

JXplorer

Administrator Guide

Contents

Chapter 1: About JXplorer	1
What Is JXplorer?.....	1
What Can You Do with JXplorer?	2
Requirements and Supported Platforms.....	2
Supported Specifications	3
Chapter 2: The JXplorer Browser	5
The Tree Pane	5
The Quick Search Bar	9
The Entry Display.....	10
Chapter 3: Connecting to a Directory	13
The Connect Dialog	14
Security Levels for LDAP Connections.....	15
Save Connection Details in a Template.....	16
Chapter 4: Searching a Directory	17
Quick Searches	17
Complex Searches.....	18
Search Operators.....	22
Search Limits	24
Bookmarks.....	24
Chapter 5: Editing the Directory	25
Directory Tree Operations	25
Modify Attributes in an Entry	28
Attribute Editors	31
Binary Values	35
Add a New Entry	37
Submit an Entry to the Directory.....	38
Chapter 6: Importing and Exporting Data	39
Binary Values in LDIF Files	39
Use an LDIF File Without a Directory.....	40

Chapter 7: Resolving Aliases	41
How JXplorer Displays Aliases	41
Create a New Alias Entry	42
Resolve Aliases While Browsing	42
Resolve Aliases While Searching	45
Chapter 8: Logging and Troubleshooting	47
Logging	47
Troubleshooting	47
Chapter 9: Customizing JXplorer	49
Why Customize the JXplorer Interface?	49
Customize Tree Icons	50
Create HTML Viewing Templates	52
Customize HTML Forms	56
Add Custom HTML Pages	61
Internationalize JXplorer	62
Supply Customized Files	66
Chapter 10: How JXplorer Reads the Schema	69
Data in Each Schema Object	70
Checking Entries for Schema Conformance	71
Chapter 11: How JXplorer Handles Passwords	73
Password Storage	73
Password Hashing	74
Chapter 12: How JXplorer Handles SSL, SASL, and Certificates	77
SSL and SASL	77
Manage Certificates and Keystores	79
Chapter 13: Extending JXplorer	81
Pluggable Attribute Editors	81
Pluggable Entry Editors	91
Plug-ins with Data Listeners	94
Plug-ins with Threads	97
Localize JXplorer Plug-ins	101

Add Help Files to Plug-ins	102
----------------------------------	-----

Chapter 14: LDAP and Directory Resources	103
---	------------

Chapter 1: About JXplorer

This section contains the following topics:

[What Is JXplorer?](#) (see page 1)

[What Can You Do with JXplorer?](#) (see page 2)

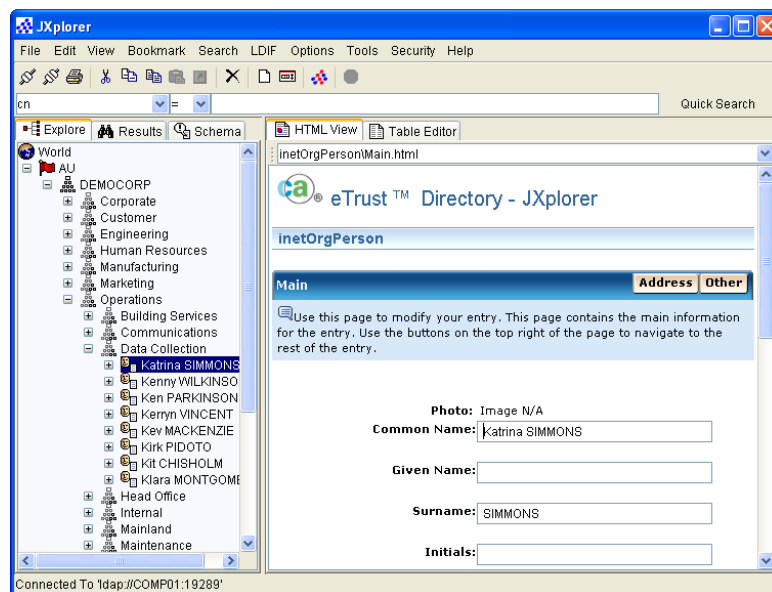
[Requirements and Supported Platforms](#) (see page 2)

[Supported Specifications](#) (see page 3)

What Is JXplorer?

JXplorer is an open source Java application that allows you to browse and search any LDAP directory.

It displays the structure of the directory data as a tree view in the left panel, and the data of any particular entry in the directory in the right hand pane.



JXplorer includes a number of directory-related utility functions, such as secure SSL connectivity, LDIF file reading and writing, graphical cut, copy, paste, and delete, and the Unicode international character set.

JXplorer has advanced security integration and support for the more difficult and obscure parts of the LDAP and DSML protocols.

What Can You Do with JXplorer?

With the JXplorer browser, you can:

- Connect to any directory that supports LDAP and navigate, search, and modify the directory.
- Read the directory's schema directly, rather than relying on schema configuration files.
- Visually cut, paste, and edit subtrees in the directory, including drag and drop on Windows platforms.
- Import and export LDIF files from a directory and even view them offline.
- Configure the browser in many ways, including its appearance and logging information. For example, you can configure the look of the browser to a company standard by using company-specific icons for the directory and company graphics within the HTML templates.
- Display directory data within configurable HTML templates using a simple extension to the HTML language.
- Run in debug mode, permitting full tracing of the LDAP BER protocol.
- Run on a wide variety of operating system platforms, since JXplorer is written in the Java programming language.
- SSL to communicate securely, and SASL for secure certificate-based authentication.

Requirements and Supported Platforms

JXplorer has been tested and run on Windows, Solaris, Linux, OS390, and Macintosh OSX.

Because JXplorer is a Java client, it can also be run on any platform that supports Java. This is up to developers to test for themselves.

JXplorer uses Java 1.4.2. To check what version you are running, go to a command prompt and type:

```
java -version
```


Supported Specifications

JXplorer supports the following LDAP specifications:

- RFC 2251: Core LDAP description
- RFC 2252: Attribute syntax
- RFC 2253: UTF-8 distinguished names
- RFC 2254: Search Filters
- RFC 2255: LDAP URLs
- RFC 2256: Default LDAP user schema
- RFC 2849: LDIF file format

JXplorer also supports the DSML 2.0 specification: Directory Services Markup Language v2.0.

Chapter 2: The JXplorer Browser

When you start JXplorer, the main browser window appears.

The menu bar at the top of the browser provides access to a full range of browser functions through pull-down menus.

Two toolbars below the menu bar give shortcuts to commonly used functions. The first is a button bar, with shortcuts to commonly used menu functions. The second, the quick search toolbar, lets you quickly execute simple searches (such as searching a directory for an employee's name).

The status bar at the very bottom of the browser displays the status of the browser. For example, it shows whether the browser is connected or not.

The main body of the browser is divided into two panes. The left pane is the directory tree, which you can navigate by using the mouse to click the entries. The right pane shows the selected entry from the directory, shown either as an HTML page or as a table of attributes and values.

This section contains the following topics:

[The Tree Pane](#) (see page 5)

[The Quick Search Bar](#) (see page 9)

[The Entry Display](#) (see page 10)

The Tree Pane

The tree display pane (the left pane) displays the directory tree, and allows you to graphically browse the directory. There are usually three tree views available:

Explore

Displays the data in the current directory

Results

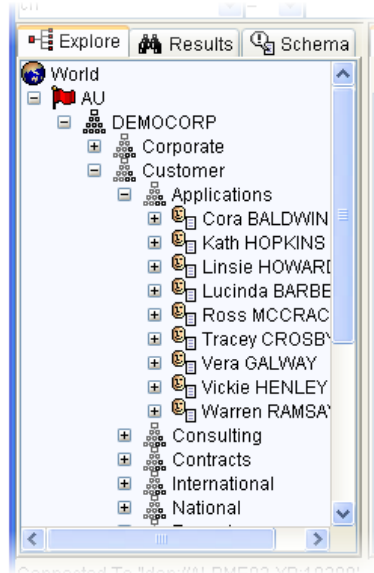
Shows the results of the most recent search

Schema

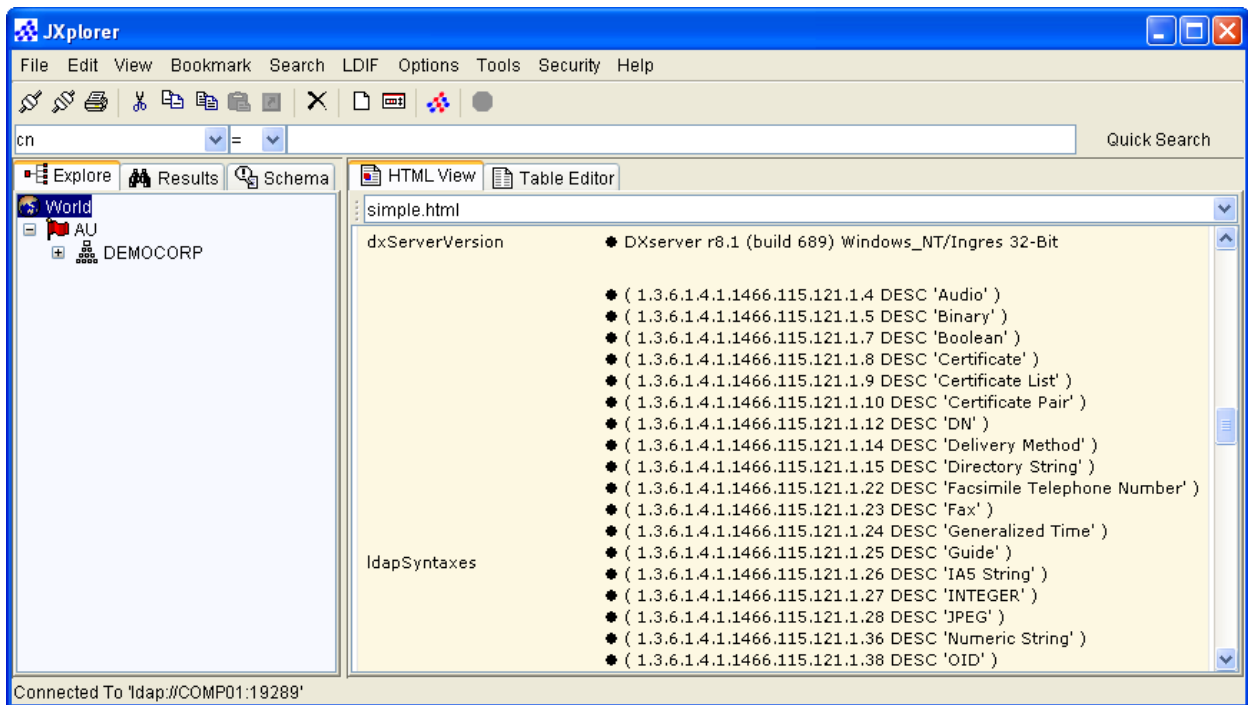
Displays the schema currently in use by the directory

The Explore Tree

In the Explore tree, you can browse the directory tree:

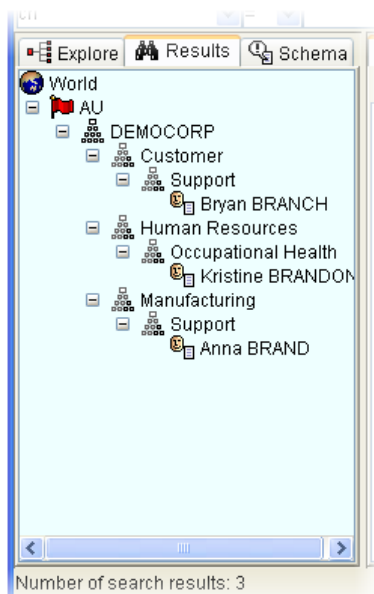


You can also display the schema rules for this directory by clicking on World, which is always shown at the top of the Explore pane:



The Results Tree

Regardless of whether the search is run using the quick search bar or a search menu option, once it is run the matching entries are displayed as a results tree in the Results view of the directory tree pane.



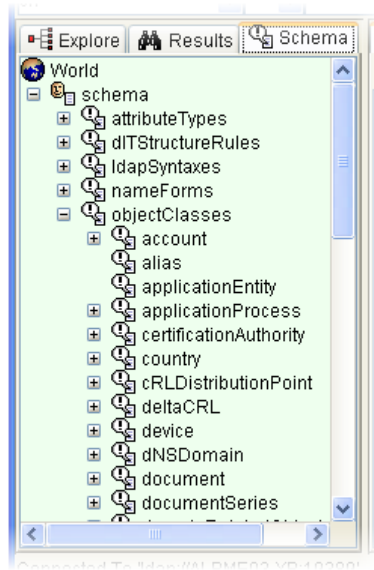
Parents of search results appear as empty entries in the tree. You can browse and save the search result tree (including LDIF format) just like the directory tree. You can edit the results.

You can set the number of entries returned from a search, and the timeout. See Search Limits (see page 24) for more information.

The Schema Tree

In addition to viewing the data entries held in a directory, you can directly view the schema that is read from the directory.

The Schema pane lets you examine attribute definitions, class definitions, and syntax definitions.



Note: Some of these options may not be available with directories other than eTrust Directory, because not all servers implement full schema publishing.

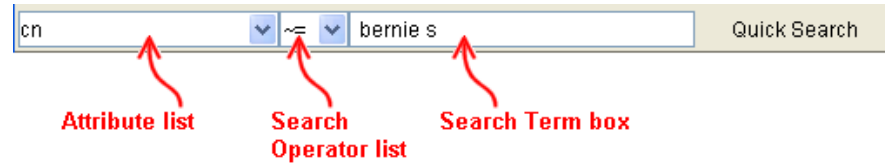
You can display schema entries either in the HTML View tab or the Table Editor tab.

Unlike the other tree displays, however, you cannot edit the schema directly through the browser. For security and administration reasons, many servers do not permit their schema to be edited online and require an administrator to perform schema maintenance at the server.

You can export schema to an LDIF file, but this is not the usual way to store schema information and most directories cannot use it without further processing.

The Quick Search Bar

The quick search bar lets you execute simple searches.



The operators are:

- Equal to =
- Approximately equal to ~=
- Greater than or equal to >=
- Less than or equal to <=
- Not equal to !=

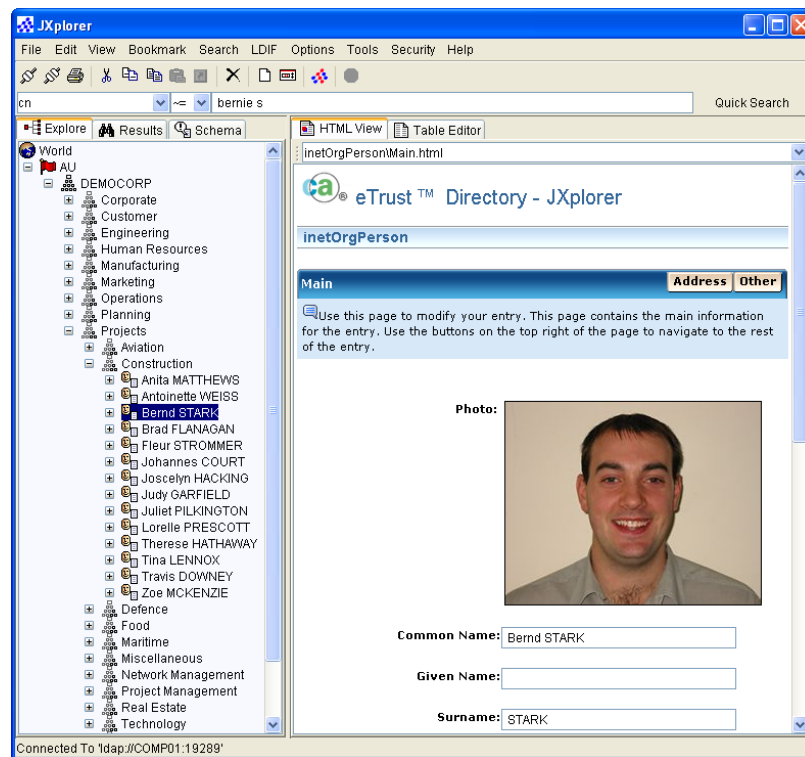
The Entry Display

The entry display pane (the right pane) displays the currently selected directory entry, either in an HTML template, or in an editable table of attributes and values.

When you select an entry, whether from the Explore view, the Results view, or even the Schema view, the browser queries the directory for the attribute values of the entry and displays the results in the entry display pane. The results can be displayed either in the HTML template view or in the attribute/value table view.

The HTML Viewer

The following is an employee record displayed in an HTML template. The template contains three tabs (Main, Address, and Other) that display the attribute information for the selected entry.



