepts, but does not enforce a concrete implementation. Thus each product comes with its own authorization feature. So does ApacheDS. A powerful authorization subsystem is provided since version 0.9.3, but disabled as a default.

Authorization for directory operations vs. group membership

In order to accomplish their authorization functionality, software components often take advantage of LDAP groups stored within the directory. *groupOfNames* and *groupOfUniqueNames* are common object classes for groups entries; they contain the DN's of their members (users, other groups) as attribute values.

In order to illustrate this, the "Seven Seas" example partition contains such group entries below "ou=groups,o=sevenSeas". Here the entry of a group describing the HMS Bounty crew (before the mutiny) in LDIF format.

```
dn: cn=HMS Bounty,ou=crews,ou=groups,o=sevenSeas
objectclass: groupOfUniqueNames
objectclass: top
cn: HMS Bounty
uniquemember: cn=William Bligh,ou=people,o=sevenSeas
```

```
uniqueMember: cn=Fletcher Christian,ou=people,o=sevenSeas
uniqueMember: cn=John Fryer,ou=people,o=sevenSeas
...
```

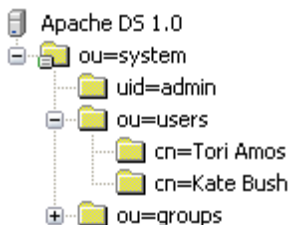
In such a scenario, a user, who is directly or indirectly member of a certain group is permitted to do something. The software component acts as a normal LDAP client and determines group belonging with the help of ordinary search operations. This is widely used but has nothing to do with the authorization for directory operations as described in this section (except that the client needs the permission to search the data). Learn more about best practices in this area in the article [Practices in Directory Groups](#). Further examples in this guide are the Tomcat and Apache HTTPD integration sections.

Default authorization behavior for directory operations

Without access controls enabled all entries are accessible and alterable by all: even anonymous users. There are however some minimal built-in rules for protecting users and groups within the server without having to turn on the ACI subsystem.

Sample data within "ou=users,ou=system"

In addition to our brave sailors below *ou=people,o=sevenSeas*, assume the following to entries present within *ou=users,ou=system*:



```
dn: cn=Tori Amos,ou=users,ou=system
objectclass: person
objectclass: top
sn: Amos
cn: Tori Amos
userpassword: amos
```

```
dn: cn=Kate Bush,ou=users,ou=system
objectclass: person
objectclass: top
sn: Bush
cn: Kate Bush
userpassword: bush
```

They are used in the following examples, in conjunction with *o=sevenSeas*, to describe the default authorization rules.

Rules and sample operations

Without ACIs the server automatically protects, hides, the admin user from everyone but the admin user.

Here a sample search operation in order to demonstrate this protection. The same command is submitted three times with different users.

```
$ ldapsearch -h zanzibar -p 10389 -D "uid=admin,ou=system" -w secret \\  
-b "ou=system" -s one "(uid=admin)" dn  
version: 1  
dn: uid=admin,ou=system  
  
$ ldapsearch -h zanzibar -p 10389 -D "cn=William Bush,ou=people,o=sevenSeas" -w pass \\  
-b "ou=system" -s one "(uid=admin)" dn  
  
$ ldapsearch -h zanzibar -p 10389 -D "cn=Tori Amos,ou=users,ou=system" -w amos \\  
-b "ou=system" -s one "(uid=admin)" dn  
  
$
```

Users cannot see other user entries under the 'ou=users,ou=system' entry. So placing new users there automatically protects them. Placing new users anywhere else exposes them.

```
$ ldapsearch -h zanzibar -p 10389 -D "uid=admin,ou=system" -w secret \\  
-b "ou=users,ou=system" -s one "(objectclass=*)" dn  
version: 1  
dn: cn=Tori Amos,ou=users,ou=system  
  
dn: cn=Kate Bush,ou=users,ou=system  
  
$ ldapsearch -h zanzibar -p 10389 -D "cn=Kate Bush,ou=users,ou=system" -w bush \\  
-b "ou=users,ou=system" -s one "(objectclass=*)" dn  
version: 1  
dn: cn=Kate Bush,ou=users,ou=system  
  
$ ldapsearch -h zanzibar -p 10389 -D "cn=William Bush,ou=people,o=sevenSeas" -w pass \\  
-b "ou=users,ou=system" -s one "(objectclass=*)" dn  
  
$ ldapsearch -h zanzibar -p 10389 -D "cn=William Bush,ou=people,o=sevenSeas" -w pass \\  
-b "ou=people,o=sevenSeas" -s one "(objectclass=*)" dn  
version: 1  
dn: cn=Horatio Hornblower,ou=people,o=sevenSeas  
  
dn: cn=William Bush,ou=people,o=sevenSeas  
  
dn: cn=Thomas Masterman Hardy,ou=people,o=sevenSeas  
  
dn: cn=Cornelius Buckley,ou=people,o=sevenSeas  
  
dn: cn=William Bligh,ou=people,o=sevenSeas  
...  
$
```

Groups defined using *groupOfNames* or *groupOfUniqueNames* under the 'ou=groups,ou=system' are also protected from access or alteration by anyone other than the admin user. Again this protection is not allowed anywhere else but under these entries.

Is this sufficient?

For simple configurations the described rules should provide adequate protection but it lacks flexibility. For advanced configurations users should enable the ACI subsystem. This however shuts down access to everything by everyone except the admin user which bypasses the ACI subsystem. Directory administrators should look at the documentation on how to specify access control information in the Advanced User's Guide.

Simple example for the ACI subsystem

As an appetizer for the stunning ACI subsystem (ACI = access control item) within ApacheDS, we provide a simple yet realistic example. It manifests the following requirements

Requirements met

1. Suffix "o=sevenSeas" used as Access Control Specific Area
2. User "cn=Horatio Nelson,ou=people,o=sevenSeas" should be able to perform all operations (delete, add, ...) below the base "o=sevenSeas"
3. Other users and anonymous users should only be able to search and compare (no add, modify etc.)
4. Other users and anonymous users should not be able to read the userPassword attribute

Enable the ACI Subsystem

The authorization (ACI) subsystem is disabled by default. If you use the server standalone configured with a *server.xml* file, you can enable it by changing the value for property *accessControlEnabled* in the Spring bean definition for bean *configuration*, as depicted in the following fragment:

```
<bean id="configuration"
class="org.apache.directory.server.configuration.MutableServerStartupConfiguration">
    ...
    <property name="accessControlEnabled" value="true" />
    ...
</bean>
```

A restart of the server is necessary for this change to take effect.

Further configuration tasks to perform afterwards

1. Create an operational attribute *administrativeRole* with value "accessControlSpecificArea" in the entry "o=sevenSeas".
2. Create a subentry subordinate to "o=sevenSeas" to grant all operations' permissions to "cn=Horatio Nelson,ou=people,o=sevenSeas", who acts as directory manager

The subentry should contain the following attributes and values:

```
cn="sevenSeasAuthorizationRequirementsACISubentry"
subtreeSpecification="{ }"
prescriptiveACI="{
    identificationTag "directoryManagerFullAccessACI",
    precedence 11,
    authenticationLevel simple,
    itemOrUserFirst userFirst:
    {
        userClasses
        {
            name { "cn=Horatio Nelson,ou=people,o=sevenSeas" }
        },
        userPermissions
        {
            {
```

```

        protectedItems
        {
            entry, allUserAttributeTypesAndValues
        },
        grantsAndDenials
        {
            grantAdd, grantDiscloseOnError, grantRead,
            grantRemove, grantBrowse, grantExport, grantImport,
            grantModify, grantRename, grantReturnDN,
            grantCompare, grantFilterMatch, grantInvoke
        }
    }
}
}
} "

```

3. A new attribute value should added to the previously created Subentry's prescriptiveACI attribute to grant search and compare permissions to all users.

The new value:

```
prescriptiveACI="{
  identificationTag "allUsersSearchAndCompareACI",
  precedence 10,
  authenticationLevel simple,
  itemOrUserFirst userFirst:
  {
    userClasses
    {
      allUsers
    },
    userPermissions
    {
      {
        protectedItems
        {
          entry, allUserAttributeTypesAndValues
        },
        grantsAndDenials
        {
          grantRead, grantBrowse, grantReturnDN,
          grantCompare, grantFilterMatch, grantDiscloseOnError
        }
      }
    }
  }
}"
```

4. A new attribute value should added to the previously created Subentry's prescriptiveACI attribute to deny search and compare permissions for *userPassword* attribute to all users.

The new value:

```
prescriptiveACI="{
  identificationTag \"preventAllUsersFromReadingUserPasswordAttributeACI\",
  precedence 10,
  authenticationLevel simple,
  itemOrUserFirst userFirst:
  {
    userClasses
    {
      allUsers
    },
    userPermissions
    {
      {
        protectedItems
```

```

        {
            attributeType { userPassword }
        },
        grantsAndDenials
        {
            denyRead, denyCompare, denyFilterMatch
        }
    }
}
}
} "

```

The two values given in 3 and 4 can be combined in a single value as:

```
prescriptiveACI="{
    identificationTag "allUsersACI",
    precedence 10,
    authenticationLevel none,
    itemOrUserFirst userFirst:
    {
        userClasses
        {
            allUsers
        },
        userPermissions
        {
            {
                protectedItems { entry, allUserAttributeTypesAndValues },
                grantsAndDenials { grantRead, grantBrowse, grantReturnDN,
                                   grantCompare, grantFilterMatch,
grantDiscloseOnError }
            },
            {
                protectedItems { attributeType { userPassword } },
                grantsAndDenials { denyRead, denyCompare, denyFilterMatch }
            }
        }
    }
}"
```

LDIF for this configuration

The following LDIF file ([authz_sevenSeas.ldif](#)) provides a set of changes made to directory entries in the "Seven Seas" data. In total it performs the steps described above.

```
# File authz_sevenSeas.ldif
#
# Create an operational attribute "administrativeRole"
# with value "accessControlSpecificArea" in the entry "o=sevenSeas".
#
dn: o=sevenSeas
changetype: modify
add: administrativeRole
administrativeRole: accessControlSpecificArea

# Create a subentry subordinate to "o=sevenSeas" to grant all operations' permissions
# to "cn=Horatio Nelson,ou=people,o=sevenSeas", to grant search and compare permissions
# to all users and to deny search and compare permissions for userPassword attribute to all
# users.
#
dn: cn=sevenSeasAuthorizationRequirementsACISubentry,o=sevenSeas
changetype: add
objectclass: top
objectclass: subentry
objectclass: accessControlSubentry
cn: sevenSeasAuthorizationRequirementsACISubentry
subtreeSpecification: {}
prescriptiveACI: {
```

```

identificationTag "directoryManagerFullAccessACI",
precedence 11,
authenticationLevel simple,
itemOrUserFirst userFirst:
{
  userClasses
  {
    name { "cn=Horatio Nelson,ou=people,o=sevenSeas" }
  },
  userPermissions
  {
    {
      protectedItems
      {
        entry, allUserAttributeTypesAndValues
      },
      grantsAndDenials
      {
        grantAdd, grantDiscloseOnError, grantRead,
        grantRemove, grantBrowse, grantExport, grantImport,
        grantModify, grantRename, grantReturnDN,
        grantCompare, grantFilterMatch, grantInvoke
      }
    }
  }
}
}
}
prescriptiveACI: {
  identificationTag "allUsersACI",
  precedence 10,
  authenticationLevel none,
  itemOrUserFirst userFirst:
  {
    userClasses
    {
      allUsers
    },
    userPermissions
    {
      {
        protectedItems { entry, allUserAttributeTypesAndValues },
        grantsAndDenials { grantRead, grantBrowse, grantReturnDN,
                           grantCompare, grantFilterMatch, grantDiscloseOnError }
      },
      {
        protectedItems { attributeType { userPassword } },
        grantsAndDenials { denyRead, denyCompare, denyFilterMatch }
      }
    }
  }
}
}

```

To apply this configuration to the sample data partition, you can perform an *ldapmodify* with the LDIF as argument:

```

$ ldapmodify -h zanzibar -p 10389 -D "uid=admin,ou=system" -w secret -f authz_sevenSeas.ldif
modifying entry o=sevenSeas

adding new entry cn=sevenSeasAuthorizationRequirementsACISubentry,o=sevenSeas
$

```

It is also possible to use graphical tools; some of them offer the feature to perform operations given in LDIF.

Verification, that it works

After successfully applying the changes to the sample partition, one may ask how to check whether it works. We therefore perform some operations with the help of command line tools. Some will be permitted, some will not (and cause an appropriate error message). It would also be able to check this with the help of graphical tools (you might like to do this instead). But it is easier to document the parameters used with the help command line arguments.

Performing some search operations in order to read data

Bind as user "William Bush" and search for entries which match "(uid=hhornblo)". Expected behavior: We are able to read the attributes of entry "cn=Horatio Hornblower,ou=people,o=sevenSeas" (the only entry which matches the filter). The password attribute should not be visible. It works as desired:

```
$ ldapsearch -h zanzibar -p 10389 -D "cn=William Bush,ou=people,o=sevenSeas" -w pass \\  
-b "o=sevenSeas" -s sub "(uid=hhornblo)"  
version: 1  
dn: cn=Horatio Hornblower,ou=people,o=sevenSeas  
mail: hhornblo@royalnavy.mod.uk  
objectclass: person  
objectclass: organizationalPerson  
objectclass: inetOrgPerson  
objectclass: top  
cn: Horatio Hornblower  
uid: hhornblo  
givenname: Horatio  
description: Capt. Horatio Hornblower, R.N  
sn: Hornblower
```

In the described configuration, the user "Horatio Nelson" acts as a directory manager below "o=sevenSeas". Hence he should basically be allowed to do everything. He should even be able to see other users' *userPassword* values. In our case, the hash function *SHA* was applied to them:

```
$ ldapsearch -h zanzibar -p 10389 -D "cn=Horatio Nelson,ou=people,o=sevenSeas" -w pass \\  
-b "o=sevenSeas" -s sub "(objectclass=person)"  
" uid userPassword  
version: 1  
dn: cn=Horatio Hornblower,ou=people,o=sevenSeas  
userpassword: {SHA}nU4eI7lbcnBGqe00t9tXvYlu5oQ=  
uid: hhornblo  
  
dn: cn=William Bush,ou=people,o=sevenSeas  
userpassword: {SHA}nU4eI7lbcnBGqe00t9tXvYlu5oQ=  
uid: wbush  
  
dn: cn=Thomas Quist,ou=people,o=sevenSeas  
userpassword: {SHA}nU4eI7lbcnBGqe00t9tXvYlu5oQ=  
uid: tquist  
...
```

But "Horatio Nelson" is not able to perform searches in other areas than "o=sevenSeas" to see the entries. Of course our global ApacheDS administrator "uid=admin,ou=system" is still able to see them:

```
$ ldapsearch -h zanzibar -p 10389 -D "cn=Horatio Nelson,ou=people,o=sevenSeas" -w pass \\  
-b "ou=system" -s sub "(objectclass=person)"  
  
$ ldapsearch -h zanzibar -p 10389 -D "uid=admin,ou=system" -w secret \\  
-b "ou=system" -s sub "(objectclass=person)"  
version: 1  
dn: uid=admin,ou=system  
sn: administrator  
cn: system administrator
```

```
objectClass: top
objectClass: person
objectClass: organizationalPerson
objectClass: inetOrgPerson
userpassword: secret
uid: admin
displayName: Directory Superuser

dn: cn=Tori Amos,ou=users,ou=system
cn: Tori Amos
userpassword: amos
objectclass: person
objectclass: top
sn: Amos
...
```

Trying to manipulate data

Until now the authorization only hid data (entries, attributes) from users with insufficient access rights. Let's perform some operations which try to manipulate the directory data!

Adding an entry

First we try to add a new user to the "Seven Seas" partition. The data for the entry is inspired by "Peter Pan" and provided by this LDIF file ([captain_hook.ldif](#)):

```
# File captain_hook.ldif
dn: cn=James Hook,ou=people,o=sevenSeas
objectclass: person
objectclass: top
cn: James Hook
description: A pirate captain and Peter Pan's nemesis
sn: Hook
mail: jhook@neverland
userpassword: peterPan
```

An anonymous user is not allowed to create new entries, as the following error message shows:

```
$ ldapmodify -h zanzibar -p 10389 -a -f captain_hook.ldif
adding new entry cn=James Hook,ou=people,o=sevenSeas
ldap_add: Insufficient access
ldap_add: additional info: failed to add entry cn=James Hook,ou=people,o=sevenSeas: null
$
```

The same holds true for all "Seven Seas"-user other than "Horatio Nelson". The latter is permitted to do so:

```
$ ldapmodify -h zanzibar -p 10389 -D "cn=William Bush,ou=people,o=sevenSeas" -w pass \\
-a -f captain_hook.ldif
adding new entry cn=James Hook,ou=people,o=sevenSeas
ldap_add: Insufficient access
ldap_add: additional info: failed to add entry cn=James Hook,ou=people,o=sevenSeas: null

$ ldapmodify -h zanzibar -p 10389 -D "cn=Horatio Nelson,ou=people,o=sevenSeas" -w pass \\
-a -f captain_hook.ldif
adding new entry cn=James Hook,ou=people,o=sevenSeas
$
```

Afterwards a new entry is successfully created within the "Seven Seas" partition by user "Horatio Nelson". The '+' sign in the attributes list of the *ldapsearch* command causes ApacheDS to return the operational attributes, which demonstrate this.

```
$ ldapsearch -h zanzibar -p 10389 -b "o=sevenSeas" -s sub "(cn=James Hook)" +
version: 1
dn: cn=James Hook,ou=people,o=sevenSeas
accessControlSubentries: cn=sevenSeasAuthorizationRequirementsACISubentry,o=sevenSeas
creatorsName: cn=Horatio Nelson,ou=people,o=sevenSeas
createTimestamp: 20061203140109Z
```

Modifying an entry

As a further example which tries to write to the directory, we add a new value to the description attribute of the freshly created entry for Captain Hook. With a change entry in an LDIF file, it looks like this (file [captain_hook_modify.ldif](#)):

```
# File captain_hook_modify.ldif
dn: cn=James Hook,ou=people,o=sevenSeas
changetype: modify
add: description
description: Wears an iron hook in place of his right hand
-
```

Performing the modification with the *ldapmodify* command line tool again fails for users other than "Horatio Nelson" (who is allowed to do due to the authorization configuration) and "uid=admin,ou=system".

```
$ ldapmodify -h zanzibar -p 10389 -f captain_hook_modify.ldif
modifying entry cn=James Hook,ou=people,o=sevenSeas
ldap_modify: Insufficient access
ldap_modify: additional info: failed to modify entry cn=James Hook,ou=people,o=sevenSeas: null

$ ldapmodify -h zanzibar -p 10389 -D "cn=William Bush,ou=people,o=sevenSeas" -w pass \\\
-f captain_hook_modify.ldif
modifying entry cn=James Hook,ou=people,o=sevenSeas
ldap_modify: Insufficient access
ldap_modify: additional info: failed to modify entry cn=James Hook,ou=people,o=sevenSeas: null

$ ldapmodify -h zanzibar -p 10389 -D "cn=Horatio Nelson,ou=people,o=sevenSeas" -w pass \\\
-f captain_hook_modify.ldif
modifying entry cn=James Hook,ou=people,o=sevenSeas
```

Deleting an entry

Now it is finale time. A demonstration on how to delete the villain's entry from the directory. With an LDIF file ([captain_hook_delete.ldif](#)) with an appropriate change entry, this can easily be accomplished, if the bind user is allowed to do so.

```
# File captain_hook_delete.ldif
dn: cn=James Hook,ou=people,o=sevenSeas
changetype: delete
```

Applying this file with the help of *ldapmodify* results in a behavior comparable to the modification.

