EVIDEN

Identity and Access Management

DirX Identity

Meta Controller Reference

Version 8.10.14, Edition March 2026

Table of Contents

Preface

This manual is reference for the DirX Identity controller and its associated programs and files. It consists of the following chapters:

- Chapter 1 provides reference pages for the identity controller command-line program **metacp** and for the binary trace log file analyzers **metacpdump** and **metahubdump**.
- Chapter 2 describes the identity controller attribute configuration file format.
- Chapter 3 describes the data file formats supported by the meta controller.
- Chapter 4 describes the use of change log information.
- Chapter 5 describes the string representations and use of attribute syntaxes for LDAP binds by the Identity controller to the meta directory store. It also contains a reference page for simple syntaxes and reference pages for each of the structured syntaxes.
- Chapter 6 describes the files used by the **metacp** and **metacpdump** programs.
- Chapter 7 describes the environment variables used by the **metacp**, **metacpdump** and **metahubdump** programs.
- Chapter 8 describes the synchronization templates supplied with DirX Identity.
- Chapter 9 describes procedures for use of data file formats like XML and change log.
- Appendix A provides a list of country codes.
- Appendix B provides detailed information on code conversion.

DirX Identity Documentation Set

*Version 8.10.14 | Build 1858 | Date 2026-03-26 *

The DirX Identity document set consists of the following manuals:

- *DirX Identity Introduction*. Use this book to obtain a description of DirX Identity architecture and components.
- *DirX Identity Release Notes*. Use this book to understand the features and limitations of the current release. This document is shipped with the DirX Identity installation as the file **release-notes.pdf**.
- *DirX Identity History of Changes*. Use this book to understand the features of previous releases. This document is shipped with the DirX Identity installation as the file **history-of-changes.pdf**.
- *DirX Identity Tutorial*. Use this book to get familiar quickly with your DirX Identity installation.
- *DirX Identity Provisioning Administration Guide*. Use this book to obtain a description of DirX Identity provisioning architecture and components and to understand the basic tasks of DirX Identity provisioning administration using DirX Identity Manager.
- *DirX Identity Connectivity Administration Guide*. Use this book to obtain a description of DirX Identity connectivity architecture and components and to understand the basic tasks of DirX Identity connectivity administration using DirX Identity Manager.
- *DirX Identity User Interfaces Guide*. Use this book to obtain a description of the user interfaces provided with DirX Identity.
- *DirX Identity Application Development Guide*. Use this book to obtain information how to extend DirX Identity and to use the default applications.
- *DirX Identity Customization Guide*. Use this book to customize your DirX Identity environment.
- *DirX Identity Integration Framework*. Use this book to understand the DirX Identity framework and to obtain a description how to extend DirX Identity.
- *DirX Identity Web Center Reference*. Use this book to obtain reference information about the DirX Identity Web Center.
- *DirX Identity Web Center Customization Guide*. Use this book to obtain information how to customize the DirX Identity Web Center.
- *DirX Identity Meta Controller Reference*. Use this book to obtain reference information about the DirX Identity meta controller and its associated command-line programs and files.
- *DirX Identity Connectivity Reference*. Use this book to obtain reference information about the DirX Identity agent programs, scripts, and files.
- *DirX Identity Troubleshooting Guide*. Use this book to track down and solve problems in your DirX Identity installation.
- *DirX Identity Installation Guide*. Use this book to install DirX Identity.

- *DirX Identity Migration Guide*. Use this book to migrate from previous versions.

Notation Conventions

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

In examples, bold words and characters represent user input.

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

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

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

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

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

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

userID_home_directory
The exact name of the home directory. The default home directory is the home directory of the specified UNIX user, who is logged in on UNIX systems. In this manual, the home pathname is represented by the notation *userID_home_directory*.

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

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

dxi_java_home
The exact name of the root directory of the Java environment for DirX Identity. This location is specified while installing the product. For details see the sections "Installation" and "The Java for DirX Identity".

tmp_path

The exact name of the tmp directory. The default tmp directory is /tmp on UNIX systems. In this manual, the tmp pathname is represented by the notation *tmp_path*.

tomcat_install_path
The exact name of the root of the directory where Apache Tomcat programs and files are installed. This location is defined during product installation.

mount_point
The mount point for DVD device (for example, **/cdrom/cdrom0**).

1. DirX Identity Commands

DirX Identity provides the following commands:

- **metacp** - Invokes the meta controller. The meta controller is a directory client program that system administrators can use to synchronize directories and manage the entries in a Identity store. This program supports the following objects:
  - **ats** - Manages the asynchronous transport service
  - **ldapargs** - Manages service controls associated with an LDAP directory operation
  - **meta** - Performs directory synchronization operations
  - **obj** - Manages Identity store entries
  - **util** - Implements the **metacp** utilities.
- **metacpdump** - Displays the contents of the binary trace log files generated by the meta controller (**metacp**).
- **metahubdump** - Displays the contents of the binary trace log files generated by the Server (IdS-C) (dxmmsssvr.exe and dxmsvr.exe).

The remainder of this section describes the DirX Identity commands and provides command syntax and examples. The **metacp** command is described first, followed by descriptions of all the objects it supports in alphabetical order. Next, the **metacpdump** and **metahubdump** commands are described.

DirX Identity commands in scripts can use the backslash (\) as the line continuation character. In interactive mode, there is no continuation character. Instead, you must continue typing. The line automatically wraps if your characters extend beyond the line end. If you press the **Enter**/**Return** key, the information you have typed is sent to the system for processing. You should press the **Enter**/**Return** key only when you have typed all information required for the command to process.

The majority of the sample commands in this chapter uses continuation characters and appears as they would appear in a script. The commands are presented this way so that they can be formatted for readability.

1.1. metacp

Synopsis

metacp [*script_name* [*arg1* [*arg2* ...]]
[**-noaudittrail**]
[**-Enc** *encryption_mode*
-Timeout *timeout_value*
-Audit *audit_level*
-CryptLogLevel *crypt_level*] |
-c *command* |

-V

Purpose

Provides a set of operations for directory synchronization and manages entries in an Identity store. The meta controller (**metacp**) supports the following objects:

ats
: Manages the asynchronous transport service

ldapargs
: Manages service controls associated with an LDAP directory operation

meta
: Performs directory synchronization operations

obj
: Manages Identity store entries

The meta controller also supports the utilities described in the **util** page.

Arguments

script_name
: Filename of a user-defined script containing **metacp** commands.

  The given Tcl script will be loaded using the **utf-8** encoding as default. If the Tcl script holds data in a different encoding, then the following additional arguments must be passed:

  -encoding *encoding*

-encoding *encoding* is part of ***arg[n]***. It is passed along to the Tcl script as global Tcl variables. In order to write compatible Tcl scripts, it is recommended to pass these two arguments at the end of the argument list.

arg[n]
: Argument to the user-defined script specified in *script_name*. Script arguments are stored in global Tcl variables and can be evaluated in Tcl scripts. See the Tcl Variables section in this chapter for more information.

Options

-c *command*
: Executes the **metacp** or Tcl commands specified by *command*.

-V

Displays the **metacp** build version, in the format:

'metacp' build version: *product_version build_id date time*

For example:

```
'metacp' build version: 5.0B 19 1999:05:05 13:14
```

The **metacp initialize** operation also writes the build version to the **metacp** trace file.

-noaudittrail

Usuaally **metacp** evaluates the audit policies to decide whether or not audit information should be supplied. If **-noaudittrail** is specified no audit information is generated (even if the audit policies would allow to create audit information).

When running **metacp** in a security environment, the following options can be used:

-ENC *encryption_mode*

Specifies the security mode. Valid modes are ATTRIB_ADMIN_PW or ADMIN_PW.

ATTRIB_ADMIN_PW - bind passwords and attributes that are marked as encrypted (in the corresponding attribute configuration file) are decrypted.

ADMIN_PW - only bind passwords are decrypted.

If metacp processes several LDAP directories in parallel (e.g. in case of an LDAPtoLDAP workflow) and it runs with one of the encryption modes described above, then the bind credentials for all the bind connections need to be provided as encrypted values. It's not possible to send encrypted bind credentials to one of the directories and to send clear text (or scrambled) bind credentials to the other directory.

This functionality only works correctly in an appropriate security environment like in the DirX Identity environment configured in security mode. (See DirX Identity Connectivity Administration Guide).

-Timeout *timeout_value*

Specifies the timeout value for the security mode. Values have to be given in microseconds.

This functionality only works correctly in an appropriate security environment like in the DirX Identity environment configured in security mode. (See *DirX Identity Connectivity Administration Guide* for details.)

-AuditLevel *audit_level*

Specifies the audit level value for the security mode. Valid values are in the range of 0 and 4.

This functionality only works correctly in an appropriate security environment like in the

DirX Identity environment configured in security mode. (See DirX Identity Connectivity Administration Guide).

-CryptLogLevel ***crypt_level***

Specifies the logging level of the crypt library for the security mode. Valid values are greater or equal to 0.

This functionality only works correctly in an appropriate security environment like in the DirX Identity environment configured in security mode. (See DirX Identity Connectivity Administration Guide).

Description

The meta controller is a command-line tool that system administrators can use to perform directory synchronization. Use it to perform the following tasks:

- Initialize and clean up the **metacp** workspace
- Open connections to directories
- Create and manage **metacp** handles
- Read attribute configuration files
- Perform entry management during directory synchronization
- Create and manage directory search results lists
- Perform string-handling operations
- Create trace files that record directory synchronization progress
- Create and manage index lists

You can also use **metacp** as a command-line entry administration tool to communicate with a Identity store and manage the entries contained within it.

Auditing Update Operation

metacp generates audit record information and stores it internally in the attribute **dxrHistory** of the current object whenever there are attribute changes (for a modify operation) or new objects with attributes that are relevant for auditing.

Therefore **metacp** reads the audit policies at startup time. It uses the bind DN and extracts the DirX Identity domain from that information. Once a valid domain name can be evaluated, it checks whether auditing is enabled at the domain (attribute **dxrWriteHistory** is set to **TRUE**). If auditing is enabled, it reads all the relevant audit policies that are defined in that domain; it therefore performs a search operation with base object **cn=Audit Policies,cn=Audit Trail,cn=***domain* and filter **(&(objectClass=dxrAuditTrailPolicy)(dxrWriteHistory=TRUE))**

Note that the attribute **dxrHistory** is only filled when calling **meta addentry** or **meta modifyentry**; it's not filled when calling **obj create** or **obj modify** as these two operations are basic operations that have the same functionality as DirX Directory **dirxcp**.

Sending JMS Messages

metacp can send JMS messages when new objects are created and when existing objects are modified, deleted or moved. It uses information that is available in change event policies; it therefore knows the relevant LDAP object classes of the objects for which such JMS events (messages) should be generated. This feature enables the DirX Identity product to run other workflows that should process the information about the changes of the objects.

Therefore **metacp** reads the change event policies at startup time. It uses the bind DN and extracts the DirX Identity domain from that information. Once a valid domain name can be evaluated, it looks for the relevant change event policies; it therefore performs a search operation

with base object **cn=***domain*
and filter **(&(objectClass= dxrChangeEventPolicy)(dxrIsActive=TRUE))**

Tcl Command Language

The meta controller is built on a portable command language called the tool command language (Tcl). Tcl permits the use of variables, if statements, list-processing functions, loop functions, and many other features commonly found in command languages. The program extends these features to provide a set of commands for directory synchronization and structured file I/O. The program also includes task scripts to help directory users and system administrators perform some routine management functions.

Including New Tcl Packages

The meta controller provides a few Tcl packages that are located in the following folder:

*install_path***/lib/tcl8.3**

The packages are:

- encoding
- http1.0
- http2.3
- msgcat1.1

To include more Tcl packages perform the following steps:

- Copy the new Tcl package as a subfolder below *install_path***/lib/tcl8.3**
- Rename all occurrences of the Tcl “list” command to “llist”

  metacp has defined an own command “list” that returns all LDAP entries below a given base node. Therefore, the original Tcl command “list” has been renamed to “llist”.

  Check the packages listed above to see what changes have been done; for example

check the usage of the following statements:

```
if {[info commands llist] != ""} then {
     # DIR.X version with usage of “llist”
} else {
      # original version with usage of “list”
}
```

The Directory Client Configuration File

The meta controller uses the directory client configuration file (**dirxcl.cfg**) to convert symbolic names of Identity stores to network addresses and to determine network addresses of default servers. See the DirX Identity Program Files chapter for details about this file.

Bind Types and Bind IDs

The meta controller supports multiple binds by using bind IDs and-for compatibility reasons-the concept of a default bind. A bind ID is a string associated with an LDAP bind (a connection to a Identity store through the LDAP v2 or LDAP v3 protocol). Specify the string value of a bind ID in an **obj bind** operation. (See the obj bind operation for details.) Use the bind ID in other operations to refer to a particular bind.

An **obj bind** command that does not specify a bind ID establishes a default bind. A command that does not specify a bind ID refers to the default bind. Only one default bind can exist at a time.

Synchronization Profiles and Synchronization Templates

A synchronization profile is a Tcl script (or a collection of Tcl scripts) that contains Tcl statements and meta controller commands that implement a specific synchronization scenario to be carried out by the meta controller. The synchronization profile is the mechanism that drives the meta controller’s directory synchronization functions. It consists of three sections:

- The variable section
- The mapping section
- The control logic section

The contents of these sections are specific to a given synchronization scenario. The next sections describe their general content.

The directory synchronization templates supplied in the Samples/metacp subdirectory of

the DirX Identity installation include sample synchronization profiles. See the Directory Synchronization Templates chapter for more details about these templates.

The Variable Section

The variable section provides a way to configure the synchronization profile using Tcl variables. Each Tcl variable defined in this section acts as a "profile switch" and is used to establish parameters for the synchronization process. Configurable information that can be set up with a profile switch includes:

- The names of the source and target attribute configuration files to be used
- The name of the meta controller trace file and the trace level to be set
- The Identity store bind and search parameters

The Mapping Section

The mapping section of a synchronization profile specifies rules for:

- Entry mapping
- Entry inclusion and exclusion
- Entry ownership
- Attribute mapping
- Attribute inclusion and exclusion
- Attribute value manipulation
- Attribute ownership

The Control Logic Section

The control logic section is the main body of the synchronization profile. It contains the Tcl script code and **metacp** operations that perform the directory synchronization.

Attribute Configuration Files

An attribute configuration file defines the attributes that are present in a particular connected directory and supplies formatting information that the meta controller is to use when processing import and export data files associated with the connected directory. An attribute configuration file must exist for each connected directory to be synchronized by the meta controller (that is, a corresponding attribute configuration file must exist for each DirX Identity agent present in DirX Identity). The attribute configuration file contains one record for each attribute in the connected directory.

During a directory synchronization operation, the meta controller reads source and target directory attribute configuration files (via a meta controller "read attribute configuration file" operation specified in the synchronization profile) to obtain attribute and data file formatting information about the directories it is synchronizing.

The meta controller uses attribute configuration files to determine connected directory

attribute abbreviations. Chapter 2 describes attribute configuration file format.

Invoking metacp

You can invoke **metacp** commands in interactive and command mode

Interactive Mode

Activate interactive mode by entering the **metacp** command without any arguments. At the **metacp** prompt, enter a **metacp** or Tcl command; **metacp** runs the command, displays the result, and is ready to accept another command.

```
% metacp
metacp> set home_dir /home/lionel
metacp> meta initialize -file $home_dir/tracefile -tracelevel 2
metacp>
```

Command Mode

Activate command-line mode from the system prompt by using one of the following methods:

- Enter the **metacp** command with the filename of a script that contains **metacp** commands, other valid Tcl commands, or both, as follows:

  ```
  % metacp my_sync_profile.tcl
  ```

- Enter the **metacp** command with the **-c** option followed by a list that contains one or more **metacp** commands or Tcl commands, for example:

  ```
  % metacp -c "bind; modify ou=Sales/o=pqr -addattr
  telephoneNumber=00498912312312"
  ```

  Enter multiple commands by separating them with a semicolon (;) and enclosing the commands in quotation marks (" "). Remember to escape shell metacharacters (for example, by enclosing them in quotation marks). Multiple commands must be on a single line.

  When you use the **-c** option, operation results return to the interpreter, not to the shell. If you enter multiple operations, the output of only the last operation is returned to the shell. You can solve this problem by using the following workaround:

  ```
  % metacp -c "bind; puts [obj show C=de -allattr]; puts [ldapargs show]"
  ```

Terminating metacp

Terminate an interactive **metacp** session by using the **exit** and **quit** commands. Use the

following syntax:

exit *n*

quit *n*

Use the *n* argument to specify the exit value returned to the shell. The following example terminates a session and returns an exit value of 56 to the shell:

```
metacp> exit 56
```

Exit Codes

By default, **metacp** returns zero (0) on success and a non-zero value if a command fails. The exit codes on failure are:

Code	Type	Description
1	Error	An invalid command line was specified, for example, an invalid option (for example, you specified **metacp -z**)
2	Error	The specified file does not exist (for example, you specified **metacp zz.tcl,** and **zz.tcl** does not exist)
3	Error	The evaluation of the file failed (for example, you supplied an incorrect command in the file)
4	Error	The evaluation of the command failed (for example, you specified an incorrect command on the **metacp -c** command line)
5	Error	Tcl initialization failed
6	Error	The initialization of the logging component failed.
7	Error	The shared libraries (or one of it's internal functions) used for Password Synchronisation couldn't be loaded.
8	Error	Setting up the environment for Password Synchronisation failed (e.g. initialization of crypt library; retrieval of private keys)
9	Error	Internal memory allocation problem.
10	Error	A serious error occurred during the execution of a script; a **meta** operation failed.
11	Error	A serious error while running the workflow occurred. The error will be returned if illegal parameter combinations are specified in control.tcl.
12	Error	The minimum required number of source directory entries is not available.
59	Error	An error occurred while sending the notification file.
60	Warning	A workflow terminated with warnings (e.g. because an entry to be deleted belongs to another master, etc.)
61	Warning	A directory update operation failed.

Code	Type	Description
200	Warning	Some operations could not be performed correctly (for example a join retrieved more than one hit). Manual intervention is necessary.

Scripts delivered with **metacp** use exit codes between 0 and 79 (exit codes 1 to 8 are generated by **metacp** itself, the remaining ones are generated by script logic of the standard script). Exit codes between 80 and 199 can be used in customer scripts. The use of exit codes lower than 80 or higher than 199 is prohibited in customer scripts.

Startup Scripts

When you invoke **metacp**, the following script files are run in the order shown:

[info library]/init.tcl
Contains the standard Tcl initialization scripts with definitions for the **unknown** command and the **auto_load** facility. The Tcl command **info library** returns the location of the file init.tcl.

$metacp_library/init.metacp
Contains the initialization scripts that implement the **metacp** commands and tasks. The implementation sets the Tcl variable **metacp_library** to *install_path***/client/conf** on Unix and *install_path***\bin** on Windows.

$HOME/.metacprc
Contains user customizations.

$TCL_PATH/*user_script***.tcl**
Contains user-defined scripts within one or more user-specified directories defined with the **TCL_PATH** environment variable.

Command Syntax

A **metacp** command has the following syntax:

[*object*] *operation* [*argument*] [-*option* [*opt_arg*]] ...

where:

object
specifies the name of one of the following **metacp** administration objects:

ats
Manages the asynchronous transport service

ldapargs
Manages service controls associated with an LDAP directory operation

meta

Performs directory synchronization operations

obj

Manages directory objects

Specifying an administration object is optional. If you do not specify an object on the command line, the default object is **obj** for operations that exist in both the **meta** and the **obj** objects (like **help** and **operations**).

Refer to the individual object reference pages for complete descriptions of these administration objects.

operation

Specifies the name of an action such as **readattrconf**, **getentry**, or **gethandle**, that is to be performed on an administration object. Refer to the individual object reference pages for complete descriptions of the operations supported by each **metacp** object. Common operations for all **metacp** objects are **help** and **operations**.

argument

Specifies the name of one or more specific objects to operate on. Most, but not all, **metacp** objects take an argument. Refer to the individual object reference pages for descriptions of the arguments supported by various objects.

option

Specifies a qualifier that controls the precise behavior of a **metacp** command. Most, but not all, **metacp** commands take options. Some options take an argument, *opt_arg*, that can be a name or a value.