HTML Templates

When the browser initially reads an entry, it attempts to find an appropriate HTML template in which to display the entry, as follows:

- The HTML templates key on the object classes of the entry, with each HTML template specializing in displaying one object class.
- Each object class can have multiple HTML templates capable of displaying it.
- HTML templates for a given object class can also display entries whose object classes are inherited from that object class.
- When you select an HTML template for a particular entry, the browser remembers that template and uses it for other entries of the same object class.

The number of attributes and how they are displayed depend on the HTML template. When the HTML template does not have a tag for a particular attribute, that attribute is not displayed. A tag is available for displaying all attributes that have values. The tag is used to simplify the display of large numbers of attributes.

This use of HTML templates enables:

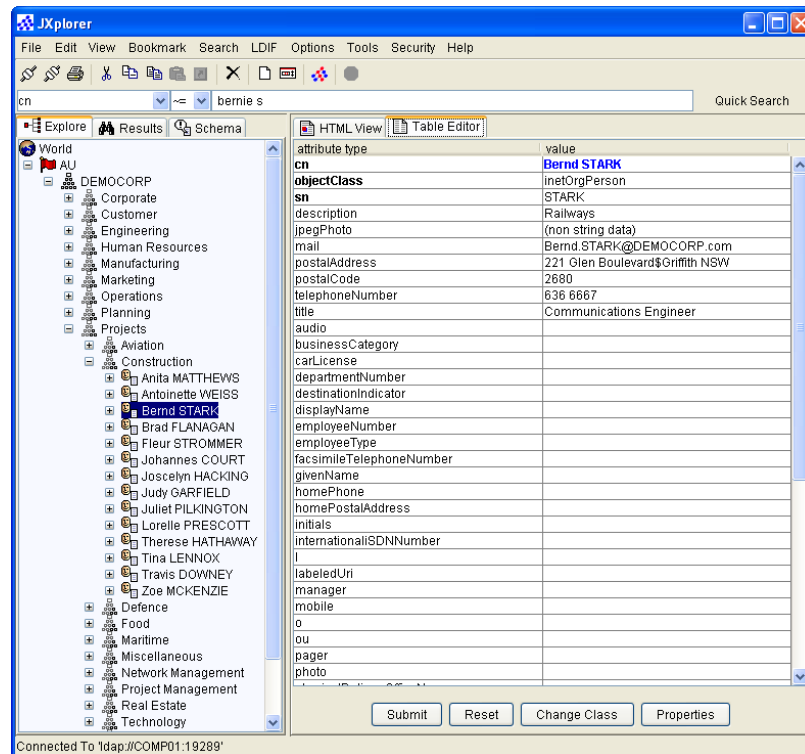
- Different types of entries to be displayed in ways appropriate to their type
- Site-specific help information to be included in the HTML
- HTML hyperlinks to be used to link to existing company resources
- Straightforward customer configuration of data display
- Special purpose templates for different types of users (administrator, help desk, office staff, and so forth)
- Use of corporate branding and logos

The Table Editor

The same employee record can be edited and displayed in the table editor.

You can also use it to view the attributes an entry can contain, because it uses schema to show all the possible attributes that the entry might have.

Attributes in bold represent mandatory attributes - these must be present for the entry to be valid. Click the Properties button to display operational attributes such as the date of the last modification.



You can configure the table viewer to include custom binary editors, which may be the only way to view complex or application-specific binary data.

Chapter 3: Connecting to a Directory

This section contains the following topics:

[The Connect Dialog](#) (see page 14)

[Security Levels for LDAP Connections](#) (see page 15)

[Save Connection Details in a Template](#) (see page 16)

The Connect Dialog

When you connect JXplorer to a directory the browser displays a connection dialog, requesting the information that JXplorer needs to connect to a directory:

Host

The name of the computer that hosts the directory.

Port

The port number of the DSA on that computer

Protocol

The protocol that is in use, which can be LDAP or DSML. For LDAP, this is usually Version 3, but can be Version 2 to support older directories.

If you select DSML, you also need to enter the path to the DSML service.

DSML Service

The path to the DSML service dsml/services/DSML

Base DN

(Optional) The base distinguished name of the directory.

If none is given, the browser is still able to connect, providing that the directory is one (like eTrust Directory) that makes this information available.

Security

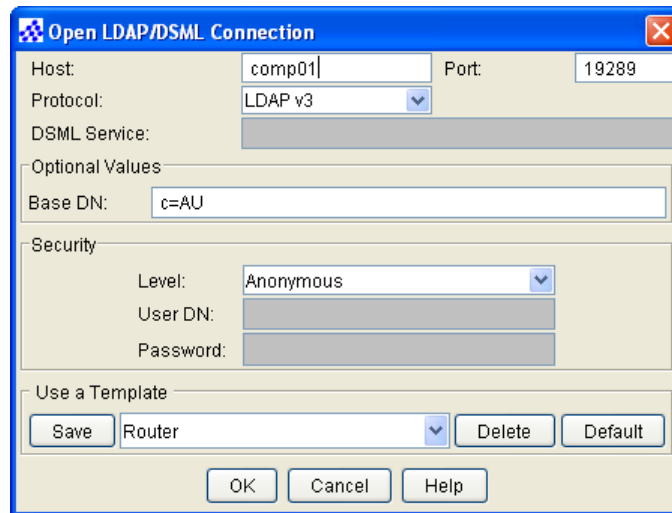
Level

The security level with which you want to connect (not applicable to DSML).

You can connect to the directory anonymously, or with your user name and password. If you require higher security, you can establish a link to the server using SSL with either of these methods. When a client keystore is available, you can use full client-authenticated SSL, combined with SASL authentication at the directory.

User DN

Password



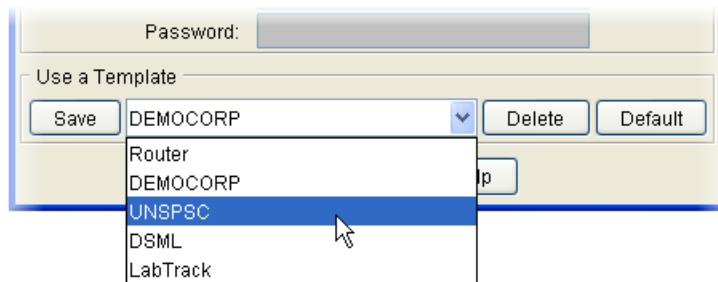
Security Levels for LDAP Connections

When you use JXplorer to connect to an LDAP server, you can choose the level of security for that connection:

Security Level	Description
Anonymous	Connects to the directory anonymously.
User + Password	Connects to the directory using your user name and password.
SSL + Anonymous	Connects to the directory anonymously via an SSL link. The SSL connection uses either the trusted public certificate of the directory server, or the public certificate of the directory server's certificate authority.
SSL + User + Password	Connects to the directory using your user name and password via an SSL link. The SSL connection uses either the trusted public certificate of the directory server, or the public certificate of the directory server's certificate authority.
SSL + SASL + Keystore Password	Connects to the directory via an SSL link. The SSL connection is authenticated using either the trusted public certificate of the directory server (or the public certificate of the directory server's certificate authority), and the client's trusted public certificate (or the public certificate of the client's certificate authority), and the client's private key.
GSSAPI	Basic GSSAPI/Kerberos support

Save Connection Details in a Template

JXplorer lets you save connection details for commonly used directories as connection templates. You can save any number of connection templates, and you can edit or delete them at any time, using the list at the bottom of the Connection dialog:



You can also choose to make one of the connection templates a default template. After you specify a default template, the connection dialog opens with the details from that template already filled in.

Chapter 4: Searching a Directory

In JXplorer, you can search for an entry in two ways:

- Quick search using simple criteria
- Complex search using a wider range of criteria. You can save complex searches as filters, join these filters to create new searches, or write your own LDAP filters.

The search results are displayed as complete directory trees, which lets you browse large numbers of search results.

You can save the search results as LDIF files.

This section contains the following topics:

[Quick Searches](#) (see page 17)

[Complex Searches](#) (see page 18)

[Search Operators](#) (see page 22)

[Search Limits](#) (see page 24)

[Bookmarks](#) (see page 24)

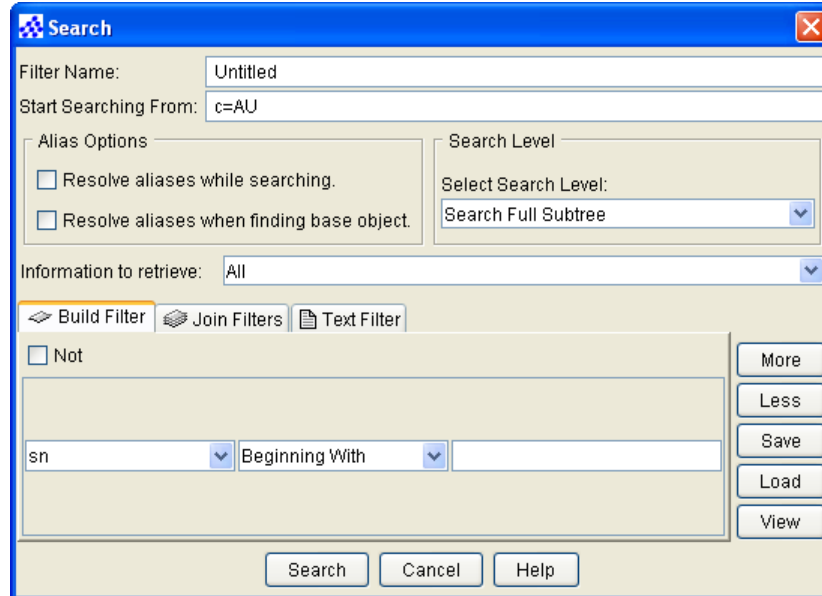
Quick Searches

You can quickly execute simple, single-attribute-value searches using the quick search bar, which contains a pull down list of common attribute types. You can add attributes to this list; the browser saves changes to the list when you exit the browser.

The quick search bar makes it possible to do common searches, for example, specific employee names, part numbers, and so on, without having to access the menu bar or enter a complete LDAP-format search request.

Complex Searches

You can perform more complex searches using the Search dialog.



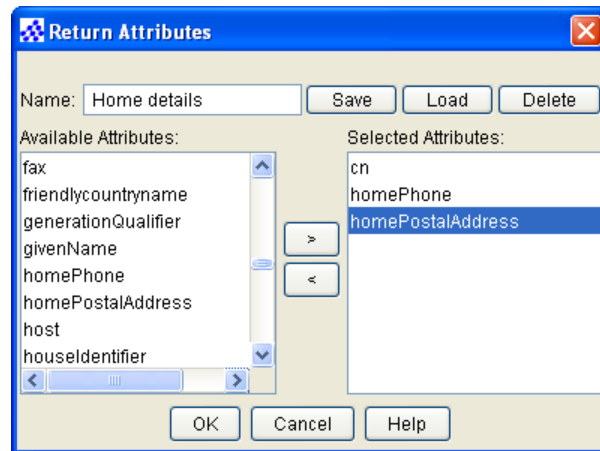
The Search dialog lets you search one of the following:

- The selected entry
- The next level from the selected entry, but not including the selected entry
- The full subtree

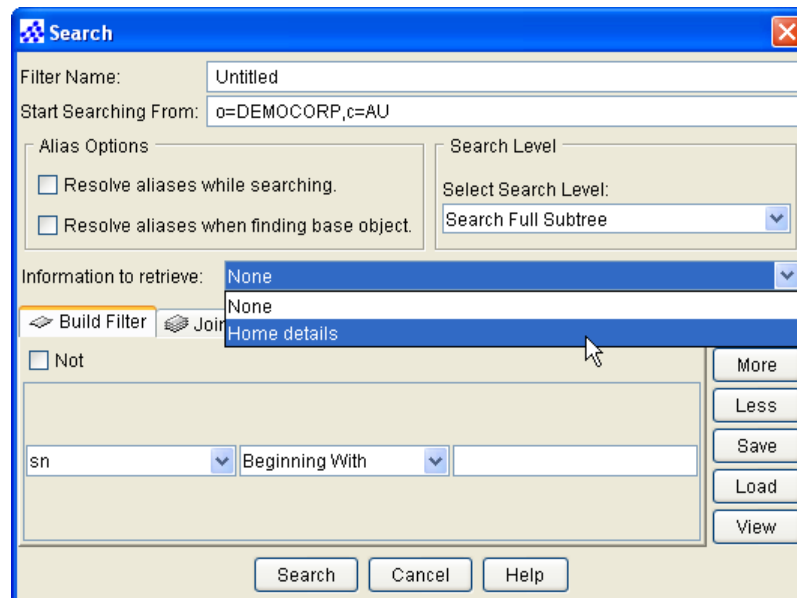
Choose Which Attributes to Return

When you search a DSA, you can define which attributes you want to have returned. The default is none.

To create these definitions, choose Return Attribute Lists from the Search menu:



You can then use the *Information to retrieve* drop-down list in the Search dialog to select the definition that contains the list of attributes you want returned:

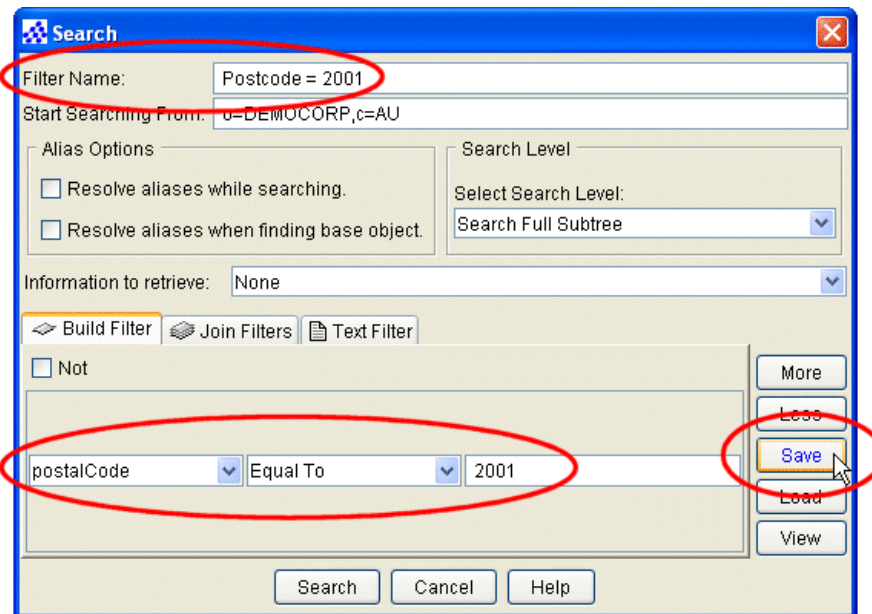


Save Searches for Later Use

JXplorer lets you save searches as filters for later use. Your saved searches appear in the Search menu, so you can access them quickly.

With saved searches, you can:

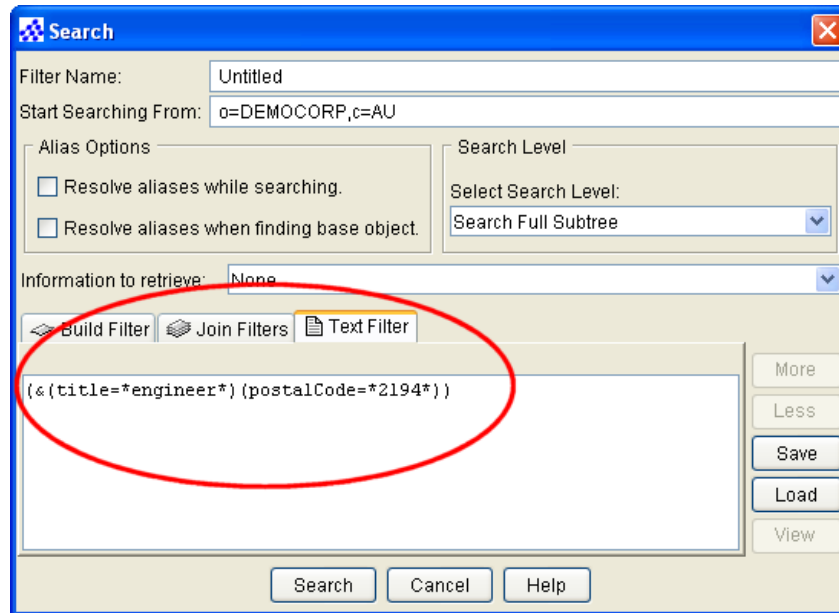
- Save any number of searches as filters
- Edit or delete them at any time
- Copy searches for modification, and save them under a different name
- View saved search filters



Write Your Own Searches

The Quick Search bar and the Search dialog are useful for helping you to build filters.