Anonymous or "normal" users (like "William Bush") are not permitted to delete Captain Hook's entry. The user "Horatio Nelson", our directory manager for "Seven Seas", is:

```
$ ldapmodify -h zanzibar -p 10389 -f captain_hook_delete.ldif
deleting entry cn=James Hook,ou=people,o=sevenSeas
ldap_delete: Insufficient access
ldap_delete: additional info: failed to delete entry cn=James Hook,ou=people,o=sevenSeas: null

$ ldapmodify -h zanzibar -p 10389 -D "cn=William Bush,ou=people,o=sevenSeas" -w pass \\
-f captain_hook_delete.ldif
deleting entry cn=James Hook,ou=people,o=sevenSeas
ldap_delete: Insufficient access
ldap_delete: additional info: failed to delete entry cn=James Hook,ou=people,o=sevenSeas: null

$ ldapmodify -h zanzibar -p 10389 -D "cn=Horatio Nelson,ou=people,o=sevenSeas" -w pass \\
-f captain_hook_delete.ldif
deleting entry cn=James Hook,ou=people,o=sevenSeas
$
```

The entry "cn=James Hook,ou=people,o=sevenSeas" has been successfully deleted from the partition. Our little demonstration on how the ACI subsystem with a realistic configuration behaves end here. Learn more about it in the Advanced User's Guide.

Resources

- [Practices in Directory Groups](#) describes how to use groups within LDAP directories. Highly recommended.
- The [ApacheDS v1.0 Advanced User's Guide](#) provides a detailed authorization chapter
- [RFC 2849](#) The LDAP Data Interchange Format (LDIF) is used extensively in this section

<< Previous:Authentication options	ApacheDS v1.0 Basic User's Guide (TOC)	Next:How to enable SSL>>
--	--	--

Basic configuration tasks

This page last changed on Jan 14, 2007 by [ck](#).

<< Previous:Installing and starting the server	ApacheDS v1.0 Basic User's Guide (TOC)	
--	--	--

This section describes some important configuration tasks. The selection is partly driven by questions frequently asked on the mailing list.

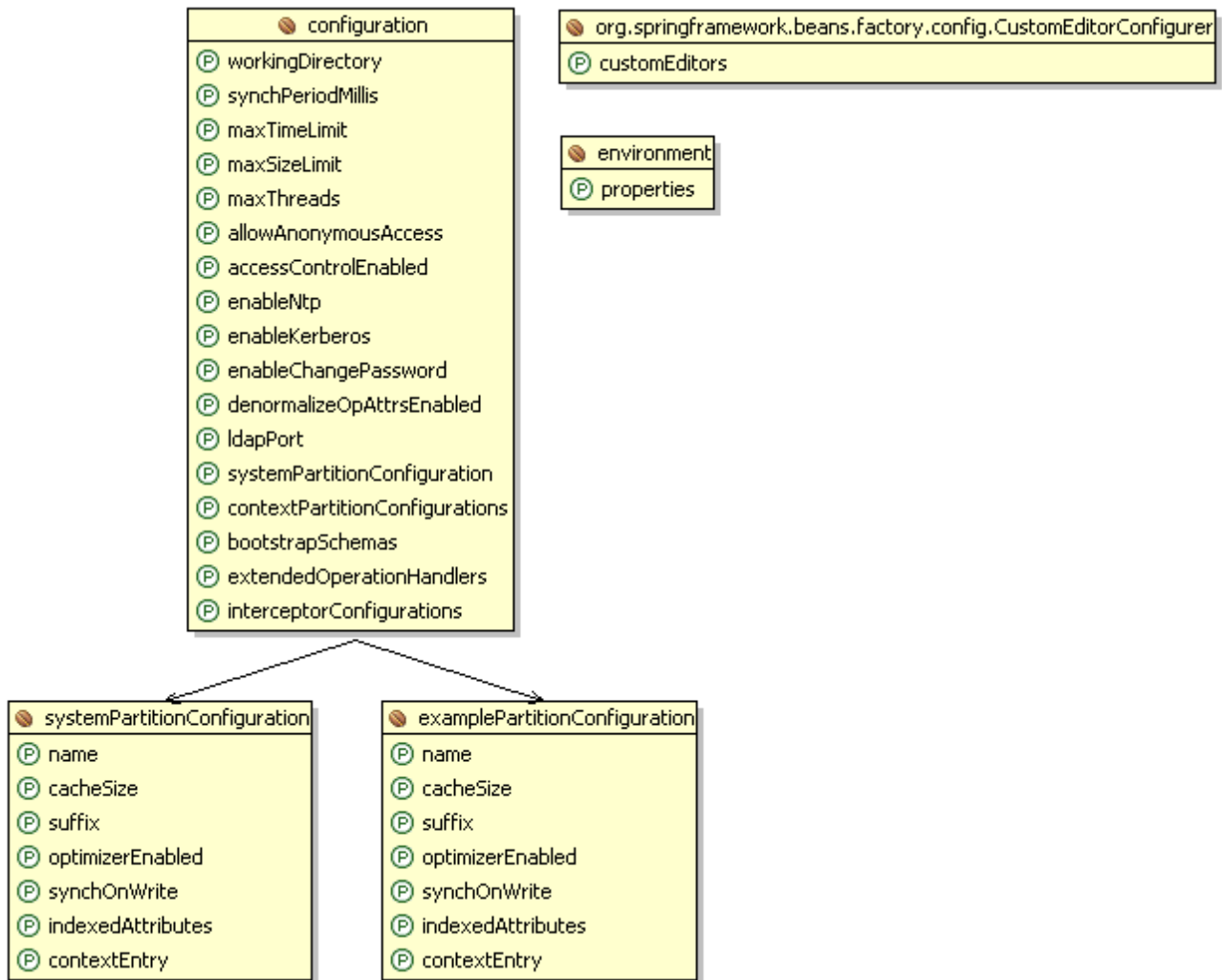
- [Configuration of ApacheDS with Spring](#)
- [Changing the server port for LDAP](#)
- [Changing the admin password](#)
- [Adding your own partition \(suffix\)](#)
- [Configure logging](#)
- [Enable/disable anonymous access](#)
- [Configure indices](#)
- [Memory Allocation](#)
- [Resources](#)

Configuration of ApacheDS with Spring

There are several options to configure ApacheDS. For instance you can practically do everything programmatically if you embed the server in a Java component.

For this guide we assume a standard installation of ApacheDS run standalone, and the default mechanism to configure this deployment option is (in almost all cases) changing the file *server.xml*, which is located in the *conf* directory of your ApacheDS installation. The file is composed of bean definitions, because configuration in ApacheDS 1.0 is done with the help of the [Spring Framework](#).

Despite the fact that the default *server.xml* shipped with the product is somewhat long, a quick look with the help of the [Spring IDE](#) displays that the structure is rather simple:



Most configuration tasks can be accomplished by modifying the properties of existing bean definitions, or (e.g. for a new partition) by adding new beans of certain types and wiring them to the configuration.

Note that the picture above does not show all properties available in the configuration. Only those are visible for which the default *server.xml* contains a value. There are more, and in case of absence the default value is chosen. Feel free to browse the file to get an impression about further options – several other features controlled by properties are commented out.

Changing the server port for LDAP

By default the LDAP server listens on port 10389. It is quite common to run LDAP on 389, which is the well-known port for this protocol. Of course other options are imaginable as well. Changing the LDAP port is a good example for adjusting the existing Spring configuration as introduced in the paragraph above.

Just pick the following line within the "configuration"-bean from the *server.xml* file

```
<bean id="configuration"
class="org.apache.directory.server.configuration.MutableServerStartupConfiguration">
    ...
    <property name="ldapPort" value="10389" />
    ...
</bean>
```

and change it to your needs. You have to restart the server afterwards in order to take this change effect.



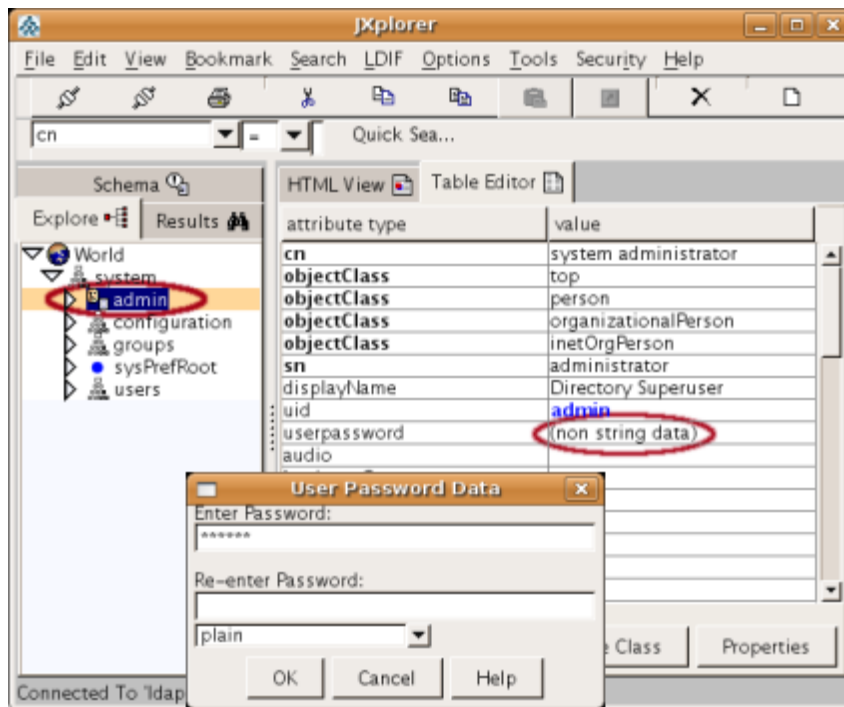
Due to traditional Unix security restrictions, ports less than 1024 were "trusted". Thus on a Unix-System, a non-root process must listen on a port greater than 1023.

Changing the admin password

Changing the admin password is a bit tricky at the moment. We will simplify that in one of the next releases:

1. First while the server is up and running log into as admin (uid=admin,ou=system) using the default password 'secret' and bind to ou=system. If you are using JXplorer this will look like the following:

2. Change the userPassword attribute in the admin entry.



3. Shutdown the server.
4. Open the conf/server.xml file and set the admin password, currently set to 'secret', to the new value.

```
<prop key="java.naming.security.credentials">secret</prop>
```

5. Start up the server.
6. Verify that you can login as admin with the new password.

Adding your own partition (suffix)

In ApacheDS entries are stored in partitions, each partition contains a complete entry tree. Multiple partitions may exist, the entry trees they contain are disconnected to each other. The entries in a particular partition are stored below some naming context called the partition suffix. The implementation of partitions is based on JDBM B+Trees (but it's possible to add custom partition implementations). By default there exist a partition with the suffix "dc=example,dc=com".

A partition contains a few data :

- An **id**, which uniquely identifies the partition
- A external **name**, which is used by managing tools
- A **cacheSize**, the cache size expressed as a number of entries.
- A **suffix** which is a LdapDN object (dc=example, dc=com, for instance)
- **indexedAttributes**, a set of attributes which will be indexed
- **contextEntry**, the root entry that will be added to the partition
- And some specific parameters associated with the backend (**synchOnWrite**, **optimizerEnabled**, etc).

For the examples in the following sections, we want to add a partition with the suffix "o=sevenSeas" and

the id "sevenSeasPartitionConfiguration". This requires editing of the conf/server.xml file. Open it in your favorite editor and look for the following element with name contextPartitionConfigurations. Add another ref element for the sevenSeas partition.

```
<property name="contextPartitionConfigurations">
  <set>
    <ref bean="examplePartitionConfiguration"/>
    <ref bean="sevenSeasPartitionConfiguration"/>
  </set>
</property>
```

Now we will add the actual partition. Just copy & paste this set of elements in the configuration file:

```
<bean id="examplePartitionConfiguration"
class="org.apache.directory.server.core.partition.impl.btree.MutableBTreePartitionConfiguration">
  <property name="name" value="example" />
  <property name="cacheSize" value="100"/>
  <property name="suffix" value="dc=example,dc=com" />
  <property name="optimizerEnabled" value="true" />
  <property name="synchOnWrite" value="true" />
  <property name="indexedAttributes">
    <set>
      <bean
class="org.apache.directory.server.core.partition.impl.btree.MutableIndexConfiguration">
        <property name="attributeId" value="1.3.6.1.4.1.18060.0.4.1.2.1" />
        <property name="cacheSize" value="100" />
      </bean>
      <bean
class="org.apache.directory.server.core.partition.impl.btree.MutableIndexConfiguration">
        <property name="attributeId" value="1.3.6.1.4.1.18060.0.4.1.2.2" />
        <property name="cacheSize" value="100" />
      </bean>
      ...
      <bean
class="org.apache.directory.server.core.partition.impl.btree.MutableIndexConfiguration">
        <property name="attributeId" value="dc" />
        <property name="cacheSize" value="100" />
      </bean>
      <bean
class="org.apache.directory.server.core.partition.impl.btree.MutableIndexConfiguration">
        <property name="attributeId" value="ou" />
        <property name="cacheSize" value="100" />
      </bean>
      ...
    </set>
  </property>
  <property name="contextEntry">
    <value>
      objectClass: top
      objectClass: domain
      objectClass: extensibleObject
      dc: example
    </value>
  </property>
</bean>
```

Now we will update the elements step by step. First change the id to the value added to the contextPartitionConfiguration element.

Before:

```
<bean id="examplePartitionConfiguration"
class="org.apache.directory.server.core.partition.impl.btree.MutableBTreePartitionConfiguration">
```

After:

```
<bean id="sevenSeasPartitionConfiguration"
class="org.apache.directory.server.core.partition.impl.btree.MutableBTreePartitionConfiguration">
```

Next give the partition a name and change the suffix to o=sevenSeas

Before:

```
<property name="name" value="system" />
<property name="cacheSize" value="100" />
<property name="suffix" value="ou=system" />
<property name="optimizerEnabled" value="true" />
<property name="synchOnWrite" value="true" />
```

After:

```
<property name="name" value="The seven seas" />
<property name="cacheSize" value="100" />
<property name="suffix" value="o=sevenSeas" />
<property name="optimizerEnabled" value="true" />
<property name="synchOnWrite" value="true" />
```

Configuration of the indexed Attributes is described at [Configure indices](#).

The last property remaining now is the context entry. The objectclasses top and extensibleObject are universal hence they remain. But the objectclass domain is replaced by the objectclass organization, because our partition shouldn't represent a domain but an organization.

Before:

```
<property name="contextEntry">
  <value>
    objectClass: top
    objectClass: domain
    objectClass: extensibleObject
    dc: example
  </value>
</property>
```

After:

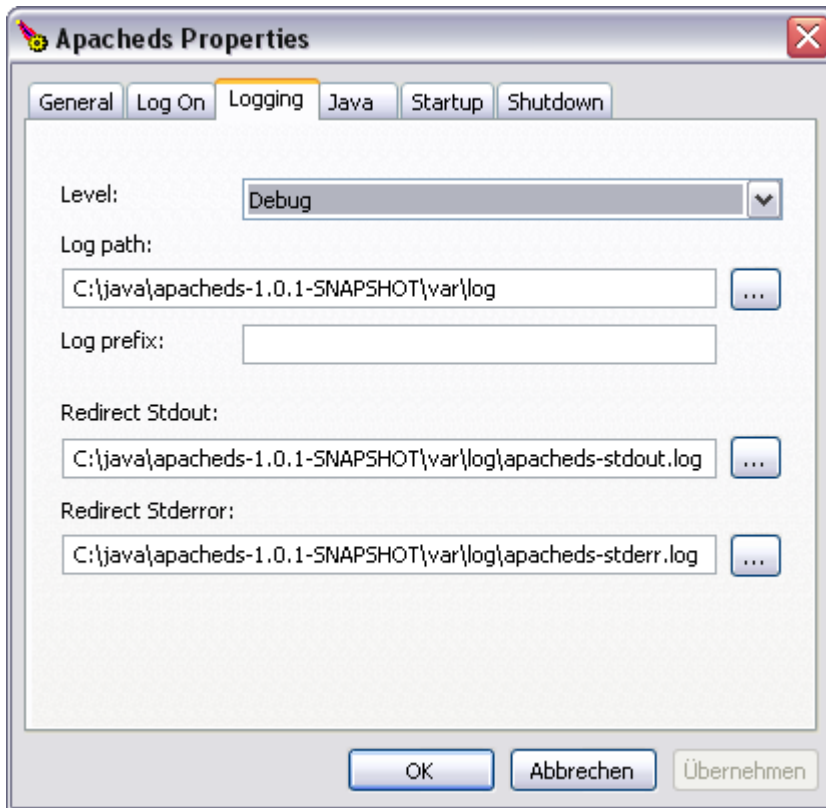
```
<property name="contextEntry">
  <value>
    objectClass: top
    objectClass: organization
    objectClass: extensibleObject
    o: sevenSeas
  </value>
</property>
```

Afterwards restart the server.

Configure logging

ApacheDS 1.0 uses [NLOG4j](#) as its logging solution. This is a simple facade for various logging APIs. The default for ApacheDS 1.0 is [log4j](#), detailed properties can be configured by modifying the file `<APACHDS_HOME>/conf/log4j.properties`.

After installation on Windows, you have the option to configure the ApacheDS Windows Service (you can do this later as well). If you do so, one option pane is dedicated to logging:



You can adjust the logging level and a log path. Note that this is for the daemon only. The server itself is configured as described below.

Default behaviour after installation of ApacheDS 1.0

By default, ApacheDS writes log files in the directory `<APACHDS_HOME>/var/log/`. Besides stdout, a [RollingFileAppender](#) is used to collect warnings and errors. It backups the log files when they reach a certain size.