Line Continuation

meta controller commands in scripts can use the backslash character (\) as the line continuation character. In interactive mode, there is no continuation character. Instead, you must continue typing. The line automatically wraps if your characters extend beyond the line end. If you press the **Enter**/**Return** key, the information you have typed is sent to the system for processing. You should press the **Enter**/**Return** key only when you have typed all information required for the command to process.

Command Processing

The **metacp** command supports the Tcl built-in commands as well as its own commands. A command unknown to **metacp** is passed to the **unknown** command, which evaluates it as follows:

- If **unknown** finds the command is in a **metacp** script file, **metacp** runs it.
- If **unknown** finds that the command is an executable UNIX or Windows program, **metacp** runs it. Consequently, you can invoke any UNIX or Windows command from the **metacp** prompt, for example,

`ls -l`.

- If **unknown** finds that you invoked the command at the top level of the **metacp** shell and that the command requests C-shell like history substitution (such as **!!**, **!***number* or **^***old***\^***new*), **metacp** emulates the C shell's history substitution.
- If **unknown** finds that you invoked the command at the top level of the **metacp** shell and the command is a unique abbreviation for another command, **metacp** runs that command.

Abbreviations

The **metacp** command uses two mechanisms to allow all object names, operation names, and options to be abbreviated to the shortest unique string in interactive commands.

The first mechanism relies on the **unknown** command, whose behavior is described in the **Command Processing** section of this reference page.

The second mechanism is built into the individual **metacp** commands themselves. This mechanism allows the operation name to be abbreviated to the shortest unique string supported by the object, and the option names to be abbreviated to the shortest unique string representing an option supported by an object and operation. For example, consider the following **metacp** commands:

```
metacp> meta getentry -source
metacp> meta operations
```

In the abbreviated form, the same operation can be entered as follows:

```
metacp> meta getentry -s
metacp> meta op
```

Although abbreviating commands is a good way to save keystrokes in typing interactive commands, abbreviations are not recommended for use in scripts. New procedures in scripts can cause abbreviations to become ambiguous. Furthermore, abbreviations are not always portable. When scripts move to other machines, some definitions may be left behind so scripts will not work correctly. Always spell out complete names in scripts.

Tcl Variables

All **metacp** commands set several variables on invocation. The variables contain the name of the directory in which startup scripts are stored, the return value of the last command, and so on. To avoid unnecessary typing, you can substitute the value of these variables into the next command.

Tcl variables behave just like other variables in **metacp**. Thus, you can trigger variable

substitution by placing a dollar sign (**$**) before the name of the variable. Alternatively, you can trigger substitution by using **set**. You can set the Tcl variables only by using the **metacp** program.

The **metacp** program defines the following variables:

_allowpartialresult

Specifies whether incomplete results are accepted as a result of a search operation.

If this variable is not specified or its value is **0** the synchronization profile script aborts subsequent processing if it receives an incomplete result.

Specify the following values:

- **0 -** Do not accept incomplete results.
- **1 -** Accept incomplete results that have been received due to a SIZE-LIMIT problem.
- **2 -** Accept incomplete results that have been received due to a TIME-LIMIT problem.
- **4 -** Accept incomplete results that have been received due to an ADMINISTRATIVE-LIMIT problem.
- **8 -** Accept incomplete results that have been received due to the presence of referrals.
- **16 -** For LDAPv2 protocol only: Accept incomplete results that have been received due to the presence of referrals.

If incomplete results due to several problems should be accepted you must combine the values by a logical OR. A value of **9** for example specifies that incomplete results due to a TIME-LIMIT problem and / or due to the presence of referrals should be accepted.

_entry_line

Specifies the current entry read from a datafile.

The **meta addentry**, **modifyentry**, **removeentries** use the last entry read from a data file as additional trace information in the tracefile. That makes sense, if an object read from a data file directly results in one of the operations just listed before.

But there are new synchronization scripts available (e.g. LDAP to LDAP synchronization) where update operations are invoked without having read a record from the data file.

Therefore the last record read from the data file should be cleared using the following command:

```
set entry_line ""
```

Otherwise the tracefile contains misleading trace information because the data record listed there has no relation to the operation that has been invoked.

_errormsg

--Specifies whether the LDAP server should return additional information both in case of errors and success:

- **TRUE** - Provide additional information from LDAP server.
- **FALSE** - Don't provide additional information from the LDAP server.

On startup, **metacp** behaves as if **_errormsg** has been set to **FALSE**.

The additional information either starts with "LDAP-Result" or "LDAP-ERROR".

Samples:

```
metacp> set _errormsg TRUE
metacp> bind -prot ldapv3
            {{LDAP-Result: Bind succeeded}}
metacp> modify cn=DomainAdmin,cn=my-company -replaceattr xxx=1234
            Error: Undefined attribute type passed in operation.
            (LDAP-ERROR: Cannot handle modification for attribute
type 'xxx')
```

_errormsgonly

Specifies whether the LDAP server should return additional information in case of errors.

- **TRUE** - Provide additional information from LDAP server.
- **FALSE** - Don't provide additional information from the LDAP server.

On startup, **metacp** behaves as if **_errormsgonly** has been set to **FALSE**.

The additional information starts with "LDAP-ERROR".

In case of errors the output is the same as described for TCL variable **_errormsg**.

_escapebackslash

Specifies whether the escape character backslash (\) is escaped by an additional backslash in requests and results. Specify one of the following keywords:

- **TRUE** - Backslashes are escaped.
- **FALSE** - Backslashes are not escaped.

_localcode

Specifies the character set that can be chosen from the character set table.

For compatibility reasons the following keywords are still valid:

- **LATIN1 -** Default Windows character set (only used in data files). This is the default value if no _localcode variable is defined.
- **UTF8 -** LDAP / Tcl character set (only used in data files).
- **PC850 -** PC DOS characer set only (only used in DOS windows).

See Code Conversion appendix for details.

_md_req_attr_limit (defined in init.metacp)

Specifies the maximum number of attributes that is passed in a search request as requested attributes. It is used to decide whether to pass the attribute list explicitly or whether to use the **-allattr** option. (For details see the **obj search** command.)

_partialresulttype

Specifies the reason for an incomplete result returned by a search operation. This variable is an output parameter of the search operation.

The search operation returns the following values:

- 1 **-** Incomplete results due to a SIZE-LIMIT problem.
- **2 -** Incomplete results due to a TIME-LIMIT problem.
- **4 -** Incomplete results due to an ADMINISTRATIVE-LIMIT problem.
- **8 -** Incomplete results due to the presence of referrals.
- **16 -** For LDAPv2 protocol only: Incomplete results due to the presence of referrals.

If incomplete results due to several problems are returned the return value is the combination of a logical OR. A value of **9** for example specifies that incomplete results due to a TIME-LIMIT problem and due to the presence of referrals are returned.

_sendchangeevents

Specifies whether change events shall be sent. Specify one of the following keywords:

- **TRUE** - send change events if change event policies are present.
- **FALSE** - disable the triggering of events even if event policies are enabled.

_trimSpaces

Specifies whether leading and trailing SPACE characters of attribute values are ignored when reading records from a data file. If **_trimSpaces** is not defined, **metacp** ignores leading and trailing SPACE characters. Specify one of the following keywords:

- **TRUE** - leading and trailing SPACE characters are ignored. (Default value.)
- **FALSE** - leading and trailing SPACE characters are not ignored.

If **_trimSpaces** is **FALSE** SPACE characters between the attribute prefix and the attribute value in the data file are considered as part of the attribute value. However, if the attribute prefix definition in the attribute configuration file contains for example trailing SPACE characters these spaces are not considered as part of the attribute value. (See section "Prefix" in chapter "Attribute Configuration File Format" for details.)

Example:

- The attribute prefix definition in the attribute configuration file is **'TelephoneNumber: '**. (Attend to the SPACE character at the end of the prefix definition.)

- **_trimSpaces** is **FALSE**.
- The data file contains the following records. (The " character is not part of the record.):

  one space behind the attribute prefix but no value

  ```
  "TelephoneNumber: "
  ```

  (one space between attribute prefix and value, two trailing spaces behind the attribute value)

  ```
  "TelephoneNumber: 12345  "
  ```

metacp evaluates the following attributes and values:

- attribute **TelephoneNumber** and no TelephoneNumber value
- attribute **TelephoneNumber** and the value **12345** (plus two trailing spaces)

metacp_library
: Holds the name of the directory in which the **metacp** startup scripts are stored. This variable cannot be set by the user.

When you specify script arguments on the **metacp** command line, the following Tcl global variables are set to the following values:

argv0
: *script_name*

argc
: number of script arguments

argv
: Tcl list of script arguments

When you specify the **-c** option on the **metacp** command line, or when you do not specify any arguments or options, these Tcl global variables are set as follows:

argv0
: program name of **metacp** as specified on the command line

argc
: **0**

argv
: An empty Tcl list (**\{ }**)

Line Recall and Editing

You can edit a line before it is sent to **metacp** by using control characters and escape sequences. To use a control character, press and hold down the **Ctrl** key while pressing the appropriate character key. To use an escape sequence, press and release the **Esc** key and then press and release one or more character keys. Escape sequences are case-sensitive; control characters are not.

You can enter an editing command anywhere on a line. In addition, you can press the **RETURN** key anywhere on the line.

To indicate that an action should be repeated a desired number of times, precede the escape or control characters with **Esc** *n,* where *n* is the number of times to repeat the action. For example, **ESC 4 Ctrl-d** (pressing the **Esc** key and the number **4** key and then pressing and holding down the **Ctrl** key while pressing the **d** key) deletes the next four characters on a line. **ESC 4 Esc DEL** deletes the previous four words on a line.

Control Characters for Editing

Use the control characters shown below for line editing. For example, press and hold down the Ctrl key while pressing the A key to move to the beginning of a line.

Ctrl a

Move to the beginning of the line.

Ctrl b

Move left (backward) one character. You can repeat this action by preceding the control characters with **Esc** *n*.

Ctrl d

Delete the character highlighted by the cursor. You can repeat this action by preceding the control characters with **Esc** *n*.

Ctrl e

Move to the end of the line.

Ctrl f

Move right (forward) one character. You can repeat this action by preceding the control characters with **Esc** *n*.

Ctrl g

Sound the terminal bell.

Ctrl h

Delete character before the cursor. You can repeat this action by preceding the control characters with **Esc** *n*.

Ctrl i

Complete the filename. (The **TAB** key performs the same function.) (See the Filename

Completion section.)

Ctrl j | Ctrl m

Send the completed line to **metacp**. (The **RETURN** key performs the same function).

Ctrl k

Delete to the end of the line (or column). You can repeat this action by preceding the control characters with **Esc** *n*.

Ctrl l

Redisplay the line.

Ctrl n

Get the next line from history You can repeat this action by preceding the control characters with **Esc** *n*.

Ctrl p

Get previous line from history You can repeat this action by preceding the control characters with **Esc** *n*.

Ctrl r

Search backward (or forward if *n*) through history for text; start a line if text begins with an up arrow.

Ctrl t

Transpose characters.

Ctrl v

Insert next character even if it is an edit command.

Ctrl w

Delete to the mark. See **Esc SPC** in the table of escape characters.

Ctrl x Ctrl x

Exchange the current location and mark. See **Esc SPC** in the table of escape characters.

Ctrl y

Restore the last deleted text to the current cursor location.

Ctrl [

Start an escape sequence. (The **Esc** key performs the same function.)

Ctrl]c

Move forward to the character indicated by c.

Ctrl ?

Delete the character before the cursor You can repeat this action by preceding the control characters with **Esc** *n*.

Escape Characters for Editing

To use an escape character, press the **Esc** key, release it, and then press the appropriate character key. For example to delete the previous word, press the **Esc** key, then the **Ctrl** key while pressing the **H** (capital H) key. Escape characters are case-sensitive, so follow the capitalization in the table.

ESC Ctrl H | ESC DEL

Delete the previous word (the action can also be performed by the **BACKSPACE** key). You can repeat this action by preceding the escape characters with **Esc** *n*.

ESC SPC

Set the mark (this action can also be performed by the **SPACE BAR**); Refer to the **Ctrl x Ctrl x** and **Ctrl w** control characters in the control character table.

ESC .

Get the last word from the previous line. You can repeat this action by preceding the escape characters with **Esc** *n*.

ESC ?

Show possible filename completions (see the Filename Completion section in this chapter).

ESC <

Move to the start of history.

ESC >

Move to the end of history.

ESC b

Move backward one word. You can repeat this action by preceding the escape characters with **Esc** *n*.

ESC d

Delete the word highlighted by the cursor. You can repeat this action by preceding the escape characters with **Esc** *n*.

ESC f

Move forward one word. You can repeat this action by preceding the escape characters with **Esc** *n*.

ESC l

Make the highlighted word lowercase. You can repeat this action by preceding the escape characters with **Esc** *n*.

ESC u

Make the highlighted word uppercase. You can repeat this action by preceding the escape characters with **Esc** *n*.

ESC y

Restore the last deleted text to the current cursor location.

ESC w

Copy the text from the cursor position up to the up to mark.

ESC *nn*

Set repeat count to the number indicated by *nn*.

Filename Completion

The **metacp** command supports filename completion. For example, suppose the root directory has the following files in it:

readme

readme.txt

If you type a command and *characters* and then press the **TAB** key, **metacp** completes as much of the filenames as possible given the characters supplied by *characters*. For example, if you type **ls** and then press the **TAB** key, **metacp** completes the filename the name as far as **readme**. However, because **readme** could be the file named **readme** or the file named **readme.txt**, **metacp** cannot complete the filenames, so it beeps to signal the conflict. If you then press **Esc-?, metacp** displays the two possible names: **readme** and **readme.txt** and prompts you with **ls readme**. You can then complete the entire filename or enough of the filename for **metacp** to complete it.

Example

The following command sequence illustrates the use of **metacp** and Tcl commands to create an entry in a connected directory. In this example, **metacp** has already been invoked.

```
puts "create new entry"
catch { meta addentry -entry rh_LDAP } status

if {$status == ""}
then {puts "operation ok"}
else {puts "$status"}
```

Return Values

All **metacp** operations return either a NULL to indicate the successful completion of an operation or a list of information requested by the user (such as the results of a search operation). If an error occurs, **metacp** returns an error message. The program uses the Tcl native error handling facility to log additional error information returned from commands.

This information is stored in two global variables:

- The **errorInfo** variable, which contains the stack trace of the error messages.
- The **errorCode** variable, which is a Tcl list that contains two elements, **METACP** (which identifies the program) and the numeric value of the error code. The following table lists the **metacp** error codes and error messages that can be returned

4401
: error while reading abbreviation file or syntax error with respect to abbreviations, e.g. abbreviation unknown

4402
: errors while converting string to internal structures and vice versa e.g. invalid attribute value

4403
: Conflicting option ...

4404
: Unknown option ...

4405
: Missing RDN.

4406
: Missing new superior.

4407
: Missing object name.

4408
: Missing attribute information.

4409
: Missing information for the new attribute values.

4410
: Missing Directory Service Agent name.

4411
: Missing Presentation Service Access Point address.

4412
: Missing user name.

4413
: Missing password.

4414
Missing filter value.

4415
Missing scope of search.

4416
Missing time limit.

4417
Missing size limit.

4418
Missing target system.

4419
Missing arguments.

4420
Invalid time limit ...

4421
Invalid size limit ...

4422
Unknown argument ...

4423
Unable to initialize workspace.

4424
Perform bind operation first.

4425
Missing value.

4426
Format error - ...

4427
Mismatched quotes - ...

4428
Invalid parameter passed as an argument.

4429
Too many arguments.

4430
Entry not found.

4431
Subordinates not found.

4432
Insufficient memory to perform operation.

4435
Ambiguous option ...

4436
Too many values.

4437
Missing authentication type.

4438
Unknown error.

4447
Invalid argument *argument_value*.

4501
Wrong bind session type. (Session type is DAP session whereas LDAP session is required (or vice versa).)

4502
Missing bind id.

4503
Invalid bind id ...

4504
Latin.1/UTF8 conversion failed.

4505
Bind session identifier already in use.

4506
Missing server name.

4507
Missing server address.

4509
Missing protocol.

4510: Invalid protocol ...

4511

Cannot set "_cwo": _cwo is supported for a default DAP bind only.

4513

A structured attribute can’t be used in the sort keys of paged results. (Only relevant for DAP protocol.)

4514

Wrong syntax for sortkey(s).

4515

Paging is not possible in this state.

4517

Invalid value (either for page size or paging type PAGING (for LDAP)).

4518

Missing path name for Certificate-DB.

4519

SSL/SASL connections not supported (on SVR4).

4520

ASN.1 BER encoding failed.

4521

ASN.1 BER initialization failed.

4522

ASN.1 BER decoding failed.

4523

LDAP result control not found.

4524

Base64-decoding/encoding failed.

4525

Missing mechanism.

4526

Missing Key3DB - file name.

4527

Missing 'key3password' value.

4528
Missing 'certsubject' value.

4529
Missing parameter '-sasl'.

4601
Unknown option.

4602
Unknown argument.

4603
Invalid argument ...

4605
Unknown operation ...

4606
Missing operation.

4607
Too many arguments.

4608
Ambiguous operation.

4609
Error: Cannot unset ...

4610
Insufficient memory.

4611
Missing bind id.

4615
Conflicting option ...

4801
Operation affects multiple DSAs.

4802
An alias is encountered where an alias is not permitted.

4803
An alias is dereferenced that names an object that does not exist.

4804
ASN.1 decoding failed.

4805
ASN.1 encoding failed.

4806
Initialization of ASN.1 utility failed.

4807
Attribute or attribute value already exists.

4808
Bad argument.

4809
Bad class.

4810
Bad context.

4811
Bad name.

4812
Bad session.

4813
Bad workspace.

4814
Directory system is busy.

4815
Operation can’t be abandoned.

4816
Operation requires chaining operation mode.

4817
Problem with communication stack.

4818
DUA configuration file missing.

4819
DUA configuration file corrupted.

4820
Connection is busy.

4821
Constraint violation.

4822
DIT is inconsistent.

4823
Object already exists.

4824
Credentials expired.

4825
Fatal error.

4826
The function does not apply to the object to which it is addressed.

4827
The function was aborted by an external force.

4828
Communication problem (ICOM-attach failed).

4829
Communication problem (ICOM-detach failed).

4830
Communication problem (ICOM-receive failed).

4831
Communication problem (ICOM-send failed).

4832
Inappropriate authentication.

4833
Inappropriate matching.

4834
Initialization failed.

4835
Insufficient access rights.

4836
Operation interrupted.

4837
Invalid attribute syntax.

4838
Invalid attribute value.

4839
Invalid credentials.

4840
Invalid memory reference.

4841
Invalid signature.

4842
Invalid workspace.

4843
Loop detected while performing operation.

4844
Insufficient memory to perform operation.

4845
Fatal error.

4846
The attribute type is not included in the AVA.

4847
An attempt was made to start a synchronous operation with outstanding asynchronous operations.

4848
Naming violation.

4849
Network problems.

4850
Security error with no other information being available.

4851
Attribute or attribute value doesn’t exist.

4852

The object doesn’t exist.

4853

Operation to be abandoned doesn’t exist.

4854

Workspace doesn’t exist.

4855

Operation not allowed on non-leaf entries.

4856

Operation not allowed on RDN.

4857

Unsupported feature.

4858

Object class modification prohibited.

4859

Object class violation.

4860

Referral or partial outcome qualifier is outside the required scope.

4861

An invalid pointer supplied as a function argument.

4862

Protection required (signed operation required).

4863

Generation of signature failed.

4864

Validation of signature failed.

4865

System error.

4866

Temporary difficulty encountered.

4867

Time Limit Exceeded

4868
Abandon failed: too late.

4869
Too many outstanding operations.

4870
Too many sessions. No more session can be started.

4871
The DSA does not have the administrative authority over the particular naming context.

4872
Directory system unavailable.

4873
Critical service extension cannot be provided.

4874
Undefined attribute type passed in operation.

4875
Unexpected PDU passed/received in operation.

4876
Directory system is unwilling to perform the operation.

4877
Warning.

4878
Local abort received.

4879
Abandon received.

4880
Can’t invoke operation.

4881
Communication problem (Can’t create subscriber).

4882
Bind to DSA failed.

4883
Can’t convert DSA name of DUA configuration file.

4884
Can’t convert requestor name.

4885
Can’t convert target name of DUA configuration file.

4886
Can’t convert DSA address of DUA configuration file.

4887
Communication problem (Can’t enable binding).

4888
Communication problem (Can’t delete binding).

4889
(Unknown) error returned from DSA.