However, if you already have an LDAP search filter, you do not need to re-create it in one of these dialogs. Instead, you can use the Text Filter tab in the Search dialog:



Search Operators

The following table lists the operators you can use in JXplorer searches, and the effects of these operators. Try applying these example filters to the Democorp DSA.

In the Search dialog	In the Quick Search bar	Example Filter	Result of Example Filter on Democorp Data
Beginning With		sn=Br*	All entries with a surname that begins with <i>Br</i> , including <i>Bruce</i> , <i>Brazier</i> , and <i>Bradley</i>
Not Beginning With		(!(sn=Br*))	All entries other than those in which the surname begins with <i>Br</i>
Containing		sn=*JOR*	All entries with a surname that includes the letter <i>JOR</i> , including <i>Major</i> , <i>Jordan</i> , and <i>Jorgenson</i> .
Not Containing		(!(sn=*a*))	All entries with a surname that does not contain the letter <i>a</i>
Equal To	=	sn=marten	All entries with a surname of <i>Marten</i> .
Not Equal To	~=	(!(title=Secretary))	All entries with a title other than <i>Secretary</i> .
Ending In		sn=*s	All entries with a surname that end with <i>s</i> , including <i>Lucas</i> , <i>Watts</i> , and <i>Giddings</i>
Not Ending In		(!(sn=*s))	All entries with a surname that do not end with <i>s</i>
Greater Than or Equal To	>=	sn>=w	All entries with a surname starting with <i>w</i> , and any subsequent letter of the alphabet, including <i>Whittle</i> , <i>Young</i> , and <i>Ziegler</i>
		postalCode>=6000	All entries with a postalCode greater than or equal to <i>6000</i> , including <i>6000</i> , <i>6018</i> , and <i>7000</i> .
Not Greater Than or Equal To		(!(sn>=w))	All entries with a surname starting with any letter after <i>w</i> , but not including <i>w</i> . This operator is equivalent to "less than".
		(!(postalCode>=2000))	All entries that have a postalCode less than <i>2000</i> , including <i>0810</i> , <i>0860</i> , but not <i>2000</i> . This operator is equivalent to "less than".

In the Search dialog	In the Quick Search bar	Example Filter	Result of Example Filter on Democorp Data
Less Than or Equal To	<=	sn<=b	All entries with a surname starting with the letters up to <i>b</i> , including <i>Anderson</i> , <i>Ash</i> , and <i>B</i> . This does not return entries starting with <i>b</i> and followed by other letters. That is, this search would not return <i>Baker</i> .
		postalCode<=2000	All entries with a postalCode less than or equal to <i>2000</i> , including <i>0810</i> , <i>0860</i> , and <i>2000</i> .
Not Less Than or Equal To		(!(sn<=b))	All entries with a surname starting with any letter before <i>b</i> , but not including <i>b</i> . This operator is equivalent to "greater than".
		(!(postalCode<=2000))	All entries that have a postalCode less than <i>2000</i> , including <i>0810</i> , <i>0860</i> , but not <i>2000</i> . This operator is equivalent to "greater than".
Present		sn=*	All entries with a surname. In the Democorp DSA this returns all of the leaf entries.
Not Present		!(sn=*)	All entries with no data in the attribute <i>sn</i> , and all entries that do not have the attribute <i>sn</i>
Similar To	~=	sn~=marten	All entries with a surname that sounds like <i>marten</i> , including <i>Martin</i> , <i>Martyn</i> , and <i>Martinez</i>
Not Similar To		(!(sn~=marten))	All entries with that do not have a surname similar to <i>marten</i> , including entries that do not have the attribute <i>sn</i>

Search Limits

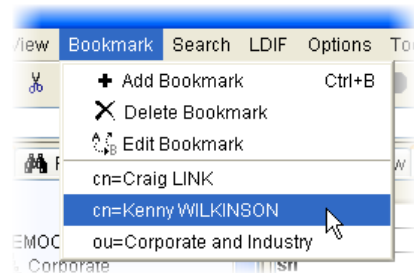
JXplorer lets you define the maximum number of entries returned from a search, and the time (in seconds) allowed to perform the search.

You can set these options from the Advanced Options dialog, which can be accessed via the Options menu. To select the search limits, click on the Search Limits tab, select the LDAP limit and LDAP timeout that you want, and then click the Apply button. To revert the search limits back to the last saved version click the Reset button.

Values can also be set in the server configuration (.dxc) files. When values are set in JXplorer and the server configuration files, the lower of the two values is accepted.

Bookmarks

A bookmark is an entry in the directory that you identify and name for future reference. You can use a bookmark to quickly jump to an entry.



Chapter 5: Editing the Directory

You can modify entries in the directory using the browser in many ways, ranging from slight modification of a single attribute value to large-scale tree operations affecting many thousands of entries.

JXplorer lets you cut, paste, and delete entire directory subtrees using the tree pane on the left. You can manipulate individual entries using the table editor.

You can rename entries from either the directory tree pane or the table editor, depending on what is most convenient at the time.

This section contains the following topics:

[Directory Tree Operations](#) (see page 25)

[Modify Attributes in an Entry](#) (see page 28)

[Attribute Editors](#) (see page 31)

[Binary Values](#) (see page 35)

[Add a New Entry](#) (see page 37)

[Submit an Entry to the Directory](#) (see page 38)

Directory Tree Operations

As you browse the directory tree, you can modify the directory using any of these items:

- The menus
- The toolbars
- Dragging and dropping
- The context menu (accessed by right-clicking on the entry) for the entry in the tree itself

Important! This is a very powerful tool, and you can affect large areas of the directory with a single operation. To avoid accidents, you should select **Confirm Tree Operations** on the Options menu.

Cut, Copy, Paste, and Delete

You can manipulate the directory tree using the cut, copy, paste, and delete operations. On Windows platforms, you can copy and move entries by dragging them with the mouse (“drag and drop”). These operations can be carried out on individual entries within the directory tree or on whole subtrees. Since most directories do not natively support operations on entire subtrees, the client reads and writes all subtree entries recursively, enabling operation on all types of LDAP-compliant directories.

When you select (and therefore display) an entry, all cut, copy, paste, and delete operations occur relative to this selected entry. Specifically:

Delete

Removes the selected entry and any subentries

Copy Branch

Copies the selected entry and any subentries

Cut Branch

Prepares the selected entry and any subentries to be moved to a new location

Paste Branch

Either moves or copies a previously cut or copied entry (and any subentries) under the selected entry as child entries

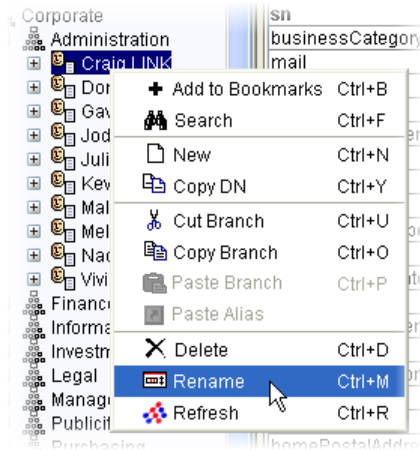
Since some subtree operations involving large numbers of entries can take a significant time to complete, the browser displays a progress bar if it estimates that the operation is extensive.

The progress bar displays the number of entries processed and estimates the proportion of the operation completed. When you want to stop the operation, you can either click Cancel in the Progress, or click the Stop button on the quick search toolbar. When you do this, any changes already made will be kept.

Rename Entries in the Directory Tree

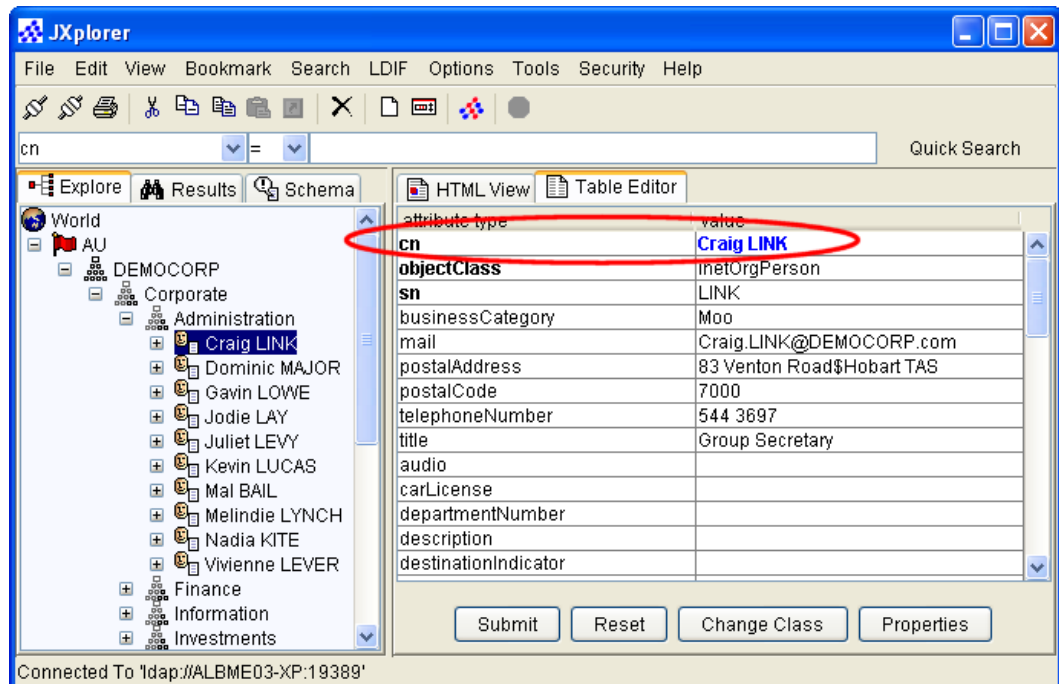
To change the name of an entry, you need to change the value of the naming attribute. The naming attribute is the single attribute used to uniquely identify each entry in the directory.

JXplorer lets you rename an entry within the directory tree by selecting the Rename option:



You can then type a new name for the entry. JXplorer then changes the naming attribute of the entry to the new name.

The naming attribute of an entry is shown in blue in the Table Editor:



When a parent is renamed, the DNs of the entries in the entire subtree under the parent also change.

If you rename an entry with subordinates, the subordinates are also renamed.

Modify Attributes in an Entry

You can modify entries in the table editor, which is a simple tabulated list of attribute names and corresponding attribute values. From the table editor, you can:

- Edit existing values
- Add new attribute/value pairs
- Delete attributes and values
- Copy and paste attribute values
- Manipulate binary attributes
- Submit the results to the directory
- Add or remove naming attributes

These user modifications do not affect the directory until the entry is submitted. When you have finished modifying the entry and checked your work, click the Submit button to send the changed entry to the directory. Only at this point is the data in the directory changed.

Change Attribute Values

You can edit existing values where they are by selecting the appropriate cell in the table and retyping the value.

Note: Binary values, attributes that contain an address, and user passwords are handled differently. See [Using Binary Values](#) (see page 35), [Using the Postal Address Editor](#) (see page 33), and [Entering a User Password](#) (see page 34) for more information.

Delete Values and Attributes

You can delete values (including binary values) using one of the following methods:

- Select the text in the cell, and then press the Delete key to leave an empty cell.
- Right-click on the table row, and then choose Delete Value Attribute from the Context menu.

When you delete the last value of a given attribute, the attribute is also deleted; however, it is not possible to delete the last value of a mandatory attribute, and the browser does not let you submit an entry that does not include mandatory attributes, unless the Ignore Schema Checking option is active. See Mandatory Attributes (see page 29) for more information about mandatory attributes.

Add Values and Attributes

When an attribute does not already have a value, but is available to a particular entry type, you can create it by finding the attribute in the list of blank-valued attributes at the bottom of the table and filling in the missing value. When an attribute already exists and has values, you can add a new value by right-clicking on the attribute name and choosing Add Another Value from the Context menu.

You can add new binary values and addresses this way. See Using Binary Values (see page 35) and Using the Postal Address Editor (see page 33) more information.

Mandatory Attributes

Some attribute names are represented in bold type. These are mandatory attributes, which must have at least one value for the entry.

The browser does not submit an entry if it has any mandatory attributes without at least one value, unless you have selected Ignore Schema Checking on the Options menu..

Naming Attributes

Some attributes are used to name an entry, by forming its relative distinguished name (RDN). Each entry must have at least one naming attribute. Although attributes can have more than one attribute value, only one of these can be chosen as the naming attribute. For example, common name may have two values, Fred and Freddie, but only one can be used as the naming attribute.

To make an entry a naming attribute, select the entry in the table editor, right-click the mouse button, and choose Make Naming Value from the Context Menu. Naming attributes are displayed in the table editor in blue. Multiple naming attributes appear in the tree display with a + symbol joining them.

For example, you may want to name a person by the commonName, or cn attribute, and the surname, or sn attribute. In the case of Craig Link, Craig LINK is the value of the cn naming attribute, and LINK is the value of the sn naming attribute. Both entries appear in the table editor in blue, and in the tree display as Craig LINK + LINK. The order in which the naming attributes appear in the tree display depends on the order in which they are originally entered into the directory.

Change Classes

The object class attribute of an entry determines the attributes that are available for an entry; therefore, you must modify them separately using the Change Classes button, located at the bottom of the table editor. This displays the same list of available object classes as is available in the New Entry panel.

Important! You must be careful when deleting object class values because you also remove all related attributes.

Attribute Editors

Some attributes cannot be easily edited in the Table Editor fields. Instead, JXplorer provides special editors for some kinds of attributes and syntaxes.

When you click on any of the following attributes, an appropriate editor is launched:

- userPassword
- userCertificate
- odSpreadSheetXLS
- odSoundWAV
- odMusicMID
- odMovieAVI
- odDocumentDOC
- ocspRSAPrivateKey
- jpegPhoto
- audio

When you click on any attribute with one of the following syntaxes, and appropriate editor is launched:

- 1.3.6.1.4.1.1466.115.121.1.24 Generalized Time
- 1.3.6.1.4.1.1466.115.121.1.41 Postal Address
- 1.3.6.1.4.1.1466.115.121.1.8 Certificate
- 1.3.6.1.4.1.1466.115.121.1.5 Binary
- 1.3.6.1.4.1.1466.115.121.1.7 Boolean

Work with Audio Files

JXplorer lets you import and export audio files into and out of the directory, using the Audio dialog, which is launched from the Table View:



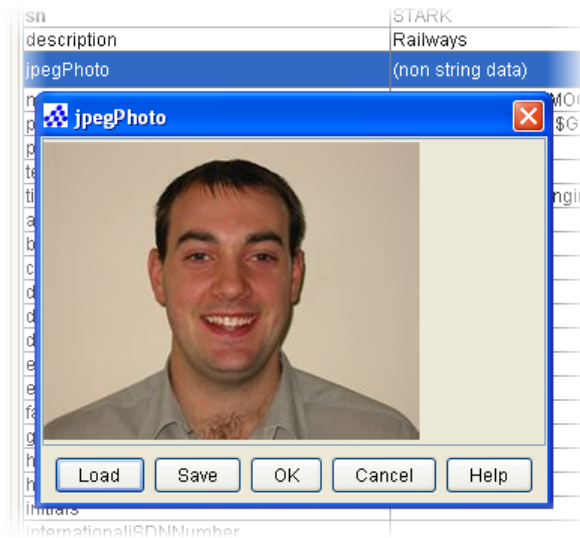
You can import and export audio files of any format. JXplorer also lets you play audio files with the following formats, using the HTML view:

- .aiff
- .au
- .it
- .mid
- .mp3
- .ram
- .rmi
- .s3m
- .stm
- .voc
- .wav
- .xm

Work with Photos

JXplorer lets you import a photo into the directory, and display it in an HTML template. In table editor view, the photo is displayed using the photo editor:

JXplorer lets you import photos through the jpegPhoto dialog, which appears when you select the jpegPhoto attribute type in the Table Editor.

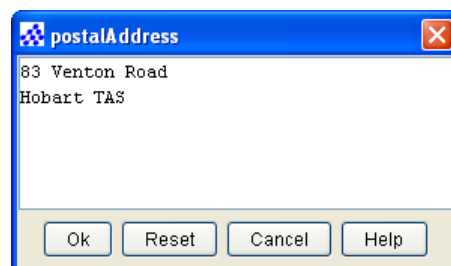


All photos must be in .jpeg or .jpg format.

You can also use this dialog to save the photo to another location.