Here is what the default configuration file `log4j.properties` looks like. The name of the *RollingFileAppender* is "R":

```
log4j.rootCategory=WARN, stdout, R

log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout

log4j.appender.R=org.apache.log4j.RollingFileAppender
log4j.appender.R.File=apacheds-rolling.log
```



```
log4j.appender.R.MaxFileSize=1024KB
# Keep some backup files
log4j.appender.R.MaxBackupIndex=5

log4j.appender.R.layout=org.apache.log4j.PatternLayout
log4j.appender.R.layout.ConversionPattern=[%d{HH:mm:ss}] %p [%c] - %m%n

log4j.appender.stdout.layout.ConversionPattern=[%d{HH:mm:ss}] %p [%c] - %m%n

# with these we'll not get innundated when switching to DEBUG
log4j.logger.org.apache.directory.shared.ldap.name=WARN
log4j.logger.org.springframework=WARN
log4j.logger.org.apache.directory.shared.codec=WARN
log4j.logger.org.apache.directory.shared.asn1=WARN
```

In this file "R" is configured like this:

Property name	Value in file above	Meaning
File	apacheds-rolling.log	path to the output log file, in our case relative to <i>var/log</i>
MaxFileSize	1024KB	maximum size that the output file is allowed to reach before being rolled over to backup files
MaxBackupIndex	5	number of backup files kept
layout.ConversionPattern	[%d{HH:mm:ss}] %p [%c] - %m%n	format string for logging events

If the default logging does not meet your requirements, you can easily adjust the configuration to your needs.

Adjusting logging to your needs

Log file location (where the log files are placed)

By default the log files are placed at **<apacheds_home>/var/log**, but that can be changed.

Linux/MacOS/Solaris

On this systems the location of the log files is configured via an entry in **/bin/server.init**. Look for the following lines and change it to your preferences:

```
$DAEMON_HOME/apacheds \
...
-outfile $SERVER_HOME/var/log/apacheds-stdout.log \
-errfile $SERVER_HOME/var/log/apacheds-stderr.log \
...
$APACHEDS_HOME start
```

Windows

On Windows you can use the configuration wizard for the service as shown in the screenshot above. To

adjust the log path you have to adjust the values of **Redirect Stdout** and **Redirect Stderror**

Log level (how detailed the logs are)

The following log levels from log4j are used for messages within ApacheDS:

Level	Description from log4j documentation
DEBUG	designates fine-grained informational events that are most useful to debug an application
INFO	designates informational messages that highlight the progress of the application at coarse-grained level
WARN	designates potentially harmful situations
ERROR	designates error events that might still allow the application to continue running
FATAL	designates very severe error events that will presumably lead the application to abort

The default (global) log level in the configuration is *WARN*. All messages of level *WARN* and more severe (*ERROR*, *FATAL*) are written to the rolling log file. The easiest way to get finer log messages is to change it like this

```
log4j.rootCategory=DEBUG, stdout, R
...
```

These detailed log messages took much file space and time and therefore should only be enabled globally in order to analyze problems.

It is possible to configure the logging more fine grained by using categories. Within the default configuration there are some examples:

```
...
# with these we'll not get innundated when switching to DEBUG
log4j.logger.org.apache.directory.shared.ldap.name=WARN
log4j.logger.org.springframework=WARN
log4j.logger.org.apache.directory.shared.codec=WARN
log4j.logger.org.apache.directory.shared.asn1=WARN
```

If the global level is switched to *DEBUG*, these definitions override the setting with *WARN* for certain areas and therefore keep the file a little bit smaller. Learn more about the concept of categories in the [Short introduction to log4j](#).

Format for log messages

The format of each line within a log file is controlled by a pattern. For the *RollingFileAppender* in the default configuration it looks like this

```
...
log4j.appender.R.layout=org.apache.log4j.PatternLayout
log4j.appender.R.layout.ConversionPattern=[%d{HH:mm:ss}] %p [%c] - %m%n
...
```

Some examples lines within the log file, formatted with the pattern "[%d{HH:mm:ss}] %p [%c] - %m%n" are:

```
...
[12:29:03] WARN [org.apache.directory.server.core.DefaultDirectoryService]
- You didn't change the admin password of directory service instance 'default'.
Please update the admin password as soon as possible to prevent a possible security breach.
...
[12:29:05] INFO [org.apache.directory.server.jndi.ServerContextFactory]
- Successful bind of an LDAP Service (636) is complete.
[12:29:05] INFO [org.apache.directory.server.Service] - server: started in 6750 milliseconds
...
```

The pattern uses the following conversion characters:

Character	Outputs
%d	date of the logging event in the given format. like "12:29:05" for %d{HH:mm:ss}
%p	priority (level) of the logging event, like "INFO" or "WARN"
%c	category of the logging event, like "org.apache.directory.server.Service"
%m	application supplied message associated with the logging event
%n	platform dependent line separator

The [javadoc of log4j](#) contains a table with all valid %-characters and their meaning.

Simple adjust the pattern in the *log4j.properties* file to get the log format of your choice, for instance

```
log4j.appender.R.layout.ConversionPattern=[%d{dd.MM.yyyy HH:mm:ss}] %p: %c{1}.%M() - %m%n
```

leads to messages of this form:

```
...
[29.12.2006 13:50:44] INFO: ServerContextFactory.startLDAP0() - Successful bind of an LDAP
Service (636) is complete.
[29.12.2006 13:50:44] INFO: Service.init() - server: started in 3016 milliseconds
...
```



Warning

"Generating caller location information like with %M or %L is extremely slow. Its use should be avoided unless execution speed is not an issue." (from the log4j documentation)

Advanced log4j configuration

You can take advantage of other features of log4j as well, such as other appenders like the daily rolling file appender. And you can configure logging to make it easier for you to view the messages with tools like Log Factor 5 or [Chansaw](#).

Learn more about log4j and related tools at its [homepage](#).

Enable/disable anonymous access

Anonymous access to the server is disabled by default. All clients have to provide their name (distinguished name) and password in order to bind to the directory service. If you use the server standalone configured with a *server.xml* file, you can enable anonymous binds by changing the value for property *allowAnonymousAccess* in the Spring bean definition for bean *configuration*, as depicted in the following fragment:

```
<bean id="configuration"
class="org.apache.directory.server.configuration.MutableServerStartupConfiguration">
    ...
    <property name="allowAnonymousAccess"><value>true</value></property>
    ...
</bean>
```

A restart of the server is necessary for this change to take effect. Learn more about authentication option in the corresponding section of this guide [here](#).

Configure indices

(-> <http://cwiki.apache.org/confluence/display/DIRxSRVx10/Performance+Tuning>)
(-> <http://cwiki.apache.org/confluence/display/DIRxSRVx11/Backend>)

(Re)Building Indices

Sometimes after loading a database you want to index another attribute but forgot to set up the index configuration for that attribute. Let's presume you have a partition called "example" with 5 million inetOrgPersons in it. All of a sudden your application starts conducting searches for the state (the st attribute) of the user. Reloading this partition after adding the index configuration for the st attribute is out of the question.

Luckily building the st index is pretty easy to do using the apacheds-tools supplied with an ApacheDS binary installation. Here's the command you have to execute on your system to build the st attribute index for the "example" partition:

```
java -jar apacheds-tools index -i <installation_directory> -p example -a st
```

Handling of the index operation is described in detail at [ApacheDS tools](#). The index operation basically builds a new index on the st attribute within the "example" partition. This new index will **not** be utilized until the index for the st attribute is also configured within the server.xml file. To make use of the index add the following index configuration to the "example" partition:

```
<bean class="org.apache.directory.server.core.partition.impl.btree.MutableIndexConfiguration">
  <property name="attributeId"><value>st</value></property>
  <property name="cacheSize"><value>100</value></property>
</bean>
```

Although this can be performed while the server is online without corrupting the partition it is recommended that you run this command offline. This way new entries cannot be added while you're building the index.

Memory Allocation

By default the initial and the maximum heap size for the JVM are set to **384 MB**. To change this settings in **Linux** open the file **/bin/server.init** and look for the following lines:

```
#
# Start apacheds
#
...
$DAEMON_HOME/apacheds \
...
-Xms384m -Xmx384m \
...
$APACHEDS_HOME start
```

Change the values at -Xms384m -Xmx384m to your preferences. Xms is the initial heap size, Xmx is the maximum heap size. It's recommendable to use not less than **128 MB** for the initial heap size.

In **Windows** you can easily change the settings using the Procrun Service Manager. Open the **Service Settings** and go to the Java tab.

!memory_windows.png!

Resources

- [Spring Framework 1.2.x Reference Documentation](#)
- [Short introduction to log4j](#)

Command line tools

This page last changed on Dec 20, 2006 by ck.

<< Previous: Graphical tools	ApacheDS v1.0 Basic User's Guide (TOC)	Next: Connecting to ApacheDS with Java components >>
--	--	--

An alternative to UI tools for connecting to your directory and perform operations are command line tools. The traditional commands are part of many applications (for instance Lotus notes, many LDAP servers) and even operating systems (e.g. Sun Solaris 8 ff.). The following table lists the names and functions of common commands. All of them open a connection to an LDAP server, bind, and perform one or more LDAP operations.

Command	short description from man page
ldapsearch	Performs a search using specified parameters.
ldapmodify and ldapadd	Modifies or adds entries. When invoked as ldapadd the -a (add new entry) flag is turned on automatically.
ldapmodrdn	Modifies the RDN of entries.
ldapdelete	Deletes one or more entries.

Open a shell and type "ldapsearch" to see whether these tools are already available on your system. This may be true on UNIX systems, or LINUX systems (with OpenLDAP client tools installed). If not (especially if you are using Windows, this is probably the case), you have different options to get such tools. One is to download the [Sun ONE Directory SDK for C](#), which is available for many platforms (among them Windows). It also contains executables of the command line tools (ldapsearch etc.).

Here is an example for a search command, which displays the o=sevenSeas entry of our tutorial partition. You will learn more about LDAP searches later on.

```
$ ldapsearch -h zanzibar -p 10389 -b "o=sevenSeas" -D "uid=admin,ou=system" -w *****
"(objectClass=*)"
o=sevenSeas
description=Contains Apache Directory Tutorial example data
objectClass=organization
objectClass=top
o=sevenSeas
$
```

One big advantage of command line tools is that you can use them within scripts. It is also much easier if you have to document changes to the directory (configuration, for instance). Therefore administrators like them a lot. We will use them within this tutorial as well, but always as an alternative to UI tools (which LDAP newbies normally prefer).



Not all command line tools are equal

Although the command line tools of different operating systems and LDAP clients (shipped with LDAP servers) normally have the same name, there are often differences in the command line

options. If any problems arise if you try out examples from this tutorial (e.g. "illegal option"), consult the man pages or documentation of your tools.

Import sample data using a command line tool

Here is an example usage of `ldapmodify`.

```
$ ldapmodify -h zanzibar -p 10389 -D "uid=admin,ou=system" -w ***** -a -f
apache_ds_tutorial.ldif
adding new entry ou=people,o=sevenSeas
adding new entry ou=groups,o=sevenSeas
adding new entry ou=crews,ou=groups,o=sevenSeas
adding new entry ou=ranks,ou=groups,o=sevenSeas
adding new entry cn=Horatio Hornblower,ou=people,o=sevenSeas
...
adding new entry cn=John Fryer,ou=people,o=sevenSeas
adding new entry cn=John Hallett,ou=people,o=sevenSeas
adding new entry cn=HMS Bounty,ou=crews,ou=groups,o=sevenSeas
$
```

The following table contains descriptions for the options used. See the [manpage of ldapmodify](#) for details.

Option	Meaning
-h magritte	Hostname
-p 10389	Port
-D "uid=admin,ou=system"	Distinguished name to bind (user with appropriate privileges needed)
-w *****	Password of bind user
-a	add new entries
-f apache_ds_tutorial.ldif	Name of LDIF file to load

The following operation demonstrates that your directory now contains the sample data. It searches for all entries below `o=sevenSeas` (`-b` = search base, `-s` = search scope), which have an attribute occurrence of `givenName` with value "William". The output contains the distinguished names (dn) of the result entries and their common name (cn) values.

```
$ ldapsearch -h zanzibar -p 10389 -D "uid=admin,ou=system" -w ***** -b "o=sevenSeas" -s sub
"(givenName=William)" cn
version: 1
dn: cn=William Bligh,ou=people,o=sevenSeas
cn: William Bligh

dn: cn=William Bush,ou=people,o=sevenSeas
cn: William Bush
$
```

Connecting to ApacheDS with graphical tools (3rd party)

This page last changed on Jan 11, 2007 by ck.

<< Previous:ApacheDS tools	ApacheDS v1.0 Basic User's Guide (TOC)	Next:Command line tools >>
--	--	--

This sections describes how to connect tp ApacheDS with some graphical LDAP clients, which are not part of the distribution.

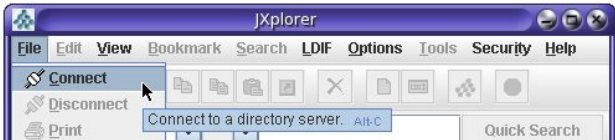
- [Options, Options](#)
- [JXplorer](#)
- [Softerra LDAP Browser](#)
- [Resources](#)

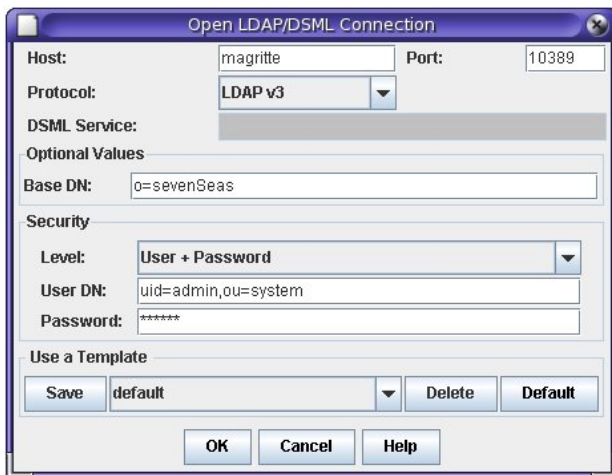
Options, Options

Especially if you are new to LDAP, you might like to use GUI tools to connect to Apache Directory Server. Some LDAP servers come with graphical tools, Apache DS 1.0 does not. But you can use any standard compliant client (free or commercial) to explore your directory. The [Resources](#) paragraph contains a collection of free tools with grapical user interface. In the following the connection to the sample directory is demonstrated for some of them.

JXplorer

JXplorer is an open source LDAP browser written in Java. For its graphical user interface the Swing libraries are used. JXplorer is feature rich and only needs a J2SE 1.4 to run. Native installers are availabe for many platforms as well. Visit the [JXplorer Homepage](#) to learn more about this client and how to download it.

	<p>In order to connect to our sample directory server with this tool, open the "Open LDAP/DSML Connection" dialog after vstarting the browser. This can be done either by selecting the menu item within the File menu (see the screen shot to the left), by pressing Alt+C, or by hitting the corresponding button in the tool bar.</p>
---	--



The dialog to the left appears. It provides input fields for hostname, port and Base DN. You may select "Anonymous" via the security level drop down. If you decide to do so it is not necessary to provide credentials afterwards, but the server must be configured to allow you to bind like that. If not, you won't be able to browse the data stored in the "o=sevenSeas" partition.

Below the dropdown you provide credentials for the user to connect, if you have decided not to connect anonymously to the directory. Because currently no user is imported to your directory, you have to use the administrator to authenticate. Enter the DN of this user "uid=admin,ou=system" and the corresponding password.



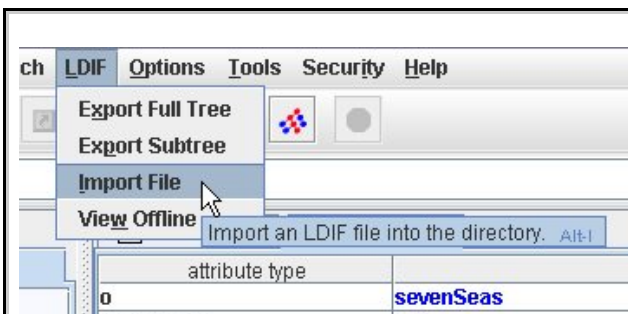
Warning

Using this configuration with a so called "simple bind" causes JXplorer to send the credentials (User DN, password) unencrypted to the server. Look at the Security sections if you want to learn more about how to secure credentials.

You might want to save the connection data as a template. Pressing the OK button causes JXplorer to connect to the server.

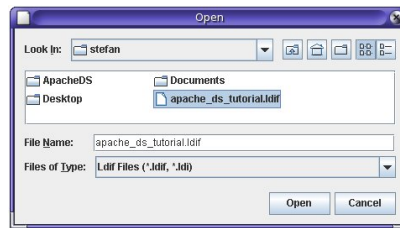
Import sample data using JXplorer

In order to load the sample data to ApacheDS with JXplorer, you have to connect to the server first. Use uid=admin,ou=system to connect (you need write access).

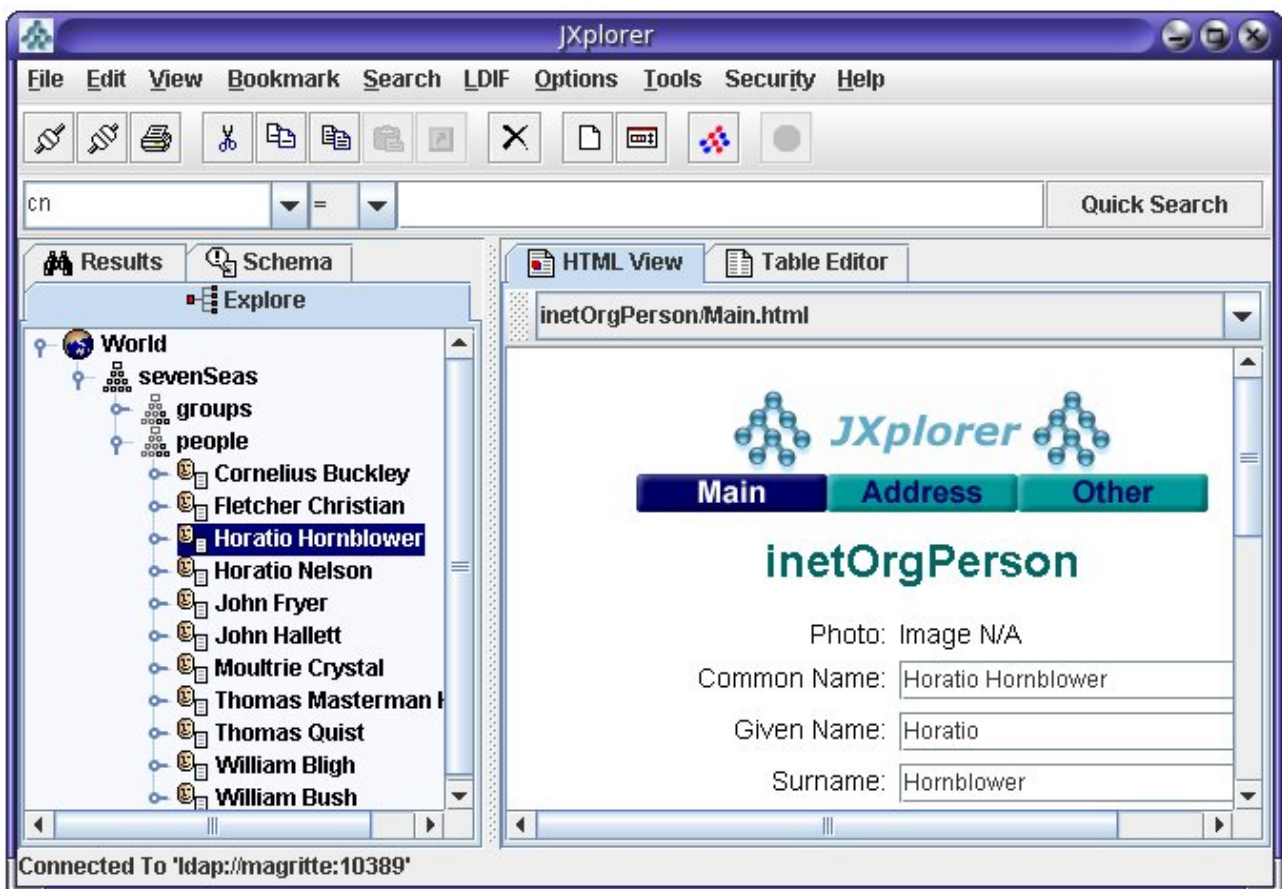


After you successfully connect to your server with JXplorer, use the menu entry *LDIF | Import File*.