4890
Incoming authentication failed.

4891
Outgoing authentication failed.

4892
Remote abort received.

4893
(Unknown) service error.

4894
(Unknown) attribute error.

4895
(Unknown) name error.

4896
Referral returned.

4897
(Unknown) security error.

4898
Abandon failed.

4899
(Unknown) update error.

4900
Administrative Limit Exceeded

4901
Invalid query reference.

4902
Unknown error from DSA received.

4903
Reject received.

4904
Missing default DSA / LDAP server.

4905
Illegal LDAP filter.

4906
LDAP protocol error.

4907
Unknown DUA error.

4908
New superior object doesn't exist.

4909
LDAP operation error.

4910
Offset range error.

4911
Missing sort control.

4912
Evaluation Copy expired.

Attribute Configuration File Errors

Attribute configuration file errors are returned in one of two formats:

1. *error* - Error in line number *number* of attribute configuration file *file*
2. *error* - Error in attribute configuration file *file*

The errors are listed below.

6001

Cannot open file (error format 2)

6002

Cannot read from file (error format 2)

6003

Missing quotes (error format 1)

6004

Missing attribute abbreviation (error format 1)

6005

Missing attribute name (error format 1)

6006

Missing attribute prefix (error format 1)

6007

Missing attribute suffix (error format 1)

6008

Missing structured information for an attribute (error format 1)

6009

Illegal structured information for an attribute (error format 1)

6010

Missing recurring separator (error format 1)

6011

Missing matching rule (error format 1)

6012

An abbreviation is used that is not yet defined in the attribute configuration file (error format 1)

6013

Illegal field length of an attribute (error format 1)

6014

Illegal matching rule of an attribute (error format 1)

6015

Illegal value for number of lines to be skipped (error format 1)

6016

Invalid octal value (error format 1)

6017

Duplicate abbreviation of an attribute. (error format 1)

6018

Duplicate attribute name of an attribute. (error format 1)

6019

Duplicate prefix of an attribute. (error format 1)

Command-Line Syntax Errors

The command-line syntax errors are listed below.

6050

Option *option* not specified (the option has not been specified, but is mandatory in this context)

6051

Attribute configuration file does not contain abbreviation *abbreviation* (an abbreviation has been used that is not defined in the related attribute configuration file)

6052

Unknown flag - "*flag*" (an invalid flag value has been specified)

6053

Invalid sort order - "*order*" (an invalid sort order has been specified)

6054

Invalid mark tag - "*tag*" (an invalid mark tag has been specified)

6055

Missing RDN (a required RDN argument has not been specified)

6056

Missing attribute information

6057

Conflicting argument - "*argument*"

6058

Illegal combination of flags - "*string*"

6059

Invalid initialization mode - "*string*"

Miscellaneous Error Conditions

The miscellaneous error conditions are listed below.

6100

Format error in TCL list - "*list*"

6101

Memory insufficient

6102

Please call meta initialize first

6103

Inappropriate file mode of handle "*handle_name*" (A connection handle of type FILE does not have the appropriate file mode for performing the operation)

6104

Superior information already exists (superior information has already been specified for the given DN)

6105

Superior information not found

6106

Inappropriate connected directory type of handle *handle_name* (the handle specified by *handle_name* is not of the correct subtype for performing the operation)

6107

'Add' operation failed (consult the trace file for the reason)

6108

'Remove' operation failed (consult the trace file for the reason)

6109

'Modify' operation failed (consult the trace file for the reason)

6110

'Modify-DN' operation failed (consult the trace file for the reason)

6111

Cannot write object into dump file (returned by **meta writerecord**)

6112

Unknown handle passed as input parameter - "*handle_name*"

6113

Handle is not the correct type - "*handle_name*"

6114

Cannot replace or delete handle. It is still used for handle "*handle_name*" (an attempt was made to overwrite or delete a handle that is used by one or more dependent handles)

6115

Handle name used for input and result handle - "*handle_name*"

6116

Operation code must be defined if only new entry is specified (a **meta modifyentry** operation was attempted without specifying the **-oldentry** option

6117

Cannot set Tcl variable array "*handle___name*" (consult the exception log file for the reason)

6118

Cannot get Tcl variable array "*handle_name*" (consult the exception log file for the reason)

6119

Cannot locate record with the given file offset

6120

Cannot determine file offset of the input data record

6121

Illegal file offset provided for index list

6122

Cannot append new elements to the already sorted index list

6123

Index list has already been sorted

6124

Invalid record (entry in data file is not in the required format)

6125

Error while skipping lines at the beginning of file "*file_name*"

6126

Unknown operation code "*string*" encountered

6127

Unknown attribute modification "*string*" encountered

6128

Handling of LDIF changes is not enabled

6129

Old entry must not be specified if new entry represents LDIF changes

6130

Tag specification not supported for the type of the given handle - "*handle_name*". (**gethandle** with -**reset** on a handle whose type is different from RESULT)

6131

Operation not supported for unsorted results

6132

Operation not supported for input handles with the LDIF change property

6133

Error while parsing LDAP name "*DN-string*"

6134

Name specification not supported for the type of the given handle - "*handle_name*"

6135

Index list is not sorted.

6136

Names have not been kept in index list.

6137

No values available for attribute '*attribute_value*'

6138

Invalid component in last RDN value.

6139

Base64-decoding/encoding failed.

6141

Data file is locked.

6142

Incomplete search result returned.

6143

Sorting criteria is not a single valued attribute.

6144

Unknown encoding.

6149

Sort modes are different.

6150
Sort keys are different.

6151
Bound connections are different.

6152
Exact action flags are different.

6400
'ats initialize' has to be called first.

6402
Error while opening the protocol file.

6403
Error while reading the protocol file.

6405
Error while writing the protocol file.

6405
Illegal timeout value.

The program also provides a **catch** command to help scripts catch errors and invoke error handlers.

Logging

The meta controller uses the DirX Identity client logging configuration files **dirxlog.on**, **dirxlog.off** and **dirxlog.cfg**. To set up these files to log information for the meta controller, insert the following line after the tracing specification of component **dir** in the file *install_path***/client/conf/dirxlog.cfg** or in the file specified in the **DIRX_LOGCFG_FILE** environment variable:

mdi:meta.1.9:GOESTO:dir

Supported logging levels are:

- **0** - Disable trace logging
- **1** - Perform entry and exit logging; at this level, all **metacp** functions except those which are frequently called are logged
- **2** - Log frequently-called **metacp** functions as well as performing entry and exit logging
- **9** - Log structures of functions

Here is an example of a **dirxlog.cfg** file; the **metacp** logging line is highlighted in boldface text:

```
dir:sys.1,sock.1,vthr.1,icom.1,shr.0,ctx.0,\
osi.1,rpc.1,ros.1,adm.1,\
sec.1,api.1,daspp.1,\
bth.1,norm.0,match.0,\
conf.1,asn1.0,util.1:BINFILE.6.2000:%s/client/log/LOG%d
mdi:meta.1.2:GOESTO:dir
FATAL:GOESTO:dir
ERROR:TEXTFILE.1.100:%s/client/log/USR%d
NOTICE:GOESTO:ERROR
WARNING:GOESTO:FATAL
NOTICE_VERBOSE:GOESTO:FATAL
```

Use the **metacpdump** program to view the contents of the generated log file.

For complete information about the logging components discussed here, see:

- The DirX Identity Program Files chapter for a description of logging configuration file format.
- The Environment Variables chapter for a description of the directory client program environment variables.
- The description of metacpdump in this chapter for program usage information.

Viewing Output

In interactive mode, Tcl lists are written to **stdout**. However, in command-line mode, the Tcl lists are not displayed. To display them, use the Tcl command **puts**. For example, the commands:

```
% metacp -c "bind; puts [show cn=naik,ou=Sales,o=pqr,c=de]"
% metacp -c "bind; catch {show cn=naik,ou=Sales,o=pqr,c=de}
result; puts $result"
```

print the results of a **obj show** operation to **stdout**.

Many of the **metacp** operations take a **-pretty** option, which causes the results of the operation to be formatted into tables. Each page of **-pretty** output is 23 lines in length. Use the following **metacp** scrolling commands to view multiple pages of output:

n

View the *n*th page

-[*n*]

Skip [n pages] backward

+[*n*]
Skip [n pages] forward

$
View the last page

q
Exit viewing

SPACE
Advance to the next page

CR
Advance one line

If you specify the **-pretty** option, the return value is empty (not a Tcl list).

LDAP Session Tracking

Session Tracking was introduced to improve LDAP audit logging. For each LDAP operation, it enables the user to identify the DirX Identity component, the directory user and the client address of the computer where DirX Identity is running.

metacp supports LDAP session tracking.

If the LDAP server of the DirX Directory installation supports the **Session Tracking Control**, **metacp** provides the following information in the LDAP audit record for each operation:

Controls # :1
Ctrl Type :1.3.6.1.4.1.21008.108.63.1 (Session Tracking Control)
Critical :no
SID-IP :*IP_address_of_computer_where_metacp_is_running*
SID-Name :DXI metacp [*process_ID_of_metacp_process***]**
SID-Oid :1.3.6.1.4.1.21008.108.63.1.3 (Sasl-Auth-Username)
SID-Info :*user name_of_connected_user*

SID-Name :DXI metacp [*process_ID_of_metacp_process***]** helps you to identify all LDAP operations that **metacp** invoked. The *process_ID_of_metacp_process* and **SID-IP :***IP_address_of_computer_where_metacp_is_running* identifies a single **metacp** process that invoked the LDAP operation.

In the following excerpt of an LDAP audit logfile the user with the common name **cn=DomainAdmin,cn=MyCompany** performed a bind operation over the **metacp** process with the process id **7596** from the computer with the IP address **10.93.25.163**. Then the user performed a search and finally an unbind operation. The session tracking information identifies the user and the **metacp** process involved:

```
----------------- OPERATION 000002 ----------------
```

```
 Create Time    :Tue Sep 8 16:02:18.977999 2015
 Start Time     :Tue Sep 8 16:02:18.977999 2015
 End Time       :Tue Sep 8 16:02:18.977999 2015
 OpUUID         :400b39df-3c0e-4e74-baf8-545864b4a93a
 DapBindId      :00160008
 Concurrency    :1
 OpStackSize    :1
 OpFlow In/Out  :0/0
 Duration       :0.000000 sec
 User           :cn=DomainAdmin,cn=My-Company
 IP+Port+Sd     :[127.0.0.1]+52645+1568
 Op-Name        :LDAP_Con34_Op0
 Operation      :BIND
 Version        :3
 MessageID      :1
 Bind-Type      :simple
 Security       :normal
 DAP-Share-Count:1
 Controls #     :1
   Ctrl Type    :1.3.6.1.4.1.21008.108.63.1 (Session Tracking
Control)
   Critical     :no
   SID-IP       :10.93.25.163
   SID-Name     :DXI metacp [7596]
   SID-Oid      :1.3.6.1.4.1.21008.108.63.1.3 (Sasl-Auth-Username)
   SID-Info     :cn=DomainAdmin,cn=My-Company
 Bytes Received :178
 Bytes Returned :29
 Socket Mode    :plain
 Abandoned      :no
 Result Code    :0 (success)
 Error Message  :Bind succeeded.

---------------- OPERATION 000003 ----------------
 Create Time    :Tue Sep 8 16:02:37.384999 2015
 Start Time     :Tue Sep 8 16:02:37.384999 2015
 End Time       :Tue Sep 8 16:02:37.384999 2015
 OpUUID         :ba682d7b-216e-452c-be18-ba266a70ca8e
 DapBindId      :00160008
 Concurrency    :1
 OpStackSize    :1
```

```
 OpFlow In/Out  :0/0
 Duration       :0.000000 sec
 User           :cn=DomainAdmin,cn=My-Company
 IP+Port+Sd     :[127.0.0.1]+52645+1568
 Op-Name        :LDAP_Con34_Op1
 Operation      :SEARCH
 Version        :3
 MessageID      :2
 Base Obj       :cn=My-Company
 Scope          :baselevel
 Filter         :(objectclass=PRES
 Size Limit     :0
 Time Limit     :0
 Deref Alias    :always
 Types Only     :no
 Req Attr #     :1
   Req Attr     :* (all user attributes)
 Found Entries  :1
 Found Attrs    :41
 Found Values   :50
 Controls #     :1
   Ctrl Type    :1.3.6.1.4.1.21008.108.63.1 (Session Tracking
Control)
   Critical     :no
   SID-IP       :10.93.25.163
   SID-Name     :DXI metacp [7596]
   SID-Oid      :1.3.6.1.4.1.21008.108.63.1.3 (Sasl-Auth-Username)
   SID-Info     :cn=DomainAdmin,cn=My-Company
 Bytes Received :187
 Bytes Returned :1574
 Socket Mode    :plain
 Cached Result  :no
 Abandoned      :no
 Result Code    :0 (success)
 Error Message  :Search succeeded. Found 1 Entries (0 Aliases), 41
Attributes, 50 Values. (ChainedResult=no)

----------------- OPERATION 000004 ----------------
 Create Time    :Tue Sep 8 16:02:45.388000 2015
 Start Time     :Tue Sep 8 16:02:45.388000 2015
 End Time       :Tue Sep 8 16:02:45.388000 2015
```

```
 OpUUID          :873c7078-e9c1-4831-b470-e9bcbff7f153
 DapBindId       :00160008
 Concurrency     :1
 OpStackSize     :1
 OpFlow In/Out   :0/0
 Duration        :0.000000 sec
 User            :cn=DomainAdmin,cn=My-Company
 IP+Port+Sd      :[127.0.0.1]+52645+1568
 Op-Name         :LDAP_Con34_Op2
 Operation       :UNBIND
 Version         :3
 MessageID       :3
 Controls #      :1
   Ctrl Type     :1.3.6.1.4.1.21008.108.63.1 (Session Tracking
Control)
   Critical      :no
   SID-IP        :10.93.25.163
   SID-Name      :DXI metacp [7596]
   SID-Oid       :1.3.6.1.4.1.21008.108.63.1.3 (Sasl-Auth-Username)
   SID-Info      :cn=DomainAdmin,cn=My-Company
 Bytes Received :139
 Bytes Returned :0
 Socket Mode     :plain
 Abandoned       :no
 Result Code     :0 (success)
```

See Also

ats, ldapargs, meta, obj, util

1.1.1. ats (metacp)

Synopsis

ats acknowledge
[**-handle** *handle_name*]
-messageid *message_ID*
-topic *topic_name*
-timeout *timeout*

ats initialize
[**-bindid** *bind_session_id*]
[**-handle** *handle_name*]

[**-prot** *protocol_file_name*]
[**-repeat** *repeat_count*]
[**-servername** *MSS_servername*]
[**-timeout** *timeout*]
[**-topicprefix** *topic_prefix*]
[**-type** *type*]
[**-cluster** *cluster*]
[**-resource** *resource*]

ats publish
[**-handle** *handle_name*]
-header *message_header_information*
[**-message** *message_body*]
-topic *topic_list*

ats receive
[**-handle** *handle_name*]
-timeout *timeout*
-topic *topic*

ats replay
-prot *protocol_file_name*

ats send
[**-handle** *handle_name*]
-header *message_header_information*
[**-message** *message_body*]
-topic *topic*

ats subscribe
[**-handle** *handle_name*]
-topic *topic_list*
-timeout *timeout*
[**-onemessage**]
[**-nodestroy**]

ats terminate
[**-handle** *handle_name*]

ats help

ats operations

Purpose

A **metacp** object that manages the asynchronous transport service. This is a messaging service that is based on JMS (Java messaging service). It supports the publish - subscribe model. A unlimited number of publisher clients can send messages to a defined topic to the message server. Other clients can subscribe to topics. That is that they get all messages that were published to a specific topic. There are no specific rules how to define topics and

messages.

Arguments

operation

The name of the **ats** operation for which to display help information.

Operations

ats acknowledge

Destroys a message from the message server. The syntax is as follows:

ats acknowledge
[**-handle** *handle_name*]
-messageid *message_ID*
-topic *topic_name*
-timeout *timeout*

Options

-handle **handle_name**

Specifies an ATS session identifier in order to support multiple ATS sessions in parallel

-messageid **message_ID**

Specifies the message id of the message that is to be destroyed from the server. The *message_ID* is the result returned by the corresponding **ats subscribe** operation. It is a base64 encoded string that represents the internal message identifier of the underlying messaging system, for example QU1RIFFNX3NjaG1pZDAyIBS8nz8gABYF.

-topic **topic_name**

Specifies the name of the topic the client uses for deleting the message, for example dxm.event.pwd.changed.

-timeout **timeout**

The operation stops after *timeout* seconds. A value of **0** implies that the underlying message system returns after destroying the message.

The **ats acknowledge** operation destroys the message with the specified ID from the server.

Example

```
ats acknowledge
      -messageid QU1RIFFNX3NjaG1pZDAyIBS8nz8gABYF
      -topic dxm.event.pwd.changed
```

```
      -timeout 1
```

ats initialize

Initializes a connection to the message server. The syntax is as follows:

ats initialize
[**-bindid** *bind_session_id*]
[**-handle** *handle_name*]
[**-prot** *protocol_file_name*]
[**-repeat** *repeat_count*]
[**-servername** *MSS_servername*]
[**-timeout** *timeout*]
[**-topicprefix** *topic_prefix*]
[**-type** *type*]
[**-cluster** *cluster*]
[**-resource** *resource*]

Options

-bindid **bind_session_id**

-handle **handle_name**

Specifies an ATS session identifier in order to support multiple ATS sessions in parallel

-prot **protocol_file_name**

Specifies the filename of the protocol file. All messages sent and all messages received are written into this file. This file can be used for the **ats replay** operation if the same messages should be re-sent.

Publish operations are written into the protocol file in the following format:

Operation=publish
Header=*message_header_1*
Header=*message_header_2*
...
Message=*message*
Topic=*topic_1*
Topic=*topic_2*

Subscribe operations are written into the protocol file in the following format:

Operation=subscribe
Header=*message_header_1*
Header=*message_header_2*
Topic=*topic_1*
...
TimeOut=*time_out_in_seconds*
Message=*message*

Currently only one ATS connection is supported in the protocol file. Furthermore no other options can be specified together with the options –prot.

***-repeat* repeat_count**

Specifies the number of retries for retrieving the message queue server information. Default value, if missing, is 10.

***-timeout* timeout**

Specifies the timeout to be used for retrieving the message queue server information. Default value, if missing, is 30.

***-topicprefix* topic_prefix**

Specifies the topic prefix to be used as part of a JMS topic if **metacp** should send automatically JMS messages (containing the SMPL representation of an update request), for example "dxm.event" as part of the topic

dxm.event.LDAP.cluster='my-cluster'.resource='my-resource'

***-type* type**

Specifies the type to be used as part of a JMS topic if **metacp** should send automatically JMS messages (containing the SMPL representation of an update request), for example "LDAP" as part of the topic

dxm.event.**LDAP**.cluster='my-cluster'.resource='my-resource'

***-cluster* cluster**

Specifies the cluster to be used as part of a JMS topic if **metacp** should send automatically JMS messages (containing the SMPL representation of an update request), for example "my-cluster" as part of the topic

dxm.event.LDAP.cluster='**my-cluster**'.resource='my-resource'

***-resource* resource**

Specifies the resource to be used as part of a JMS topic if **metacp** should send automatically JMS messages (containing the SMPL representation of an update request), for example "my-resource" as part of the topic

dxm.event.LDAP.cluster='my-cluster'.resource='**my-resource**'

The **ats initialize** operation initializes connection to the message server. The connection must be terminated by performing an **ats terminate** operation.

The message server is identified by using an MSS server name and retrieving its related message server. The relevant information is read from the directory and therefore an LDAP connection needs to be available.

There are two ways to select a message server:

- If no options are given, the **ats initialize** operation reads the bind information stored in

the configuration file *install_path*/server/conf/dxmmsssvr.ini* to retrieve all relevant information from the DIT for setting up a connection to the messaging system. In that case, the complete set of bind information must be available in the configuration file, namely the password (and optionally PIN).

- In order to setup a JMS connection to any other message server, the MSS server name can be given in the parameter **–servername**. An existing LDAP connection is required and is identified either by the bind session given in **–bindid**, or the default LDAP bind session is used. Furthermore the parameters **–repeat** and **–timeout** will be evaluated.