Work with Postal Addresses

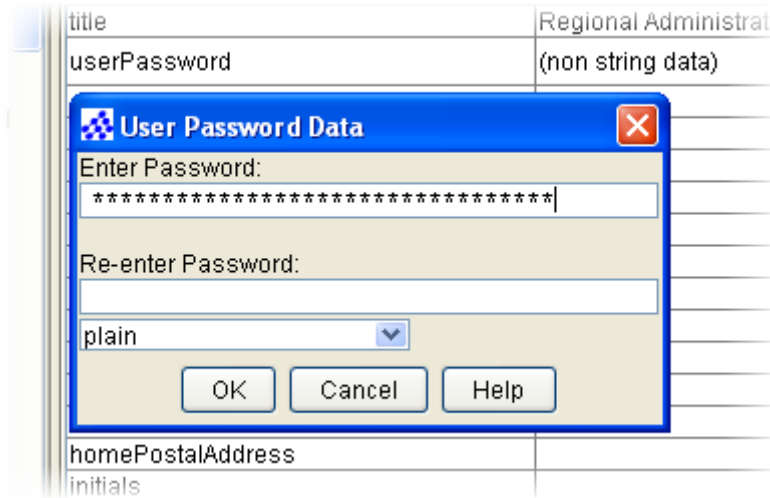
All attributes that have an *address* value, for example, homePostalAddress, are entered through the Postal Address Editor dialog, which appears when you select an address field in the Table Editor:



The dialog lets you enter the details, as they appear in a template with the relevant line breaks and spacing.

Work with User Passwords

If you use the Table Editor view to edit a user password, a dialog opens that lets you create a new password and change the way the password is stored (plain text, MD5 encryption or SHA encryption):



If you edit a user password from the HTML view, you can only change the password:



Access to an entry is controlled via the access control settings in the directory's configuration file (.dxc). This means that you do not have to enter the existing password before changing it.

Binary Values

LDAP is primarily a string handling protocol and many attribute values are simple text strings. However, it is often necessary to load other types of data, for example, certificates and images.

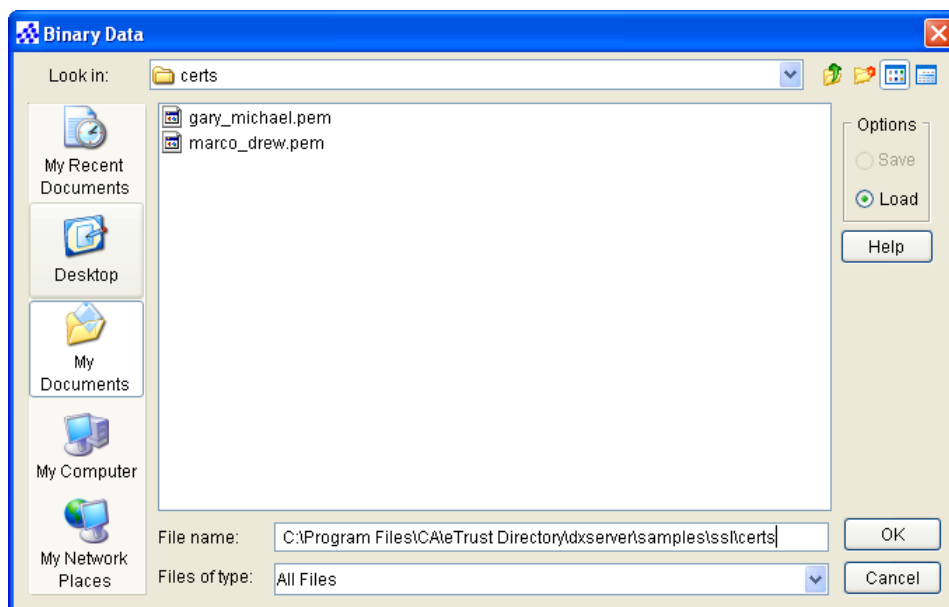
JXplorer allows you to load binary files to some attributes, such as jpegPhoto and userPKCS12.

The browser also supports custom binary editors (written in Java using a provided minimal API) that dynamically loads at run time. You key such binary editor extensions to a particular object class. This would let you write, for example, an editor for a custom certificate object class.

Standard editors are provided for X.509 certificates, and a number of standard image and audio formats.

Import Binary Files

With the following Binary Data dialog, you can import and export binary files:



Files that Can Be Launched

eTrust Directory provides the odMultimedia object class that contains attribute types that let you launch files of the following formats:

.avi

odMovieAVI

.doc

odDocumentDOC

.mid

odMusicMID

.wav

odSoundWAV

.xls

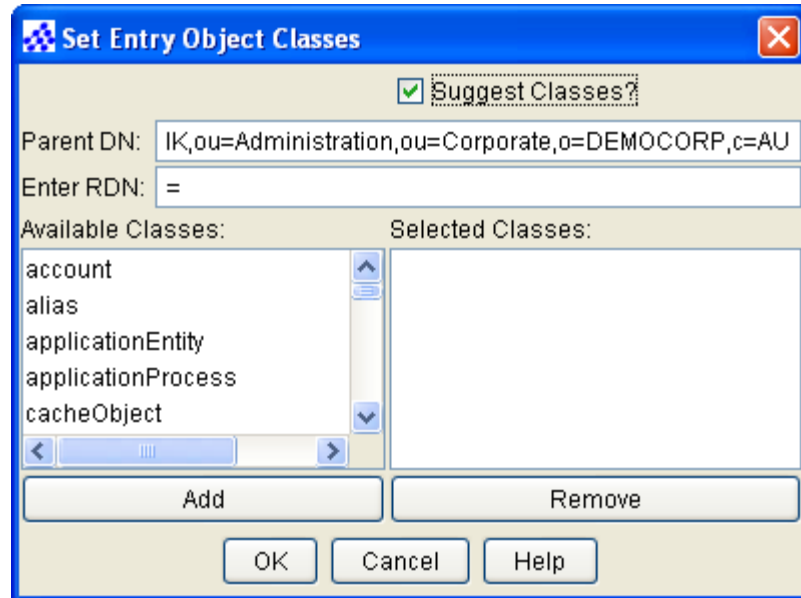
odSpreadSheetXLS

A dialog appears when you click the value field of one of these attribute types in the table editor.

Add a New Entry

You can add an entry by selecting the New option.

The new entry is created as a child of the currently selected entry in the tree. When a new entry is created, you must specify the entry's RDN and list its object classes.



Choose Object Classes

When there are other children of the selected entry, the browser suggests object classes based on those children; otherwise, you must select them. (To turn this behavior off, clear the Suggest Classes checkbox). Since the browser is schema aware, it can fill in any required parent classes.

The choice of object classes must conform to the restrictions laid down by the schema. The server may also have additional schema rules restricting where entries of a particular type are created. For example, it may not be possible to create a country entry underneath an organizationalUnit entry.

Set Initial Attribute Values

When all information is entered in the new entry dialog, the entry is set up in the browser, and you can fill in the entry's attributes in the table editor. Before the entry is actually created in the directory, you must enter the information for the attributes and submit them-especially mandatory attributes.

Submit an Entry to the Directory

Once a new entry has been filled in, or an existing entry has been modified, you must submit the result to the directory. The browser checks for consistency using schema information (unless you have selected Ignore Schema Checking in the Options menu), and the directory checks the entry again when it is submitted.

If the entry is invalid, the browser reports the directory error to you, but leaves your changes unaltered in the edit table. To discard your changes, click the Reset button.

Submitted entries are:

- Checked for gross errors by the browser.
- Submitted to the directory through LDAP.
- Checked for errors by the directory, after which either:
 - The user is informed if an error has occurred.
 - If no error has occurred, the browser display tree is updated.

Chapter 6: Importing and Exporting Data

You can import an LDIF file into JXplorer, edit it, and then save the LDIF file.

When JXplorer is connected to a directory, it reads the values from the selected file and adds them to the directory, or reads the values from the directory and writes them to an LDIF file.

JXplorer does the following:

- Lets the prefix of the DNs in the LDIF file be replaced when reading or writing an entry to assist in data migration between directories
- Automatically handles base-64-encoded binary LDIF data
- Flags (with the help of the directory) when LDIF data entries are invalid

In addition, JXplorer provides a status display when it estimates that a large import or export operation is taking place. The status display shows the number of entries processed and the estimated proportion processed. Click Cancel when you want to quit a long operation.

This section contains the following topics:

[Binary Values in LDIF Files](#) (see page 39)

[Use an LDIF File Without a Directory](#) (see page 40)

Binary Values in LDIF Files

Binary values in LDIF files are stored in base 64 format. This means that you can copy and paste binary values between JXplorer and LDIF files with the same ease as other string values.

For more information about base 64 encoding, see MIME (Multipurpose Internet Mail Extensions) Part One (<http://www.ietf.org/rfc/rfc1521.txt>).

Use an LDIF File Without a Directory

An added feature of JXplorer is that it lets you use an LDIF file directly as a miniature directory without any LDAP connection to a directory server. Using an LDIF file offline in this way can be useful for:

- Editing during data migration
- Caching data over a slow communication link
- Stand-alone demonstrations (on laptops, for example)
- Reviewing data before committing it to a production environment

Because there is no communication lag, you may navigate the offline LDIF file faster than using a directory. You can also edit the LDIF file, and add and manipulate binary values. The only restriction in the use of offline LDIF files is that they cannot be searched. To search an LDIF file, you must load the file in a directory (or the raw LDIF file viewed using a text editor).

Chapter 7: Resolving Aliases

An alias is a directory entry that contains the name of another entry. When you search or browse a directory, you can decide whether to resolve aliases (show the details of the target entry) or to show the details of the alias entry itself.

Aliases are similar to shortcuts and are used in some directories to link different parts of the tree. You can also use the Copy Branch and Copy DN functions to copy the name of an entry for pasting into any of JXplorer's text entry fields, or into your own documents.

This section contains the following topics:

[How JXplorer Displays Aliases](#) (see page 41)

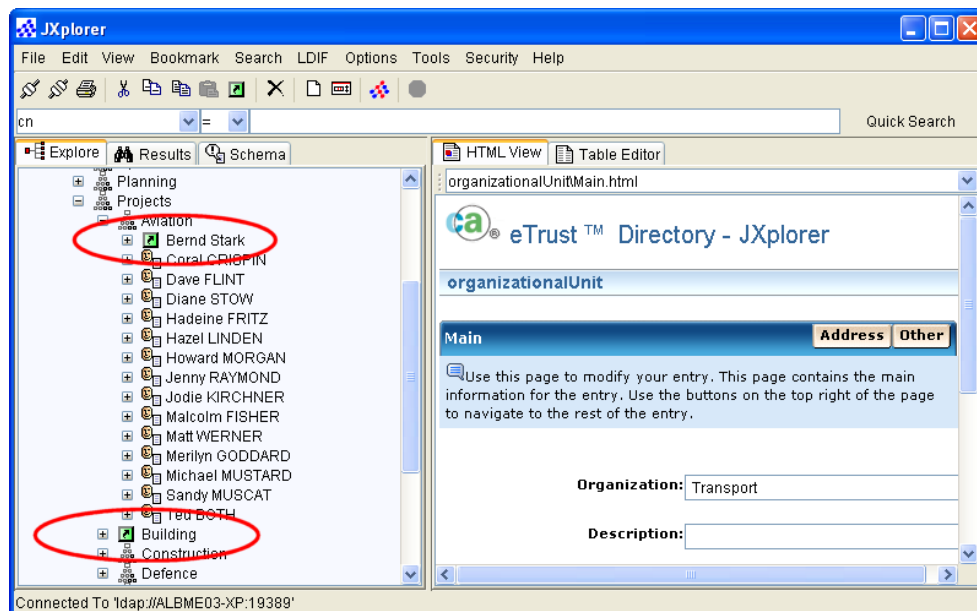
[Create a New Alias Entry](#) (see page 42)

[Resolve Aliases While Browsing](#) (see page 42)

[Resolve Aliases While Searching](#) (see page 44)

How JXplorer Displays Aliases

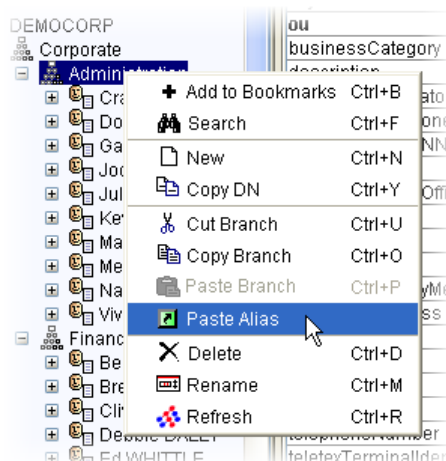
Alias entries are displayed with a green icon in the tree pane. The following screenshot shows two alias entries:



You can choose to resolve aliases while browsing and while searching.

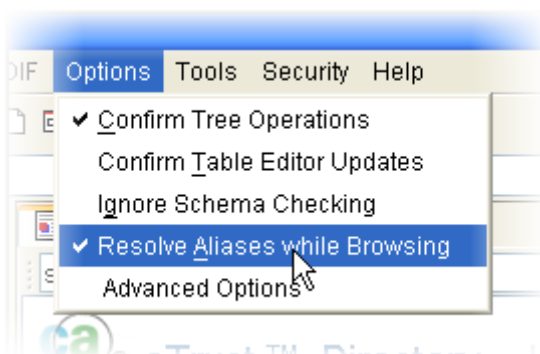
Create a New Alias Entry

You can create a new alias entry by copying an entry, and then using the Paste Alias option:



Resolve Aliases While Browsing

JXplorer lets you choose whether to show resolve aliases while you are browsing the tree.



If you choose to resolve aliases, when you browse to an entry that is an alias, the details of the target entry will be displayed. There will be no indication that you have browsed to an alias entry.

If you choose to *not* resolve aliases, when you browse to an entry that is an alias, the details of the selected entry will be displayed. This means that you will see the DN of the target entry, instead of the details of the target entry.

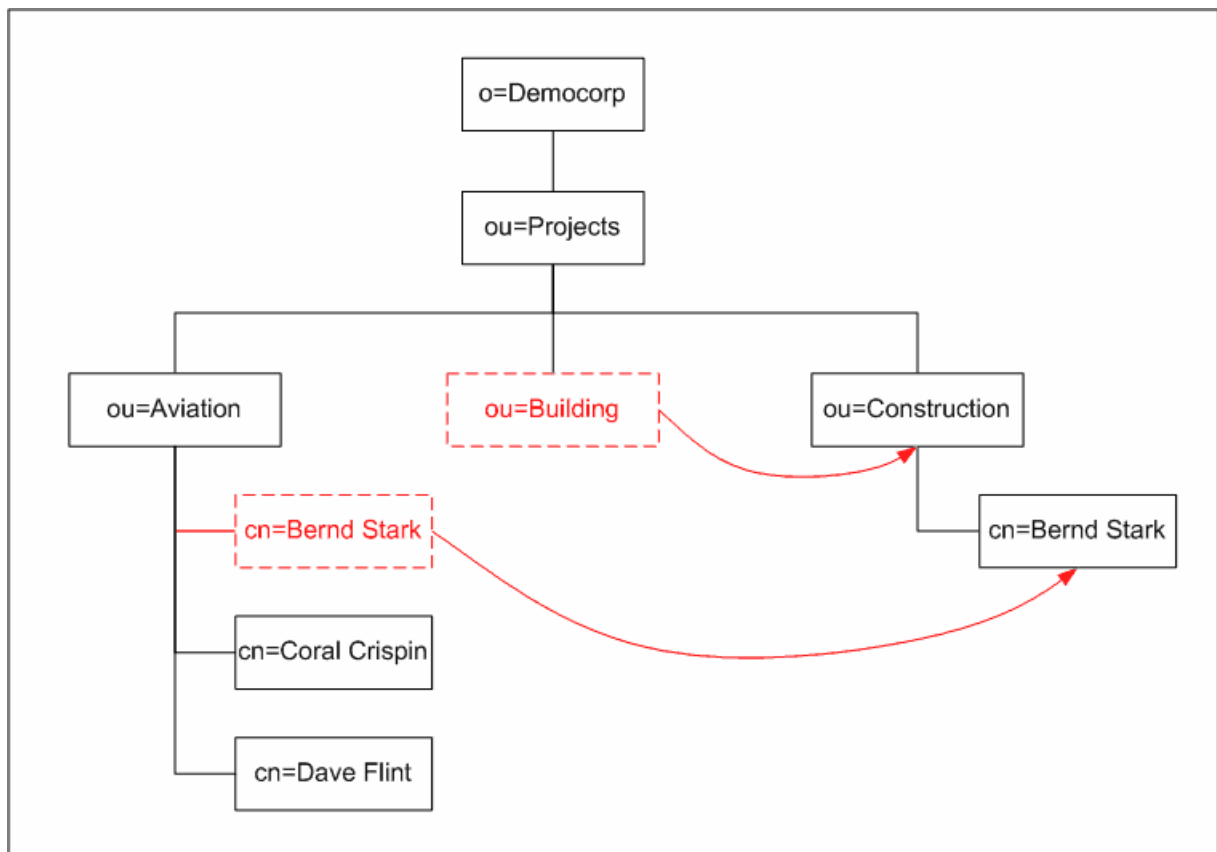
Note: This only applies to browsing the tree. When you search the directory, this option is ignored.