Browse the file system with the Open dialog provided by JXplorer (see image) and select the file with the [sample data](#).



Press the Open button. JXplorer reads the LDIF file and performs some LDAP add operations against your server in order to create the entries within the sample partition. After a successful import, you can immediately browse the directory with JXplorer and see the imported entries:



Softerra LDAP Browser

[Softerra LDAP Browser](#) is a lightweight version of the commercial [Softerra LDAP Administrator](#). It is available for free. Functionality of the browser is limited to exploring directories and schemas, searching, and exporting of directory data to disk. Entry creation and modification is reserved to LDAP Administrator only, together with advanced functionality like group management. Check the vendor's Homepage for details about both offerings.

Softerra platform support

Note that the Softerra clients (Browser and Administrator) are available on Windows platforms only.



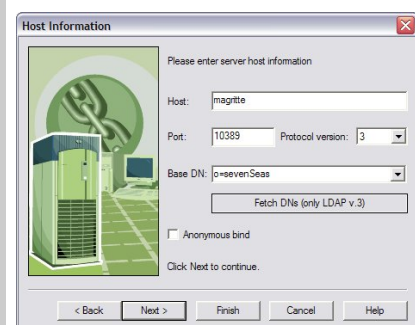
In order to connect to our sample directory server with this tool, start the "Server Connection Profile Wizard". This can be done either by selecting the menu item within the File menu (see the screen shot to the left), by pressing Strg+N, or by hitting the corresponding button in the tool bar. The following dialog appears.



Enter an appropriate profile name (we chose "Seven Seas") and press the "Next" button to continue.

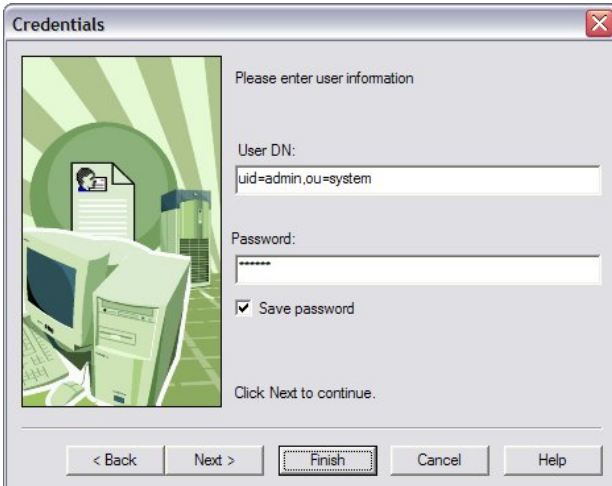
The next wizard step provides input fields for hostname, port and Base DN. The Button "Fetch DNs" allows you to fill the Base DN dropdown with the suffix names available, if the LDAP server supports it. So does Apache Directory Server, hence "o=sevenSeas" should appear in the list after pressing the button.

You may select "Anonymous bind" via the according check box. If checked it is not necessary to provide credentials afterwards, but the server must be configured to allow you to bind like that. If not, you won't be able to browse the data stored in the "o=sevenSeas"



partition.

After entering the parameters for your directory press the "Next" button to continue.



In this step you provide credentials for the user to connect, if you decided not to connect anonymously to the directory. Because currently no user is imported to your directory, you have to use the administrator to authenticate. Enter the DN of this user "uid=admin,ou=system" and the corresponding password.

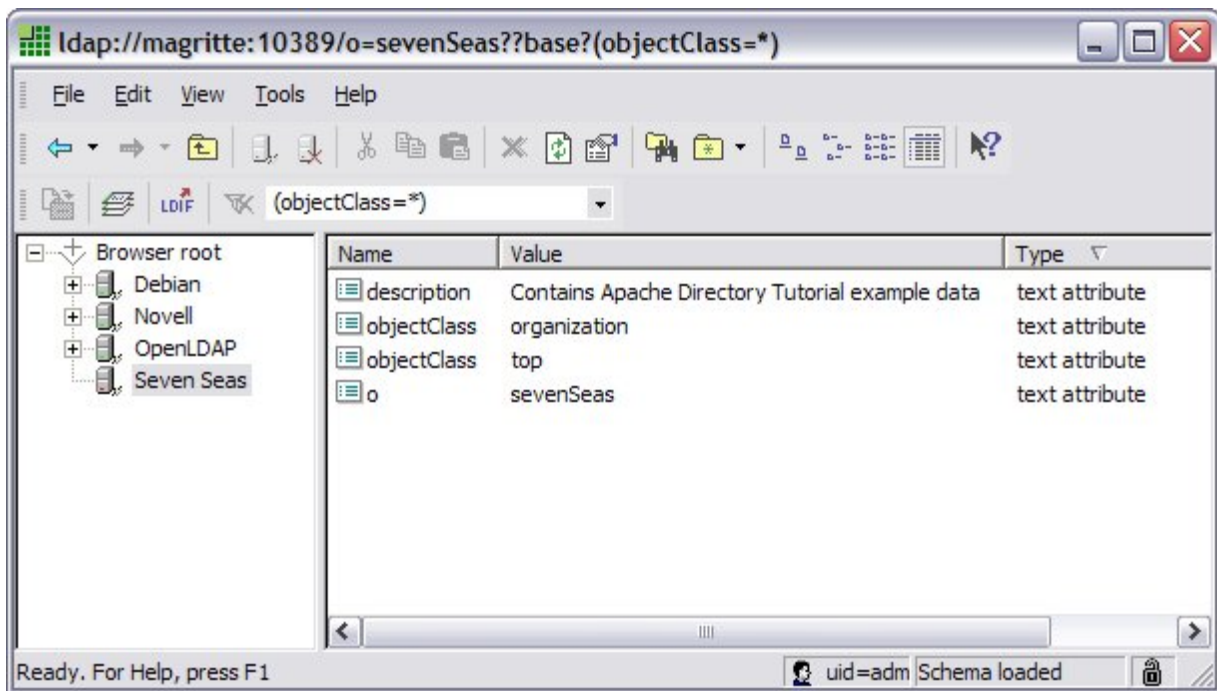


Warning

Using this configuration with a so called "simple bind" causes the client to send the credentials (User DN, password) unencrypted to the server. Look at the security sections if you want to learn more about how to secure credentials.

If you hit the "Next" button, a next step wizard the wizard offers some advanced LDAP settings. We decided to accept the defaults here and therefore pressed "Finish" to create the connection profile.

Done. You should now be able to see the entry provided by the "o=sevenSeas" suffix of the tutorial partition:



Softerra LDAP Browser does not allow LDIF imports (it does not allow adding entries to the directory at all), but Softerra LDAP Administrator does (Menu: *File | Import Data...*).

Resources

- [JXplorer](#)
- [LDAP Browser/Editor](#) by Jarek Gawor
- [Softerra LDAP Browser](#)

Connecting to ApacheDS with Java components

This page last changed on Dec 20, 2006 by [ck](#).

<< Previous: Command line tools	ApacheDS v1.0 Basic User's Guide (TOC)	Next: Other programming languages >>
---	--	--

- [Which option to choose?](#)
- [Resources](#)

Which option to choose?

JNDI

A natural choice for LDAP integration in Java components is [JNDI](#), the Java Naming and Directory Interface. It has been part of the J2SE since version 1.3, and it is available as a separate download for even more ancient JDKs.

JNDI offers a common interface to connect to different naming and directory services, including DNS, file systems, CORBA naming services and LDAP. Therefore it represents an abstraction layer, which sometimes causes trouble if special LDAP features have to be exploited.

"Native" LDAP libraries for Java

On the other hand several libraries exist which take advantage of the basic Java network functionality directly. The interfaces are organized the LDAP protocol operations, which leads to an LDAP programming experience without abstraction. Examples for pure Java libraries like that are

- [Netscape Directory SDK for Java](#)
- [JLDAP](#), derived from Novell Developer Kit

But because JNDI is present in practically every Java runtime environment, these libraries are not widely used any longer, although they have some advantages versus JNDI.

DSML

TBD

A first example with JNDI

```

import java.util.Hashtable;

import javax.naming.Context;
import javax.naming.NamingEnumeration;
import javax.naming.NamingException;
import javax.naming.directory.Attributes;
import javax.naming.directory.InitialDirContext;

public class SimpleJndiExample {

    public static void main(String[] args) throws NamingException {

        Hashtable env = new Hashtable();

        env.put(Context.INITIAL_CONTEXT_FACTORY, "com.sun.jndi.ldap.LdapCtxFactory");
        env.put(Context.PROVIDER_URL, "ldap://zanzibar:10389/o=sevenSeas");

        env.put(Context.SECURITY_PRINCIPAL, "uid=admin,ou=system");
        env.put(Context.SECURITY_CREDENTIALS, "secret");
        env.put(Context.SECURITY_AUTHENTICATION, "simple");

        InitialDirContext ctx = new InitialDirContext(env);

        Attributes attrs = ctx.getAttributes("");
        NamingEnumeration enm = attrs.getAll();
        while (enm.hasMore()) {
            System.out.println(enm.next());
        }
    }
}

```

Output:

```

$ java SimpleJndiExample
objectClass: organization, top
description: Contains Apache Directory Tutorial example data
o: sevenSeas
$

```

An excellent resource to learn more about JNDI is the [JNDI Tutorial](#) at Sun Microsystems.

Resources

- [JNDI Tutorial](#) at Sun Microsystems

Custom application development

This page last changed on Dec 02, 2006 by [szoerner](#).

- [Java](#)
- [Other programming languages](#)
- [Resources](#)

Java

A natural choice for LDAP integration in Java components is [JNDI](#), the Java Naming and Directory Interface. It has been part of the J2SE since version 1.3, and it is available as a separate download for even more ancient JDKs.

JNDI offers a common interface to connect to different naming and directory services, including DNS, file systems, CORBA naming services and LDAP. Therefore it represents an abstraction layer, which sometimes causes trouble if special LDAP features have to be exploited.

Several libraries exist which take advantage of the basic Java network functionality directly. The interfaces are organized the LDAP protocol operations, which leads to an LDAP programming experience without abstraction. Examples for pure Java libraries like that are

- [Netscape Directory SDK for Java](#)
- [JLDAP](#), derived from Novell Developer Kit

But because JNDI is present in practically every Java runtime environment, these libraries are not widely used any longer, although they have some advantages versus JNDI.

A simple JNDI example

```
import java.util.Hashtable;

import javax.naming.Context;
import javax.naming.NamingEnumeration;
import javax.naming.NamingException;
import javax.naming.directory.Attributes;
import javax.naming.directory.InitialDirContext;

public class SimpleJndiExample {

    public static void main(String[] args) throws NamingException {

        Hashtable env = new Hashtable();

        env.put(Context.INITIAL_CONTEXT_FACTORY, "com.sun.jndi.ldap.LdapCtxFactory");
        env.put(Context.PROVIDER_URL, "ldap://zanzibar:10389/o=sevenSeas");

        env.put(Context.SECURITY_PRINCIPAL, "uid=admin,ou=system");
        env.put(Context.SECURITY_CREDENTIALS, "secret");
        env.put(Context.SECURITY_AUTHENTICATION, "simple");

        InitialDirContext ctx = new InitialDirContext(env);
```



```
Attributes attrs = ctx.getAttributes("");
NamingEnumeration enm = attrs.getAll();
while (enm.hasMore()) {
    System.out.println(enm.next());
}
}
```

Output:

```
$ java SimpleJndiExample
objectClass: organization, top
description: Contains Apache Directory Tutorial example data
o: sevenSeas
$
```

An excellent resource to learn more about JNDI is the [JNDI Tutorial](#) at Sun Microsystems.

Other programming languages

If you connect to an LDAP server from a program, you normally use a library which encapsulates the LDAP calls. It depends on your programming language and platform, which options you have here. LDAP is widely adopted, so there should be at least one solution available for you. Here are some examples:

Language	Library
C, C++	An RFC exists for LDAP C APIs, RFC 1823 , although different implementation vary. The OpenLDAP project offers a library for many platforms.
Perl	Perl LDAP

Resources

- [JNDI Tutorial](#) at Sun Microsystems

How to enable SSL

This page last changed on Jan 14, 2007 by [szoerner](#).

<< Previous: Basic authorization	ApacheDS v1.0 Basic User's Guide (TOC)	
--	--	--

This section describes the transport layer security options for LDAP, and especially how to enable LDAPS on ApacheDS.

- [Transport layer security and LDAP](#)
- [Server configuration](#)
- [Verification, Clients](#)
- [Resources](#)

Transport layer security and LDAP

Several requirements related to security can be easily accomplished with the help of **SSL** technology (Secure Socket Layer) or its standardized successor **TLS** (Transport Layer Security, RFC 2246). Among these are the protection of data against eavesdropping and modification, when on transit between client and server (data integrity), and the authentication of a server toward a client with the help of a certificate.

There are two approaches to utilize these technologies in the LDAP world.

1. Idaps (LDAP over SSL/TLS, port 636)
2. StartTLS (extended operation)

The first option is comparable to HTTPS and inserts an SSL/TLS layer between the TCP/IP protocol and LDAP. Establishing a connection like this is normally provided via a different server port (port 636 is common, it is a well-known port, like port 389 is for LDAP). In URIs the schema "Idaps" is specified (for instance *ldaps://zanzibar:636/*) instead of "ldap". It is possible to write programs which switch between ldap and ldaps without changes in the source, if the connection data is configured external.

In the second option a client establishes at first a "normal" LDAP connection. With a special request (extended operation StartTLS) it tries to switch to secure communication afterwards. It is not necessary to change the port for this, the communication continues on the established connection. The client may go back to the original connection state ("TLS Closure Alert"), in doing so protecting only selected parts of the communication.

Both ways to utilize SSL/TLS within LDAP require the configuration of the server with an appropriate certificate.

Server configuration

ApacheDS 1.0 supports only the first option (ldaps) and only if it runs with JDK 1.5 or above. The feature is disabled by default, you need to configure it. There are some steps to follow in order to obtain a SSL enabled server.

Of capital importance: You need a **certificate** for your server. A certificate is signed public key (signed normally by a third party, a certificate authority, CA).

There are different options

- either you buy a certificate from a Certificate Authority (like Verisign, etc.), or you obtain one from your enterprise CA, if available
- or you ask for a free certificate from [CACERT organisation](#)
- or you create your own certificate, self-signed or signed by your private CA, which will not be trusted.

We will do it the last way (self-signed), primarily because it's easy and fast (you won't have to pay nor to wait to obtain your certificate)

Key creation

First it is necessary to create a key pair (public/private key) for your server, *zanzibar* in our case. One option is to use the JDK tool *keytool* for this task. In the following example, we use these options

Option	value	Description
-genkey		command to generate a key pair
-keyalg	"RSA"	algorithm to be used to generate the key pair, in our case, default is "DSA"
-dname	"cn=zanzibar, ou=ApacheDS, o=ASF, c=US"	the X.500 Distinguished Name to be associated with alias, used as the issuer and subject fields in the self-signed certificate
-alias	zanzibar	name to refer the entry within the keystore
-keystore	zanzibar.ks	keystore file location
-storepass	secret	password used to protect the integrity of the keystore
-validity	730	number of days for which the certificate should be considered valid, default is 90

Learn more about keytool at the [manpage](#).

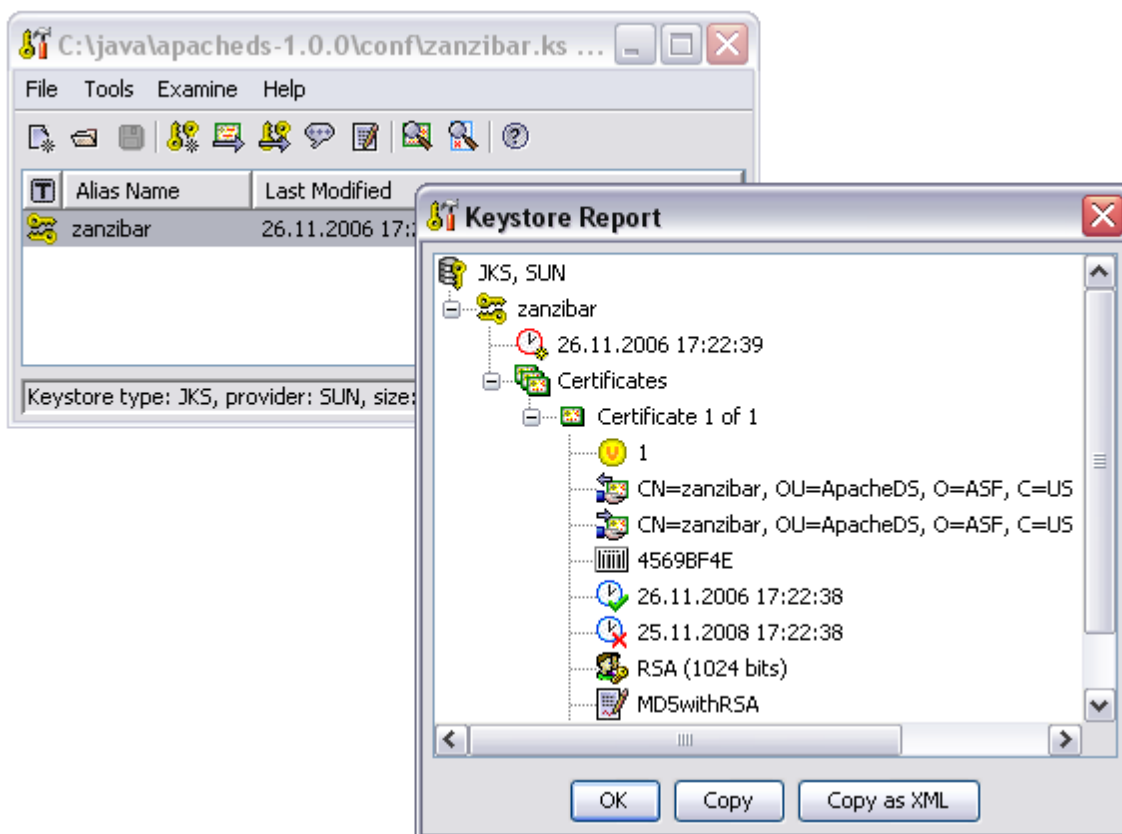
```
$ keytool -genkey -keyalg "RSA" -dname "cn=zanzibar, ou=ApacheDS, o=ASF, c=US" -alias zanzibar
-keystore zanzibar.ks -storepass secret -validity 730
Enter key password for <zanzibar>
(RETURN if same as keystore password):
$ ls -l
total 4
-rw-r--r-- 1 stefan users 1275 Nov 26 17:22 zanzibar.ks
$ keytool -list -keystore zanzibar.ks
Enter keystore password: secret

Keystore type: jks
Keystore provider: SUN

Your keystore contains 1 entry

zanzibar, Nov 26, 2006, keyEntry,
Certificate fingerprint (MD5): 28:B8:F9:41:94:DB:86:F1:9F:77:13:77:45:27:6A:89
$
```

Another option is to use graphical tools for key creation like [Portecle](#), which is basically a user-friendly front-end for keytool with comparable functionality. For a first impression see a screenshot below.