If **metacp** should propagate JMS requests (containing the SPML representation of an update request) to a given JMS topic then specify the options **-topicprefix**, **-type**, **-cluster**, and **-resource**. When **metacp** later on performs the update operations (using **meta addentry**, **meta modifyentry**, **meta removeentries**, or **meta modifydn**) JMS messages are propagated automatically to the given topic.

Example

```
ats initialize -prot ATS_trace01
ats initialize -servername myMSS -bindid myLDAPbind
```

ats publish

Publishes a message. The syntax is as follows:

ats publish
[**-handle** *handle_name*]
-header *message_header_information*
[**-message** *message_body*]
-topic *topic_list*

Options

-*handle* handle_name

Specifies an ATS session identifier in order to support multiple ATS sessions in parallel.

-*header* message_header_information

Is a comma separated list of header fields, for example DXMVersion=1.0,DXMInstId=100,DXMOperation=...,...

You can define your own messages. See the event manager utility script in the DirX Identity default applications for examples for messages.

-*message* message_body

Specifies the message text. There are no specific rules how to define the message text.

-*topic* topic_list

Is a comma separated list of topics for that the message is published, for example

dxm.command.main,DXM.stateTracker,DXM.myserver.fileservice.

The **ats publish** operation publishes a message. The operation returns the message ID on success.

Example

1. The following **ats publish** operations creates a workflow instance at the server. It returns the message ID main_1067593458_2888_0:

```
ats publish
   -topic dxm.command.main
   -header DXMReplyTo=dxm.command.event.mss_reply,
       JMSReplyTo=dxm.command.event.mss_reply,
       DXMVersion=1.10,
       DXMObjectName=cn=BA_MetaStore2CSVfile_Full\,
         dxmC=CSVfile\,
         dxmC=uid-c0671b98-96a4c-e98fe4f404--7f95\,
         dxmC=Workflows\,
         dxmC=DirXmetahub,
       DXMObjectType=3,
       DXMInitiatorType=1,
       DXMType=1,
       JMSType=1
```

2. The following **ats publish** operation executes the instance created in example 1. It returns the message ID main_1067593458_2888_1:

```
ats publish
   -topic dxm.command.main
   -header DXMReplyTo=dxm.command.event.mss_reply,
        JMSReplyTo=dxm.command.event.mss_reply,
        DXMVersion=1.10,
        DXMObjectName=cn=BA_MetaStore2CSVfile_Full\,
          dxmC=CSVfile\,
          dxmC=uid-c0671b98-96a4c-e98fe4f404--7f95\,
          dxmC=Workflows\,
          dxmC=DirXmetahub,
        DXMOperation=10,
        DXMWorkflowActivity=,
        DXMType=8,
        JMSType=8,
```

```
        DXMInstId=main_1067593458_2888_0
```

3. The following **ats publish** operation destroys the instance created in example 1. It returns the message ID main_1067593458_2888_2:

```
ats publish
   -topic dxm.command.main
   -header DXMReplyTo=dxm.command.event.mss_reply,
       JMSReplyTo=dxm.command.event.mss_reply,
       DXMVersion=1.10,
       DXMType=2,
       JMSType=2,
       DXMInstId=main_1067593458_2888_1
```

ats receive

Receives messages from the message service's queue using specified topic (that identifies the message service's queue). The syntax is as follows:

ats recive
  [**-handle** *handle_name*]
 -timeout *timeout*
 -topic *topic*

Options

-handle **handle_name**

Specifies an ATS session identifier in order to support multiple ATS sessions in parallel.

-timeout **timeout**

The operation stops after *timeout* seconds. A value of **0** implies that operation returns all currently available messages and then stops.

-topic **topic**

Specifies the topic name that gets mapped to the message service's queue name.

The **ats receive** operation reads a message from the message service's queue defined in the option **-topic**.

Example

```
ats receive -topic dxm.my.topic -timeout 5
```

If you look at the sample of **ats send** then the result is the following:

```
{dxm.my.topic DXMVersion=1,DXMType=10 {Hello world}}
```

ats replay

Re-sends all publish operations that have been written to the specified protocol file in another session. The syntax is as follows:

ats replay
  -prot *protocol_file_name*

Options

***-prot* protocol_file_name**

Specifies the filename of the protocol file that has been written in another session.

The **ats replay** operation performs all publish operations that it finds in the specified protocol file. (See also the **ats initialize** operation.)

Example

```
*ats replay -prot ATS_trace01*
```

ats send

Sends a message to the queue. The syntax is as follows:

ats send
  [**-handle** *handle_name*]
  -header *message_header_information*
  [**-message** *message_body*]
  -topic *topic*

Options

***-handle* handle_name**

Specifies an ATS session identifier in order to support multiple ATS sessions in parallel.

***-header* message_header_information**

Is a comma separated list of header fields, for example DXMVersion=1.0,DXMInstId=100,DXMOperation=...,...

You can define your own messages.

***-message* message_body**

Specifies the message text. There are no specific rules how to define the message text.

-*topic* topic

Specifies the topic name that gets mapped to the message service's queue name.

The **ats send** operation sends a message to a message service's queue. The operation returns the message ID on success.

Example

```
ats send -topic dxm.my.topic
              -header DXMVersion=1,DXMType=10
              -message "Hello world"
```

ats subscribe

Subscribes to the specified topic list. The syntax is as follows:

ats subscribe
[**-handle** *handle_name*]
-topic *topic_list*
-timeout *timeout*
[**-onemessage**]
[**-nodestroy**]

Options

-*handle* handle_name

Specifies an ATS session identifier in order to support multiple ATS sessions in parallel

-*topic* topic_list

Is a comma seperated list of topics the client wants to subscribe to, for example dxm.command.main,DXM.stateTracker,DXM.myserver.fileservice.

-*timeout* timeout

The operation stops after *timeout* seconds. A value of **0** implies that operation returns all currently available messages and then stops.

-onemessage

The operation stops after the first message has been returned.

-nodestroy

The operation does not destroy the messages returned. These messages must be destroyed by performing an **ats acknowledge** operation. The corresponding message Ids are returned.

The **ats subscribe** operation subscribes to the topic list specified in the option **-topic**.

The operation returns all messages available on success. If the **-nodestroy** option is

specified the returned messages contain also the base 64 encoded message IDs that must be used when performing the **ats acknowledge** operations to destroy the messages from the server. The messages are returned in aTcl list with the following format:

{*topic1 header_info_t1_1 message_t1_1* [*message_ID_t1_1*]}
{*topic1 header_info_t1_2 message_t1_2* [*message_ID_t1_2*]}
...
{*topic2 header_info_t2_1 message_t2_1* [*message_ID_t2_1*]}
{*topic2 header_info_t2_2 message_t2_2* [*message_ID_t2_2*]
...

Example

```
ats subscribe
      -topic dxm.event.pwd.changed
      -timeout 20
      -onemessage
      -nodestroy
```

The return value is as follows:

```
{dxm.event.pwd.changed
{JMSType=513,
DXMVersion=1.10,
DXMUserData=
<event name="dxm.event.PWD_CHANGED">
<source application="APETA" type="ADS" resource="ASW" />
<entry>
<identifier type="urn:oasis:names:tc:SPML:1:0#UserIDAndOrDomainName">
    <id>1002</id>
</identifier>
<operationalAttributes>
    <attr name="encryptedAttributes"> <value>password</value> </attr>
</operationalAttributes>
<attributes>
    <attr name="username"> <value>test</value> </attr>
    <attr name="password">
            <value>{E:00000001:cn=server-admin,
                        1.3.12.2.1107.1.3.102.4.13.17=DirXmetahub}
                         i7qivioOPuaAlicWdTZztGHewT8WSP65iix
                         VvpPT4txUz17IMAnavlb7deREKVYU
                         /ak2FjuIv815ZLltfFpz8Sm7nA4qpeZBW
```

```
                            2mj67dwWIqvBH2QRkeAOOd+RGRbw
                            0nzAlM93rNcpNnSrPqj8qiwr2RnY2WvX
                            j+izJmNiKoZsYk=
            </value>
    </attr>
    <attr name="passwordexpired"> <value>0</value> </attr>
</attributes>
</entry>
</event>,
JMSExpiration=86400,
JMSMessageID=_1067432986_-1_1,
JMSDeliveryMode=0,
JMSDestination=dxm.event.pwd.changed,
JMSPriority=1,
JMSRedelivered=0,
JMSTimestamp=1067433251}
QU1RIFFNX3NjaG1pZDAyIBS8nz8gABYF}
```

where

"dxm.event.pwd.changed" represents the topic,

"QU1RIFFNX3NjaG1pZDAyIBS8nz8gABYF" at the end represents the base 64 encoded message ID that must be used when performing the **ats acknowledge** operation

and the remaining information represents the message.

ats terminate

Terminates a connection to the message server. The syntax is as follows:

ats terminate
  [**-handle** *handle_name*]

The **ats terminate** operation terminates a connection to the message server. It releases all resources. The connection was previously initialized by an **ats initialize** operation.

Options

-handle **handle_name**
  Specifies an ATS session identifier in order to support multiple ATS sessions in parallel

Example

```
ats terminate
```

ats help

Returns help information about the **ats** object and its operations. The syntax is as follows:

ats help

Example

```
ats help
```

ats operations

Returns a list of operations that can be performed on the **ats** object. The syntax is as follows:

ats operations

The list of available operations is in alphabetical order except for **help** and **operations**, which are listed last.

Example

```
ats operations
```

The output of the sample command is as follows:

```
acknowledge initialize publish replay subscribe terminate help
operations
```

1.1.2. ldapargs (metacp)

Synopsis

ldapargs help [*operation* | **-verbose**]

ldapargs operations

ldapargs modify
[-**bindid** *bid*]
{-**default** |
[-**dereferencealias** {**ALWAYS** | **NEVER** | **SEARCHING** | **FINDING**}]}

[-**followreferral** {**TRUE** | **FALSE**}]
[-**sizelimit** {*limit* | **INFINITE**}]
[-**timelimit** {*limit* | **INFINITE**}]}

ldapargs show [**-bindid** *bid*] [**-pretty**]

Purpose

A **metacp** object that manages the service controls for LDAP binds. The **ldapargs** object operations can only be issued for established LDAP binds.

Arguments

operation

The name of the **ldapargs** operation for which to display help information.

Operations

ldapargs help

Returns help information about the **ldapargs** object and its operations. The syntax is as follows:

ldapargs help [*operation* | **-verbose**]

Options

-verbose

Displays information about the **ldapargs** object.

Used without an argument or option, the **ldapargs** help command returns brief information about each **ldapargs** operation. Use the *operation* argument to return a description of the options associated with the operation you specify. Alternatively, you can use the **-verbose** option to return a description of the **ldapargs** object itself.

Example

```
ldapargs help
```

The output of the sample command is as follows:

```
modify          Modifies the service controls.
show            Shows the service controls.
```

help	Displays help text for the 'ldapargs' object and its operations.
operations	Lists the operations that can be performed on the 'ldapargs' object.

ldapargs modify

Changes the service controls for LDAP binds. The syntax is as follows:

ldapargs modify
[-**bindid** *bid*]
{-**default** |
[-**dereferencealias** {**ALWAYS** | **NEVER** | **SEARCHING** | **FINDING**}]}
[-**followreferral** {**TRUE** | **FALSE**}]
[-**sizelimit** {*limit* | **INFINITE**}]
[-**timelimit** {*limit* | **INFINITE**}]}

Options

***-bindid* bid**

The name (*bid*) of the bind to be used. If this option is omitted, the default bind is used. (See the Bind Types and Bind IDs section in this chapter for details.)

-default

Sets the service controls and options to the following values:

Dereference Alias	ALWAYS
Follow Referral	TRUE
Size Limit	INFINITE
Time Limit	INFINITE

Do not use this option with any other options except with the option -bindid.

-dereferencealias {ALWAYS | NEVER | SEARCHING | FINDING}

Controls handling of aliases in directory operations. An alias is an entry that refers to another entry (the target). During directory operations, aliases can be replaced by the target entry (dereferenced). To control when this occurs, specify one of the following keywords:

- **ALWAYS**
- **NEVER**
- **SEARCHING** - Aliases are dereferenced during an **obj search** operation but are not dereferenced when locating the base object of the search.
- **FINDING** - Aliases are dereferenced when locating the base object of the search but are not dereferenced during the **obj search** operation.

The default value is **ALWAYS**.

-followreferral **{TRUE | FALSE}**

Controls whether referrals returned by an **obj search** operation are followed automatically. The default value is **TRUE**.

-sizelimit **{limit | *INFINITE*}**

Sets the maximum number of objects returned for **obj list** and **obj search** operations. Specify a non-negative integer or the keyword **INFINITE**. The default value is **INFINITE**.

-timelimit **{limit | *INFINITE*}**

Sets the maximum elapsed time in seconds for completion of an **obj list** or **obj search** operation. If an **obj list** or **obj search** operation does not complete by the specified limit, the operation returns an arbitrary selection of results accumulated before exceeding the time limit. Specify a non-negative integer or the keyword **INFINITE**. The default value is **INFINITE**.

The **ldapargs modify** operation changes one or more service control settings to be used in all subsequent operations of the specified LDAP bind. Specify one or more of the options that correspond to the service controls you want to set. Alternatively, you can specify the **-default** option to set the control services to their default values.

Service control settings for a specific bind ID are lost when an **obj unbind** operation is issued for this bind ID. A subsequent **obj bind** operation resets the service controls to the default values for the specified bind ID. The service controls are not reset to the default values if the default bind is used. (See the obj bind operation for details.)

Examples

The following sample command performs the following tasks for the LDAP bind corresponding to the binding id **hawkL3**:

- Limits the size of the results returned to 500 objects
- Limits the amount of time to 30 seconds in which an operation can complete

```
ldapargs modify -bindid hawkL3 -sizelimit 500 -timelimit 30
```

The following sample command sets the service controls for the LDAP bind corresponding to the bind ID **hawkL3** to the default values:

```
ldapargs modify -bindid haekL3 -default
```

ldapargs operations

Returns a list of operations that can be performed on the **ldapargs** object. The syntax is as follows:

ldapargs operations

The list of available operations is in alphabetical order except for **help** and **operations**, which are listed last.

Example

```
ldapargs operations
```

The output of the sample command is as follows:

```
modify show help operations
```

ldapargs show

Shows the service control settings currently in use for LDAP binds. The syntax is as follows:

ldapargs show [**-bindid** *bid*] [**-pretty**]

Options

***-bindid* bid**

The name (*bid*) of the bind to be used. If this option is omitted, the default bind is used. (See the Bind Types and Bind IDs section in this chapter for details.)

-pretty

Displays the results of the operation in tabular format.

By default, the results of the **ldapargs show** operation are returned as a Tcl list. Use the **-pretty** option to display the results in a tabular, more readable format.

Example

```
bind -prot LDAPv3
ldapargs show -pretty
```

The output of the sample command is as follows:

```
Dereference Alias - ALWAYS
Follow Referral - TRUE
Size Limit - INFINITE
Time Limit - INFINITE
```

1.1.3. meta (metacp)

Synopsis

[**meta**] **addentry**
  -entry *entry_handle*
  [**-updateattr** *attribute_list*]

[**meta**] **appendresult**
  -**source** *source_search_result_handle*
  -**target** *target_search_result_handle*

[**meta**] **createindexlist**
  [**-exactaction**]
  -conn *connection_handle*
  -handle *index_list_handle*
  [**-type** *distinguished_name_syntax*]

[**meta**] **encryptdata**
  -result *result_handle*
  -databindid *data_bind_id*
  -keybindid *key_bind_id*
  [**-oldserialnumber** *old_serial_number*]

[**meta**] **encryptvalue** *string*

[**meta**] **escapechar** *string*
  -characters *characters*

[**meta**] **findentry**
  -result *result_handle*
  -name *distinguished_name* or *attribute_value*
  -entry *entry_handle*
  [**-getattr** *flag*]
  [**-tag** *tag*]
  [**-numbermatches** *number_variable*]

[**meta**] **getchangelog**
  -name *distinguished_name*
  -changetype *operation_change_type*
  -changes *list of changes*
  -source *handle_to_pseudo_LDIF-file*
  -target *entry_handle*

[**meta**] **getentry**
  -source *source_handle*
  {**-target** *entry_handle* [**-tag** *tag*] [-**mark**]
  [**-name** *distinguished_name* [**-countmatches** *flag*]] |
  -reset}

[**meta**] **gethandle**
  -entry *entry_handle*
  -conn *connection_handle*

[**meta**] **getnumentries**
  -result *result_handle*

[**meta**] **getstatistic** *statistic_variable*

meta help [*operation* | **-verbose**]

[**meta**] **initialize**
  -file *trace_file*
  [**-tracelevel** *level*]
  [**-maxtracerecords** *recordnumber*]
  [**-maxentries** *number*]
  [**-mode** *initialization_mode*]
  [-**nostatistics**]

[**meta**] **modifydn**
  -entry *entry_handle*

[**meta**] **modifyentry**
  -newentry *entry_handle*
  [**-oldentry** *entry_handle*]
  [**-tag** *tag*]
  [**-allowrename** *rename_flag*]
  [**-updateinfo** *update_flag*]
  [**-updateattr** *attribute_list*]

[**meta**] **openconn**
  -type *directory_type*
  [**-file** *filename* **-mode** *file_open_mode*
  [**-format** *file_format*] [**encoding** *encoding*]]
  [**-attribute** *attribute_list* [**-processallattr**]]
  -attrconf *attr_conf_handle*
  -conn *connection_handle*
  [**-bindid** *binding_id*]
  [-**ldifhandle** *LDIF_file_handle*]

[**meta**] **openldif**
  -**file** *filename*
  -**ldifhandle** *LDIF_file_handle*

meta operations

[**meta**] **readattrconf**
  -file *filename*
  -attrconf *attr_conf_handle*
  [**-encoding** *encoding*]

[**meta**] **releasehandle** *handle*

[**meta**] **removeentries**
  -handle *handle*
  [**-tag** *tag*]

[**meta**] **sortindexlist -handle** *index_list_handle*
  [**-keepnames** *index_list_access_flag*]

[**meta**] **sortresult**
  -result *result_handle*
  -order *sort_order*
  [**-key** *sort_key*]

[**meta**] **supinfo**
  -rdn *attribute_type*
  {**-attribute** *attribute_value* | **-remove**}
  [**-bindid** *binding_id*]

[**meta**] **terminate**

[**meta**] **unescapechar** *string*

[**meta**] **writeindexlist**
  -handle *index_list_handle*
  -name *distinguished_name*
  -entry *entry_handle*

[**meta**] **writerecord**
  -entry *entry_handle*

[**meta**] **writetext**
  -conn *connection_handle*
  -text *string*
  [**-nonewline** *flag*]

[**meta**] **writetrace**
  -text *string*
  [**-handle** *handle*]

Purpose

A **metacp** object that performs directory synchronization operations.

Arguments

handle

A meta handle to be released by the **releasehandle** operation. A meta handle is a handle created by a **meta** operation that is associated with **metacp** information. The following table describes the types of meta handles.

Handle Type	Associated Information	Created by	Dependencies
attribute configuration file handle	Attribute configuration file	**meta readattrconf**	none
connection handle	connected directory (file, DAP or LDAP)	**meta openconn**	attribute configuration file handle
result handle	search results list	**obj search** **meta appendresult**	connection handle
entry handle	directory entry directory entry entry template	**meta findentry** **meta getentry** **meta gethandle**	result handle result or connection handle connection handle
index list handle	index list	**meta createindexlist**	connection handle
LDIF file handle	LDIF-CHANGE output	**meta openconn**	none

operation

The name of the **meta** operation for which to display help information.

string

The string to be processed (for example, in an **escapechar** operation).

Operations

meta addentry

Creates a new entry in the Identity store (a connected directory of type **LDAP**). The syntax is as follows:

[**meta**] **addentry**
-entry *entry_handle*
[**-updateattr** *attribute_list*]

Options

-*entry* entry_handle

A required option that specifies the handle to the entry to be added. Specify the name of an entry handle created by a **meta gethandle** operation called with a **-conn** option that specifies a connection handle to a connected directory of type **LDAP**.

-*updateattr* attribute_list

An optional parameter that specifies an additional attribute list. All these attributes are created together with the information associated with entry_handle. The attribute list has to be specified in native LDAP syntax, e. g.

```
[meta] addentry entry new_entry updateattr dxmState=NEW
```

Using the -updateattr switch allows to handle additional attributes which normally should not be synchronized under any circumstances. (Otherwise these attributes must have been selected in the associated entry_handle and therefore are always subject to synchronization. Thats not desirable in any cases.)