Example: Resolving Aliases While Browsing

This example shows part of the directory information tree for the Democorp DSA. The tree has been edited to include the following two aliases:

- In the Aviation subtree, a new alias entry has been created. This alias entry points to the Bernd Stark entry in the Construction sub-tree.
- In the Projects subtree, a new alias entry that points to the Construction entry.

If you select **Resolve Aliases While Browsing** in the Options menu and then browse the directory tree to Bernd Stark's entry under `ou=Aviation`, you will see all of his details. However, if you de-select **Resolve Aliases While Browsing** in the Options menu and then do the same thing, you will see the details of the alias entry (that is, "`cn=Bernd STARK,ou=Construction,ou=Projects, o=DEMOCORP,c=AU`"), but no details about Bernd Stark.

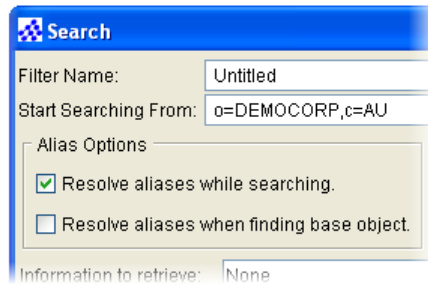


This table shows the data that will be displayed in different circumstances:

Resolve Aliases While Browsing	Browse To	Display
Selected	cn=Bernd STARK,ou=Construction, ou=Aviation , o=DEMOCORP,c=AU	Full details of Bernd Stark, as stored in his entry in the Construction subtree
De-selected	cn=Bernd STARK,ou=Construction, ou=Aviation , o=DEMOCORP,c=AU	Details of the alias entry (that is, "cn=Bernd STARK,ou=Construction,ou=Projects,o=DEMOCORP,c=AU"), but no details about Bernd Stark
Selected	cn=Bernd STARK,ou=Construction, ou=Projects , o=DEMOCORP,c=AU	Full details of Bernd Stark
De-selected	cn=Bernd STARK,ou=Construction, ou=Projects , o=DEMOCORP,c=AU	Full details of Bernd Stark

Resolve Aliases While Searching

When you set up a complex search using the Search dialog, you can choose whether to resolve aliases during the search:



When you resolve aliases, JXplorer returns the real entry to which the alias points. When you do not resolve aliases, JXplorer returns all alias entries as regular entries.

However, each search operation has two components: the search for the base object (if you specified one), and the search for the entries to return. you can choose whether to resolve aliases in each of these components.

Chapter 8: Logging and Troubleshooting

This section contains the following topics:

[Logging](#) (see page 47)

[Troubleshooting](#) (see page 47)

Logging

There are a number of logging options available, ranging from no logging at all to complete tracing of all the LDAP communications with the directory.

The client can log data to a log file, to the console window, to both, or not at all.

To change the level and target of the logging, use the Advanced Options dialog. For more information, see [Set the Logging Level](#) and [Set the Logging Method](#).

Troubleshooting

If you experience trouble while using JXplorer, you can run the *console.bat* utility provided in the JXplorer home directory, which displays any problems.

Chapter 9: Customizing JXplorer

You can customize the following JXplorer features:

- HTML templates
- Logging and tracing
- Directory tree icons

This section contains the following topics:

[Why Customize the JXplorer Interface?](#) (see page 49)

[Customize Tree Icons](#) (see page 50)

[Create HTML Viewing Templates](#) (see page 52)

[Customize HTML Forms](#) (see page 55)

[Add Custom HTML Pages](#) (see page 61)

[Internationalize JXplorer](#) (see page 62)

[Supply Customized Files](#) (see page 66)

Why Customize the JXplorer Interface?

You do not have to customize JXplorer if you do not want to. JXplorer already works with any LDAP directory.

You can already use JXplorer to modify the structure of your directory with the tree view (cut/copy/paste/delete of entries and sub-trees), and you can edit individual entry attribute values using the table editor.

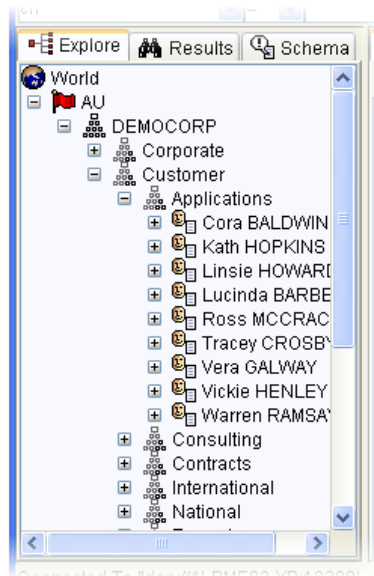
However, you might want to customize JXplorer:

- To change the JXplorer interface
- To change the way JXplorer behaves, to provide more advanced functionality.

Customize Tree Icons

You can add or modify the icons that JXplorer displays in the tree in the left pane.

A small set of icons is included in the standard release.



The icons displayed in the JXplorer tree are stored in the icons sub-directory.

The browser reads the icons used in the directory display tree from the /icons subdirectory of the JXplorer home directory. The icons are 16-pixel-square images in the form:

object_class_name.gif

You can easily replace the icons.

Add a Custom Icon

To make sure the browser recognizes the new icons, you must restart it.

To add a new icon

1. Create a 16 x 16 pixel GIF file of the icon.
2. Give the icon the same name as the naming attribute you want the icon to appear next to.
For example, you could create an icon named **cn.gif**, which would appear next to entries that use **cn** as their naming attribute.
3. Save the GIF file in the **jxplorer\icons** directory

The next time you start JXplorer, the new icon will be used.

Display Entry-Specific Icons

The icons are linked with entries using the object class of the entry, or the LDAP naming attribute in the lowest RDN of the entry.

For example, this means that an entry with a name **ou=R&D,o=acmecorp,c=us** will be labeled with the icon **organizationalUnit.gif**, or if that doesn't exist, with **ou.gif**. The **organizationalUnit.gif** icon is shown as a gray org tree symbol in the tree above.

Create HTML Viewing Templates

You can create HTML templates and place them in a file directory hierarchy under the /templates subdirectory of the JXplorer root directory. When a shared template directory exists on the file system, you can configure JXplorer to use that directory instead by editing the dxconfig.txt configuration file in the JXplorer directory.

Place templates in file directories corresponding to object class names. Within these file directories, the templates can have any name (although names that include spaces are not recommended). Templates placed directly in the /templates directory are common to all object classes. For example, the following file directory structure provides a general template, two templates for person, and a default template for organizationalUnit:

/templates/

general.html

/templates/person

employee.html

contractor.html

/templates/organizationalUnit

default.html

You can add any number of new HTML templates, including templates for newly defined object classes by creating (if necessary) the appropriate subdirectory under the /templates directory and placing the new HTML files in that subdirectory. After you add new files, you may need to restart the browser to recognize them.

HTML Tag Extensions

The HTML language is extended using a modification to the HTML <comment> tag to set placeholders where attribute values can be filled in.

These extension tags:

- Position individual attribute names and values in the page
- List of all attribute names and values to be set with a single tag
- Provide a variety of tabulated formatting options, such as HTML tables and lists

See the JXplorer online help for more information.

HTML Forms

You can also use standard HTML forms; however, you must give each form element a name value that corresponds to an attribute, for example, title. JXplorer will display existing values and make updates to the directory when you click Submit.

Important! JXplorer submits the changes to the directory, not to a Web server.

A number of example HTML forms are in the templates sub-directory. For more information, see [Customize HTML Forms](#) (see page 55).

Example: Customized JXplorer GUIs

There are a number of reasons you might want to change the GUI, including:

- Corporate Branding - providing a distinct look and feel for a client or a product.
- Simpler Interface - providing tools to do commonly used activities quickly.
- Providing Extended Help - you may want to provide more information to the user.
- Data Specific Display - you may want special icons for some types of data, or special displays for specific entry types.

Corporate Branding Example

The following example shows JXplorer with custom icons (see page 50) and corporate-specific graphics. These graphics were incorporated using a simple extended version (see page 61) of HTML

The screenshot shows a web browser window with a tab labeled 'person_individual'. The page features the 'Computer Associates' logo and a navigation menu with options: Person, Home, Work, Delivery, and Advanced. The main content area is titled 'Individual Record' and includes a legend: '* = Mandatory Field'. The form contains the following fields:

Title:	<input type="text"/>	Personal Title
Name:	<input type="text" value="Craig LINK"/>	* Common Name E.g., "Robin McLeod"
Surname:	<input type="text" value="LINK"/>	* Surname E.g., "McLeod"
Password:	<input type="password"/>	User Password
Email:	<input type="text" value="Craig.LINK@DemoCorp.com"/>	Email Address E.g., "Some.One@ca.com"
Mobile Phone:	<input type="text"/>	Mobile Phone Number E.g., "0400 007 007"
Pager:	<input type="text"/>	Pager Number
Drink:	<input type="text"/>	Favorite Drink

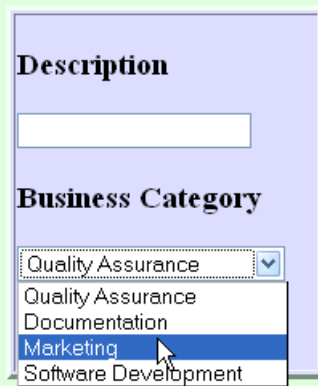
At the bottom of the form are two buttons: 'Update' and 'Reset'.

Data Entry Example

If you were using JXplorer in a help desk or data entry environment, it is straight-forward to build custom HTML forms that only allow data entry for a handful of attributes, and possibly restrict the allowed values, or provide more on screen help information.

Organisational Unit Data Set

Please Update Entry



Description

Business Category

Quality Assurance

Quality Assurance

Documentation

Marketing

Software Development

Customize HTML Forms

You can use HTML forms to enter data into the directory. Unlike the HTML page display, you do not need any custom tags at all; it is possible to use plain HTML forms.

The Java HTML component does not support any active scripting. It works with plain HTML only.

It is quite legitimate to combine the special DXAttribute tag of the previous chapter with these custom forms.

Be careful not to include extra fields containing attributes that don't exist in the directory. If you do this, and a user attempts to enter data into the field, the directory will show an 'unknown attribute' error. This is not particularly serious, but it will confuse your users.

Form elements that have a 'value=""' clause will (where possible) have this value filled in by the existing entry value. This is not done for non-text components however.

Example: Customized HTML

Showing the value of description:

```
<dxtemplate:get-attribute name="description"/>
```

Showing all addresses in an HTML list:

```
<dxtemplate:get-attribute name="address" style="list"/>
```

Showing all attribute values in a table:

```
<dxtemplate:get-all-attributes style="table"/>
```

Listing all available data in the entry, without needing to explicitly name the various attributes:



Simple view: displaying all attributes...

<code>cn</code>	* Brigitte HOUGH
<code>description</code>	* Periodicals
<code>mail</code>	* Brigitte.HOUGH@DEMOCORP.com
<code>objectClass</code>	* inetOrgPerson

Sample Templates

The above page for viewing 'person' data was saved under `/templates/person/`. Look at the existing subdirectories under `templates` to see some examples.

If you have a template that you want to be available for all entries, regardless of object class, place it at the base `'templates'` subdirectory level, and it will always appear as an option.

Name the Buttons

Anything with a `'name'` tag is parsed as an attribute. This means that form submission buttons and so on will be treated as attributes if they are given names.

To avoid this, don't give your Submit buttons specific names, unless you have a special reason to do this.

While you shouldn't give buttons a `'name'` clause, you can certainly give them a `'value'` clause, to get the button to display something other than `'submit'`.

Name the Form Elements

The form elements must be named after the attribute they are to modify. For example, to modify a description attribute, you could use the following form HTML

```
<input type="text" name="description" size=20 value="">
```

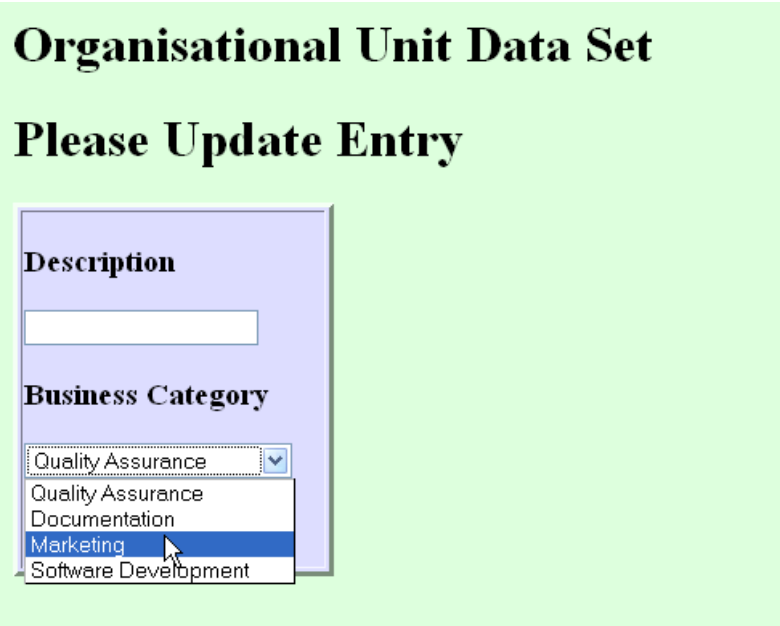
This gives a normal text input field that modifies the description attribute when the user clicks the Submit button.

Example Customized HTML Forms

Any form component can be used, although the data returned may not always be appropriate for a particular attribute type.

Example: organizationUnit HTML Form

The organizationUnit example HTML file shows how to use the description field above as well as a pull-down list:



Organisational Unit Data Set

Please Update Entry

Description

Business Category

- Quality Assurance
- Quality Assurance
- Documentation
- Marketing
- Software Development

You must use the normal HTML form syntax, including a submission URL. However, the form does not get submitted to that URL - JXplorer intercepts the submission event and parses the data itself.

Example: Script to Create HTML Form

The following code is the script used to generate the directory-enabled HTML form shown above. It is a complete HTML page, including both an attribute tag and a form.

```
<html>
  <head><title>First Template</title></head>
  <body bgcolor="#DDFFDD">
    <h1>Organisational Unit Data Set</h1>
    <h2><dxtemplate:get-attribute name="ou" style="list"/></h2>
    <p>
      <h1> Please Update Entry </h1>
      <table width=200 border=3 bgcolor="#DDDDFF">
        <tr>
          <td>
            <form name="temp" action="http://www.pegacat.com" method=get>
              <h3>Description</h3>
              <input type="text" name="description" size=20 value="">
              <h3>Business Category</h3>
              <select name="businessCategory">
                <option value="Quality Assurance">Quality Assurance
                <option value="Documentation">Documentation
                <option value="Marketing">Marketing
                <option value="Software Development">Software
                Development</select>
              <p>
                <input type="submit">
              </form>
            </td>
          </tr>
        </table>
      </body>
    </html>
```

Add Custom HTML Pages

JXplorer loads HTML pages into the HTML view panel based on the object class(s) of the entry being viewed. For example, when viewing a 'person' entry, it looks for a 'person' subdirectory under the 'templates' subdirectory, and lists all the available html files within that person subdirectory as options to the user.

Within these HTML files, an 'extended' HTML tag is used to allow the display of entry data. Within the HTML code, the following tag may be used:

```
<dxtemplate:get-attribute name="[attribute name]" style="[display type]"/>.
```

[Attribute name] is the exact name of the attribute to display values of. The optional

[display type] is one of either 'list', 'table', or 'plain', which sets how multiple values will be displayed. To show a list of all available attributes. To display all the attributes and their values a similar tag is used:

```
<dxtemplate:get-all-attributes style="[display type]"/>
```

You can insert hyperlinks in your HTML pages. For example, you could link to extended help, or further resources.

However, that the Java HTML component does not support Javascript or VB script, and is not as robust as a commercial web browser.

Internationalize JXplorer

It is easy to add new languages to JXplorer. You can choose what parts of JXplorer you internationalize.

You can internationalize the following:

- The HTML templates
This means that the text strings on each JXplorer page appears in the new language.
- The Welcome page
- The online help

Translate HTML Templates

To internationalize an HTML template

1. Create the new templates in UTF8, 16 bit UNICODE, or a local encoding (such as Shift-JIS in Japan).
UTF8 is preferable, because local encodings won't work outside computers set to that locale.
2. If you want to check your work in an English-language browser, make sure you use the correct HTML meta header. For example:

```
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
```

Translate Help Files

The online help system for JXplorer uses the JavaHelp 1.1.3 format. The online help is supplied as a helpset.