Configuring ApacheDS

Enabling SSL in Apache Directory Server and using the key pair created as above is quite easy. Simply put the keystore file in the *conf* directory of ApacheDS, and enable *ldaps*. Here is the fragment from *server.xml* on how to do so.

```
<bean id="configuration"
class="org.apache.directory.server.configuration.MutableServerStartupConfiguration">
```

```

...

<!-- SSL properties -->
<property name="enableLdaps" value="true" />
<property name="ldapsPort" value="10636" />
<property name="ldapsCertificateFile" value="C:/java/apacheds-1.0.0/conf/zanzibar.ks" />
<property name="ldapsCertificatePassword" value="secret" />

...

```

The following properties were used

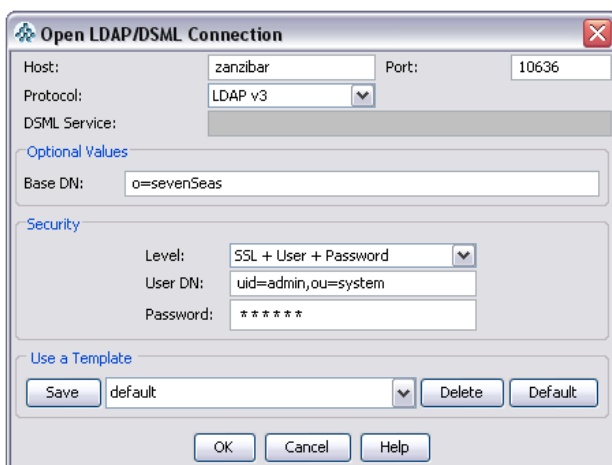
Property	default value	Description
enableLdaps	false	sets if LDAPS is enabled or not
ldapsPort	636	LDAPS TCP/IP port number to listen to
ldapsCertificateFile	<WORKDIR>/certificates/server.cer	path of the X509 (or JKS) certificate file for LDAPS
ldapsCertificatePassword	changeit	password which is used to load the LDAPS certificate file

After modification of the *server.xml*, the server has to be restarted in order to take effect.

Verification, Clients

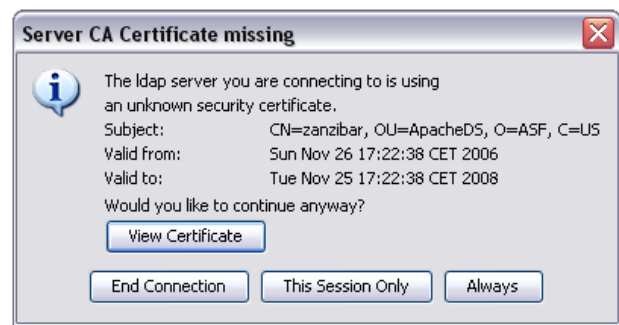
After restarting the server, you should have server offering both ldap and ldaps. How to verify whether it works?

Using JXplorer to connect



JXplorer happily supports ldaps connections. Enter the connection data (hostname and port) and select SSL from the dropdown (different options here, we choose to provide user and password). Other options/parameters as usual.

Because our self-signed certificate is not trustworthy, JXplorer will present a warning. This is a good sign – ApacheDS with SSL obviously works. You can view the certificate, and decide to continue (accepting the certificate always or this session only)



Afterwards the connection behaves like LDAP does. No difference in functionality, but the transmission is secured by SSL.

Other clients, Java programs using JNDI

If you use other graphical clients, the behaviour will be comparable. Sometimes clients don't allow to connect to a server, if the certificate is not trustworthy. This is for instance the case for Java clients using JNDI.

The following simple Java program tries to connect via JNDI/JSSE (Java Secure Socket Extension) and LDAPS to `ldaps://zanzibar:10636`

```
import java.util.Hashtable;
import javax.naming.*;
import javax.naming.directory.*;

public class ConnectWithLdaps {

    public static void main(String[] args) throws NamingException {

        Hashtable env = new Hashtable();

        // Simple bind
        env.put(Context.SECURITY_AUTHENTICATION, "simple");
        env.put(Context.SECURITY_PRINCIPAL,
            "cn=Horatio Hornblower,ou=people,o=sevenSeas");
        env.put(Context.SECURITY_CREDENTIALS, "pass");

        env.put(Context.INITIAL_CONTEXT_FACTORY,
            "com.sun.jndi.ldap.LdapCtxFactory");
        env.put(Context.PROVIDER_URL, "ldaps://zanzibar:10636/o=sevenSeas");

        DirContext ctx = new InitialDirContext(env);
        NamingEnumeration enm = ctx.list("");
        while (enm.hasMore()) {
            System.out.println(enm.next());
        }
        ctx.close();
    }
}
```

It causes a *CommunicationException*, if the certificate is not trusted:

```
$ java ConnectWithLdaps
Exception in thread "main" javax.naming.CommunicationException:
    simple bind failed: zanzibar:10636
    [Root exception is javax.net.ssl.SSLHandshakeException:
```

```
sun.security.validator.ValidatorException: PKIX path building failed:
sun.security.provider.certpath.SunCertPathBuilderException:
unable to find valid certification path to requested target]
at com.sun.jndi.ldap.LdapClient.authenticate(Unknown Source)
...
```

In order to make the client trust our server, one option is to share a self signed certificate. So we export the certificate (DER format) using *keytool* like this:

```
$ keytool -export -keystore zanzibar.ks -alias zanzibar -file zanzibar.cer
Enter keystore password: secret
Certificate stored in file <zanzibar.cer>
$ ls -l
total 6
-rw-r--r--  1 stefan  users      504 Dec  1 21:51 zanzibar.cer
-rw-r--r--  1 stefan  users     1275 Nov 26 17:22 zanzibar.ks
$
```

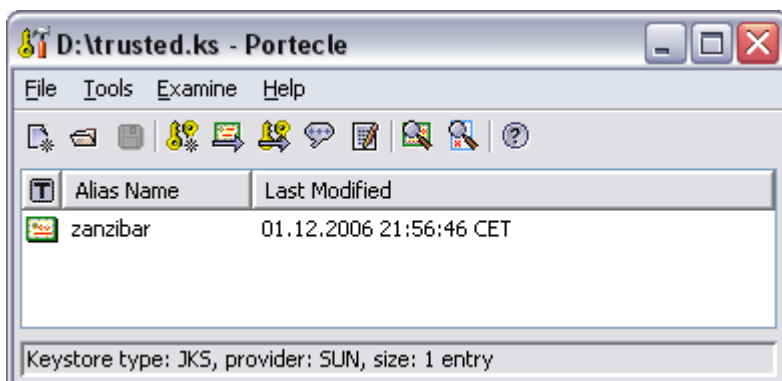
Please note that you don't want to share the server keystore file itself with arbitrary clients, because it holds the private key. Instead we create a separate keystore *trusted.ks* with the help of *keytool*. We import the certificate *zanzibar.cer* like this:

```
$ keytool -import -file zanzibar.cer -alias zanzibar -keystore trusted.ks -storepass secret
Owner: CN=zanzibar, OU=ApacheDS, O=ASF, C=US
Issuer: CN=zanzibar, OU=ApacheDS, O=ASF, C=US
Serial number: 4569bf4e
Valid from: Sun Nov 26 17:22:38 CET 2006 until: Tue Nov 25 17:22:38 CET 2008
Certificate fingerprints:
    MD5:  28:B8:F9:41:94:DB:86:F1:9F:77:13:77:45:27:6A:89
    SHA1: 76:BC:7C:E6:11:57:F4:86:A5:6C:A1:F7:C1:CF:1A:1E:6C:4A:15:BC
Trust this certificate? [no]: yes
Certificate was added to keystore
$ keytool -list -keystore trusted.ks -storepass secret
Keystore type: jks
Keystore provider: SUN

Your keystore contains 1 entry

zanzibar, Dec 1, 2006, trustedCertEntry,
Certificate fingerprint (MD5): 28:B8:F9:41:94:DB:86:F1:9F:77:13:77:45:27:6A:89
$
```

Instead of using the command line version of *keytool*, it is also possible to perform the certificate export and import operations with Portecle or any other graphical frontend. This is for instance how the *trusted.ks* files with the imported certificate looks like in Portecle.



Clients may use this keystore in order to connect to the server. Therefore they can configure *trusted.ks* as the trusted store via the environment like this:

```
$ java -Djavax.net.ssl.trustStore=trusted.ks ConnectWithLdaps
ou=people: javax.naming.directory.DirContext
ou=groups: javax.naming.directory.DirContext
```

Another option would be to import the certificate in the default keystore of the JRE installation (within \$JAVA_HOME/jre/lib/security). For a test certificate this proceeding is not appropriate.

Troubleshooting

In practice connection establishment with LDAP over SSL may lead to various problems. In order to eliminate the errors it is helpful to see communication-specific debug information. The system property *javax.net.debug* is available for this task. The value "ssl" provides information about the certificates in the used key store, the server certificate, and the steps during establishing of the SSL connection (handshake):

```
$ java -Djavax.net.ssl.trustStore=trusted.ks -Djavax.net.debug=ssl ConnectWithLdaps
setting up default SSLSocketFactory
use default SunJSSE impl class: com.sun.net.ssl.internal.ssl.SSLSocketFactoryImpl
class com.sun.net.ssl.internal.ssl.SSLSocketFactoryImpl is loaded
keyStore is :
keyStore type is : jks
keyStore provider is :
init keystore
init keymanager of type SunX509
trustStore is: trusted.ks
trustStore type is : jks
trustStore provider is :
init truststore
adding as trusted cert:
  Subject: CN=zanzibar, OU=ApacheDS, O=ASF, C=US
  Issuer:  CN=zanzibar, OU=ApacheDS, O=ASF, C=US
  Algorithm: RSA; Serial number: 0x4569bf4e
  Valid from Sun Nov 26 17:22:38 CET 2006 until Tue Nov 25 17:22:38 CET 2008

init context
trigger seeding of SecureRandom
done seeding SecureRandom
instantiated an instance of class com.sun.net.ssl.internal.ssl.SSLSocketFactoryImpl
%% No cached client session
*** ClientHello, TLSv1
...
```

You should be able to determine any SSL-related configuration problem with the help of this log.

Resources

- [Java Secure Socket Extension \(JSSE\)](#)
- [Portecle](#) a free UI application for creating, managing and examining keystores
- [SSL 3.0 Specification \(Netscape\)](#)

<< Previous:Basic authorization	ApacheDS v1.0 Basic User's Guide (TOC)	
---	--	--

Installing and starting the server

This page last changed on Jan 11, 2007 by ck.

<< Previous: Sample configurations and sample directory data	ApacheDS v1.0 Basic User's Guide (TOC)	Next: Basic configuration tasks >>
--	--	--

This section describes how ApacheDS can be installed and started at different platforms.

- [Prerequisites](#)
- [Downloading the server](#)
- [Installing as Windows service](#)
- [Installing on Linux](#)

Prerequisites

- **Java 5.0.** We recommend using [Sun's JDK](#) , but the server has also been successfully tested with JRockit 5.0 and with IBM Java 5.0.
You can check your java installation with:

```
java -version
```

this should response something like:

```
java version "1.5.0_06"  
Java(TM) 2 Runtime Environment, Standard Edition (build 1.5.0_06-b05)  
Java HotSpot(TM) Client VM (build 1.5.0_06-b05, mixed mode)
```

- using Linux: you must have a X11 graphical interface
- **384 MB RAM** for the JVM. That's the default setting, how to change that is described [here](#)

Downloading the server

There exist installers for different platforms:

- MacOS (jar)
- Solaris (jar)
- Windows (exe)
- Debian package
- RPM package
- Linux (jar)

You can download the installers from [here](#).

Installing as Windows service

To install the ApacheDS as Windows service you need **Administrator** privileges. Installing can be done easily using the Windows installer. Its interface and functionality is similar to other wizard based installers.

Starting and Stopping the server

The server can be started and stopped with the Service Configuration Manager (**Programs -> apacheds -> Service Settings**). You must be admin to do this.

The Service Configuration Manager also allows the configuration of several windows service settings such as:

- Changing the display name & description of the service
- Configuring the service startup type of **Manual**, **Automatic**, or **Disabled**
- Changing the user the service runs as
- Changing logging settings for the service
- Changing the version of the Java Runtime used by the service
- Changing the properties for the JVM

For details look at the next page [Basic configuration tasks](#)

Installing on Linux

If you want to install ApacheDS as daemon in /etc/init.d, you need **root privileges**. There are two possibilities installing ApacheDS on Linux:

- using the Debian package
- using the RPM package
- using the jar file

Using the jar file

```
// Get the apacheds Linux jar, and copy it where you want the server be installed.  
java -jar apacheds-1.0.0-linux-i386-setup.jar
```

Now, you'll see a graphical installer displayed. Interface and functionality of this installer is similar to other wizard based installers. First, you get some information how to run the server as daemon, tools included in ApacheDS and so on. Next you can Select a directory where the server will be installed. As

you are user `*root*`, you will be able to install the server almost anywhere. One solution could be to install it in your home directory, if you want to test the server. Another solution could be to install the server in `/opt` or in `/usr/local` if you are using Ubuntu or Debian. It's up to you. Be aware that the database backend will also reside within the same directory, so make sure you will have enough place. On the next screen you can choose the packages to install. The init script can only be installed if you are root. The Documentation packages also contains the whole Java API for Apache DS. Last you can add some shortcuts in your environment.

Starting and stopping the server

After installing the application there is one step left to start the server:

```
[root]# /etc/init.d/apacheds start
```

or

```
[user]# /usr/local/apacheds-1.0.0/bin/server.init start
```

Stopping the server works equivalent

```
[root]# /etc/init.d/apacheds stop
```

or

```
[user]# /usr/local/apacheds-1.0.0/bin/server.init stop
```

LDAP Operations Overview

This page last changed on Dec 21, 2006 by [szoerner](#).

[ApacheDS v1.0 Basic User's Guide \(TOC\)](#)

[Next: Searching >>](#)

This section gives an overview of the LDAP operational model

- [How LDAP works](#)
- [All LDAP operations](#)
- [Resources](#)

How LDAP works

In principle the LDAP protocol follows a request/reply scenario between client and server. Unlike HTTP this not necessarily takes place synchronously in a ping-pong exchange. A client may also execute several requests by a server within a session simultaneously. The client supplies a MessageID with each request. The server uses this ID within the corresponding reply. This allows the client to match requests and replies.

All LDAP operations

The operational model of LDAP v3 is comprised of the the following 10 operations.

LDAP Operation	Short description
Bind Operation	exchange authentication information between client and server
Unbind Operation	terminate a protocol session
Search Operation	perform a search by the server
Modify Operation	modify attributes of an entry
Add Operation	add an entry into the directory
Delete Operation	remove an entry from the directory
Modify DN Operation	change the leftmost component of the name of an entry, or move a subtree of entries to a new location
Compare Operation	compare an assertion provided with an entry in the directory

Abandon Operation	abandon an outstanding operation
Extended Operation	perform an additional operation, defined in RFCs or be private to particular implementations

Resources

- [RFC 2251](#) Lightweight Directory Access Protocol (v3)

LDAP Operations, Modification

This page last changed on Jan 06, 2007 by [szoerner](#).

<< Previous: Searching	ApacheDS v1.0 Basic User's Guide (TOC)	
--	--	--

This section gives an overview on how to manipulate entries within your directory. Manipulating data with the help of [graphical tools](#) is straight forward. This section concentrates on using [LDIF](#) and [command line tools](#).