The **addentry** operation adds the entry associated with *entry_handle* into the Identity store (a connected directory of type **LDAP**). The Tcl variable array associated with *entry_handle* must contain a value for the Directory-Distinguished-Name (DDN) attribute. The **add entry** operation uses this value for the entry name when it creates the entry. The attributes that the operation creates for the entry correspond to the set of attributes present in the Tcl variable array associated with *entry_handle*.

The **addentry** operation creates the necessary superior entries when creating the entry provided that the required attribute information has been specified with a **supinfo** operation.

If the underlying connection handle (derived from *entry_handle*) is an LDAP directory with the parameter **-ldifhandle** set, the **addentry** is not applied to the LDAP directory; instead, an LDIF-CHANGE operation is created in the LDIF-CHANGE file associated with **-ldifhandle** parameter.

metacp automatically propagates a JMS message (containing the SPML add request) if at the beginning of a synchronization scenario **ats initialize** has been called with the options **-topicprefix**, **-type**, **-cluster**, and **-resource**. (See **ats initialize** for details.)

Return Values

Check the Tcl variable **errorCode** for error details. If this variable is not set or its value is empty, the entry was successfully created. Otherwise the error is logged in the trace file and the entry was not created.

0	**The object creation succeeded, or it failed with an error that did not prevent further processing of operations in the Tcl script logic.**
1	The object creation failed with a serious error; for example, "LDAP server not available". Examine your Tcl script logic to determine whether it makes sense to terminate the Tcl synchronization script.

Example

```
meta addentry -entry en
```

See Also

gethandle, initialize, openconn, openldif, supinfo, ats initialize

meta appendresult

Appends a new search result to an already existing search result. The syntax is as follows:

[**meta**] **appendresult**
-**source** *source_search_result_handle*
-**target** *target_search_result_handle*

Options

-*source* source_search_result_handle

A required option that specifies a search result (identified by its *source_search_result_handle* name) that should be appended to an already existing search result (identified by its target search result handle name). Specify the name of a search result handle created by a **obj search** operation called with a **-conn** option that specifies a connection handle to a connected directory of type LDAP.

-*target* target_search_result_handle

A required option that specifies a search result (identified by its *target_search_result_handle* name) that should be extended by another search result. Specify the name of a search result handle created by a **obj search** operation called with a **-conn** option that specifies a connection handle to a connected directory of type LDAP. If the target search result handle does not exist, then it will be created and the final search result is an exact copy of the source search result.

The **appendresult** operation appends a search result (identified by the *source_search_result_handle*) to an already existing target search result. If the target result does not exist as yet, the target search result handle will be created and consists of all the entries from the source search result.

The two search results can only be merged if the following conditions are satisfied:

- Both search results are based on the same connection identified by the **-openconn connection** option.
- If the search results are sorted, both of them are sorted using the same sort key.
- If the search results are sorted, both of them are sorted using the same sort order.
- If the option **-exactaction** was used in the **obj search** opration, then it must have been used either for both the search operations or for none of them.

Only if these conditions are satisfied, the final search result is consistent again.

If the target search result was already sorted (using **meta sortresult**), then the final search result will be sorted again with the same sort criterias.

- When using **appendresult** the user must guarantee that the same object does not exist several times in the final search result. Otherwise its not predictable, whether the operation **meta addentry**, **meta modifyentry** and **meta removeentries** are correctly called by a synchronization TCL script.
- If **meta getentry** has already been applied to the target search result before calling **meta appendresult** and therefore the current position in the search result is not the beginning of the search result, the current position is of no use any longer. Therefore the current position is set to the beginning of the search result; **meta getentry** will return the entries from the beginning of the search result again.

Return Values

None.

Example

```
meta appendresult -source new_result -target merged_result
```

See Also

obj search

meta createindexlist

Creates a handle to an index list. The syntax is as follows:

[**meta**] **createindexlist**
  [**-exactaction**]
  -conn *connection_handle*
  -handle *index_list_handle*
  [**-type** *distinguished_name_syntax*]

Options

-exactaction

Normally DNs are defined as CaseIgnore strings which implies that leading and trailing spaces are ignored and several spaces are reduced to one space during comparisions. There are no problems when modifying such an object and a reduced number of spaces is passed in the DN; the objects are still considered to be the same and the MODIFY operation is performed successfully. There are directory systems, that want to have the exact number of spaces when calling the MODIFY operation. Therefore, multiple spaces must not be dropped.

Using the option -exactaction guarantees that multiple spaces in the DNs are not

dropped. The DNs are provided to the user the same way as they are retrieved from the directory server.

-*conn* connection_handle

A required option that specifies a handle to a connected directory. Specify the name of a connection handle created by a **meta openconn** operation called with the **-type** option **FILE**.

-*handle* index_list_handle

A required option that specifies the name to be assigned to the index list handle returned by the operation.

-*type* distinguished_name_syntax

Specifies the distinguished name syntax for entries in the index list. Specify the keyword **LDAP** to create an index list for entry distinguished names in LDAP format.

The **createindexlist** operation initializes an empty index list template, creates a handle to it, and assigns it the name specified with the **-handle** option. The index list can then be used to sort the entries in the connected directory, rather than sorting the connected directory (which is a file) itself. The process for using an index list to sort directory entries is as follows:

1. Create the index list with **meta createindexlist**
2. For each entry in the connected directory:
    - Use the **meta getentry** operation to read the entry. The **meta getentry** operation returns an entry handle to the entry. Obtain the entry's distinguished name either from the entry itself (if it is stored in the entry, it is available in a Tcl variable) or through a Tcl mapping procedure.
    - Use the **meta writeindexlist** operation to write the distinguished name of the entry and its entry handle returned by **meta getentry** into the index list.
3. Use the **meta sortindexlist** operation to sort the entries in the index list according to distinguished name or any other attribute, in alphabetical (ascending) order.

You can then continue the directory synchronization process using the sorted index list rather than using the connected directory. For example, you can perform **meta getentry** operations on the sorted index list by specifying its index handle rather than specifying the connection handle to the connected directory.

The **createindexlist** operation stores the distinguished name syntax keyword specified in the **-type** option in the index list handle. The **meta sortindexlist** operation uses this information to determine how to process the distinguished names in the index list. If the **-type** option is not specified, the operation uses DAP distinguished name processing.

Return Values

None.

Example

```
meta createindexlist -conn file_ch -handle ih
```

See Also

getentry, sortindexlist, writeindexlist

meta encryptdata

Performs data encryption for a search result handle based on the private keys and certificates stored for the server (IdS-C) admistrator object cn=server_admin,dxmc=DirXmetahub. There is a support of both an initial mode (encryption of cleartext data with the public key having the highest serial number) and a migration mode which includes

- encryption of cleartext data with the public key having the highest serial number
- data previously encrypted with a public key of older serial number is decrypted and encrypted again with the latest public key.

The command syntax is as follows:

[**meta**] **encryptdata**
  -result *result_handle*
  -databindid *data_bind_id*
  -keybindid *key_bind_id*
  [**-oldserialnumber** *old_serial_number*]

Options

-result **result_handle**

Specifies the ldap search result handle which includes relevant objects and attributes relevant for data encryption. The *result_handle* must be a handle which respresents an ldap search result,

-databindid **data_bind_id**

Specifies the bind identifier required for performing the attribute modifications in the Directory. This bind identifier must be the same as the bind identifier related to *result_handle*, and a bind with protocol LDAPv3 must have been performed with bindid *data_bind_id*.

-keybindid **key_bind_id**

Specifies the bind identifier required for reading the keys and certificates from the Server (IDS-C) administrator object. A suitable bind with protocol LDAPv3 must have been performed with bindid *key_bind_id*.

***-oldserialnumber* old_serial_number**

This parameter is optional, the default is NONE. Either omit this option or specify NONE if you intend to perform initial encryption. Specify a hexadecimal value (e.g. 6E) or the keyword PREVIOUS, if you intend to perform data encryption in migration mode. In this case, the environment variable *_DXC_OLD_PIN must contain the PIN suitable to the private key with _old_serial_number*.

Return Values

None.

Example

An encryptdata operation for initial encryption:

```
meta encryptdata result resultHandle \
      -databindid dataBindId \
      -keybindid keyBindId
```

An encryptdata operation for migration from old serial number 6E to current serial number:

```
meta encryptdata result resultHandle \
      -databindid dataBindId \
      -keybindid keyBindId \
      -oldserialnumber 6E
```

See Also

obj bind, obj search, initialize, encryptvalue

meta encryptvalue

Performs data encryption for a string based on the certificates stored for the Server (IDS-C) administrator object cn=server_admin,dxmc=DirXmetahub. It uses the public key (certificate) with the highest serial number.

That function is only available when metacp runs in encryption mode.

The syntax is as follows:

[**meta**] **encryptvalue** *string*

Options

none

Return Values

The encrypted string.

In case of errors, an error message is returned and the errorCode variable is set.

Example

The encryptvalue operation

```
meta encryptvalue dirx
```

returns the encrypted value base on the currently available public key.

See Also

encryptdata

meta escapechar

Escapes one or more characters in a string. The syntax is as follows:

[**meta**] **escapechar** *string*
  -characters *characters*

Options

-*characters* characters

Escapes the characters specified in *characters*.

The **escapechar** operation escapes one or more characters by prefixing them with the backslash character (\) and returns the escaped string as a result. The backslash character itself is escaped as well.

Use the **-characters** option to escape only those characters specified in *characters*. To avoid conflicts with the Tcl shell escape mechanism, enclose the string that contains the characters to be escaped in curly braces (\{ }). If curly braces are to be a part of the string, they must be escaped.

Return Values

The escaped string.

Example

The **escapechar** operation

```
meta escapechar f{1}=2 -characters {\{\}=}
```

returns the string f\{1\}\=2

See Also

unescapechar

meta findentry

Retrieves an entry from the sorted results of an **obj search** operation. The syntax is as follows:

[**meta**] **findentry**
  -result *result_handle*
  -name *distinguished_name* or *attribute_value*
  -entry *entry_handle*
  [**-getattr** *flag*]
  [**-tag** *tag*]
  [**-numbermatches** *number_variable*]

Options

-*result* result_handle

A required option that specifies a handle to a directory search results list. Specify the name of a result handle created by an **obj search** operation that has been sorted by a **meta sortresult** operation.

-*name* distinguished_name or attribute_value

A required option that specifies the distinguished name of the entry to be retrieved if the result list has been sorted according to DDN. Specify the complete distinguished name in the format:

distinguished_name

For example:

cn=mueller,ou=board,o=pqr,c=de

or the attribute type and value to be retrieved if the result has been sorted according to a sort criteria different than DDN. Specify the value in the format

attribute_type=attribute_value

For example:

employeeNumber=98765

-*entry* entry_handle

A required option that specifies a name to associate with the entry handle created by the operation (if it finds the entry in the search results list).

-*getattr* flag

Controls whether the entry handle created by the operation is associated with a Tcl variable array containing the entry's attributes. Specify one of the following keywords:

- **TRUE**-associate the entry handle with a Tcl variable array
- **FALSE**-do not associate the entry handle with a Tcl variable array (default)

-*tag* tag

Specifies a marking action to be performed on the found entry in the search results list. Specify the keyword **MARK** in the *tag* argument to this option in order to mark the entry.

-*numbermatches* number_variable

If an LDAP result is sorted according to a sort criteria different than DDN, then multiple matches could be present.

Therefore, if **-numbermatches** with a TCL variable name *number_variable* is specified, then **findentry** returns the first matched entry together with the number of matched entry that is stored in a TCL variable with the given name. In that case, **findentry** internally sets the search result pointer to the matched entry. Using subsequent calls of **getentry** then allow to retrieve all the matching entries one after the other.

If **-numbermatches** is not specified, then the matching entry that is returned is unpredictable (as described in the current release of the document).

The **findentry** operation attempts to match the entry specified in the **-name** option to an entry within the directory search results specified in the **-result** option. On success, the operation creates an entry handle associated with the matched entry and assigns it the name specified in the **-entry** option. By default, the entry handle created by **meta findentry** is not associated with a Tcl variable array. Use the **-getattr** option to associate the handle with a Tcl variable array.

If the result list has been sorted according to any attribute type, then that attribute must be single-valued for all the entries of the search result. Entries that don't hold that specific attribute type will be listed at the end of the sorted list (if sorting has been done in ascending order) or at the beginning of the sorted list (if sorting has been done in descending order). If several such entries exist, their position at the beginning or end of the sorted list is undefined. As a consequence, it is unpredictable which entry will be returned by a "meta findentry" operation that uses such an ambiguous value. Therefore, it is strongly recommended to use only attributes as sort_key (in meta sortresult) that are unique, like employee number.

Return Values

0 The entry was found

1 The entry was not found

Example

```
meta findentry -name cn=zahn,ou=sales,o=ibis,c=us \
        -result ldap_res -entry eh
```

See Also

getentry, modifyentry, removeentries, obj search

meta getchangelog

Processes an LDIF change log entry as provided, for example, by iPlanet or Oracle Internet Directory (OID). The syntax is as follows:

[**meta**] **getchangelog**
  -name *distinguished_name*
  -changetype *operation_change_type*
  -changes *list of changes*
  -source *handle_to_pseudo_LDIF-file*
  -target *entry_handle*

Options

-*name* distinguished_name
  Specifies the target DN of the object that is listed in the change log entry.

-*changetype* operation_change_type
  Specifies the kind of LDIF change operation: add, delete, modify, moddn, modrdn

-*changes* list_of_changes
  Specifies the list of changes that have been applied to the given DN

-*source* handle_to_pseudo_LDIF-file
  Specifies the name of a handle to a pseudo LDIF file

-*target* entry_handle
  Specifies the name of an entry handle that is created by the operation in case of success

Both iPlanet and OID store all the changes of directory entries and attributes in the directory itself as directory entries (changelog entries). The getchangelog operation processes a change log entry that has been read either from an iPlanet or OID directory

using an **obj search** operation. The relevant attributes (targetDN, changetype, changes) of such a changelog entry are passed to getchangelog; as the format of these parameters are exactly the same as if an LDIF change file had been read, getchangelog will internally concatenate the values of these parameters as a byte stream in memory and then use its internal LDIF file parser. (Therefore, a handle to a pseudo LDIF file must be passed as **-source** handle.)

On success, getchangelog creates a handle to the entry and associates it with the name in the **target** option. The entry's DN and its attributes become accessible as an array of Tcl variables. For details, see the description of **getentry**.

Return Values

0 The entry was successfully processed

1 The entry could not be processed

Example

In this example, the change log entry has been read by **obj search** and its data is available in the Tcl array named **chlog**:

```
meta getchangelog
      -name           {[lindex $chlog(targetdn) 0]}     \
      -changetype {[lindex $rh_x500(changetype) 0]}   \
      -changes       {[lindex $rh_x500(changes) 0]}      \
      -source   file_ch     \
      -target    rh"]
```

See Also

getentry, Chapter: Processing of ChangeLogEntries

meta getentry

Reads the next entry from a connected directory, a search results list, or an index list, reads a specific entry from an index list, or resets a search results list or index list. The syntax is as follows:

[**meta**] **getentry**
  -source *source_handle*
  {**-target** *entry_handle* [**-tag** *tag*] [-**mark**]
  [**-name** *distinguished_name* [**-countmatches** *flag*]] |
  -reset}

Options

-*source* source_handle

A required option that specifies a handle to a connected directory, a search results list, or an index list.

- To operate on a connected directory, specify the name of a connection handle created by a **meta openconn** operation called with the **-type** option **FILE** and the **-mode** option **READ**.
- To operate on a search results list, specify the name of a result handle created by an **obj search** operation.
- To operate on an index list, specify the name of an index list handle created by a **meta createindexlist** operation. The connection handle inherited by the specified index list handle must specify a connected directory that has been opened with the **-type** option **FILE** and the **-mode** option **READ**.

-*target* entry_handle

Specifies a name to assign to the entry handle created by the operation on success.

-*tag* tag

Specifies whether the next marked or unmarked entry is to be read, if the operation is reading entries from a search results list or an index list. Specify one of the following keywords:

- **MARKED**-read next marked entry
- **NOTMARKED**-read next unmarked entry (default)

-mark

Specifies that an entry that is returned from a search result or an index list should be marked or not.

If -**mark** is not given, the entry will not be marked.

The -**mark** option is ignored, if the source handle does not represent a search result or an index list.

-*name* distinguished_name

Specifies the distinguished name of the entry to be read, if the operation is reading entries from an index list. Specify the complete distinguished name in the format:

distinguished_name

For example:

```
cn=mueller,ou=board,o=pqr,c=de
```

***-countmatches* flag**

Specifies whether a status code or the number of entries that match the distinguished name specified in the **-name** option is to be returned, if the operation is reading entries from an index list. Specify one of the following keywords:

- **TRUE**-return the number of matched entries found in the index list
- **FALSE**-return **0** on success or **1** on end of index list (default)

-reset

Specifies that the list is to be reset to its beginning, if the operation is reading entries from a search results list or an index list.

Use the **getentry** operation to:

- Read entries sequentially from a connected directory (for example, an import file on disk) or a search results list
- Read entries sequentially or randomly by distinguished name from an index list
- Reset an index list or a search results list to the beginning of the list

Specify the **-reset** option to reset an index list or a search results list to the beginning of the list.

Specify the **-target** option to read an entry from a connected directory, a search results list, or an index list into the **metacp** workspace. On success, the operation creates a handle to the entry and associates it with the name specified in the **-target** option. When the **getentry** operation reads an entry into the **metacp** workspace, the entry's attributes to be synchronized become accessible as an array of Tcl variables. Each Tcl variable in the array represents a specific attribute to be synchronized; the variable is a Tcl list of the attribute's values. The Tcl variable array corresponds to the set of attributes to be synchronized (SSA); the SSA is defined:

- In the connection handle specified in the **-source** option, when operating on a connected directory
- In the connection handle referred to by the result handle specified in the **-source** option, when operating on a search results list
- In the connection handle referred to by the index list handle specified in the **-source** option, when operating on an index list

The Tcl variable array created by the **getentry** operation also contains two special attributes:

- A Directory-Distinguished-Name (**DDN**) attribute. A value exists for this attribute when operating on a search results list, on an index list, or on a connected directory of type **FILE** that supplies the entry distinguished name as a specific attribute.
- An LDIF "changetype" (**_ldif_opcode**) attribute. A value exists for this attribute when operating on a connected directory opened with the **-format** option **LDIF-CHANGE** and indicates that the entry specifies a type of directory modification. For valid handles with format **LDIF-CHANGE,** the value is one of **ADD**, **DELETE**, **MODIFY** or **MODIFY-DN**. If **metacp** fails to recognize the "changetype" attribute value when it reads an LDIF change entry, then the value will be an empty string. It is the responsibility of the user (or the **metacp** script author) to take the appropriate action (or error handling) depending on the attribute value. See the "LDIF Change Format" section in Chapter 3 (Directory Data File Formats) for more information about LDIF change file format and

the types of directory modifications an entry in an LDIF change file can represent.

Use the Tcl notation ***$*entry_handle*(*abbreviation*)*** to reference an element of the Tcl variable array, where *entry_handle* is the name of the handle and *abbreviation* is the attribute abbreviation for the attribute defined in the Abbreviation (Abbr) field of the attribute configuration file. For example, suppose that:

- The set of attributes to be synchronized are the attribute types Given Name, Surname, and Telephone Number
- The attribute abbreviations defined in the attribute configuration file are **givenName**, **sn**, and **telephoneNumber**
- The attribute abbreviation for Directory-Distinguished-Name defined in the attribute configuration file is **DDN**
- The attribute abbreviation defined in the attribute configuration file for the attribute that holds an entry operation code is **LCHNGTYPE**
- The name supplied for the entry handle created by the **getentry** operation is **eh**

Then the indexes to the values for these attributes in the Tcl array are

$eh(DDN) - to access the attribute value(s) for Directory-Distinguished-Name

$eh(LCHNGTYPE) - to access the attribute value for the entry operation code

$eh(givenName) - to access the attribute value(s) for Given Name

$eh(sn) - to access the attribute value(s) for Surname

$eh(telephoneNumber) - to access the attribute value(s) for Telephone Number

An attribute with a single value is represented as a Tcl list with one element. A multi-valued attribute is represented as a Tcl list of multiple elements, one for each value. Use Tcl statements for accessing list variables to manipulate the values in the Tcl variable array; for example, **lindex**.