To internationalize the JXplorer online help

1. Create a new version of the help in a different language.
2. Give the root 'helpset.hs' file a country-specific name of 'JXplorerHelp_[my language].hs'. For example, the Chinese helpset would be named JXplorerHelp_ch.hs.
3. Allow JXplorer to see the translated helpset. You can do either of the following:
 - Make a ZIP file of the translated helpset, then place it in the /plugins directory.
 - Add it to the JXplorer class path.

For information about the JavaHelp format, see <http://java.sun.com/products/javahelp/> (<http://java.sun.com/products/javahelp/>).

For information about internationalizing JavaHelp, see the JavaHelp User Guide. Look in the Localizing Help Information section.

Translate the Welcome Page

You can internationalize the Welcome page that appears when JXplorer starts.

To do this, replace the /htmldocs/start.html file with language specific files of the form 'start_[language code].html'. For example, for Japanese, this file would be start_ja.html.

Make sure that these new files are written in UTF8, Unicode, or the default local encoding.

Files Used in Translation

JXplorer is written using English strings. It then uses resource files to map the English strings to the translated strings. These resource files are located in the /language subdirectory.

Each resource file is made up of lines of words and phrases. Each line has a word or phrase on the left, an equals sign, and the translation on the right. For example, a map file for German might contain the following two lines:

```
HeIp = Hilfe  
Welcome to JXplorer = Willkommen du JXplorer
```

Any symbol on the *left* of the equals sign (including another equals sign) may be escaped with a backslash, but for most strings this is not necessary.

Use the Template Map File

JXplorer comes with a template translation file named JX.translateMe.txt. This file provides a list of strings on the left side, with the translations on the right side remaining blank.

To use this template file

1. Fill in the translated strings on the right side of each line.
2. Save the file in UTF8 format if possible.
You can also use 16-bit Unicode, local character encoding, or Java Unicode escape format.
3. Name the file according to the following format: JX[_country abbreviation][_dialect].properties.
For example, a Japanese translation file should be named /language/JX_ja.properties, where ja is the standard two letter internet code for Japanese.
In another example, a French Canadian translation file should be named /language/JX_fr_CA.
4. Place the file in the /language directory.

Test Your Translation

To test your translation file

1. Back up the existing files in the /languages directory.
2. Add the translated strings to the translation files.
3. Run JXplorer
 - If you are on an English platform and you made changes to the right side translations in the file JX.properties, you will see those changes the next time JXplorer runs.
 - If you are in a non-English locale, copy the JX.properties file to a JX_[my language].properties file, and start making changes there. These changes will be visible the next time you run JXplorer.

For more information, see the Sun documentation on Java resource bundles (<http://java.sun.com/j2se/1.3/docs/api/java/util/ResourceBundle.html>).

File Formats

Many products have very restrictive rules about what file formats they will accept non-English text in. JXplorer prefers files to use UTF-8 or 16 bit unicode, but local character encodings (such as Japanese 'Shift-JIS' or Chinese 'Big-5') will usually work as well.

If your file might be used outside your locale, or if you need to include multiple languages, you should use Unicode (either UTF-8 or 16 bit). This is because locale-specific formats will only work on computers that are set to that locale. That is, 'Shift-JIS' only works on computers that are set to the Japanese region setting.

Unicode is used internally throughout JXplorer.

Occasionally you may see text of the form '`...\u30AF\u309A...`'. This is Java-escaped Unicode format. You can use this format, but most word processors will not save to this format, and it is not recommended.

Fonts

While many fonts may exist on a system, Java does not automatically pick them up.

If you need to install new fonts, use this Sun document for instructions: <http://java.sun.com/j2se/1.4.2/docs/guide/intl/fontprop.html> (<http://java.sun.com/j2se/1.4.2/docs/guide/intl/fontprop.html>).

Supply Customized Files

Depending on how much customization you've done, you may have new icon files, HTML forms, HTML templates, and images.

If you are writing your own Java plug-in, and you need to use external data files, you can access them using normal java zip handling methods. In addition a utility class `'/com/cai/od/cbutil/CBJarResource.class'` is available (see the API documentation) and can be imported by your plug-in.

To supply the changed files in a ZIP file:

1. Put the changed files in a ZIP file.
2. Send the ZIP file to the end users.
3. Ask the users to place the ZIP file in the plugins directory.

JXplorer automatically scans the plug-ins directory for .zip and .jar files when JXplorer is started. It reads the indexes of the zip files, and afterwards treats any HTML templates, icons, and plug-ins in a ZIP file as being part of the normal class path.

Automatic loading of tree icons and plug-ins is available in JXplorer 1.2.
Automatic loading of HTML templates from ZIP files is available in JXplorer 1.3.

Supply the Changed Files Directly

If you have customized only a small number of files, you can send your end users the files directly, with instructions or a batch file to place them in the correct directory.

Supply the Changed Files in a ZIP File

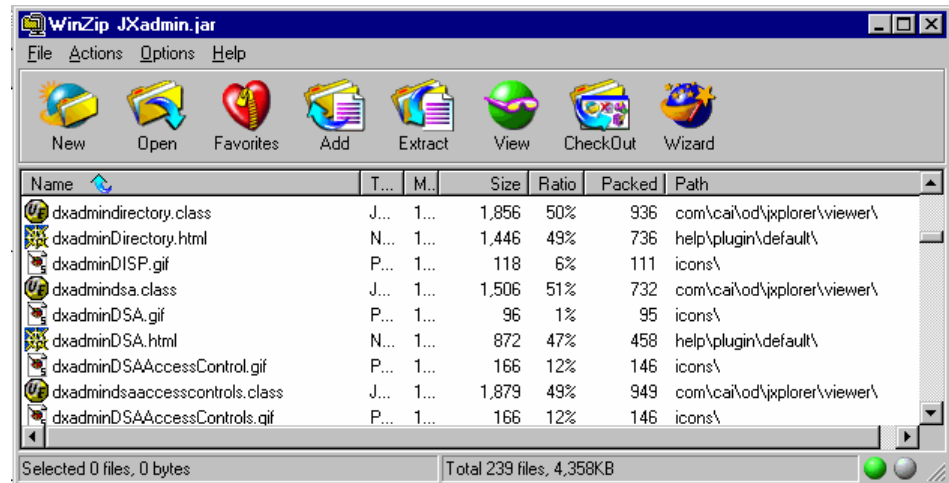
1. If you have a large number of files, you may need to make changes to the /icons and /plugins directory, and create new subdirectories in the /templates directory. To supply the changed files in a ZIP file:
2. Put the changed files in a ZIP file.
3. Send the ZIP file to the end users.
4. Ask the users to place the ZIP file in the plugins directory.

JXplorer automatically scans the plugins directory for .zip and .jar files when JXplorer is started. It reads the indexes of the zip files, and afterwards treats any HTML templates, icons, and plugins in a ZIP file as being part of the normal class path.

Structure of the ZIP File

If you use a ZIP file to supply the changed files to your users, the ZIP file must have the same structure as the external directories.

That is, templates must be in the ZIP file under a /templates directory, icons under a /icons directory, and so on. For example:



Chapter 10: How JXplorer Reads the Schema

After the connection details are obtained, the browser attempts to contact the directory.

After a connection to the directory has been made (through LDAP v3 or DSML), JXplorer obtains the directory's current schema. Directories that support LDAP v3 (such as eTrust Directory) download the schema to the browser.

This lets the browser correctly create, display, and edit entries without requiring any independent browser configuration. Since the browser gets the schema from the directory, it is always up-to-date, and there is no possibility of inconsistency.

If you connect to the directory using DSML instead of LDAP, JXplorer can still get the schema, as long as the DSML server can use LDAP v3 to communicate with the DSA.

This section contains the following topics:

[Data in Each Schema Object](#) (see page 70)

[Checking Entries for Schema Conformance](#) (see page 71)

Data in Each Schema Object

A schema contains the following schema objects:

Attribute Types

Attribute types define the attribute's syntax and how the attribute is compared and sorted.

LDAP Syntaxes

LDAP syntaxes describe the representation of the attribute's value.

Object Classes

Object classes define what kind of attributes an entry can contain.

DIT Structure Rules

DIT structure rules define where entries appear in the directory information tree. DIT structure rules are part of the name bindings.

Name Forms

Name forms specify which attributes the entry can be named by. Name forms are part of the name bindings.

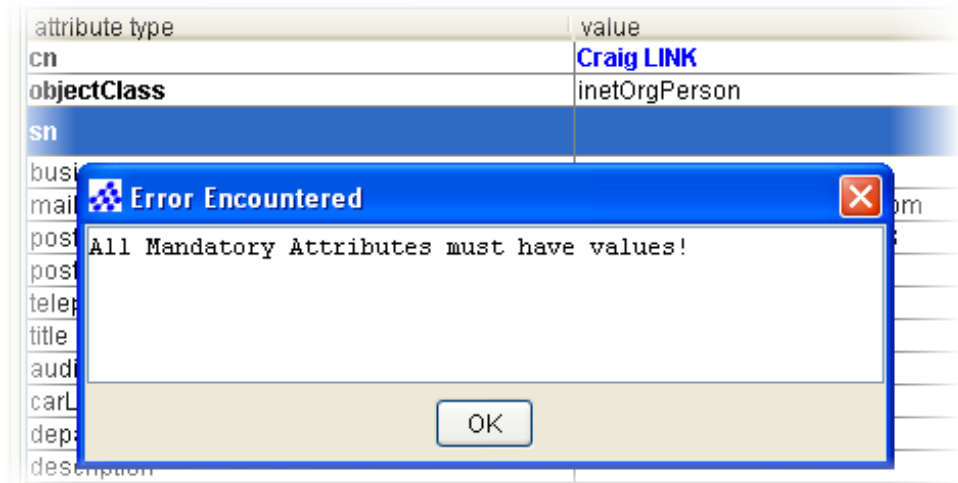
This table lists the data contained in each of these schema objects:

	Attribute Types	LDAP Syntaxes	Object Classes	DIT Structure Rules	Name Forms
OID	Yes	Yes	Yes	Yes	Yes
NAME	Yes		Yes	Yes	Yes
DESC	Yes	Yes	Yes		
SUP	Yes		Yes	Yes	
SYNTAX	Yes				
SYNTAX Description	Yes				
EQUALITY	Yes				
SINGLE-VALUED	Yes				
OC					Yes
MUST			Yes		Yes
MAY			Yes		Yes
FORM				Yes	

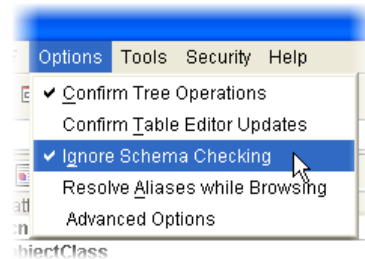
Checking Entries for Schema Conformance

eTrust Directory automatically checks all entries submitted to the directory to ensure that they conform to the directory schema.

JXplorer also checks that each entry conforms to the schema. For example, if you try to submit an entry that has a mandatory attribute with no value, JXplorer rejects the change you have made:



If you do not want JXplorer to check that each entry conforms to the schema, select the Ignore Schema Checking option in the Options menu:



Chapter 11: How JXplorer Handles Passwords

This section discusses how JXplorer handles passwords.

This section contains the following topics:

[Password Storage](#) (see page 73)

[Password Hashing](#) (see page 74)

Password Storage

When you use the Connection dialog to connect to a DSA using a password, JXplorer saves the password.

These saved passwords are not stored in connections.txt. Instead, JXplorer stores these passwords internally.

When JXplorer is shut down, any stored passwords are discarded.

For instructions, see [Turn Off Password Storage](#).

Password Hashing

When you create or edit a user password using JXplorer, this password has to be sent from JXplorer to the underlying directory.

To make sure that the password is kept secure:

- Do not send unencrypted passwords over unsecure connections
- Do not store passwords in clear text (unencrypted).

JXplorer with eTrust Directory

If you use JXplorer with eTrust Directory, the following happens when you create a new user password:

1. JXplorer binds to an eTrust Directory DSA. You should use SSL or another secure connection for this binding.
2. In JXplorer, you create a user password, using plain encryption, MD5 or SHA.
3. JXplorer sends the password to the DSA, using the SSL connection.
4. The eTrust Directory DSA hashes the password, then stores it.

This means that if you use eTrust Directory and bind to it using SSL, JXplorer does not need to hash the password. The password is kept secure during transmission because the connection to the DSA uses SSL, and the password is stored securely because eTrust Directory hashes it.

However, if you do choose to hash the password, an eTrust Directory DSA will recognize the hash format, and can compare hashes to check that the password is correct. Make sure you use the same hash algorithm for the password in both JXplorer and the DSA.

JXplorer with Other LDAP Directories

Not all directories are capable of hashing passwords before they are stored. This means that if you use JXplorer with another LDAP directory, you may need to set JXplorer to hash the password before it is sent to the directory.

If you use JXplorer with a directory that doesn't hash passwords, the following should happen when you create a new user password:

1. JXplorer binds to the directory. You should use SSL or another secure connection for this binding.
2. In JXplorer, you create a user password.
3. JXplorer hashes the password using MD5 or SHA.
4. JXplorer sends the hashed password to the directory, using the SSL connection.
5. The directory stores the password exactly as it was received.

To set JXplorer to hash a user password, use the drop-down list in the User Password Data dialog. For more information, see [Adding a User Password](#).

Chapter 12: How JXplorer Handles SSL, SASL, and Certificates

You can use Secure Sockets Layer (SSL) authentication to communicate securely with a directory server. Two variants are allowed:

- SSL with server authentication only (simple SSL)
- SSL with both client and server authentication (authenticated SSL)

This section contains the following topics:

[SSL and SASL](#) (see page 77)

[Manage Certificates and Keystores](#) (see page 79)

SSL and SASL

Simple SSL authenticates the server only, whereas authenticated SSL authenticates both the client and the server.

Both variants require the client to be initialized with the trusted public certificate of the directory server, or the public certificate of the directory server's certificate authority. The trusted public certificates of servers are stored in the cacerts keystore file, located in the security directory, under JXplorer.

In addition to the above, Authenticated SSL requires the registration of the client's trusted public certificate (or the public certificate of the client's certificate authority) with the directory server, and use of the client's private key. Trusted public certificates and private keys of clients are stored in the clientcerts keystore file, located in the security directory, under JXplorer.

When you add or delete a certificate, or private key, the keystore files are updated and encrypted. You can set a password to stop unauthorized changes to these files.

Server-Authenticated SSL

For server-authenticated SSL to work, you must initialize the client with the trusted public certificate of the directory server, or the server's certificate authority.

The default keystore for trusted certificates is the `security/cacerts` file, which comes initialized with the certificate authority certificate used to create the demonstration DXserver certificates. While you can change this, the default setup lets the demonstration directories be contacted immediately using SSL.

You can connect to the directory using server authenticated SSL, as either an anonymous user, or with your user name and password.

Client-Authenticated SSL and SASL

Client-authenticated SSL requires the registration of the server's certificate with the browser, and in addition, the registration of the browser's certificate (or certificate authority) with the server.

Client-authenticated SSL also requires the use of the browser's private key, which is held in the `.../security/clientcerts` file. This file is password-protected, and requires the password to be entered in the connection dialog for client-authenticated SSL to work.

A demonstration client certificate *marjorie.pem* is provided in the *security* directory.

eTrust Directory can use SASL authentication to authenticate a user, rather than a user name and password. This implementation of SASL uses the certificates previously exchanged by SSL, and will only work when client-authenticated SSL is used. This differs from server-authenticated SSL where no client certificate is produced, so the directory is not able to use it to establish identity.

The secure use of client-authenticated SSL requires creating a new private key for the browser rather than using the default private key. This requires using a public key infrastructure tool, such as eTrust® PKI or Open SSL, to produce a PKCS8 private key.

Manage Certificates and Keystores

To use SSL in either form, you must manage a variety of certificates and private keys. These are kept in two Java keystores, which are password protected data stores. The first keystore, `.../security/cacerts`, with the password `changeit`, is used for storing the public certificates of trusted certificate authorities and servers. The second keystore, `.../security/clientcerts`, is used for storing the certificates and private keys of the JXplorer browser, and it has the password `passphrase`. Manage these keystores from the Security menu in JXplorer, where you can change the default keystore passwords.