- [Adding an entry](#)
- [Modifying an entry](#)
- [Resources](#)

Adding an entry

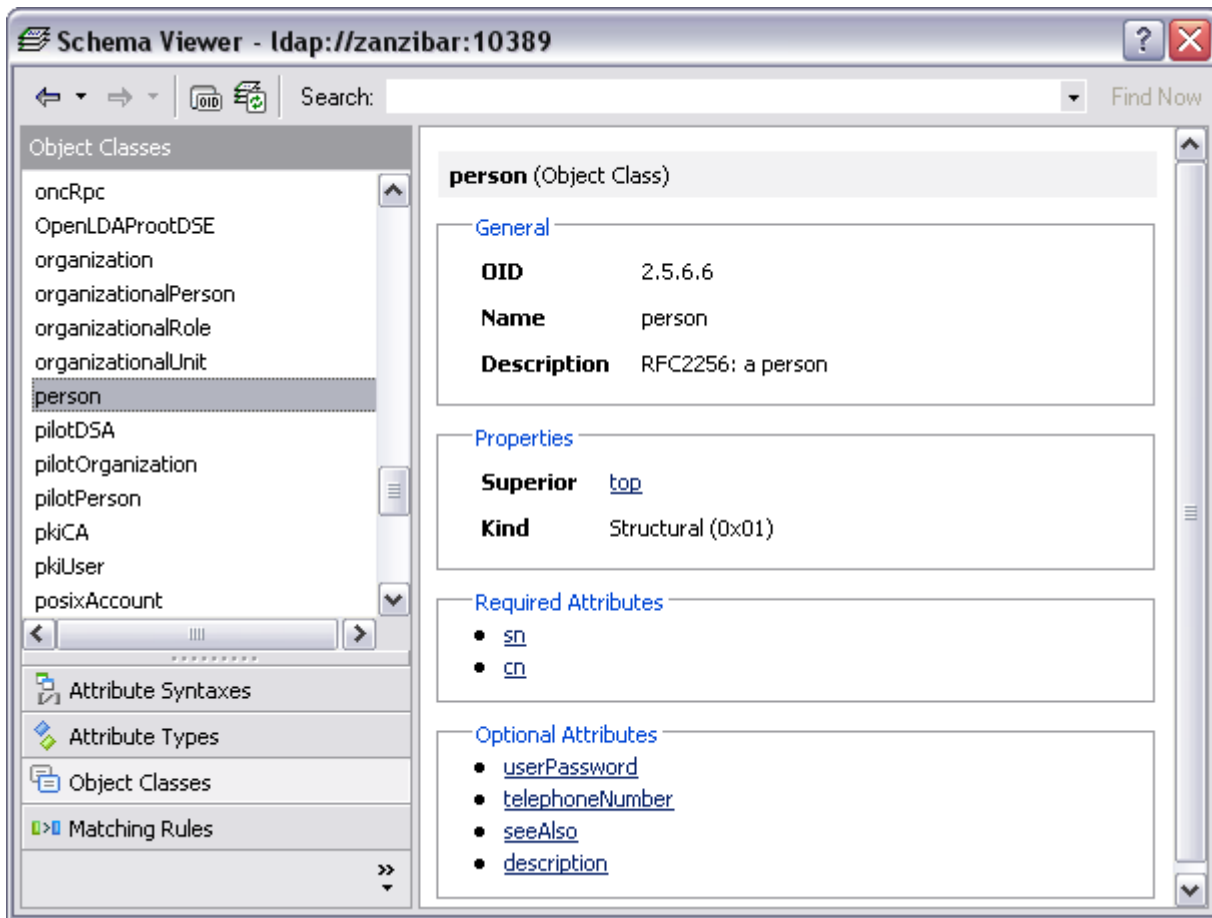
Let's start with adding a new entry to the "Seven Seas" partition (it is therefore assumed that you have already imported the sample data).

A person to add

The data is inspired by "Peter Pan" and provided by this LDIF file ([captain_hook.ldif](#)):

```
# File captain_hook.ldif
dn: cn=James Hook,ou=people,o=sevenSeas
objectclass: person
objectclass: top
cn: James Hook
description: A pirate captain and Peter Pan's nemesis
sn: Hook
userpassword: peterPan
```

The entry with distinguished name "cn=James Hook,ou=people,o=sevenSeas" describes a person. In the default schema of ApacheDS (as defined in RFC 2256), object class *person* requires attribute values for *cn* (common name) and *sn* (surname). The other attributes are optional. The following screenshot of a schema browser illustrates this:



Using a command line tool to add the entry

It depends on your [authorization configuration](#), which directory users are allowed to add entries (or generally to manipulate data). The administrator `uid=admin,ou=system` is always allowed to do anything; thus we use him for authentication.

With `ldapmodify`, the data above can be added to the sample partition like this:

```
$ ldapmodify -h zanzibar -p 10389 -D "uid=admin,ou=system" -w secret -a -f captain_hook.ldif
adding new entry cn=James Hook,ou=people,o=sevenSeas
$
```

The following table contains descriptions for the options used. See the [manpage of ldapmodify](#) for details.

Option	Meaning
-h zanzibar	Hostname
-p 10389	Port
-D "uid=admin,ou=system"	Distinguished name to bind (user with appropriate privileges needed)
-w *****	Password of bind user

-a	add new entries
-f captain_hook.ldif	Name of LDIF file to load

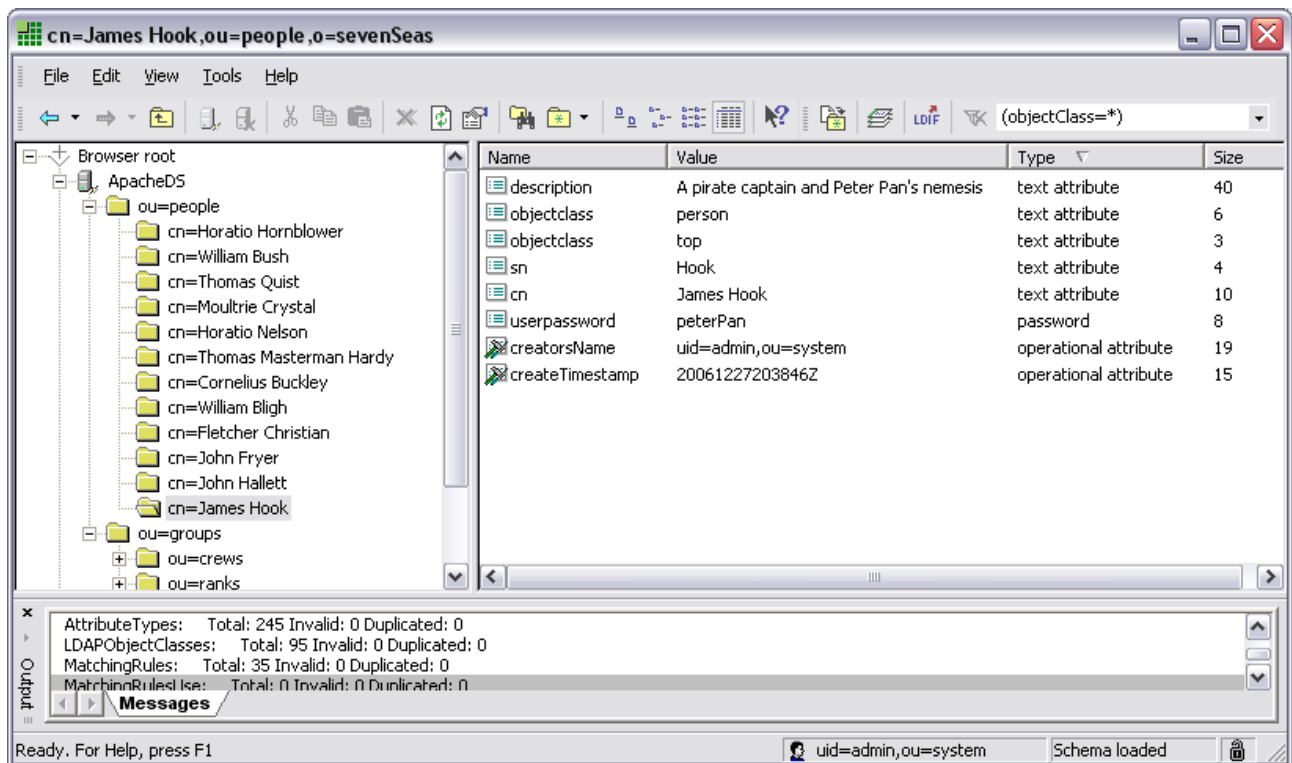
Note that the file contains only one entry, but it is possible to add several entries at once with a single *ldapmodify* call. An LDIF file can contain an arbitrary number of entries, separated by an empty line. an *ldapmodify* call as above would try to add them one by one.

Verification

With the help of the *ldapsearch* command, you can verify that the entry is indeed present in the directory.

```
$ ldapsearch -h zanzibar -p 10389 -D "uid=admin,ou=system" -w secret -b "o=sevenSeas" -s sub
"(cn=James*)"
version: 1
dn: cn=James Hook,ou=people,o=sevenSeas
userpassword: peterPan
description: A pirate captain and Peter Pan's nemesis
objectclass: person
objectclass: top
sn: Hook
cn: James Hook
$
```

Learn more about LDAP search operations [here](#). Another option for verification is to use a graphical tool like [Softerra LDAP Browser](#):



Modifying an entry

Modifications with the help of LDIF

LDIF can either be used to describe complete entries, like Captain Hook in the example before, or to describe a set of changes made (or to be made) to directory entries. In the following we use the latter variant. We present simple LDIF files with changes to an entry (Hook again, the samples assume his existence within the tree) and apply them to the directory.

Adding attribute values

Let's add a telephone number and a second description to the entry "cn=James Hook,ou=people,o=sevenSeas".

A corresponding LDIF file ([captain_hook_modify_addAttrs.ldif](#)) looks like this:

```
# File captain_hook_modify_addAttrs.ldif
dn: cn=James Hook,ou=people,o=sevenSeas
changetype: modify
add: description
description: Wears an iron hook in place of his right hand
-
add: telephoneNumber
telephoneNumber: 254-20
-
```

We apply these changes to the directory with the help of the *ldapmodify* command:

```
$ldapmodify -h zanzibar -p 10389 -D "uid=admin,ou=system" -w secret -f
captain_hook_modify_addAttrs.ldif
modifying entry cn=James Hook,ou=people,o=sevenSeas
$
```

The arguments are the same as in the add example above, except the missing *-a* switch (because we perform modifications, not additions).

After successfully applying these changes, we can verify the effect with a search operation.

```
$ ldapsearch -h zanzibar -p 10389 -D "uid=admin,ou=system" -w secret -b "o=sevenSeas" -s sub
"(cn=James Hook)"
version: 1
dn: cn=James Hook,ou=people,o=sevenSeas
sn: Hook
telephonenumber: 254-20
userpassword: peterPan
objectclass: person
objectclass: top
cn: James Hook
description: A pirate captain and Peter Pan's nemesis
description: Wears an iron hook in place of his right hand
$
```

to be continued

Changing attribute values

Replacing attribute values

Removing attribute values

Recording modify operations with the help of ELBE

Execution of LDIF files with graphical tools

Deleting an entry

Resources

- [RFC 2849](#) RFC 2849 - The LDAP Data Interchange Format (LDIF) - Technical Specification
- [RFC 2256](#) RFC 2256 - A Summary of the X.500(96) User Schema for use with LDAPv3
- [RFC 4511](#) RFC 4511 - Lightweight Directory Access Protocol (LDAP): The Protocol

LDAP Operations, Searching

This page last changed on Dec 21, 2006 by [szoerner](#).

<<Previous:Overview	ApacheDS v1.0 Basic User's Guide (TOC)	Next:Modification>>
---	--	---

This section gives an overview on how to search your directory

- [Searching with LDAP](#)
- [Important parameters in LDAP searches](#)
- [Searching with a command line tool](#)
- [Searching with a Java program](#)
- [Searching with a graphical client tool](#)
- [Resources](#)

Searching with LDAP

From a users point of view, searching is the most important LDAP operation at all. Normally end users perform queries against a directory with the help of custom applications, which fit their specific needs. E-Mail programs like Thunderbird for instance allow the look up of addresses in a convenient way:



The actual LDAP search operation with all its parameters is hidden here (which is desired in this situation).

So, when do you need the original LDAP syntax and parameters? Here are some use cases:

- command line tools (ldapsearch)
- LDAP clients (browsers, administrative tools)
- custom application development (using an API like JNDI)
- configuration of software products, which integrate LDAP directories

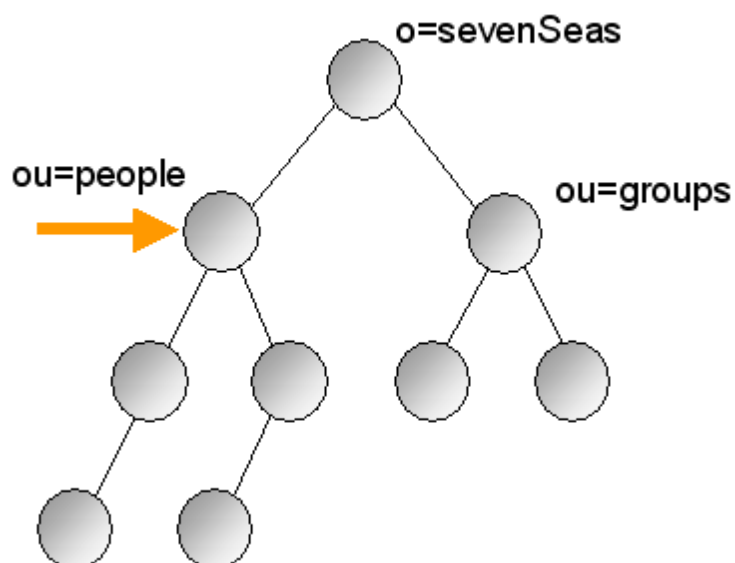
The following description about LDAP searches matches all of the above.

Important parameters in LDAP searches

The search base and the search scope are used to reduce the amount of entries, which should take into account as result entries. To the set of entries defined by base and scope, a filter is applied.

Search base

The search base determines an entry as the starting point within the tree. No entries above this point will be returned within the search result.

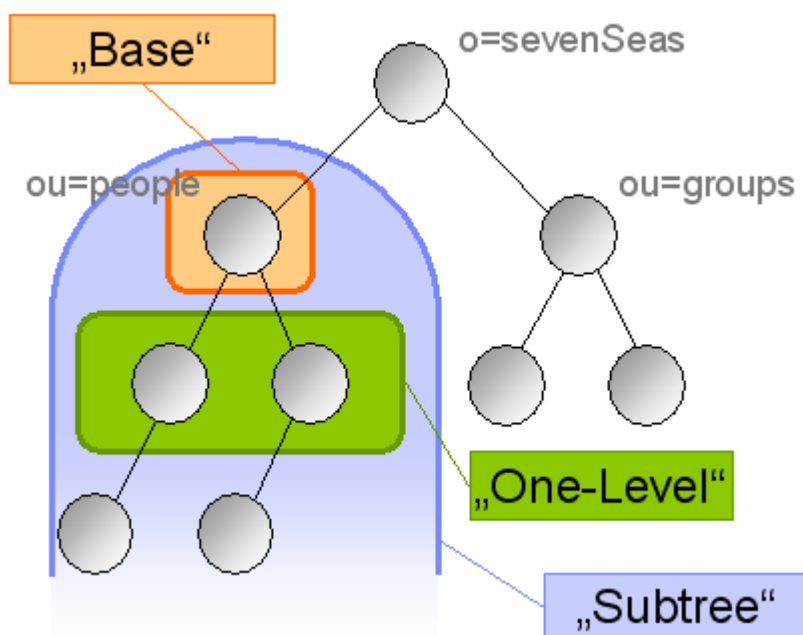


Search scope

The search scope defines the set of entries below the base, which is examined by the server.

Scope	Description
Base	Only the base entry
One-Level	One level below, all direct children of the base

	entry
Subtree	The whole subtree below the base, including the base entry



Base scope is useful for reading attribute values of a particular entry. One example for using *One_Level* scope is expanding the tree in graphical LDAP clients. For extensive investigations within your data, *Subtree* is the most powerful option.

Search Filters

If you look for entries with certain properties (attribute values), you need search filters to delimit the result set in order to contain the entries which meet your search criteria.

Within a filter expression, the following operators are used to combine an attribute name with a value

Name	Operator	Example	Example matches
Presence	= *	<i>(mail=*)</i>	all entries which have at least one <i>mail</i> attribute value
Equality	=	<i>(givenName=William)</i>	all antries where a <i>givenName</i> attribute value equals "William"
Substring	=	<i>(sn=H*)</i>	all entries where a <i>sn</i> (surname) attribute value starts with "H"
Order	<=, >=	<i>(ou>=N)</i>	all entries where a <i>ou</i> attribute value is greater than "N"

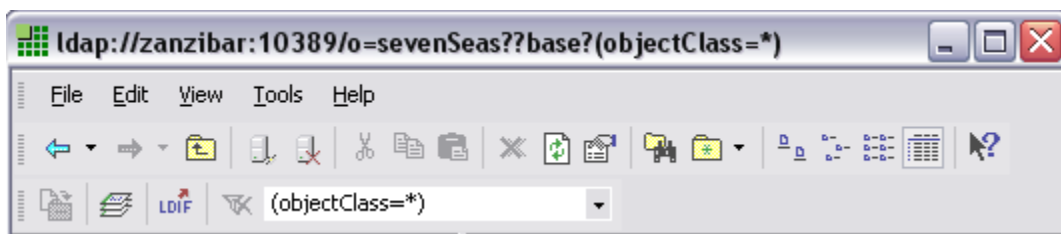
Approximatly	<code>~=</code>	<code>(sn~=Mayer)</code>	all entries where an <i>sn</i> attribute value is similar to "Mayer"
--------------	-----------------	--------------------------	--

Simple filter expressions like the examples above can be combined with the help of boolean operators. To form expressions with them, the prefix notation is used. Prefix means that the operator does not stand between the operands (like in "(3 + 6)"), but before them (like in "(+ 3 6)"), which allows you to use more than two operands within a single expression (like in "(+ 3 6 2)").

Name	Operator	Example	Example matches
and	<code>&</code>	<code>(&(objectClass=person)(sn=H*))</code>	all person entries with surnames starting with "H"
or	<code> </code>	<code>(&(objectClass=person)(sn=H*sn=W*))</code>	all person entries with surnames starting with "H" or "W"
not	<code>!</code>	<code>(&(objectClass=person)(!mail=*))</code>	all person entries without mail address

(objectClass=*)

Quite often, for instance as default in UI tools, you will see the search filter `(objectClass=*)`. It is used if a filter expression is required as a parameter, but all entries should be returned regardless of their attribute values. `(objectClass=*)` does the job, because each entry has at least one *objectClass* value occurrence (normally two or more), the filter therefore matches all entries.



Searching with a command line tool

If you connect to ApacheDS with [command line tools](#), you execute searches against the directory with the `ldapsearch` command.

An example

The query searches all entries below "ou=people,o=sevenSeas" which match the filter `(&(objectClass=person)(givenName=William))`, and therefore are persons called William.

```
$ ldapsearch -h zanzibar -p 10389 -D "uid=admin,ou=system" -w secret \\\
```

```

-b "ou=people,o=sevenSeas" -s sub "(&(objectClass=person)(givenName=William))" uid mail
dn: cn=William Bush,ou=people,o=sevenSeas
uid: wbush
mail: wbush@royalnavy.mod.uk

dn: cn=William Bligh,ou=people,o=sevenSeas
uid: wbligh
mail: wbligh@royalnavy.mod.uk
$

```

The tool return the attribute values for the attributes given in the command line (uid and mail). If no attributes are provided, all attributes the user is allowed to read will be returned.