For the "add" "delete" and "modify DN" LDIF change operations, each variable in the Tcl array contains a list of attribute values. For "modify" operations, each variable in the Tcl array contains a list of attribute changes. Each change is of the form:

{ *changetype* [*values*] }

where *changetype* is

ADD-VALUE

DELETE-VALUE

REPLACE

to indicate the three different types of attribute value modifications. The Tcl list:

```
{ { ADD-VALUE 1111 2222 } { DELETE-VALUE 3333 } }
```

represents the addition of two attribute values and the removal or one attribute value for a specific attribute. If the variable **eh(telephoneNumber)** has this list value, the changes apply to the attribute **telephoneNumber**. The first element of **eh(telephoneNumber)** is:

```
{ ADD-VALUE 1111 2222} # Three elements. The first is the
# changetype followed by two attribute
# values to be added.
{ DELETE-VALUE 3333 } # Two elements. The first is the changetype
# and the second is the attribute value to
# be deleted
```

The "add" and "replace" Tcl lists must contain one or more attribute values as elements that follow the *changetype* element. A "delete" Tcl list with one element (the *changetype*) indicates that the attribute itself is to be removed.

The Tcl variable array for a "modify DN" LDIF operation contains three additional attributes:

- A "new RDN" attribute-Supplies the new relative distinguished name for the entry
- A "new superior" attribute-Supplies the new superior distinguished name of the entry
- A "delete old RDN" attribute-Supplies the boolean value that specifies whether or not to delete the old relative distinguished name

The attribute abbreviations for these attributes are specified in the attribute configuration file.

The elements of the Tcl variable array remain accessible as Tcl variables until they are unset with the Tcl **unset** command or until the handle to the entry is released with **meta releasehandle**.

If the **-target** option specifies a result handle to a search results list, use the **MARKED** keyword in the **-tag** option to read the next entry from the Identity store (a connected directory of type **LDAP**) that the **meta modifyentry** or **meta findentry** operations have marked as "processed" in the results list. Use the **NOTMARKED** keyword to read the next entry that has not been marked by **meta modifyentry** or **meta findentry**.

If the **-target** option specifies an index list handle to an index list, and a **meta sortindexlist** operation with the **-keepnames** option has been performed on the index list specified in the **-source** option, you can use the **-name** option to read an entry with a specific distinguished name from the list. If there is more than one entry in the index list that matches the specified distinguished name, the entry handle returned by the operation represents the first matched entry in the index list. To retrieve the next matching entry, use the **meta getentry** operation without any options. Use the **mark** option to mark the entry in the index list before **meta getentry** is going to return that entry. Use the **tag** option in order to decide which entries should be returned from the index list: use the **MARKED**

keyword, if an already marked entry should be returned once again; use the **NOTMARKED** keyword, if the next unmarked entry has to be returned.

Use the **-countmatches** option in conjunction with the **-name** option to select whether the **meta getentry** operation returns the number of entries that match the specified distinguished name, or whether it returns the status codes described in **Return Values**. The setting for the **-countnames** option affects how the operation's return values are to be interpreted. For example, if the operation finds one matching entry, and **-countnames** is set to **FALSE** (or not specified at all), the operation returns **0**, which represents success (and not the number of matching entries found). If the operation finds one matching entry, and **-countnames** is set to **TRUE**, the operation returns **1**, which represents the number of matching entries found (and not "end of list".)

The type of handle specified in the **-source** option affects the handle dependencies for the handle created by **getentry**:

- If the **-source** option specifies a connection handle, the entry handle depends on this connection handle.
- If the **-source** option specifies a result handle, the entry handle has no dependencies.
- If the **-source** option specifies an index list handle, the entry handle depends on the connection handle to which the index list handle refers.

Note: If the **meta getentry** operation retrieves an entry from an LDIF change file that does not contain an LDIF operation code (**ADD**, **DELETE**, **MODIFY**, or **MODIFY-DN**), it creates an empty **_ldif_opcode** Tcl variable when it creates the Tcl variable array.

Return Values

0	The entry was read or the search results list was reset to the beginning of the list
1	No more entries to read (end of file, end of search results list, or end of index list)
any non-negative number	The number of matched distinguish names, if the countnames option is set to TRUE and an index list is used

Example

```
meta getentry -source LDAP_rh -target eh
```

See Also

addentry, createindexlist, modifydn, modifyentry, obj search, openconn, removeentries, writerecord

Attribute Configuration File Format (Chapter 2)

meta gethandle

Creates a handle to an empty entry. The syntax is as follows:

[**meta**] **gethandle**
  -entry *entry_handle*
  -conn *connection_handle*

Options

-*entry* entry_handle

A required option that specifies a name to be assigned to the entry handle created by the operation.

-*conn* connection_handle

A required option that specifies a handle to a connected directory. Specify the name of a connection handle created by a **meta openconn** operation.

The **gethandle** operation creates an entry handle associated with a "template" directory entry and assigns it the handle name specified in the **-entry** option. The "template" entry is an array of Tcl variables initialized as empty lists in the same format as that created by the **getentry** operation. Subsequent Tcl statements or **meta** operations (issued in the synchronization profile) read values into the array. The elements of the Tcl variable array remain accessible as Tcl variables until they are unset with the Tcl **unset** command or until the handle to the entry is released with **meta releasehandle**.

Return Values

None.

Example

meta gethandle -entry eh -conn ch

See Also

addentry, getentry, modifydn, modifyentry, openconn, removeentries, writerecord

Attribute Configuration File Format (Chapter 2)

Section **Synchronization Profile** (*DirX Identity Connectivity Administration Guide*)

meta getnumentries

Returns the number of entries in a search result. The syntax is as follows:

[**meta**] **getnumentries**
  -result *result_handle*

Options

***-result* result_handle**

A required option that specifies a handle to a search results list. Specify the name of a result handle returned by an **obj search** operation.

The **getnumentries** operation returns the number of entries contained in the results of a search operation.

Return Values

None.

Example

```
meta getnumentries -result LDAP_rh
```

See Also

obj search

meta getstatistic

Returns the statistic information as a (global) TCL array, if the statistic was not turned off in meta initialize. The syntax is as follows:

[**meta**] **getstatistic** *statistic_variable*

Options

none

Return Values

Fills the given TCL array with the following components:

- **DITEntries** - Denotes the complete number of entries that were found in the DIT
- **FileEntries** - Denotes the complete number of entries that were read from file (in case of an Import operation) or that were written to file (in case of an Export operation)
- **AddedOk** - Denotes the complete number of entries that were added successfully to the DIT.
- **ModifiedOk** - Denotes the complete number of entries that were modified successfully in the DIT.
- **ModdnOk** - Denotes the complete number of entries that were renamed successfully in the DIT.

- **DeletedOk** - Denotes the complete number of entries that were deleted successfully from the DIT.
- **UpToDate** - Denotes the complete number of entries that were already up to date.
- **AddedFailed** - Denotes the complete number of entries that could not successfully be added to the DIT.
- **ModifiedFailed** - Denotes the complete number of entries that could not successfully be modified in the DIT.
- **ModdnFailed** - Denotes the complete number of entries that could not successfully be renamed in the DIT.
- **DeletedFailed** - Denotes the complete number of entries that could not successfully be deleted from the DIT.
- **AddedIgnored** - Denotes the complete number of entries that normally should be added to the DIT, but that actually were not added as TRIAL mode is switched on.
- **ModifiedIgnored** - Denotes the complete number of entries that normally should be modified in the DIT, but that actually were not modified as TRIAL mode is switched on.
- **ModdnIgnored** - Denotes the complete number of entries that normally should be renamed in the DIT, but that actually were not renamed as TRIAL mode is switched on.
- **DeletedIgnored** - Denotes the complete number of entries that normally should be deleted from the DIT, but that actually were not deleted as TRIAL mode is switched on.
- **AddAttrIgnored** - Denotes the complete number of attribute creations that normally should be done, but that actually were not done as TRIAL mode is switched on.
- **DeleteAttrIgnored** - Denotes the complete number of attribute deletions that normally should be done, but that actually were not done as TRIAL mode is switched on.

Example

The function

```
meta getstatistic statInfo
```

sets the following TCL array:

```
statInfo(Processed)
statInfo(AddedOk)
statInfo(ModifiedOk)
statInfo(DeletedOk)
statInfo(AddedFailed)
statInfo(ModifiedFailed)
statInfo(DeletedFailed)
statInfo(AddedIgnored)
statInfo(ModifiedIgnored)
```

```
statInfo(DeletedIgnored)
```

See Also

Initialize

meta help

Returns help information about the **meta** object and its operations. The syntax is as follows:

meta help [*operation* | **-verbose**]

Options

-verbose

Displays information about the **meta** object.

Used without an argument or option, the **help** command displays a list of the **meta** operations. The operations are listed in alphabetical order, except for **help** and **operations**, which are listed last. Use the *operation* argument to return a brief description of an operation and its associated options. Alternatively, you can use the **-verbose** option to return a description of the **meta** object itself.

Return Values

None (The requested information is displayed).

Example

```
meta help
```

The output of the sample command is as follows:

```
The following operations are available:
addentry createindexlist encryptdata encryptvalue
escapechar findentry getchangelog getentry
gethandle getnumentries getstatistic initialize
modifyentry modifydn openconn openldif
readattrconf releasehandle removeentries sortindexlist
sortresult supinfo terminate unescapechar
writeindexlist writerecord writetext writetrace
help operations
meta help readattrconf
```

```
meta help readattrconf
```

The output of the sample command is as follows:

```
readattrconf - Reads an attribute configuration file and
               creates an attribute configuration file handle.

-attrconf -    Specifies the name of the handle to be created.

-file -        Specifies the name of the attribute
               configuration file.
```

meta initialize

Initializes the **metacp** workspace and opens a trace file for writing. The syntax is as follows:

[**meta**] **initialize**
  -file *trace_file*
  [**-tracelevel** *level*]
  [**-maxtracerecords** *recordnumber*]
  [**-maxentries** *number*]
  [**-mode** *initialization_mode*]
  [-**nostatistics**]

Options

-file **trace_file**

The name of the file to which trace information is to be written. The operation creates the file relative to the current working directory if it is a relative pathname; otherwise, the full pathname is used.

-tracelevel **level**

The level of tracing to be performed for operations invoked after **meta initialize** is called. Specify one of the following values in *level*:

- **1**-To perform error tracing (default)
- **2**-To perform full tracing
- **3**-To perform full tracing excluding the tracing of entries that are up-to-date

-maxtracerecords **recordnumber**

The maximum number of records in the tracefile.

-maxentries **number**

The maximum number of entries in a search results list to be processed by a **meta**

writetrace operation.

***-mode* initialization_mode**

The mode in which **metacp** is to run when an import procedure from a connected directory to the Identity store (a connected directory of type **LDAP**). Specify one of the following values in *initialization_mode*:

- **REAL**-Updates the Identity store with the imported data (default)
- **TRIAL**-Bypasses actual update of the Identity store, and logs the update operations that would otherwise be performed

-nostatistics

If set, the statistic information that metacp internally calculates while performing the synchronization tasks is suppressed.

For more details about the default statistic information please refer to meta terminate.

Depending on the complex logic of the synchronization scripts, there are sometimes good reasons to suppress the statistics, because it may be confusing, e.g.

- if a file is (for any purposes) read twice or
- several files are processed in parallel or
- searches in source and target directories are performed simultaneously or
- searches for the same objects need to be repeated because additional attribute information is needed or
- searches with different filter conditions overlap in the objects found in the DIT

The confusion in the first two cases is the following:
The "number of file entries" is misleading because it is the sum of all records ever read from any file.

The confusion in the remaining cases is the following:
The "number of X.500 entries" is misleading, because the number listed here is the count of entries ever read from any directory. That number may not be the exact number that is currently available in the DIT. Furthermore, if more than one directory is handled, that number has no meaning, because one does not know how many entries have been found in one directory and how many have been found in the other.

Therefore the user is advised to integrate his own statistic counters in the TCL logic itself, if he builds complex synchronization scripts because he is the only one that knows the information that is worth to be printed at the end of the synchronization.

The **initialize** operation initializes **metacp** data structures and creates a **metacp** trace file to be used to store information about the directory synchronization process. A synchronization profile must invoke the **initialize** operation before it invokes any **metacp** operations that operate on meta handles.

When they are invoked in a synchronization profile, the **meta addentry**, **meta modifydn**, **meta modifyentry** and **meta removeentries** operations automatically write information

about their operation into the trace file specified with the **-file** option. The **writetrace** operation can also be used to write explicit trace information into the trace file.

Use the **-tracelevel** option to control the level of trace information that the **meta addentry**, **meta modifydn**, **meta modifyentry** and **meta removeentries** operations write to the trace file. A trace level of 1 directs the operations to perform error tracing. An error trace on an operation traces:

- The original entry in the source
- The attempted directory operation, prefixed with the # character
- The error message, prefixed with the # character

With error tracing, if an error occurs on an entry being processed, the operation writes the entry as it appears in the original source file into the trace file together with the associated directory operation and error message, prefixed with the # character.

Here is a sample error trace record:

```
dn: cn=Filler, ou=Development, o=PQR
objectclass: person
objectclass: organizationalPerson
telephoneNumber: +44 1922 222222
facsimileTelephoneNumber: +44 1922 222223
userPassword: two
sn: Filler
description: Craftsman
postalCode: WS1 9YY
streetAddress: 17 Kelly Road
st: Walsall
# MODIFY operation: Mon Jan 4 10:19:49 1999 [Bind-ID: (default)]
# "cn=Filler,ou=Development,o=PQR " "-removeattr" "sn=Filler"
# ERROR: Object class violation.
```

A trace level of 2 directs the operations perform full tracing. A full trace on an operation traces:

- The original entry in the source
- The entry handle that was input to the operation, for a **meta modifyentry** operation on an entry in a connected directory of type **LDAP**
- The entry handle for the matched entry, for a **meta modifyentry** operation on an entry in a connected directory of type **LDAP**
- The attempted directory operation
- No message, if the operation was successful; otherwise an informational message or an error message

With full tracing, the operations write both error trace information and successful operation information to the trace file. Entries that are successfully processed are written to the file and delimited with # characters.

Here is an excerpt from a full trace operation:

```
#dn: cn=Digger T., ou=Development, o=PQR
#cn: Digger T.
#telephoneNumber: +44 1902 111111
#postalAddress: FLAT 86B$24 Dougan
Street$Fordhouses$Wolverhampton$West Midlands$WV1 9ZZ
#facsimileTelephoneNumber: +44 1902 111112
#userPassword: PASS1
#sn: Digger
#description: Salvage Clerk
#postalCode: WV1 9ZZ
#street: 24 Dougan Street
#st: Wolverhampton
#objectClass: inetOrgPerson
#objectClass: organizationalPerson
#objectClass: person
#objectClass: top
#createTimestamp: 200003081209472
# ADD operation: Wed Jul 12 12:34:21 2000 [Bind-ID: (default)]
# "cn=Digger T., ou=Development, o=PQR" "-attribute"
"objectClass=inetOrgPerson;organizationalPerson;person;top"
"telephoneNumber=+44 1902 111111"
"postalAddress=FLAT 86B$24 Dougan
Street$Fordhouses$Wolverhampton$West
Midlands$WV1 9ZZ"
"facsimileTelephoneNumber=+44 1902 111112"
"userPassword=PASS1"
"sn=Digger"
"description=Salvage Clerk"
"postalCode=WV1 9ZZ"
"street=24 Dougan Street"
"st=Wolverhampton"
"cn=Digger T."
#dn: cn=Smith John, ou=Sales, o=PQR
#cn: Smith John
#sn: Smith
#description: Sales Manager
```

```
#telephoneNumber: +12 34 567 891
#telephoneNumber: +12 34 567 890
#userPassword: jps123
#labeledURI: http://www.sni.de
#mail: John.Smith@sales.pqr.de
#objectClass: inetOrgPerson
#objectClass: organizationalPerson
#objectClass: person
#objectClass: top
#createTimestamp: 20000308120946Z
# Mapped Entry:
# "cn=Smith John, ou=Sales, o=PQR"
# "objectClass=inetOrgPerson;organizationalPerson;person;top"
# "telephoneNumber=+12 34 567 891;+12 34 567 890"
# "userPassword=jps123"
# "sn=Smith"
# "description=Sales Manager"
# "mail=John.Smith@sales.pqr.de"
# "labeledURI=http://www.sni.de"
# "cn=Smith John"
# X500 Entry:
# "cn=Smith John,ou=Sales,o=PQR"
# "objectClass=person;organizationalPerson;inetOrgPerson;top"
# "cn=Smith John"
# "sn=Smith"
# "description=Sales Manager"
# "telephoneNumber=+12 34 567 890;+12 34 567 891"
# "userPassword=jps123"
# "labeledURI=http://www.sni.de"
# "mail=John.Smith@sales.pqr.de"
# "collectiveTelephoneNumber=+12 34 567 0"
# TRIAL operation: Wed Jun 28 12:41:35 2000 [Bind-ID: (default)]
# No update necessary !
```

The update operations that really result in some action in the DSA, are logged as "**ADD operation:**", "**DELETE operation:**", "**MODIFY operation:**" and "**MOD-DN operation:**".

In case a (potential) MODIFY operation needs to be sent to the server and metacp detects, that the object is already up to date, then such an operation is logged as "**TRIAL operation:**".

For readability, the previous example shows the attributes added to the entry on separate lines; in the trace file, all the attributes added to an entry appear on a single line.

A trace level of 3 directs the operation to perform full tracing except for entries that are already up-to-date. For example, the "add" operation shown in the previous example appears in the trace file, but the "modify" operation for which no update was necessary does not appear in the trace file.

If the **-tracelevel** option is not specified, the operations by default perform level 1 (error) tracing.

Use the **-maxtracerecords** option to limit the maximum number of records in a trace file. *recordnumber* specifies the maximum number of records. If the option is missing or the value is **0** only one trace file is written with an unlimited number of records.

If required several trace files are written each of them containing at least *recordnumber* trace records. The maximum number of records written to the trace files may differ slightly from the specified number because the maximum number is checked after the complete information about a synchronization entry is written. For example the source entry, the entry in the directory, the generated operation, and the result of the operation represent a logical unit and is written completely to the trace file before the maximum number of records written is checked.

If more than one trace file is written an internal sequence number in the format ***-number*** is inserted into the filename of the next trace file when a trace file exceeds the maximum number of records. Use the wildcard character (*****) in the **-file** *file_name* option to specify the position of this sequence number. For example you specify the following options:

```
...
-file trace*_file.trc
...
-maxtracerecords 50000
```

After writing at least the 50000th record to the trace file currently in use the file is closed. A new trace file is created. The internal sequence number is inserted at the position of the wildcard character into the filename of the new file. After finishing the synchronization process you find for example the following trace files:

```
trace_file.trc
trace-1_file.trc
trace-2_file.trc
trace-3_file.trc
```

where "trace_file.trc" is the first trace file written and closed after exceeding the maximum number of records.

If you do not use a wildcard character to specify the position of the sequence number this number is appended to the filename (before its extension, if there is one), for example "trace_file-1.trc" or "trace-1".

Use the **-maxentries** option to set a limit on the number of entries in a search results list that will be processed by **meta writetrace** operations. When the **-maxentries** option is specified, the **meta writetrace** operation writes only the number of entries specified in *number* to the **metacp** trace file. If this option is not specified, the default is "no limit".

Use the **-mode** option to control whether or not **metacp** performs update operations on the Identity store (a connected directory of type **LDAP**) during an import process. When the **-mode** option is set to TRIAL, **metacp** logs update operations with the message "operation ignored" and does not make the changes to the Identity store. When the **-mode** option is set to REAL, **metacp** makes the updates to the Identity store.

The generated trace file contains the source entry information. Consequently, if errors occur during an import from a connected directory of type **FILE** into the Identity store (a connected directory of type **LDAP**), administrators can re-run the synchronization profile using the trace file as input (after first fixing the errors reported in the trace file)

See the "Return Values" section in the **metacp** reference page for a description of **metacp** error messages.

Return Values

None.