JXplorer uses the standard Java cryptography tools for its SSL support. These two files are standard Java keystores, which you can maintain using the Java `keytool` utility. This is a command line utility produced by Sun Microsystems. For more information, see `keytool - Key and Certificate Management Tool` (<http://java.sun.com/j2se/1.3/docs/tooldocs/solaris/keytool.html>).

How Certificates Are Stored

By default, the trusted public certificates of servers and certificate authorities are stored in the `cacerts` keystore file, and the trusted public certificates and private keys of clients are stored in the `clientcerts` keystore file, which are located in the JXplorer security directory.

However, you can change the location and type of the keystore file. See the online help for instructions.

How to Decide Where to Store Certificates

If you want to use Secure Sockets Layer (SSL) authentication to communicate securely with a directory server, you must add the trusted public certificate of the directory server, or the public certificate of the directory server's certificate authority, to the `cacerts` keystore file.

If you want to strengthen the security and validate the client as well, you must add the client's trusted public certificate (or the public certificate of the client's certificate authority), and the corresponding private key, to the `clientcerts` keystore file.

How to Work with Private Keys

The password protecting the client's private key must be the same as that used to protect the *clientcerts* keystore file; therefore, you cannot change the keystore password after entering the private key password. If you want to change the keystore password, you must export the private key first, change the keystore password, and then re-import the private key into the keystore.

Private keys can only be imported as a special type of password-protected file called *pkcs8*. The password of the private key must be the same as that protecting the keystore.

How to Work with Public Keys

The *cacerts* keystore file has a default password of *changeit*; the *clientcerts* keystore file has a default password of *passphrase*. For instructions for setting a new password to stop unauthorized access to these files, see the online help

Chapter 13: Extending JXplorer

JXplorer can be extended to load small Java programs, similar to Web applets, based on an entry's object class.

This section contains the following topics:

[Pluggable Attribute Editors](#) (see page 81)

[Pluggable Entry Editors](#) (see page 91)

[Plug-ins with Data Listeners](#) (see page 94)

[Plug-ins with Threads](#) (see page 97)

[Localize JXplorer Plug-ins](#) (see page 101)

[Add Help Files to Plug-ins](#) (see page 102)

Pluggable Attribute Editors

LDAP is primarily a string handling protocol and many attribute values are simple text strings. However, it is often necessary to load other types of data, for example, certificates and images.

eTrust Directory can be used to store many types of non-string data types, including graphics formats, audio formats, cryptography formats, and application-specific data types.

JXplorer allows you to load binary files to some attributes, such as userPKCS12.

The general-purpose JXplorer cannot be used to modify binary data, but you can create your own binary editor for a particular data type.

The browser also supports custom binary editors (written in Java using a provided minimal API) that dynamically loads at run time. You key such binary editor extensions to a particular object class. This would let you write, for example, an editor for a custom certificate object class.

These editors will only work for binary value data attributes and no other data types.

Standard editors are provided for X.509 certificates, and a number of standard image and audio formats.

Ways to Edit Binary Attributes

To create a pluggable attribute editor takes time, and you may not need to do it. JXplorer comes with two binary editors, or you can use a Windows editor.

Using a Windows Editor

You can choose to use a Windows editor instead of creating a pluggable attribute editor. To do this, save the file, then edit it, then load it again.

However, if you do this, your application may not be able to run on a non-Windows platform. A possible path out of this is to check the operating system in the pluggable attribute editor, and issue an appropriate native command depending on the OS.

Using the File Launching Feature

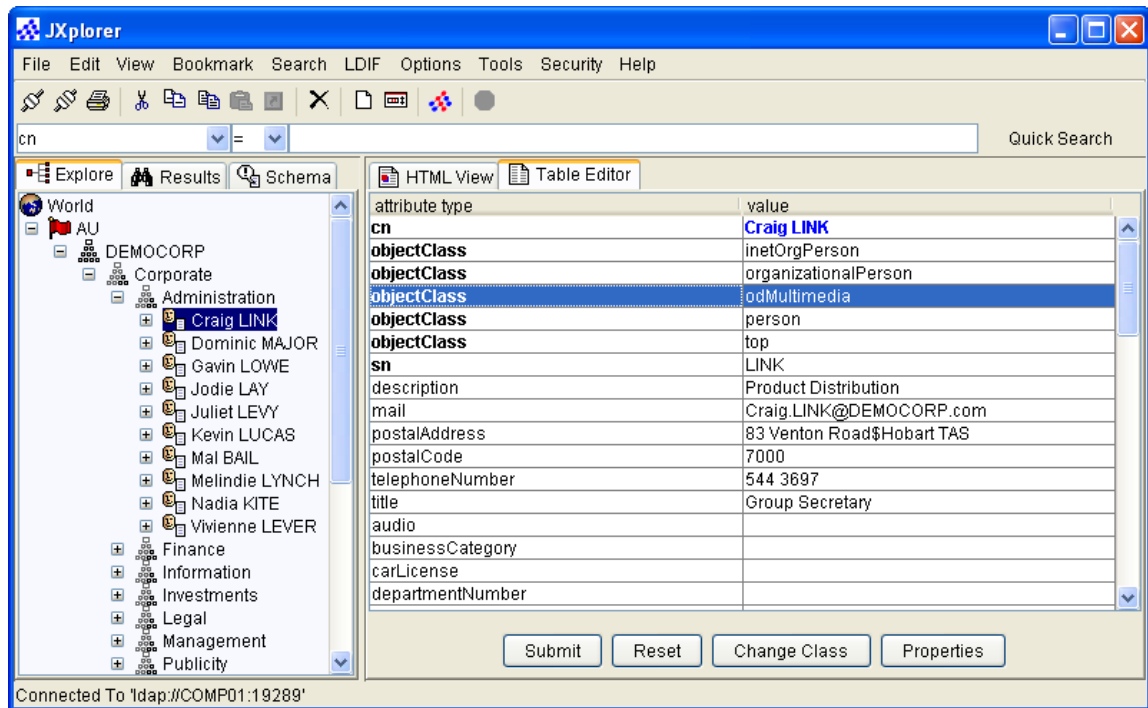
To modify binary data, you can use the default binary editor to save the data to disk, use your normal tool to manipulate the data, and load it back into the default binary editor.

The binary editor is very limited. JXplorer lets you edit the attributes in the odMultimedia schema, which lets you save and launch files of the following formats:

Format	Attribute Type
.avi	odMovieAVI
.doc	odDocumentDOC
.mid	odMusicMID
.wav	odSoundWAV
.xla	odSpreadSheetXLS

To use the attribute types in `odMultimedia` in the Table Editor, add the `odMultimedia` object class to an entry.

You can then add binary files to that entry, and launch those files from the Table Editor view of the entry.



For more information, see [Launch Binary Files](#) in the JXplorer online help.

Foreign Languages Do Not Require a Pluggable Editor

You don't need to write a pluggable editor to use different languages with JXplorer.

JXplorer has been successfully tested with European and Asian languages on correctly installed localized platforms.

Java's normal string handling of Unicode automatically translates between the locale-specific character encoding and the Unicode format used internally in the browser. The Unicode is read from and written to the directory in a transformation format called UTF-8, but this is invisible to the user.

For information about internationalizing JXplorer, see [Internationalize JXplorer](#) (see page 62).

Writing a Pluggable Attribute Editor

Writing a pluggable editor simply involves writing a Java class that does one of the following:

- Implements the `com.ca.directory.jxplorer.editor.abstractbinaryeditor` interface
- Extends the `com.ca.directory.jxplorer.editor.basicbinaryeditor` class

The class must be given the name of the attribute type, in lower case, with the suffix `editor`.

For example, if you create an editor for the attribute `ocspRSAPrivateKey`, you must name the final class `ocsprsaprivatekeyeditor`.

Extending the `abstractbinaryeditor` class

The `abstractbinaryeditor` interface has only one method: `public void setValue(editablebinary editMe)`. This passes an `editablebinary` object, which is a data object used to store a byte array. The `editablebinary` object has two methods, `public byte getValue()` and `public void setValue(bytebytes)`, used respectively to read and write to the object.

When an attribute value of the appropriate type is found by JXplorer, and a user wishes to edit it (by clicking on the value cell in the editor pane) JXplorer will run the pluggable editor, passing the byte array to it using the `setValue` method of `abstractbinaryeditor`. That Editor can then manipulate the object as it wishes (usually by creating a GUI and allowing the user to edit the data in some way). As long as the `editablebinary` object is updated with the final value, when the user click the Submit button in JXplorer, the new value is entered into the directory.

Extending `DefaultBinaryEditor`

The class `defaultbinaryeditor` is a default implementation of `abstractbinaryeditor` using a swing `JFileChooser`. For some purposes it may be easier to extend it, rather than implement a GUI from scratch and using `abstractbinaryeditor` directly.

Example: Handling Masked Binary Password Data

The following is an example of a pluggable editor that could be used to handle masked, binary password data. Almost all of this code is simply setting up the GUI.

```

/**
 * All pluggable editors must be in this package
 */
package com.ca.directory.jxplorer.editor;

import com.ca.commons.cbutil.*;    // custom version of JPanel
import com.ca.directory.jxplorer.AdvancedOptions;

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.io.*;

/**
 * Password Editor.
 * Allows user to enter a masked binary password. The dialog contains two
 * password fields. The first is for displaying the password that is stored in
 * the directory. The second is to confirm that the first has been entered
 * correctly. To change the password the user needs to click on the change
 * button. Both password fields are then cleared (we don't want the user to edit
 * the encoded password). When the user clicks the save button a check is done.
 * If the user hasn't changed the password, it gets stored. Otherwise if the user
 * did change it, the two password fields must match before it is saved (a
 * warning message is displayed if they don't match).
 *
 */

public class userpasswordeditor extends JDialog
    implements abstractbinaryeditor
{
    protected JPasswordField oldPwd, newPwd;
    protected CButton btnOK, btnCancel, btnHelp;
    protected EditableBinary editMe = null;
    protected CPanel display;
    protected JLabel oldLabel, newLabel;
    protected boolean firstClick = true;

    /**
     * Constructor - sets up the gui.
     */

    public userpasswordeditor(Frame owner)
    {

```

```
super(owner);

setModal(true);
setTitle(CBIntText.get("User Password"));

display = new CBPanel();

oldPwd = new JPasswordField();
oldPwd.addMouseListener(new MouseListener()
{
    public void mouseClicked(MouseEvent e) { }
    public void mouseEntered(MouseEvent e) { }
    public void mouseExited(MouseEvent e) { }
    public void mouseReleased(MouseEvent e) { }
    public void mousePressed(MouseEvent e)
    {
        if (firstClick)
        {
            oldPwd.setText("");
            firstClick = false;
        }
    }
});

newPwd = new JPasswordField();

oldLabel = new JLabel(CBIntText.get("Enter Password:"));
newLabel = new JLabel(CBIntText.get("Re-enter Password:"));

btnOK = new CButton(CBIntText.get("OK"), CBIntText.get("Click here to
save the changes (remember to click Submit in the table editor)."));
btnOK.addActionListener(new ActionListener()
{
    public void actionPerformed(ActionEvent e)
    {
        load();
    }
});

btnCancel = new CButton(CBIntText.get("Cancel"), CBIntText.get("Click
here to exit.));
btnCancel.addActionListener(new ActionListener()
{
    public void actionPerformed(ActionEvent e)
    {
        quit();
    }
});
```

```

        btnHelp = new CButton(CBIntText.get("Help"), CBIntText.get("Click here
for Help.));
        CBHelpSystem.useDefaultHelp(btnHelp, "edit.password");

        display.makeHeavy();
        display.addln(oldLabel);
        display.addln(oldPwd);
        display.addln(new JLabel(" "));
        display.addln(newLabel);
        display.addln(newPwd);
        display.makeLight();

        JPanel buttonPanel = new JPanel();
        buttonPanel.add(btnOK);
        buttonPanel.add(btnCancel);
        buttonPanel.add(btnHelp);
        display.addln(buttonPanel);
display.getInputMap(JComponent.WHEN_IN_FOCUSED_WINDOW).put(KeyStroke.getKeyStroke
("ENTER"), "enter");

display.getInputMap(JComponent.WHEN_IN_FOCUSED_WINDOW).put(KeyStroke.getKeyStroke
("ESCAPE"), "escape");
        display.getActionMap().put("enter", new MyAction(CBAction.ENTER));
        display.getActionMap().put("escape", new MyAction(CBAction.ESCAPE));

        setSize(300, 170);
        CBUtility.center(this, owner); //Centres the window.
        setTitle(CBIntText.get("User Password Data"));
        getContentPane().add(display);
    }

/**
 * Apparently it is better to use key bindings rather than adding a
 * KeyListener... "for reacting in a special way to particular keys, you
 * usually should use key bindings instead of a key listener".
 * This class lets the user set the key as an int. If a key is pressed and
 * it matches the assigned int, a check is done for if it is an escape or
 * enter key.
 * (27 or 10). If escape, the quit method is called. If enter, the apply
 * method is called.
 * Bug 4646.
 */
private class MyAction extends CAction
{
    /**
     * Calls super constructor.
     * @param key
     */
    public MyAction(int key)

```

```
{
    super(key);
}

/**
 * quit is called if the Esc key pressed,
 * load is called if Enter key is pressed.
 * @param e never used.
 */
public void actionPerformed(ActionEvent e)
{
    if (getKey() == ESCAPE)
        quit();
    else if (getKey() == ENTER)
        load();
}
}

/**
 * This is the abstractbinaryeditor interface method which is
 * called when the user wants to edit the password
 */
public void setValue(editablebinary editMe)
{
    this.editMe = editMe;
    oldPwd.setText(stringEncode(editMe.getValue()));
}

/**
 * converts between text and a byte array
 */
protected byte[] stringDecode(String s)
{
    if (s == null)
        return (new byte[0]);
    else
        try
        {
            return s.getBytes("UTF-8");
        }
        catch (UnsupportedEncodingException e)
        {
            CBUtility.log("Unexpected error encoding password " + e);
            e.printStackTrace();
            return new byte[0];
        }
}
}

/**
```

```
* converts between a byte array and text
*/
protected String stringEncode(byte[] b)
{
    if (b == null || b.length == 0)
        return new String();
    else
        try
        {
            return new String(b, "UTF-8");
        }
        catch (UnsupportedEncodingException e)
        {
            CBUtility.log("Unexpected error decoding password " + e);
            e.printStackTrace();
            return new String(b);
        }
}

/**
 * sets the value of the editablebinary object with whatever the
 * user has entered into the password text field.
 */
protected void load()
{
    if (passwordConfirm())
    {
        editMe.setValue(stringDecode(new String(newPwd.getPassword())));
        quit();
    }
}

/**
 * Does some checks on the password.
 * @return True - if the two password fields match.
 * False - if the new password field is empty (an error message is
 * displayed).
 * False - if the password fields don't match (an error message is
 * displayed).
 */
protected boolean passwordConfirm()
{
    if (new String(newPwd.getPassword()).equals(new
String(oldPwd.getPassword())) //if the two password fields match carry on
saving the password.
    {
        return true;
    }
    else if (new String(newPwd.getPassword()).equals("")) //if the new
```

```
password field is empty display error message.
    {
        JOptionPane.showMessageDialog(display, CBIIntText.get("Empty password
field, please fill in both fields"), CBIIntText.get("Warning message"),
JOptionPane.INFORMATION_MESSAGE);
        newPwd.setText("");
        return false;
    }
    else //if the password fields don't match display error message.
    {
        JOptionPane.showMessageDialog(display, CBIIntText.get("Password typed
incorrectly, please try again"), CBIIntText.get("Warning message"),
JOptionPane.INFORMATION_MESSAGE);
        newPwd.setText("");
        return false;
    }
}

/**
 * Shuts down the gui.
 */
protected void quit()
{
    setVisible(false);
    dispose();
}
}
```


Pluggable Entry Editors

Pluggable entry editors are similar to pluggable attribute editors, except that they are triggered on an entire entry, rather than just an attribute of an entry. They take up the entire right-hand editing pane, and can suppress the normal HTML and table display if desired, giving a 'custom application' look.

There may be no need to write a pluggable editor. If all you need to do is give a custom look-and-feel to your data, you may find it easier to use the HTML templates and forms. But, if you need to do any complex client-side processing such as wizards or form validation, it may be useful to write your own editors.

Editor Names and Locations

The name of the editor must be the same as the object class of the type of entry it is to be used for. So if the editor is to be used for 'people', the name of the class is simply 'people.class'.

Entry editors must be completely in lower case, and there is no suffix.

The classes made need to be packaged into the JavaBrowser.jar file in the directory C:\Program Files\CA\Trust Directory\jxplorer\jars and they need to be added into the JavaBrowser.jar file with the correct path: com\ca\directory\jxplorer\viewer.