Command line options used in the example

Option	Example value	Meaning
-h	zanzibar	Hostname of LDAP server to connect to (default is localhost)
-p	10389	Port on which the server listens (default is 389)
-D	"uid=admin,ou=system"	Distinguished name of bind user
-w	secret	Password of bind user
-b	"ou=people,o=sevenSeas"	search base
-s	sub	search scope, one of "base", "one", or "sub" (default is sub)

Searching with a Java program

Here is the same search as above, performed against the "Seven seas" example from within a Java program with the help of JNDI.

```

package search;

import java.util.Hashtable;

import javax.naming.Context;
import javax.naming.NamingEnumeration;
import javax.naming.NamingException;
import javax.naming.directory.Attribute;
import javax.naming.directory.Attributes;
import javax.naming.directory.DirContext;
import javax.naming.directory.InitialDirContext;
import javax.naming.directory.SearchControls;
import javax.naming.directory.SearchResult;

public class SimpleSearch {

    public static void main(String[] args) throws NamingException {

        // JNDI connection data, move them to jndi.properties
        Hashtable env = new Hashtable();
        env.put(Context.INITIAL_CONTEXT_FACTORY, "com.sun.jndi.ldap.LdapCtxFactory");
        env.put(Context.PROVIDER_URL, "ldap://zanzibar:10389/");
    }
}

```



```

env.put(Context.SECURITY_AUTHENTICATION, "simple");
env.put(Context.SECURITY_PRINCIPAL, "uid=admin,ou=system");
env.put(Context.SECURITY_CREDENTIALS, "secret");

try {
    DirContext ctx = new InitialDirContext(env);

    String base = "ou=people,o=sevenSeas";
    String filter = "(&(objectClass=person)(givenName=William))";

    SearchControls ctls = new SearchControls();
    ctls.setSearchScope(SearchControls.SUBTREE_SCOPE);
    ctls.setReturningAttributes(new String[] { "uid", "mail" });

    NamingEnumeration resultEnum = ctx.search(base, filter, ctls);
    while (resultEnum.hasMore()) {
        SearchResult result = (SearchResult) resultEnum.next();

        // print DN of entry
        System.out.println(result.getNameInNamespace());

        // print attributes returned by search
        Attributes attrs = result.getAttributes();
        NamingEnumeration e = attrs.getAll();
        while (e.hasMore()) {
            Attribute attr = (Attribute) e.next();
            System.out.println(attr);
        }
        System.out.println();
    }

    ctx.close();
} catch (NamingException e) {
    System.out.println(e.getMessage());
}
}

```

The output of the program looks like this:

```

cn=William Bush,ou=people,o=sevenSeas
mail: wbush@royalnavy.mod.uk
uid: wbush

cn=William Bligh,ou=people,o=sevenSeas
mail: wbligh@royalnavy.mod.uk
uid: wbligh

```

Key for LDAP searches with JNDI are the *search* methods in the *javax.naming.directory.DirContext* interface ([javadoc](#)). Learn more about JNDI and how to perform searches against an LDAP directory with it in the [JNDI tutorial](#)

Searching with a graphical client tool

If you connect to ApacheDS with [graphical tools](#), normally a search dialog is provided which allows you to enter the parameters needed to perform your search.

Here is the search query used throughout this section, performed with [Softerra LDAP Browser](#):

Search [zanzibar:10389]

Search Settings

Search DN:

Filter:

Attributes:

Search Scope: ☐ One level ☒ Sub-tree level

DN	uid	mail
cn=William Bush,ou=people,o=sevenSeas	wbush	wbush@royalnavy.mod.uk
cn=William Bligh,ou=people,o=sevenSeas	wbligh	wbligh@royalnavy.mod.uk

Search Save Result... Stop Close

If you prefer to use [JXplorer](#), the search dialog looks like this:

Search

Filter Name:

Start Searching From:

Alias Options

☐ Resolve aliases while searching.

☐ Resolve aliases when finding base object.

Search Level

Select Search Level:

Information to retrieve:

Build Filter Join Filters Text Filter

`(&(objectClass=person) (givenName=William))`

More Less Save Load View

Search Cancel Help

Resources

- [RFC 4515](#) LDAP: String Representation of Search Filters
- [manpage of ldapsearch](#) contains the options for the OpenLDAP of this LDAP search tool
- The [JNDI tutorial](#) is a good resource to learn about LDAP client programming with Java

LDAP Studio

This page last changed on Jan 27, 2007 by [ersiner](#).

	ApacheDS v1.0 Basic User's Guide (TOC)	Next:ApacheDS tools>>
--	--	---

LDAP Studio is an Eclipse RCP based LDAP Client Application.

Resources

[LDAP Studio Documentation](#)

Mozilla Thunderbird

This page last changed on Dec 18, 2006 by [ck](#).

	ApacheDS v1.0 Basic User's Guide (TOC)	Next:Apache Tomcat>>
--	--	--

In this section you will learn how to integrate Apache Directory Server into a mail client in order to use the data as an adressbook. Mozilla Thunderbird is used as an example.

- [E-Mail clients and Mozilla Thunderbird](#)
- [Prerequisites](#)
- [Define Apache Directory Server as an adress book](#)
- [Searching your new adress book](#)
- [Other mail clients](#)
- [Recources](#)

E-Mail clients and Mozilla Thunderbird

Integrating an LDAP server in an E-Mail client is a very traditionary task, because directories are commonly used as user repositories within companies and organizations. Contact data is stored for all users of the enterprise, and it is quite common to build the companys online phone/address book on this directory. These address books are often web based application within the intranet. But many E-Mail clients allow to connect to an LDAP based directory directly and use its data as an address book. This seamless integration provides better user experience. One of these clients is Mozilla Thunderbird.

Technically, a mail program acts as a normal LDAP client, as described in earlier sections (i.e. the client connects to the server and performs LDAP search operations). Therfore the parameters you have to specify are the same. Main difference between searches with E-Mail clients and searches with LDAP Browsers like Softterra or JXplorer is that most of the complexity of the LDAP search is hidden to the user. Hence these tools are easier to use, but less powerful.

Mozilla Thunderbird

Mozilla Thunderbird is a popular open source E-Mail client which supports many platforms. Actually it is more than just an E-Mail client (e.g. a news client as well). Features include junk mail control and RSS reading. Learn more about this software at the projects Homepage: [Mozilla Thunderbird](#).

Within this lesson we use Thunderbird primarily because of its broad support for different operation systems and hardware platforms (and because it allows the integration of LDAP servers as address books, of course). You may use other E-Mail clients as well. It is likely that that allow the integration of LDAP directories as well, and even that the configuration is similar to Thunderbird. Check your product documentation for details.

Prerequisites

We assume that you have Mozilla Thunderbird installed on your system (or you use another E-Mail client and are willing to assimilate the instructions to your situation). You may wish to download the software at the homepage ([Mozilla Thunderbird](http://www.mozilla.com/thunderbird/)) and install it, before proceed with this lesson.

Furthermore you need an LDAP server up and running, which address data should be used as an address book within your E-Mail client. For the instructions it is assumed that you have installed Apache Directory Server as described in the first trail and loaded our sample data. To sum it up the following is assumed for the environment:

- Apache Directory runs on host **zanzibar**. LDAP and listens to port **10389**
- Anonymous access to the directory is allowed
- Data is imported as described in section 2, Base DN is **o=sevenSeas**

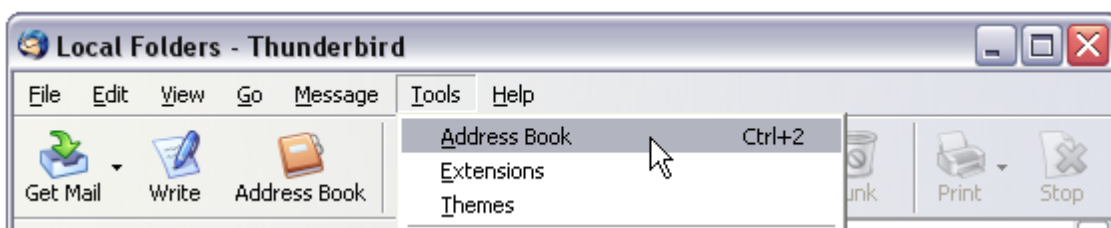
You may use this lesson as a blueprint to integrate other directory servers as well. At least you need the data given above in **bold**.

Define Apache Directory Server as an adress book

Open the adress book

After starting Mozilla Thunderbird on your workstation, go to the adress book by

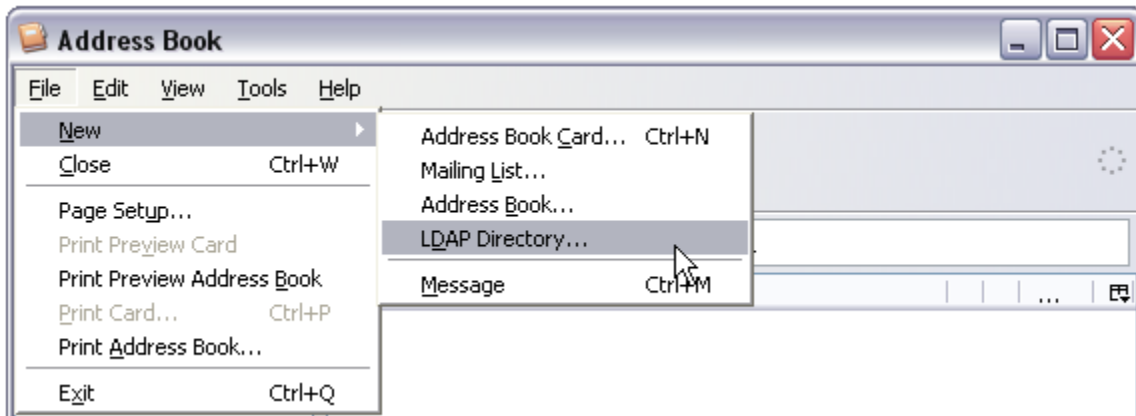
- clicking the adress book icon or
- activation of the corresponding menu item ("Tools" – "Address Book") or
- pressing Ctrl+2



Define a new LDAP directory

Within the adress book window open the dialog to define a new LDAP directory by

- activation of the corresponding menu item ("File" – "New" – "LDAP Directory...")



Thunderbird opens a dialog with three tabbed panes to provide the data of the directory.

Provide connection data

Within the "General" tab, enter basic connection data to your directory:

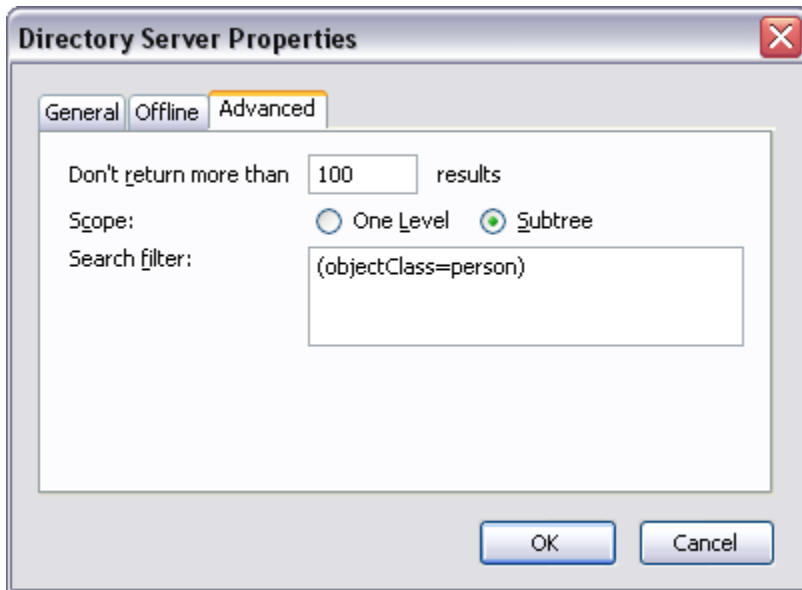
- Name: A name which is used by Thunderbird within the UI, e.g. "Seven Seas"
- Hostname: the hostname or IP address of the server, "zanzibar" in our case
- Base DN: Search base for looking up people, we choose "ou=people,o=sevenSeas"
- Port number: The port the LDAP provider of Apache Directory Server is listening on, "10389" in our case



In this example we do not provide a Bind DN but let Thunderbird look up the users within our directory anonymously. Apache Directory Server should be appropriately configured for that, or you have to provide a user here.

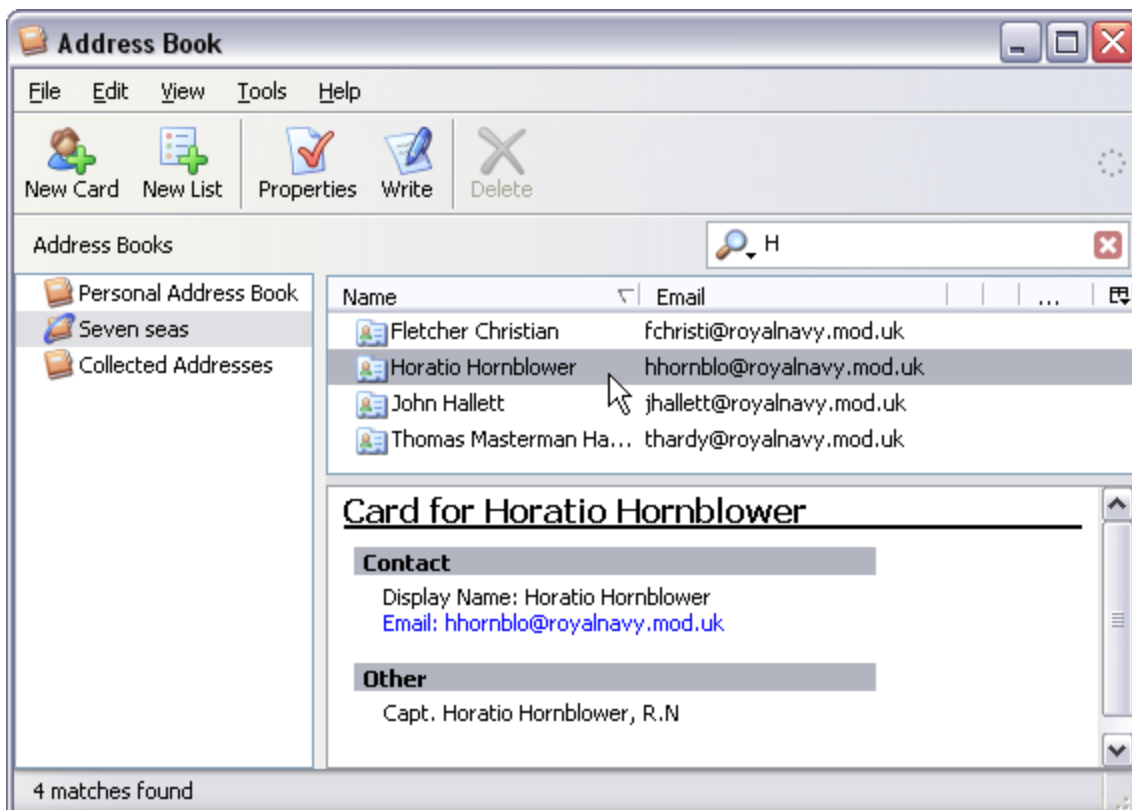
The advanced tab of the dialog provides input fields for result set limits, search scope and search filter. In our example we perform a search with subtree scope and a maximum number of 100 entries within

the result set. The search filter restricts the results to person entries only.



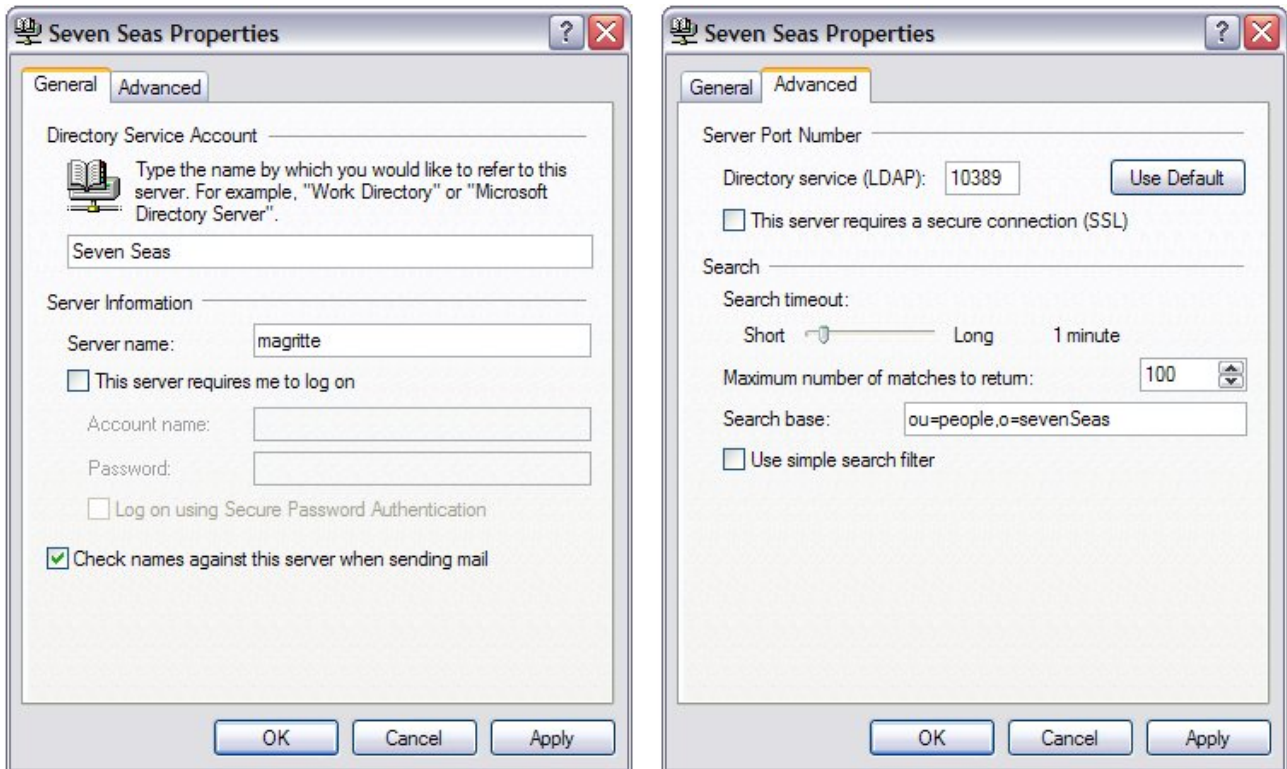
You probably have noticed that the input fields in the two tabbed panes corresponds exactly to the parameters for an LDAP search operation as described in lesson Search the directory of this trail.

Searching your new adress book



Other mail clients

As mentioned above other E-Mail clients have comparable features regarding directory integration. See below for illustration the dialog from the address book of [Microsoft Outlook Express 6](#). As you can see the parameters to define a new directory for usage within this E-Mail client are totally the same as above.



Other E-Mail clients that support LDAP integration for address books include [Eudora Email](#) and [IBM Lotus Notes](#).

Resources

- [An introduction to Thunderbird](#), Open Source Articles
- [LDAP Attribute Mapping](#) for Mozilla Thunderbird

Other programming languages

This page last changed on Dec 21, 2006 by [ck](#).