Example

meta initialize -file tracefile.log -tracelevel 2 -maxentries 10

See Also

addentry, modifydn, modifyentry, removeentries, terminate, writetrace

Section **Synchronization Profile** (Chapter 1, *DirX Identity Connectivity Administration Guide*)

Chapter 2, "Synchronization Templates", *DirX Identity Connectivity Administration Guide*

meta modifydn

Modifies an entry's name in the Identity store (a connected directory of type **LDAP**). The syntax is as follows:

[**meta**] **modifydn**
  -entry *entry_handle*

Options

-*entry* entry_handle

A required option that specifies the handle to the entry whose distinguished name is to be changed. Specify the name of an entry handle created by a **meta gethandle** operation called with a **-conn** option that specifies a connection handle to a connected directory of type **LDAP**.

The **modifydn** operation modifies the distinguished name of the entry associated with *entry_handle* in the Identity store (a connected directory of type **LDAP**). The operation uses the following attributes of the Tcl variable array associated with *entry_handle*:

- *entry_handle*(*new_rdn*)-the attribute that holds the new RDN for the entry. The name of this attribute is defined in the New-RDN field in the attribute configuration file.
- *entry_handle*(*del_old_rdn*)-the attribute that represents the "delete old RDN" flag (optionally present in the Tcl variable array). The name of this attribute is defined in the Delete Old-RDN field in the attribute configuration file. The default value **1** means that the old RDN is to be deleted and the value **0** means that the old RDN is to be preserved.
- *entry_handle*(*new_sup*)-the attribute that represents the new superior distinguished name (optionally present in the Tcl variable array). The name of this attribute is defined in the New Superior field in the attribute configuration file.

The **modifydn** operation changes the last RDN of an entry in the connected directory. The *entry_handle*(*del_old_rdn*) attribute controls whether or not the old RDN value should be preserved in the directory entry. If present, the *entry_handle*(*new_sup*) attribute directs **modifydn** to move the entry (or subtree) into a different subtree of the DIT. The pseudo attributes *new_rdn*, *del_old_rdn*, and *new_sup* must be selected with the **-attribute** option in the **openconn** operation.

If the underlying connection handle (derived from entry_handle) is an LDAP directory with the parameter **-ldifhandle** set, then the **modifydn** is not applied to the LDAP directory; instead an LDIF-CHANGE operation is created in the LDIF-CHANGE file associated with **-ldifhandle** parameter.

metacp automatically propagates a JMS message (containing the SPML add request) if at the beginning of a synchronization scenario **ats initialize** has been called with the options **-topicprefix**, **-type**, **-cluster**, and **-resource**. (See **ats initialize** for details.)

Return Values

Check the Tcl variable **errorCode** for error details. If this variable is not set or its value is empty, the entry was successfully renamed. Otherwise, the error is logged in the trace file and the distinguished name was not modified.

0	**The distinguished name modification succeeded, or it failed with an error that did not prevent the further processing of operations in the Tcl script logic.**
1	The distinguished name modification failed with a serious error; for example, "LDAP server not available". Examine your Tcl script logic to determine whether it makes sense to terminate the Tcl synchronization script.

Example

```
meta modifydn -entry en
```

See Also

gethandle, initialize, openconn, openldif, ats initialize

Attribute Configuration File Format (Chapter 2)

meta modifyentry

Updates the contents of an entry in the Identity store (a connected directory of type **LDAP**). The syntax is as follows:

[**meta**] **modifyentry**
  -newentry *entry_handle*
  [**-oldentry** *entry_handle*]
  [**-tag** *tag*]
  [**-allowrename** *rename_flag*]
  [**-updateinfo** *update_flag*]
  [**-updateattr** *attribute_list*]

Options

-*newentry* entry_handle

A required option that specifies the entry that contains the update information. Specify an entry handle created by a **meta gethandle** operation called with a **-conn** option that specifies a connection handle to a connected directory of type **LDAP**.

-*oldentry* entry_handle

Specifies the entry to be updated. Specify an entry handle created by a **meta getentry** operation called with a **-source** option that specifies a result handle, or by a **meta findentry** or **meta gethandle** operation. This option is not required if the **-newentry** option specifies an entry handle that refers to an LDIF change entry with changetype modify.

-*tag* tag

Specifies a marking action to be performed on the modified entry. Specify the keyword **MARK** in the *tag* argument to this option.

-*allowrename* rename_flag

Controls whether the distinguished name of the entry specified in the **-oldentry** option is to be renamed to the distinguished name of the entry specified in the **-newentry** option. Specify one of the following keywords:

- **TRUE**-Rename the old entry's distinguished name to the new entry's distinguished

name

- **FALSE**-Do not rename old entry's distinguished name (the default if this option is not specified)

-*updateinfo* update_flag

Controls whether a **modifyentry** operation returns a value that indicates that no operation was performed because the entry was already up-to-date. Specify one of the following keywords:

- **SIMPLE**-Return a value that indicates when an entry is already up-to-date
- **NONE**-Do not return a value when an entry is already up-to-date (default)

-*updateattr* attribute_list

An optional parameter that specifies an additional attribute list. All these attributes are created in case a difference in the attribute values of the old entry and the new entry has been detected and the necessary changes have been applied. The attribute list has to be specified in native LDAP syntax, e.g.

```
[meta] modifyentry newentry new_entry oldentry old_entry updateattr
dxmState=CHANGED
```

Using the -updateattr switch allows to handle additional attributes, which normally should not be synchronized under any circumstances. (Otherwise, these attributes must have been selected in the associated entry_handles and therefore are always subject to synchronization. That is not desirable in any cases. Only in case the object has really been modified, the additional attributes from attribute_list should be stored in the directory entry.)

The **modifyentry** operation updates the contents of an entry with the contents of the entry specified by **-newentry** *entry_handle*.

The **modifyentry** operation compares the new entry to the old entry, evaluates the connection handle for any synchronization flags set on the attributes to be processed, and generates the operations that are necessary to convert the contents of the old entry to the contents of the new entry (adding, replacing, and deleting the entry attributes, depending on the sync flags set with the **-attribute** option in the **openconn** operation).

Use the **-oldentry** option to identify the entry that is to be updated with the contents of the entry specified with the **-newentry** option. You do not need to specify **-oldentry** if **-newentry** *entry_handle* refers to an LDIF change file entry with the **modify** changetype. In this case, the entry handle contains the information about the entry that is to be modified. But in case you do provide an **oldentry** entry_handle for LDIF change files, only changes that still can be applied to the directory entry will be processed. Changes that cant be applied (e.g. removal of an attribute telephone number even if telephone number doesn't exist) will be ignored.

The Tcl variable arrays associated with both **-oldentry** *entry_handle* and **-newentry** *entry_handle* must contain the entry's distinguished name in the DDN attribute.

Use the **-tag** option to mark a modified entry as "processed" by a **modifyentry** operation. The **removeentries** operation uses the tag to determine which entries to remove.

Use the **-allowrename** option to control whether **metacp** modifies the DDN attribute of an entry when it updates its other attributes. When this option is set to **FALSE** (or is not used), the distinguished name in the DDN attribute of the entry that contains the update information must match the distinguished name in the DDN attribute of the entry to be updated.

Use the **-updateinfo** option to control the **modifyentry** return values. Specify **NONE** to return the values **0** (modification of the entry was successful) or **1** (modification of the entry failed). Specify **SIMPLE** to return the values **0** (success) **1** (failure) or **2** (no operation because the entry was already up-to-date). The extended return values (0,1,2) allow you to use the **modifyentry** function in more complex ways within your synchronization profiles.

If the underlying connection handle (derived from entry_handle) is an LDAP directory with the parameter **-ldifhandle** set, then the **modifyentry** is not applied to the LDAP directory; instead an LDIF-CHANGE operation is created in the LDIF-CHANGE file associated with **-ldifhandle** parameter.

metacp automatically propagates a JMS message (containing the SPML add request) if at the beginning of a synchronization scenario **ats initialize** has been called with the options **-topicprefix**, **-type**, **-cluster**, and **-resource**. (See **ats initialize** for details.)

Return Values

Check the Tcl variable **errorCode** for error details. If this variable is not set or its value is empty, the entry was successfully modified and/or renamed. Otherwise, the error is logged in the trace file and the entry was not modified.

0	**The entry modification succeeded, or it failed with one or more errors that did not prevent further processing of operations in the Tcl script logic.**
1	The entry modification failed with a serious error; for example, "LDAP server not available". Examine your Tcl script logic to determine whether it makes sense to terminate the Tcl synchronization script.
2	(If **-updateinfo** specifies **SIMPLE**) Modification of the entry was not performed because the entry was already up-to-date.

Example

```
meta modifyentry -newentry eh_LDAP -oldentry cur_entry -tag MARK
```

See Also

findentry, gethandle, getentry, initialize, openconn, openldif, removeentries, ats initialize

meta openconn

Opens a connected directory for synchronization. The syntax is as follows:

[**meta**] **openconn**
-type *directory_type*
[**-file** *filename* **-mode** *file_open_mode*
[**-format** *file_format*] [**encoding** *encoding*]]
[**-attribute** *attribute_list* [**-processallattr**]]
-attrconf *attr_conf_handle*
-conn *connection_handle*
[**-bindid** *binding_id*]
[-**ldifhandle** *LDIF_file_handle*]

Options

-*type* directory_type

A required option that identifies the type of connected directory. Specify one of the following keywords:

- **FILE**-A data file (an import file or an export file)
- **LDAP**-A directory that is accessible over LDAP

-*file* filename

The name of the data file to be opened. This option can only be used if the **-type** option is **FILE** and is a required option in this case.

-*mode* file_open_mode

Specifies the mode to use when opening *filename*. This option can only be used if the **-type** option is **FILE** and is a required option in this case. Specify one of the following keywords:

- **READ**-Opens the file read-only
- **WRITE**-Opens the file write-only (and write from beginning of file)
- **APPEND**-Opens the file for writing at end-of-file

-*format* file_format

The file format of the data file. Specify one of the following keywords:

- **TAGGED** - If the **-type** option is **FILE**, the data file uses a tagged file format (the default if this option is not specified)
- **NON-TAGGED** - If the **-type** option is **FILE**, the data file uses a non-tagged file format
- **LDIF-CHANGE** - If the **-type** option is **FILE**, the data file uses LDIF change format. If the **-type** option is **LDAP**, LDIF changes are to be imported to the directory.
- **DSML or DSMLv1** - If the -**type** option is **FILE**, the data file uses Directory Services Markup Language (DSML) format (version 1)
- **DSMLv2-REQ** - If the **-type** option is **FILE** and the **mode** option is either **WRITE** or

APPEND, the data file uses Directory Services Markup Language (DSML) request format (version 2). In that case, the TCL variables that are used in **meta writerecord** use the same format as if LDIF change format would need to be produced.

- **DSMLv2-RSP** If the **-type** option is **FILE** and the **mode** option is **READ**, the data file uses Directory Services Markup Language (DSML) response format (version 2). The only elements that will be processed are search responses.
- **FLAT-XML -** If the -**type** option is **FILE**, the data file uses flat XML format

-*encoding* encoding

Specifies the character set encoding of the data file. If the **encoding** parameter is missing, then the encoding as defined in the **_localcode** Tcl variable is used.

Later on while reading the file (using getentry), the data read from file is internally converted from the local encoding to the internal Tcl format UTF-8.

Tcl 8.3 supports a variety of different encodings. (see install_path\lib\tcl8.3\encoding (on Windows) or install_path/lib/tcl8.3/encoding (on UNIX) or use the command **encoding names** in metacp). The encodings listed in these encoding files can be used by dropping the file suffix .enc.

Example:

```
Encoding file name: cp850.enc
encoding must be set to cp850.
```

-*attribute* attribute_list

A list of attributes to be processed.Specify *attribute_list* as a Tcl list in the format:

{*attribute*}

Each *attribute* in the Tcl list has the format:

abbreviation [{*sync_flag*}]

where *abbreviation* is an attribute abbreviation in the attribute configuration file specified in the **-attrconf** option and each *sync_flag* is one or more of the following keywords:

DONT-ADD-ATTR

DONT-ADD-REC-ATTR-VAL

DONT-DEL-ATTR

DONT-DEL-REC-ATTR-VAL

DONT-MOD-ATTR

REPLACE-ALL

Multiple *sync_flag* arguments are represented as a Tcl list.The *sync_flag* argument is

only valid if the **-type** option is **LDAP**.

-processallattr

Specifies that all of the attributes defined in the attribute configuration file specified in the **-attrconf** option are to be processed. This option can only be used if the **-type** option is **LDAP**.

***-attrconf* attr_conf_handle**

A required option that specifies the handle to the attribute configuration file for the connected directory. Specify an attribute configuration handle created by a **meta readattrconf** operation.

***-conn* connection_handle**

A required option that specifies a name to be associated with the connection handle structure returned by the operation.

***-bindid* binding_id**

Specifies a binding ID to be associated with the connected directory. Specify a binding ID created by an **obj bind** operation. This option can only be used if the **-type** option is **LDAP**.

***-ldifhandle* LDIF_file_handle**

Specifies the name of an LDIF file handle (created by **meta openldif**) that is used when changes should not directly be applied to the LDAP directory; instead the changes are written as LDIF-CHANGE entries in the LDIF file associated with LDIF_file_handle. This option can only be used if the **-type** option is **LDAP**.

The **openconn** operation opens a connected directory for synchronization, creates a connection handle for it, and assigns it the name specified in the **-conn** option. The connection handle points to an import file, an export file, an LDAP connection to a directory, depending on the **-type** option specified in the command line. If a LDAP connection to a directory is being opened, the synchronization profile must perform an **obj bind** operation with suitable protocol (LDAPv2 or LDAPv3) before the **openconn** operation. The meta controller does not check for the bind in the **openconn** operation itself; but it does check in the **addentry, modifyentry, modifydn, removeentries** operations (which operate on handles containing a nested connection handle) and in **obj search** (with connection handle specified).

When multiple binds are open to different LDAP directories, you can use the **-bindid** option to associate the connection handle with a specific bind. The *binding_id* argument to this option specifies binding ID assigned to the bind during the **obj bind** operation. If the **-bindingid** option is not used, the **openconn** operation associates the connection handle with the default bind.

Use the **-attribute** option to select specific attributes for synchronization from the connected directory's attribute configuration file. The selected set becomes the set of synchronized attributes (SSA). Use the *sync_flag* argument (for connected directories of type **LDAP** only) to specify the operations that are allowed during synchronization on each attribute in the subset specified in the **-attribute** option. The *sync_flag* keywords have the following effects on the synchronization of attribute values:

- **DONT-ADD-ATTR**-Do not create an attribute in the target directory even if the corresponding attribute in the source directory exists. If **metacp** is processing an LDIF change entry and it encounters a changetype "modify" operation with an "add" modification for a specific attribute, **metacp** uses this flag to verify whether or not it can create the attribute. In this case, the Identity store may already hold the attribute; if it does, **metacp** creates an additional attribute value for the attribute.
- **DONT-ADD-REC-ATTR-VAL**-Do not add additional attribute values in the source directory to the attribute in the target directory
- **DONT-DEL-ATTR**-Do not delete the attribute in the target directory even if the corresponding attribute in the source directory is deleted
- **DONT-DEL-REC-ATTR-VAL**-Do not delete a recurring attribute value in the target directory even if the corresponding attribute in the source directory does not have the recurring value.
- **DONT-MOD-ATTR**-Do not modify the attribute value(s) of the attribute in the target directory. If this flag is set, no modification at all is performed (new attribute values cannot be created, and existing values cannot be removed)
- **REPLACE-ALL**-replace the existing attribute value(s) in the target directory with the attribute value(s) in the source directory. If the Identity store has no equality matching rule for an attribute, this flag needs to be set for the attribute in order to permit it to be updated. An example of an attribute for which the Identity store has no matching rule is Facsimile-Telephone-Number. The **Mrule** field in the attribute configuration file specifies whether a matching rule is defined for the attribute.

Valid combinations of *sync_flag* are:

- **DONT-MOD-ATTR** with no other flag set
- **REPLACE-ALL** with no other flag set
- Any combination of the flags **DONT-ADD-ATTR, DONT-ADD-REC-ATTR-VAL, DONT-DEL-ATTR** and **DONT-DEL-ATTR**

Use the **-processallattr** option to synchronize all of the connected directory's attributes (when the **-type** option is **LDAP**). You can use this option in conjunction with the **-attribute** option to select certain attributes and assign special processing to them.

When the **-type** option is **LDAP**, the **-attributes** option must specify all of the naming attributes used in the connected directory, unless the **-processallattr** option is specified.

The connection handle created by **openconn** defines the set of attributes to be synchronized (SSA). This set is either the same as the set of attributes to be synchronized that is specified in the attribute configuration file (if the **-attribute** option is not specified, or if **-processallattr** is specified) or it is a subset of the SSAs specified in the attribute configuration file (if the **-attribute** option is specified).

Return Values

None.

Examples

1. Open a tagged data file for reading and allow processing of all of the attributes associated with the attribute configuration file.

```
meta openconn -type FILE \
    -file msexch.data \
    -mode READ \
    -format TAGGED \
    -attrconf ah \
    -conn ch
```

2. Open a non-tagged data file for reading and select attributes for processing. The selected attributes correspond to the first through sixth fields of the data file.

```
meta openconn -type FILE \
    -file msexch.data \
    -mode READ \
    -format NON-TAGGED \
    -attrconf ah \
    -attribute {o ou sn givenName telephoneNumber
facsimileTelephoneNumber} \
    -conn ch
```

3. Open a tagged data file for writing and allow processing of all attributes.

```
meta openconn -type FILE \
    -file notes_out.data \
    -mode WRITE \
    -format TAGGED \
    -attrconf ah \
    -conn ch
```

4. Open a non-tagged data file for writing and select attributes for processing. The selected attributes correspond to the first through sixth fields of the data file.

```
meta openconn -type FILE \
    -file msexch.data \
    -mode WRITE \
    -format NON-TAGGED \
```

```
    -attribute {o ou sn givenName telephoneNumber
facsimileTelephoneNumber} \
    -attrconf ah \
     -conn ch
```

5. Open an LDAP connection and allow all attributes (except DDN) to be processed.

```
meta openconn -type LDAP \
    -attrconf ah \
     -conn ch
```

6. Open an LDAP connection and specify a set of attributes to be processed.

```
meta openconn -type LDAP \
    -attrconf ah \
    -conn ch\
    -attribute {o ou sn givenName telephoneNumber
facsimileTelephoneNumber}
```

7. Open an LDAP connection and allow all attributes (except DDN) to be processed. For attribute sn, disallow attribute deletion. For attribute givenName, disallow attribute deletion and attribute creation.

```
meta openconn -type LDAP \
    -attrconf ah \
    -attribute {{sn DONT-DEL-ATTR} \
    {givenName DONT-DEL-ATTR DONT-ADD-ATTR}} \
    -processallattr
```

See Also

getentry, gethandle, openldif, readattrconf, obj bind obj search

meta openldif

Creates an LDIF-CHANGE file. The syntax is as follows:

[**meta**] **openldif**
  -**file** *filename*
  -**ldifhandle** *LDIF_file_handle*

Options

-*file* filename

A required option that specifies the file name of the LDIF-CHANGE file.

-*ldifhandle* LDIF_file_handle

Specifies the name of an LDIF file handle that is used when changes should not directly be applied to the LDAP directory; instead the changes (calculated by **meta addentry**, **meta modifydn**, **meta modifyentry**, **meta removeentries**) are written as LDIF-CHANGE entries in the LDIF file associated with *LDIF_file_handle*.

The **openldif** operation opens the given file (as output file containg LDIF-CHANGE records) and associates that file with the given *LDIF_file_handle*. That handle can later on be used in **meta openconn**.

Return Values

None.

Example

```
meta openldif -file /homes/metadir/data/ldif.txt \
     -ldifhandle ldif_file
```

See Also

addentry, modifydn, modifyentry, openconn, removeentries

meta operations

Returns a list of operations that can be performed on the **meta** object. The syntax is as follows:

meta operations

The list of available operations is in alphabetical order except for **help** and **operations**, which are listed last.

Return Values

The requested information.

Example

```
meta operations
```

The output of the sample command is as follows:

```
addentry createindexlist escapechar findentry getentry gethandle
getnumentries initialize modifydn modifyentry openconn openldif
readattrconf releasehandle removeentries sortindexlist sortresult
supinfo terminate unescapechar writeindexlist writerecord writetext
writetrace help operations
```

meta readattrconf

Reads an attribute configuration file. The syntax is as follows:

[**meta**] **readattrconf**
  -file *filename*
  -attrconf *attr_conf_handle*
  [**-encoding** *encoding*]

Options

-file **filename**

A required option that specifies the pathname of the attribute configuration file. If only a file name is given, the file is assumed to be located the current working directory.