For example, if the person.class file is created it needs to be packaged and have the path name: com\ca\directory\jxplorer\viewer\person.class

The PluggableEditor Interface

The Pluggable Editor interface defines a number of methods that let the Pluggable Editor take control of the entirety of JXplorer. Most of the time you will only want to use a handful of these. For more information, see the JXplorer API Reference ([../jxplorer/api/index.html](http://jxplorer/api/index.html)) that was installed with JXplorer.

The DataSink Interface

The DataSink interface is used internally in JXplorer for transferring data to data consumers. Since the Pluggable Editor class does this, a pluggable editor must also implement the DataSink interface (which has only two methods).

Extending BasicPluggableEditor

Most editor writers will probably simply extend `BasicPluggableEditor`, which has default implementations of all the required methods. The class `BasicPluggableEditor` simply displays the entry passed to it as text in a `JPanel`. The following shows a trivial extension of `BasicPluggableEditor` (which in fact does exactly the same thing - simply displays the entry in a text pane).

```
package com.ca.directory.jxplorer.viewer;
import java.awt.*;
import java.awt.event.*;
import java.awt.geom.*;
import javax.swing.*;
import com.ca.directory.jxplorer.viewer.BasicPluggableEditor;
import com.ca.commons.naming.DXEntry;
import com.ca.directory.jxplorer.DataSource;

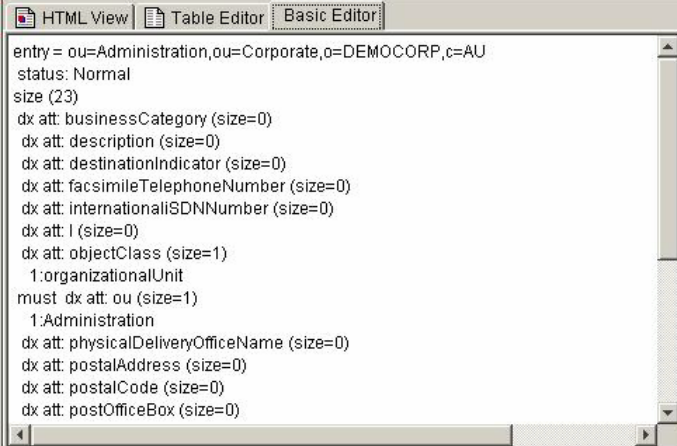
public class person extends BasicPluggableEditor
{
    JEditorPane basicDisplay;
    JScrollPane scrollPane;

    /**
     * The Constructor sets up the JEditorPane GUI
     */
    public person()
    {
        basicDisplay = new JEditorPane("text/plain","");
        scrollPane = new JScrollPane(basicDisplay);
    }

    /**
     * This method is called by JXplorer when a new 'person' entry
     * needs to be displayed.
     */
    public void displayEntry(DXEntry entry, DataSource ds)
    {
        basicDisplay.setText(entry.toString());
    }

    /**
     * This method returns a GUI component to JXplorer to display
     * in the right hand pane.
     */
    public JComponent getDisplayComponent()
    {
        return scrollPane;
    }
}
```

This results in the following editor pane:



```

HTML View | Table Editor | Basic Editor
entry = ou=Administration,ou=Corporate,o=DEMOCORP,c=AU
status: Normal
size (23)
dx att: businessCategory (size=0)
dx att: description (size=0)
dx att: destinationIndicator (size=0)
dx att: facsimileTelephoneNumber (size=0)
dx att: internationalSDNNumber (size=0)
dx att: l (size=0)
dx att: objectClass (size=1)
1:organizationalUnit
must dx att: ou (size=1)
1:Administration
dx att: physicalDeliveryOfficeName (size=0)
dx att: postalAddress (size=0)
dx att: postalCode (size=0)
dx att: postOfficeBox (size=0)

```

This is all that is required for your own pluggable editor - a GUI and some code to handle the 'displayEntry(..)' method. Check the API for the methods of the DXEntry object - it is an extension of the standard JNDI 'Attributes' object and supports all its methods, as well as a large number of utility extension methods (such as the entry.toString() method).

Sending Changes to the Directory

When your user has registered changes in your custom pluggable editor GUI, you'll need to submit them to the directory. The core pluggable editor method is public void displayEntry(DXEntry entry, DataSource ds). The entry parameter is the data to display, while the DataSource parameter is a link to a data source (usually a directory) that you can use to submit changes. The simplest call in DataSource is modifyEntry(..), which takes the original entry and the new entry, and makes the necessary calls to the directory to efficiently convert the old entry to its new state.

There are a large number of other data related operations you might want to make with the directory, ranging from reading more data, searching, modifying the directory tree structure, and reading schema details.

DataSource is actually the front end to a queued, multi-threaded directory connection. As a pluggable Editor writer you shouldn't normally need to worry about that; it just means that your GUI won't freeze up when you make a directory request! However if you need to know the result of a directory operation you'll need to know about the data event model. When a directory operation is completed (successfully or otherwise) your editor can find out by registering itself as a 'DataListener' (this metaphor will be familiar to Java programmers who have worked with GUI listeners). The details of doing this are covered in the Plug-ins with Data Listeners (see page 94) section.

Plug-ins with Data Listeners

Directory operations occur over networks, which can be slow, and may require processing on a busy directory server, which may also be slow. If your application is waiting on the results of a directory operation, it can 'freeze up'. The GUI doesn't respond to mouse clicks, or may not repaint properly. The user will assume the program has crashed, and may kill it manually. The worst that can occur is when the directory server crashes or your network dies, and the browser freezes until the connection times out.

The JXplorer Threading Model

Fortunately it is relatively easy to make a Java application multi-threaded, so, as an editor writer, it would not be much work to put your directory code in a different thread. However this isn't necessary using JXplorer's pluggable editors, since JX is already multi-threaded. In fact, pluggable editors written using the techniques of the last chapter are already multi-threaded, and will not hang.

Problems arise when you need to write code that is conditional on the results of a directory operation. For example, you may want to create a particular entry if it doesn't exist, or modify it if it does. The way to do this in a pluggable entry editor is to either use the 'DataListener' interface (for simple operations), or to pass an extended 'DataQuery' object (for more complex tasks).

This section shows you how to use the simpler DataListener interface, and the Plug-ins with Threads (see page 97) section how to use the general purpose 'DataQuery' method to run arbitrary code.

Using the DataListener Interface

The `displayEntry()` method of `DataSink` contains the entry to display, and a `'DataSource'`. The `DataSource` can be used to carry out various directory operations, such as `getEntry()` or `copyTree()`. However, since these operations occur in another thread, rather than returning data or a success code immediately, they return a `'DataQuery'` object.

The `DataQuery` object is used to communicate with the connection thread. If you need to know what happens to the operation (and you may not - for example the results of a `'copyTree()'` operation, or any errors, will be displayed by the browser without any intervention) you can use the `DataQuery` object. This is done by registering a `'DataListener'` with the `DataQuery` object, in the same way as an `'ActionListener'` might be registered with a button. When the `DataQuery` has completed (either successfully, or with an error) your `DataListener` will be called with the result.

The `DataListener` only has one method returning one object - the original `DataQuery`! However in this method you are guaranteed that the `DataQuery` has finished, and is ready for reading. So within the `DataListener` method you can use all the `DataQuery` methods such as `hasException()` or `getResult()`.

This may seem a bit of effort to go to, but is in fact fairly straightforward.

Important! Attempting to read the `DataQuery` object immediately can be very dangerous - in the best possible case it will block the current thread until the data is ready, in the worst case it will attempt to block the thread making the directory connection and will throw an exception in order to avoid thread 'deadlock'.

The following code snippet (using an anonymous inner class) is a quick example of how to use a `DataListener`:

```
public class MyPluggableEditor extends BasicPluggableEditor
{
...
...

    public void displayEntry(DXEntry entry, DataSource myDataSource)
    {
        ...
        ...
        DataQuery readQuery = myDataSource.getEntry(
                                                    new DN("cn=fred,ou=R&D,o=CA"));
        ...
        --
        readQuery.addDataListener(new DataListener()
        {
            public void dataReady(DataQuery query)
            {
```

```
        if (query.hasException())
        {
            System.err.println("couldn't read entry " +
                query.getDN()+"\nexception= "+query.getException());
            --           // prevent the browser also displaying the error.</font>
            query.squelch();

        }
        else
        {
            System.out.println("read entry " + query.getEntry());
        }
    }
    });
    ...
}
    ...
}
```

Other DataListeners

Your pluggable editor isn't the only Data Listener. The JXplorer browser tree is another, and it will respond to any data operations that occur, showing error messages or changing the tree as appropriate.

Most of the time this is what you want. However, if you'd prefer to keep your operations private (maybe you're handling your own exceptions, or you're hiding a sub-level of the directory from the user) you can 'squelch()' the query, preventing any other listeners from processing the query.

Complex Directory Interactions

Sometimes this still isn't enough - you need to make a directory request in your pluggable editor, and then, depending on the result, you need to make further requests.

There are a number of ways of doing this. One method is to use the `getBroker()` `DataQuery` method. This gives you raw access to the directory connection methods.

This method falls down if the initial directory action is non-standard, and may also be a little clumsy if you have a single unit of work to do, that could be nicely executed in one place. In this case, the best thing to do may be to extend the `DataQuery` class itself, and pass an 'extended `DataQuery`' to the `DataSource`, using `DataSource`'s `extendedRequest()` method. How to do this is covered in the Plug-ins with Threads (see page 97) section.

Plug-ins with Threads

If you have a complex piece of directory logic to execute, the preceding methods may be a bit clumsy. For example, consider the following action: "check if this entry exists, if it doesn't exist, create the following sub-tree, otherwise, check that all these components exist, and if they don't exist, create them, and finally copy the whole tree into a backup branch directory".

This could be done using the previous technique by doing the first query (an existence check) and then putting the rest of the code in the `DataListener`, while using the `DataQueries` `'getBroker()'` method.

Bundling Complex Directory Code

Instead, you can bundle all the code up in one place, and use an 'extended data query'. The method is simply to extend the `DataQuery` class by implementing the `'doExtendedRequest()'` method, and then pass the resulting query to the connection thread using the broker's `'extendedQuery()'` method.

Example: Reading an Entry and Printing It Out

The code in the Plug-ins with Data Listeners (see page 94) section, which read an entry and printed it out, could be rewritten as:

```
public class MyPluggableEditor extends BasicPluggableEditor
{
...
...
public void displayEntry(DXEntry entry, DataSource myDataSource)
{
...
...
myDataSource.extendedRequest(new DataQuery
{
public void doExtendedRequest(Broker myBroker)
{
DXEntry myEntry = myBroker.unthreadedReadEntry(
    new DN("cn=fred,ou=R&D,o=CA"));
if (myBroker.getException() != null)
{
System.err.println("couldn't read entry " + query.getDN() +
    "\nexception was: " + query.getException());
}
else
{
System.out.println("read entry " + query.getEntry());
}
}
}
...
}
}
```

Note: There is no need to 'squelch()' anything, because no listeners (except any that you might register) pay attention to extended queries.

Unthreaded Broker Methods

Since the DataQuery is being run in the connection thread, it should use the 'unthreaded' broker methods to access the directory.

These methods are:

- unthreadedExists
- unthreadedReadEntry
- unthreadedCopy
- unthreadedList
- unthreadedModify
- unthreadedSearch
- unthreadedGetAllOCs
- unthreadedGetRecOCs

For more information, see the JXplorer API Reference ([../jxplorer/api/index.html](http://jxplorer/api/index.html)) that was installed with JXplorer.

Important! Do not use of the threaded methods such as `doEntryQuery()` instead of `unthreadedRead()`. The threaded methods place a query on the connection thread queue. Since the extended request is being run by this same connection thread, if eTrust Directory tries to read the result of one of these threaded operations, the connection thread will immediately deadlock.

Example: Check for Existence, Create, and Copy to Backup Branch

As a more elaborate example, let's try the scenario outlined previously:

1. Check the existence of an entry.
2. If it doesn't exist, create it and a subtree.
3. If it does exist, check that the subtree exists and create it if it doesn't.
4. Copy everything to a backup branch.

```
public class MyPluggableEditor extends BasicPluggableEditor
{
...
...
    public void displayEntry(DXEntry entry, DataSource myDataSource)
    {
        ...
        ... /**
            * Define a new 'ExtendedDataQuery' class.
            * (XXX error checking not implemented)
        */
    }
}
```

```

*/

class ExtendedDataQuery extends DataQuery
{
    Broker currentBroker = null;
    public void doExtendedRequest(Broker myBroker)
    {
        currentBroker = myBroker;
        boolean exists = currentBroker.unthreadedExists(new DN("cn=top,o=CA"));
        if (exists == false)
        {
            writeTop();
            writeTree();
        }
        else
        {
            if (checkTree() == false)
                writeTree()
        }
        currentBroker.unthreadedCopy(new DN("cn=top,o=CA"),
            new DN("cn=top,cn=backup,o=CA"));
    }
    void writeTop()
    {
        DXEntry top = new DXEntry("cn=top,o=CA");
        top.put("objectClass", "TopEntry");
        currentBroker.unthreadedModify(null, top);
    }
    void writeTree()
    {
        DXEntry[] nodes = new DXEntry[5];
        DXEntry[0] = new DXEntry("cn=node A,cn=top,o=CA");
        DXEntry[1] = new DXEntry("cn=node B,cn=top,o=CA");
        DXEntry[2] = new DXEntry("cn=node C,cn=top,o=CA");
        DXEntry[3] = new DXEntry("cn=node D,cn=top,o=CA");
        DXEntry[4] = new DXEntry("cn=node E,cn=top,o=CA");
        for (int i=0; i<5; i++)
        {
            DXEntry[i].put("objectClass", "TreeEntry");
            currentBroker.unthreadedModify(null, DXEntry[i]);
        }
    }
    boolean checkTree()
    {
        DXNamingEnumeration list =
            currentBroker.unthreadedList("cn=top,o=CA");
        return (list != null && list.size() == 5);
    }
}

```

```
...
// Create and run our extended query.
myDataSource.extendedRequest(new ExtendedDataQuery())
...
}
}
```

Localize JXplorer Plug-ins

Plug-ins may also require their own translation files and help files. JX makes it easy to add plug-in-specific files to JX at run-time.

This is easily done using a single method call. If your plug-in is extending from `BasicPluggableEditor`, the call is:

```
addLanguageBundle("myBundleName");
```

If you are implementing the `PluggableEditor` interface, you can make the equivalent call using the code:

```
CBIntText.addBundle("myBundleName", getClass().getClassLoader());
```

In both these instances, replace the string "myBundleName" with the name of your particular plug-in's resource bundle.

If your plug-in needed to translate the word 'help' into German for display in the German locale, you would create a file called `myPlugin_de.properties`, and have in it the single line:

```
help = hilfe
```

Place the 'myPlugin_de.properties' file in your plugin zip file (at the top level), and make the call 'addLanguageBundle("myPlugin");' in the constructor of your plug-in, and a later call to 'CBIntText.get("help")' should return the string "hilfe". You could use this in a label definition: `JLabel myLabel = new JLabel(CBIntText.get("help"));`

Add Help Files to Plug-ins

Plug-ins can also add their help files to the main JXplorer help. This requires the creation of a plug-in help set (similar to the JXplorer help). When your help set is created (including foreign language version if required), the resulting help files should be added to the plug-in zip file (or equivalently, copied unzipped to the plug-ins directory).

If your plug-in is extending from `BasicPluggableEditor`, the call is:

```
addHelpSet("myHelpSet");
```

If you are implementing the `PluggableEditor` interface, you can make the equivalent call using the code:

```
CBHelpSystem.addToDefaultHelpSet("myHelpSet", getClass().getClassLoader());
```

In both these instances the string "myHelpSet" is replaced with the name of your particular plug-in's help set.

Chapter 14: LDAP and Directory Resources

For more information about JXplorer

See the following:

- The *eTrust Directory Developer Guide*
- The JXplorer API, which is installed at:
 - **Windows:** %DXHOME%\..\jxplorer\api
 - **UNIX:** \$DXHOME/../../jxplorer/api
- The JXplorer page on SourceForge (<http://sourceforge.net/projects/jxplorer>)

For more information about Java

See the following:

- Java home page (<http://java.sun.com/>)
- API Specification for Java 2 Standard Edition 1.4.2 (<http://java.sun.com/j2se/1.4.2/docs/api/>)
- Java naming and directory interface (JNDI) page (<http://www.java.sun.com/products/jndi/index.html>)

For information about LDAP

- See the Request For Comments (RFC) documents, especially rfc2251 to rfc2256, at the RFC page on the IETF home page (<http://www.ietf.org/rfc/>).