<< Previous:Connecting with Java components	ApacheDS v1.0 Basic User's Guide (TOC)	Next:LDAP Operations - Overview >>
---	--	--

If you connect to an LDAP server from a program, you normally use a library which encapsulates the LDAP calls. It depends on your programming language and platform, which options you have here. LDAP is widely adopted, so there should be at least one solution available for you. Here are some examples:

Language	Library
C, C++	An RFC exists for LDAP C APIs, RFC 1823 , although different implementation vary. The OpenLDAP project offers a library for many platforms.
C#	Novell LDAP Libraries for C#
Perl	Perl LDAP
Python	python-ldap an LDAP client API for Python
Ruby	Ruby/LDAP an extension module for Ruby.
Tcl	Several options exist, one of them is provided by the TCL library at SourceForge

Some Background. Directories, directory services and LDAP

This page last changed on Dec 18, 2006 by [ck](#).

<< Previous: What Apache Directory Server is	ApacheDS v1.0 Basic User's Guide (TOC)	Next: About the sample configurations and sample directory data >>
--	--	--

This section provides a brief overview about directories, directory services and LDAP. Furthermore you find links to different resources (books, online resources, ...), which may act as introduction to the topic. If you are already an LDAP expert, you'll probably skip this section.

- [Directories and directory services](#)
- [LDAP – the Lightweight Directory Access Protocol](#)
- [LDAP resources \(further reading\)](#)

Directories and directory services

Generally speaking, a directory is a collection or list of data. Real world examples are telephone books (public or within organizations), church/land registers and listings of works (e.g. the Koechel-index, which lists all compositions of Mozart). All these examples have the purpose to preserve information and to make it available on demand to whom it may concern.

Within information technology the term **directory** is used for a special kind of data storage. It allows the structured storage and efficient retrieval of objects which are often derived from the real world (e.g. persons, IT equipment). Characteristic:

- all data is stored in so called **entries**
- the set of entries within a directory forms a tree (hierarchical database)

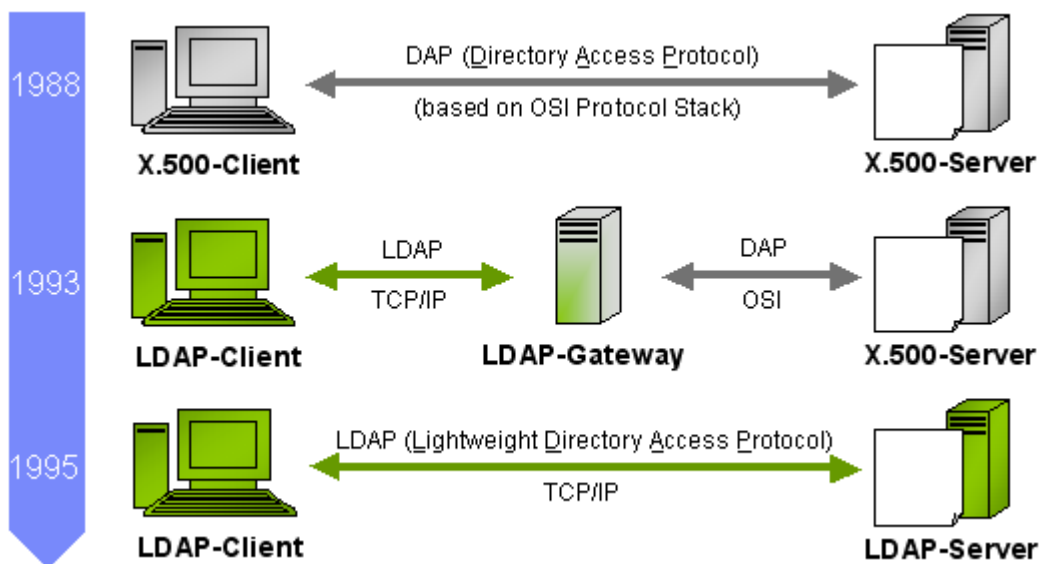
A **directory service** is a solution which offers users access to the information stored in the directory. A directory assistance (call center agent) is a good real world example for such a service. Within information technologies, such services are normally provided by software components. Directory services provide access to the content of a directory via a well-defined interface. If a network is used, an appropriate protocol has to be defined. LDAP (see below) is such a protocol.

The real world examples mentioned above may be stored in such a directory, although other types of storage systems can be more appropriate (this depends on circumstance/requirements). At first sight directories compete thereby as data storage with the established relational data bases. However in the most large enterprises and organizations both directory services and relational databases are actually used. Read how Vikas Mahajan describes directories and databases as complementary, not competitive, solutions in his excellent article ["Should I Use a Directory, a Database, or Both?"](#).

LDAP – the Lightweight Directory Access Protocol

What is it? Some history.

The comprehensive standard **X.500**, finalized in 1988, builds the foundation for many of today's directory solutions. Within this standard, the client accesses the server via the Directory Access Protocol (**DAP**), which is OSI protocol stack based. With the Internet boom in the nineties, the accessibility of directories via TCP/IP became more and more important. Hence a TCP/IP-based access method, which in functionality was a subset of DAP, was standardized in 1993: the **Lightweight Directory Access Protocol (LDAP)**. First LDAP implementations were gateway solutions, they mediated between LDAP clients and X.500 servers. In 1995 the University of Michigan presented the first native LDAP server; in the meantime the work is continued by the [OpenLDAP](#) project. 1996 Netscape followed with the first commercial LDAP server (Netscape Directory Server, foundation of several later LDAP servers). Other examples (among many others) include [Microsoft Active Directory](#) and [Novell eDirectory](#). The figure below shows the development of directory protocols from X.500/DAP to LDAP.



Information model primer

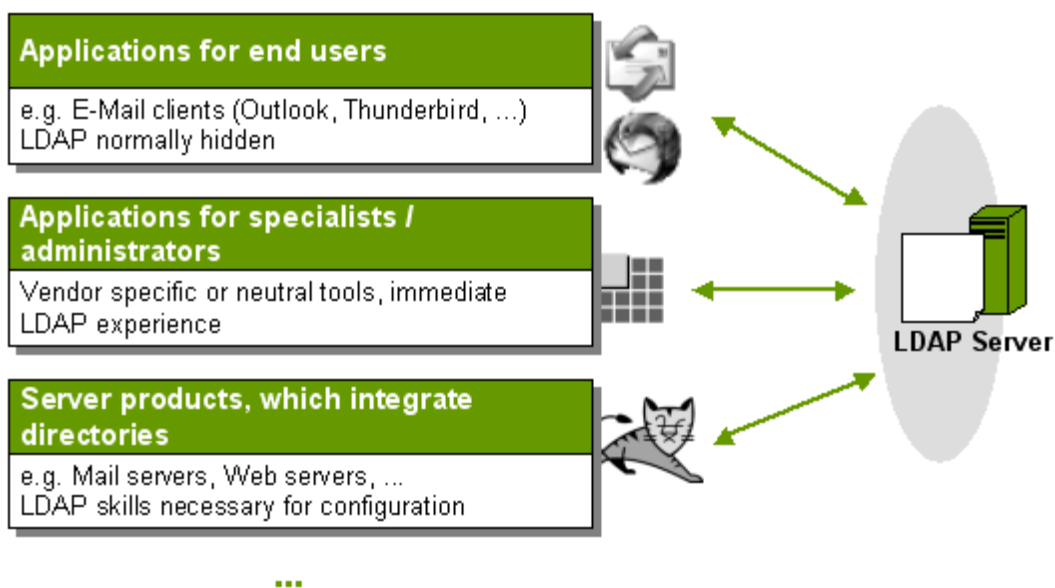
Within the information model of LDAP, data is stored in entries, which build up a hierarchical, tree like structure. Each entry has a unique name (**DN, Distinguished Name**), which depicts its position within the tree. An entry consists of key/value pairs, the **attributes**. Some attributes may occur more than once within an entry (single or multi valued, e.g. a person can have more than one telephone number). So called **object classes** define, which attributes an entry may have, and which of them are required. The classes build up a hierarchy with **top** as root; there is a parallelism to the object oriented world. **top** forces only the attribute objectclass, which assigns an entry its object classes. A **schema** consists object classes and attribute types, and therefore defines, what kind of entries can be stored within the directory. Directory servers ship a schema out-of-the-box, often with elements standardized by RFCs. In addition, most directory solutions allow you to define custom object classes and attributes. But in practise, the pre-defined elements are used. Sometimes they get extended according to special requirements.

Common applications of LDAP based directories

LDAP operations include entry creation, modification, deletion and search. As a general rule, LDAP directories are optimized for read and search operations, at the cost of write performance. Data, which will be modified often, therefore better suits in a relational database, which offers better support for transactions and referential integrity as well. Directories are rather used if comparatively stable data has to be provided centrally.

Common examples are network resources (printers, services) and user data (including credentials and rights for the resources). As a notable feature, many directory products offer replicas, which permit better access times and higher availability especially in geographically dispersed organizations. Not for nothing, the most common LDAP application is the enterprise phone book. That even Microsoft Outlook may be an LDAP client in this case - most average users probably don't know.

Examples of software components which support LDAP



Very different types of software products may act as LDAP clients, consuming data for authentication, authorization or data presentation etc.

- E-Mail clients (e.g. Mozilla Thunderbird)
- LDAP tools (e.g. JXplorer)
- Web servers (e.g. Apache Tomcat, Apache HTTP Server)
- Mail servers (e.g. Apache James)
- ...

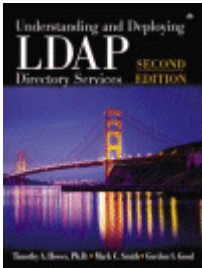
Configuration details for several of these programs in conjunction with ApacheDS are described in later sections.

LDAP resources

English

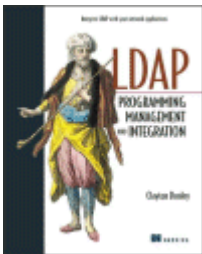
Books

There are several good LDAP books available. Here are some which provide sample chapters on their homepages.



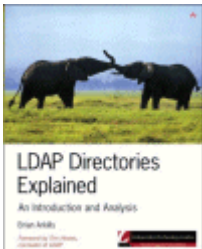
Understanding and Deploying LDAP Directory Services (2nd Edition)
by Timothy A. Howes, Mark C. Smith, Gordon S. Good, Tim Howes
Addison-Wesley Professional, 2nd Edition 2003
ISBN: 0-672323-16-8

[Book's Homepage \(Howes\)](#)



LDAP Programming, Management, and Integration
by Clayton Donley
Manning Publications Company, 2003
ISBN: 1-930110-40-5

[Book's Homepage \(Donley\)](#)



LDAP Directories Explained: An Introduction and Analysis
by Brian Arkills
Addison-Wesley Professional, 2003
ISBN: 0-201787-92-X

[Book's Homepage \(Arkills\)](#)

Articles, forums, blogs and other online resources

Forums

Blogs

- [cn=Directory Manager - All about Directory Server](#), Sun Blog

Articles and other online resources

- [Understanding LDAP - Design and Implementation](#), IBM RedBook, July 2006
- [Demystifying LDAP](#) by Brian K. Jones, O'Reilly Network

German

Bücher



LDAP verstehen, OpenLDAP einsetzen
von Dieter Klünter, Jochen Laser
dpunkt.verlag, Juni 2003
ISBN: 3-89864-217-8

[Webseite zum Buch \(Kluenter\)](#)



LDAP für Java-Entwickler – Eine praxisorientierte
Einführung
von Stefan Zörner
Software und Support Verlag, 2 überarbeitete
Auflage 2005
ISBN: 3-93504-272-8

[Webseite zum Buch \(Zoerner\)](#)

Artikel

- [LDAP verstehen mit linx](#), by Petra Haberer

What Apache Directory Server is

This page last changed on Dec 18, 2006 by [ck](#).

	ApacheDS v1.0 Basic User's Guide (TOC)	Next:Some Background>>
--	--	--

This section describes what Apache Directory Server (abbreviated ApacheDS) is, and where it comes from.

- [System vision](#)
- [Origin and Motives](#)
- [Resources](#)

System vision

ApacheDS 1.0 is an embeddable, extendable, standards compliant, modern LDAP server written entirely in Java, and available under the Apache Software License. Other network protocols like Kerberos and NTP are supported as well (and even more may be added), but basically (and especially for this introduction guide) ApacheDS is an LDAP server.

Embeddable means that it is possible to configure, start and stop ApacheDS from other Java components, especially application servers, and the server runs within the same VM. The solution has already been successfully embedded in Apache Geronimo, JBoss, and others. The fact that the server is embeddable is quite interesting, nevertheless you also have the deployment option to run the server standalone, for instance as a Windows service. Perhaps you know this situation from other LDAP servers – open source (like OpenLDAP) as well as commercial ones (like Sun Java System Directory Server). This guide is dedicated to people that are new to ApacheDS. The guide concentrates on installing, configuring and running ApacheDS in a standalone configuration.

Extendable means that the modern architecture of the solution provides many extension points. Write your own partitions to store directory data, interceptors to add functionality, etc. by implementing certain interfaces and plugging them in using Spring.

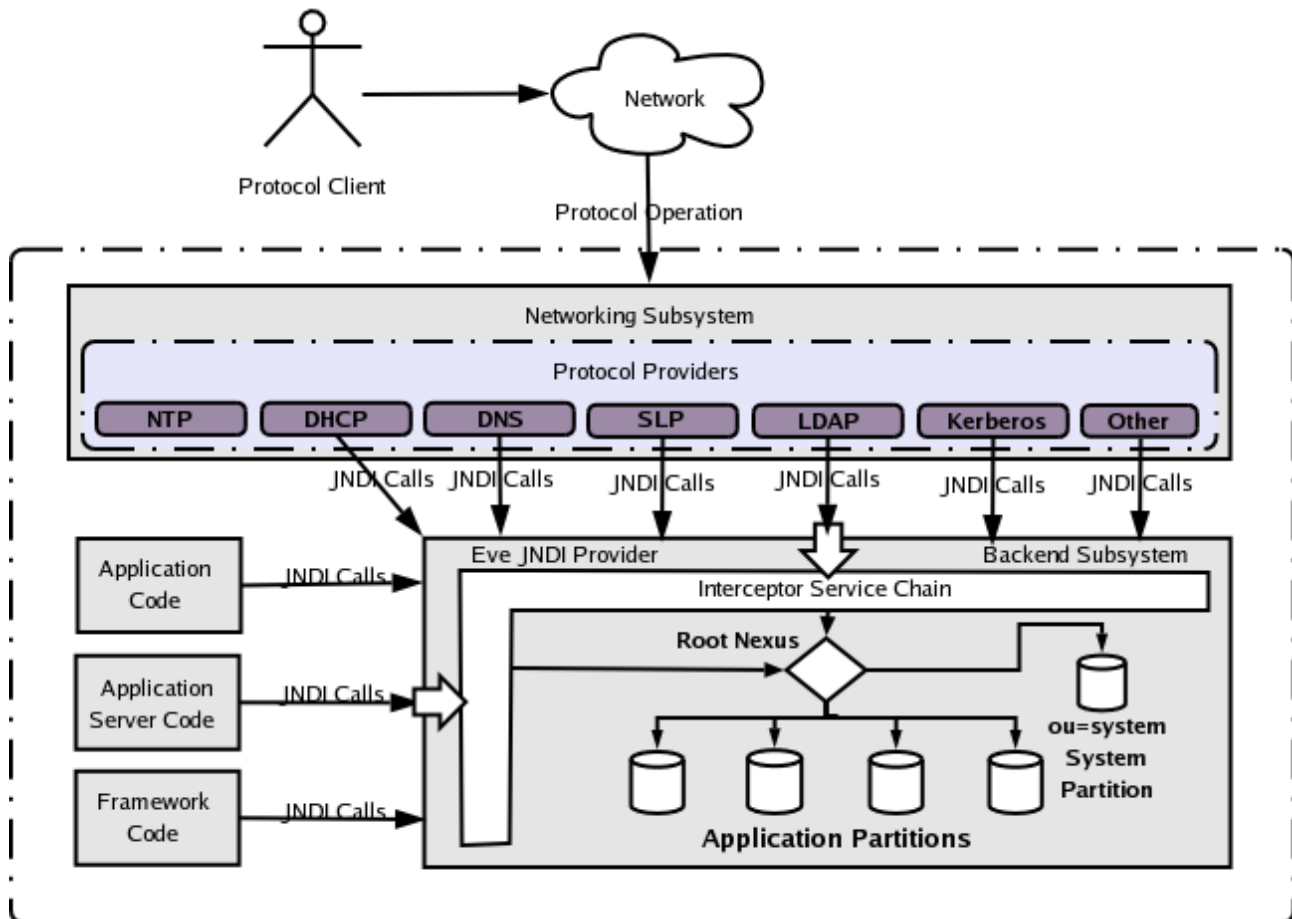
Standard compliant means that ApacheDS 1.0 adheres to all RFCs relevant to LDAPv3. Please note that the server has been successfully certified by the Open Group in September 2006 ("LDAP certified"). Thus LDAP clients may rightly expect that ApacheDS behaves like they expect.

Modern means that ApacheDS aims modernize the LDAP territory, as well as it favors standards compliance. New rich integration tier constructs like LDAP Stored Procedures and Triggers are being built on top of existing standards.

Entirely written in Java means that the software compiles and runs on a huge number of hardware and software platforms. Native installers are available for Windows, MacOS and Solaris (both SPARC and intel

platform), but in fact the set of possible targets is by far more extensive.

Architectural overview



Origin and Motives

Through his experiences with enterprise LDAP directories, Alex Karasulu, realized there is a great need for rich integration tier constructs like LDAP Stored Procedures, Triggers, and Views. In 2001 he set out to alter the OpenLDAP server to offer support for these useful facilities which are present in relational databases but missing in the LDAP world. Alex's attempts failed due to the complexity of the software which was brittle, and difficult to manage. As C code ported to several platforms, the OpenLDAP code base, had several `#IFDEF` conditional pre-compiler directives that made it difficult to change the code. At this point Alex thought about implementing a new LDAP server in pure Java. Thanks to NIO this was finally possible using the 1.4 JDK.

In October 2002 Alex Karasulu founded and registered the [LDAPd](#) project at SourceForge.net. LDAPd was a pure Java embeddable LDAP v3 protocol daemon built on the Avalon framework. Alex donated the code to the Apache Software Foundation and the code entered the [Apache Incubator](#) in October 2003. One year later in October of 2004, the Apache Directory Top Level Project (TLP) was formed after a successful incubation with the now called Apache Directory Server as its flagship product. After 4 years of development, in October 2006, Apache Directory Server 1.0 was released as an Open Group certified

LDAPv3 protocol server.

Having a standards compliant and modern LDAP server, Apache Directory Team is now working on Identity and Access Management solutions leveraging the directory technology.

Resources

- [Proposal for an Apache Directory Project](#) the original proposal for incubation, September 2003.