-encoding **encoding**

Specifies the character set encoding of the attribute configuration file. If the **encoding** parameter is missing, then the encoding as defined in the **_localcode** variable is used.

The data read from the attribute configuration file is internally converted from the local encoding to the internal Tcl format UTF-8.

Tcl 8.3 supports a variety of different encodings. (see install_path\lib\tcl8.3\encoding (on Windows) or install_path/lib/tcl8.3/encoding (on UNIX) or use **encoding names** in metacp). The encodings listed in these encoding files can be used by dropping the file suffix .enc.

Example:

```
Encoding file name: cp850.enc
encoding must be set to cp850.
```

-attrconf **attr_conf_handle**

A required option that specifies the name to be associated with the attribute configuration handle returned by the operation.

The **readattrconf** operation reads an attribute configuration file into the **metacp**

workspace, creates a handle to it, and assigns it the name specified in the **-attrconf** option. An attribute configuration file defines the attributes that are present in a particular connected directory and supplies formatting information that **metacp** is to use when processing data files associated with the connected directory.

A synchronization profile generally contains two **readattrconf** operations: one to read the attribute configuration file associated with the source directory, and one to read the attribute configuration file of the target directory.

Return Values

None.

Example

**meta readattrconf -file /homes/metadir/conf/mail/exchgattr.cfg **
-attrconf exchg_ah

See Also

openconn

Attribute Configuration File Format (Chapter 2)

meta releasehandle

Releases a meta handle. The syntax is as follows:

[**meta**] **releasehandle** *handle*

The **releasehandle** operation releases the meta handle specified in *handle*. See the *handle* argument description in this chapter for a list of meta handle types.

A meta handle's context can be dependent on the context of another meta handle. For example, the entry handle context returned by a **meta findentry** operation is dependent on the context of a result handle returned by an **obj search** operation. In the case of handles with dependencies, you can only release a handle after you have first released all of its dependent handles. For example, you must release an entry handle before you can release the result handle on which it is dependent. Similarly, you must release a connection handle before you can release the attribute configuration file handle on which the connection handle is dependent. See the *handle* argument description for the list of handle dependencies.

Return Values

None.

Example

```
meta releasehandle LDAP_results_rh
```

meta removeentries

Removes an entry or removes entries in a sorted search results list from the Identity store (a connected directory of type **LDAP**). The syntax is as follows:

[**meta**] **removeentries**
  -handle *handle*
  [**-tag** *tag*]

Options

-*handle* handle

A required option that specifies an entry handle (for removal of a single entry) or a result handle (for removal of entries in a sorted search results list). Specify an entry handle created by a **meta getentry**, **meta gethandle**, or **meta findentry** operation or a result handle created by an **obj search** operation that has been sorted by a **meta sortresult** operation.

-*tag* tag

Specifies whether marked or unmarked entries are to be removed, if the operation is removing entries from a results list. Specify one of the following keywords:

- **MARKED**-remove all marked entries
- **NOTMARKED**-remove all unmarked entries (default)

The **removeentries** operation removes an entry specified by an entry handle or the entries in a results list specified by a result handle from the Identity store (a connected directory of type **LDAP**).

If the **-handle** option specifies a result handle to a search results list, use the **MARKED** keyword in the **-tag** option to remove all entries from the Identity store (a connected directory of type **LDAP**) that the **meta modifyentry** or **meta findentry** operations have marked as "processed" in the results list. Use the **NOTMARKED** keyword to remove all entries that have not been marked by **meta modifyentry** or **meta findentry**.

The meta controller will not remove an entry for which it has information that the entry has subordinates. The meta controller will have this information for an entry in two cases:

- When it has attempted to remove a subordinate entry and the operation has failed
- When is has encountered a subordinate entry that is marked as not to be deleted

If the underlying connection handle (derived from entry_handle) is an LDAP directory with the parameter -**ldifhandle** set, then the **removeentries** is not applied to the LDAP directory;

instead an LDIF-CHANGE operation is created in the LDIF-CHANGE file associated with **-ldifhandle** parameter.

metacp automatically propagates a JMS message (containing the SPML add request) if at the beginning of a synchronization scenario **ats initialize** has been called with the options **-topicprefix**, **-type**, **-cluster**, and **-resource**. (See **ats initialize** for details.)

Return Values

Check the Tcl variable **errorCode** for error details. If this variable is not set or its value is empty, the entry (or all of the entries from the result list) was successfully removed. Otherwise the error is logged in the trace file and the entry (or entries) was not removed.

0	**The entry removal succeeded, or it failed with one or more errors that did not prevent further processing of operations in the Tcl script logic.**
1	The entry removal failed with serious error; for example, "LDAP server not available". Examine your Tcl script logic to determine whether or not it makes sense to terminate the Tcl synchronization script.

Example

```
meta removeentries -handle rh -tag MARKED
```

See Also

getentry, initialize, modifyentry, obj search, openconn, openldif, ats initialize

meta sortindexlist

Sorts the entries in an index list. The syntax is as follows:

[**meta**] **sortindexlist -handle** *index_list_handle*
  [**-keepnames** *index_list_access_flag*]

Options

-handle **index_list_handle**

A required option that specifies the name of an index list handle created by a **meta createindexlist** operation.

-keepnames **index_list_access_flag**

Selects whether random access to index list elements by distinguished name or sequential access to index list elements can be performed. Specify one of the following keywords in *index_list_access_flag*:

- **TRUE**-permit random access to index list elements by distinguished name

- **FALSE**-do not permit random access to index list elements (default)

The **sortindexlist** operation sorts the entries in the index list specified by the **-handle** option. The operation sorts the entries by distinguished name in alphabetical (ascending) order. Use the **sortindexlist** operation in conjunction with the **meta createindexlist**, **meta getentry**, and **meta writeindexlist** operations to create a sorted list (by distinguished name) of entries. Note that an index list can be sorted only once.

Use the **-keepnames** option to control whether the distinguished names specified in the **-name** option of previous **meta writeindexlist** operations can be used as indexes into the sorted index list, or whether only sequential access can be performed.

Return Values

None.

Example

```
meta sortindexlist -handle ih
```

See Also

getentry, createindexlist, writeindexlist

meta sortresult

Sorts the results of a read or search operation. The syntax is as follows:

[**meta**] **sortresult**
  -result *result_handle*
  -order *sort_order*
  [**-key** *sort_key*]

Options

-result *result_handle*

A required option that specifies a handle to a search results list. Specify the name of a result handle created by an **obj search** operation.

-*order* sort_order

A required option that specifies the order in which the results are to be sorted. Specify one of the keywords:

- **ASC**-To sort in ascending order (a-z for alphabetic sort, 1-n for numeric sort)
- **DESC**-To sort in descending order (z-a for alphabetic sort n-1 for numeric sort)

***-key* sort_key**

Specifies one or more attribute abbreviations that indicate a precedence of sort criteria. (In the current release only one sort key is supported.)

Specify the keyword DDN to sort by distinguished name or any other attribute type.

The **sortresult** operation sorts the entries in the search result specified by the **-result** option, updates the result data structure with information about how the result has been sorted, and returns a handle to the sorted result.

Use the **-key** option to sort the results list by distinguished name or by any other attribute type.

If the result list has been sorted according to any attribute type, then that attribute must be single-valued for all the entries of the search result. Entries that don't hold that specific attribute type will be listed at the end of the sorted list (if sorting has been done in ascending order) or at the beginning of the sorted list (if sorting has been done in descending order). If several such entries exist, their position at the beginning or end of the sorted list is undefined. As a consequence, it is unpredictable which entry will be returned by a "meta findentry" operation that uses such an ambiguous value. Therefore it is strongly recommended to use only attributes as sort_key that are unique, like employee number.

Return Values

None.

Examples

```
meta sortresult -result rh -order ASC -key DDN
meta sortresult -result rh -order DESC -key employeeNumber
```

meta supinfo

Supplies attribute information for creating a superior entry in the Identity store (a connected directory of type **LDAP**). The syntax is as follows:

[**meta**] **supinfo**
  -rdn *attribute_type*
  {**-attribute** *attribute_value* | **-remove**}
  [**-bindid** *binding_id*]

Options

***-rdn* attribute_type**

A required option that specifies the naming attribute for the superior entry. For a

connected directory of type LDAP, specify a valid LDAP attribute name.

-*attribute* attribute_list

Specifies one or more attributes to be applied to the superior entry. See the description of the **-attribute** option of the **obj create** operation in the **obj** reference pages for a description of *attribute_list* format. This option and the **-remove** option are mutually exclusive.

-remove

Deletes the attribute information for creating a superior entry. The **-attribute** option and this option are mutually exclusive.

-*bindid* binding_id

Specifies the binding ID to which the **supinfo** operation is to be applied. Specify a binding ID created by an **obj bind** operation.

The **supinfo** operation defines attribute information for the creation of superior entries in the Identity store. When it creates an entry in the Identity store (a connected directory of type **LDAP**), the **meta addentry** operation uses this information to create any required superior entries for the entry, if these superior entries do not already exist in the Identity store (a connected directory of type **LDAP**).

Use the **-attribute** option to specify the attributes to be applied to the superior entry. You must specify the ObjectClass (OCL for DAP connections, objectClass for LDAP connections) attribute as an *attribute_list* element; specifying additional attributes depends upon the object class (you must supply the mandatory attributes for the object class).

Use the **-remove** option to remove an attribute definition previously specified with **supinfo**.

When multiple binds are open to different LDAP directories, you can use the **-bindid** option to direct the **supinfo** operation to be applied to a specific bind. The *binding_id* argument to this option specifies the binding ID assigned to the bind during the **obj bind** operation. In this way, you can establish different sets of superior information, and then apply them to the appropriate connected directory. If the **-bindid** option is not used, the **supinfo** operation carries out the operation on the default bind.

Since there is no syntax checking, **supinfo** <-> **openconn** or **supinfo** <-> **bind** can be performed in any order. It is the responsibility of the user to keep **supinfo/openconn/bind** consistent for each bind ID.

Example

```
meta supinfo -rdn C -attribute objectClass=C -bindid ldb
meta supinfo -rdn O -attribute objectClass=organization
     -bindid ldb
meta supinfo -rdn OU -attribute objectClass=organizationalUnit
     -bindid ldb
```

See Also

addentry

meta terminate

Closes the **metacp** trace file and cleans up the **metacp** workspace. The syntax is as follows:

[**meta**] **terminate**

The **terminate** operation writes statistics to the trace file opened with the **meta initialize** operation (if -**nostatistics** not set) and closes it. Here is an example of the type of statistics written to the trace file:

```
# Statistics:
#     Number of X.500 entries: 110
#     Number of file entries: 14

#     Successful Additions: 1
#     Unsuccessful Additions: 1
#     Ignored Additions: 0
#     Successful Modifications: 3
#     Unsuccessful Modifications: 1
#      Ignored Modifications: 0
#      Ignored Modify-Attributes: 0
#      Ignored Delete-Attributes: 0
#      Ignored Add-Attributes: 0
#     Entries already up to date: 9

#     Successful Deletions: 2
#     Unsuccessful Deletions: 0
#     Ignored Deletions: 0

#     Successful Modify-DNs: 0
#     Unsuccessful Modify-DNs: 0
#     Ignored Modify-DNs: 0
```

Explanations:

- Number of X.500 entries:

  Lists the number of entries read from the directory.

  

  There is only one global counter for entries read from the directory. As in

some Tcl-Scripts several search operations are sent to the directory, the results can be overlapping and therefore that number can be misleading. In such scenarios, that number is therefore not the number of the basic search operation in the directory.

- Number of file entries:

  Lists the number of entries read from a data file.

  

  There is only one global counter for entries read from data file. As in a synchronization template (Tcl scripts) several data files could be handled in parallel, that number is the total number of entries of all files.

  If the file is read a second time (e.g. while creating index lists), the entries are counted twice.

  Therefore, the number listed here may sometimes be misleading.

- Successful Additions:

  Lists the number of successful ADD operations.

- Unsuccessful Additions:

  Lists the number of unsuccessful ADD operations.

- Ignored Additions:

  Lists the number of ADD operations that have been ignored, because the synchronization runs in TRIAL mode.

- Successful Modifications:

  Lists the number of successful MODIFY operations.

- Unsuccessful Modifications:

  Lists the number of unsuccessful MODIFY operations.

- Ignored Modifications:

  Lists the number of MODIFY operations that have been ignored, because the synchronization runs in TRIAL mode.

- Ignored Modify-Attributes:

  Lists the number of modify changes that are ignored, because the attribute flags (in the openconn operation) forbid to perform the modification.

- Ignored Delete-Attributes:

  Lists the number of delete modifications that are ignored, because the attribute flags

(in the openconn operation) forbid to perform the modification.

- Ignored Add-Attributes:

  Lists the number of add modifications that are ignored, because the attribute flags (in the openconn operation) forbid to perform the modification.

- Entries already up to date:

  Lists the number of entries that exist in the directory and whose attributes are already matching the attribute values read from the data file.

- Successful Deletions:

  Lists the number of successful DELETE operations.

- Unsuccessful Deletions:

  Lists the number of unsuccessful DELETE operations.

- Ignored Deletions:

  Lists the number of DELETE operations that have been ignored, because the synchronization runs in TRIAL mode.

- Successful Modify-DNs:

  Lists the number of successful MODIFY-DN operations.

- Unsuccessful Modify-DNs:

  Lists the number of unsuccessful MODIFY-DN operations.

- Ignored Modify-DNs:

  Lists the number of MODIFY-DN operations that have been ignored, because the synchronization runs in TRIAL mode.

If in **meta initialize** the parameter -**nostatistics** is set, the statistic information as listed above is not written to the tracefile.

In that case, the user is advised to integrate statistic information in his synchronization scripts. Before calling **terminate**, it's his task to print this information using **meta writetext**.

Return Values

None.

Example

meta terminate

See Also

initialize, writetext

meta unescapechar

Unescapes one or more characters in a string. The syntax is as follows:

[**meta**] **unescapechar** *string*

The **unescapechar** operation "unescapes" one or more characters in a string by removing the backslash (\) prefix (previously inserted with **meta escapechar** or returned by an **obj search** operation). and returns the unescaped string as a result.

Return Values

The unescaped string.

Example

The **unescapechar** operation

```
meta unescapechar f\{1\}\=2
```

returns the string f{1}=2

See Also

escapechar

meta writeindexlist

Writes information about an entry into an index list. The syntax is as follows:

[**meta**] **writeindexlist**
  -handle *index_list_handle*
  -name *distinguished_name*
  -entry *entry_handle*

Options

-handle **index_list_handle**

A required option that specifies the name of an index list handle created by a **meta createindex** operation.

-name **distinguished_name**

A required option that specifies the distinguished name of the entry to be written to the

index list.

-*entry* entry_handle

A required option that specifies the name of the entry handle for the entry to be written to the index list. Specify an entry handle created by a **meta getentry** operation.

The **writeindexlist** operation writes an entry's distinguished name and the entry handle into the index list specified in the **-handle** option. Obtain the distinguished name either from the Tcl variable created by **meta getentry** (if the distinguished name is stored in the entry) or from a Tcl mapping operation. Use the **writeindexlist** operation in conjunction with the **meta createindexlist**, **meta getentry**, and **meta sortindexlist** operations to create a sorted list (by distinguished name) of entries from a connected directory of type **FILE** for directory synchronization processing.

A **writeindexlist** operation cannot be performed on an index list that has already been sorted with the **sortindexlist** operation.

Example

```
meta writeindexlist -handle ih \
    -name cn=fou,ou=mfg,o=pqr,c=de \
    -entry eh
```

See Also

getentry, createindexlist, sortindexlist

meta writerecord

Writes the contents of an entry into a connected directory. The syntax is as follows:

[**meta**] **writerecord**
-entry *entry_handle*

Options

-*entry* entry_handle

The handle to the entry to be written. Specify the name of a handle created by a **meta gethandle** operation called with a **-conn** option that specifies a connection handle to a directory of type **FILE** (opened with **openconn -type** option **FILE**) that has been opened in **WRITE** or **APPEND** mode (opened with **openconn -mode** option **WRITE** or **APPEND**).

The **writerecord** operation writes the contents of the entry associated with *entry_handle* into the connected directory of type **FILE** inherited by *entry_handle*. The data format generated depends on the **-format** option specified in the **openconn** operation (**TAGGED** or **NON-TAGGED**) and the syntax specified in the attribute configuration file.

Example

```
meta writerecord -entry eh
```

See Also

gethandle, openconn, writetext

Attribute Configuration File Format (Chapter 2)

meta writetext

Writes text to a connected directory of type **FILE** that has been opened for **WRITE** or **APPEND**. The syntax is as follows:

[**meta**] **writetext**
  -conn *connection_handle*
  -text *string*
  [**-nonewline** *flag*]

Options

-*conn* connection_handle

A required option that specifies the name of a connection handle created by a **meta openconn** operation called with the **-type FILE** and the **-mode WRITE** or **APPEND** options.

-*text* string

The text to be written to the data file, enclosed in double quotation marks ("").

-*nonewline* flag

Controls whether a newline character is appended to *string*. Specify one of the keywords:

- **TRUE**- Do not append a newline character
- **FALSE**-Append a newline character (default)

The **writetext** operation writes the text string specified in the **-text** option to the output data file represented by the connection handle specified in the **-conn** option. Use the **writetext** operation to write headers, trailers and other text into a data file that contains entry information.

Example

```
meta writetext -conn conn_ch -text "Start of deleted entries"
```

See Also

gethandle, openconn

meta writetrace

Writes a trace message into the **metacp** trace file. The syntax is as follows:

[**meta**] **writetrace**
  -text *string*
  [**-handle** *handle*]

Options

-text **string**

The message to be written to the trace file, enclosed in double quotation marks ("").

-handle **handle** The name of an entry handle or a result handle that holds additional information to be traced. To trace an entry handle, specify the name of an entry handle created by a **meta getentry** or a **meta gethandle** operation. To trace a result handle, specify the name of a result handle created by an **obj search** operation.

The **writetrace** operation writes the message specified in the **-text** option into the trace file created and opened by the **meta initialize** operation. Use the **-handle** option to trace the current entry in a connected directory or to trace the contents of a search results list. Use the **meta initialize -maxentries** option to set a limit on the number of entries to trace in a search results list.

Example

```
meta writetrace -text "Base object 'O=pqr' does not exist"
```

See Also

getentry, gethandle, initialize, obj search

1.1.4. obj (metacp)

Synopsis

[**obj**] **bind**
  [**-bindid** *bid*]
  [**-authentication** *auth_method*]
  [**-password** *password*]
  [**-user** *username*]
  [**-protocol** *protocol*]
  [**-server** *ldap_server_name* | **-address** *ldap_server_address*]

[**-ssl**]
[**-sasl**]
[**-cert7path** *cert8_pathname*]
[**-key3path** *key3_pathname*]
[**-certsubject** *nickname*]
[**-key3password** *key3_password*]
[**-mechanism** *mechanism*]
[**-status** [**-bindid** *bid1* [*bid2* ...]]]

[**obj**] **compare** *distinguished_name*
-attribute *attribute*
[**-bindid** *bid*]

[**obj**] **create** *distinguished_name*
-attribute *attribute_list*
[**-bindid** *bid*]

[**obj**] **delete** *distinguished_name*
[**-bindid** *bid*]

[**obj**] **help** [*operation* | **-verbose**]

[**obj**] **list** *distinguished_name*
[**-bindid** *bid*]
[**-pretty**]

[**obj**] **moddn** *distinguished_name*
[**-bindid** *bid*]
-rdn *name_part*
[**-dontdeleteoldrdn**]
[**-newsuperior** *new_superior*]

[**obj**] **modify** *distinguished_name*
[**-bindid** *bid*]
{**-addattr** *attribute_list* |
-changeattr *old_attribute new_attribute* |
-removeattr *attribute_list |*
-replaceattr *attribute_list*}

[**obj**] **nextpage**
[**-bindid** *bid*]
[**-conn** *connection_handle*]
[**-exactaction**]
[**-pretty**]
[**-result** *result_handle*]
[**-terminate**]
[**-vafter** *after_count*]
[**-vbefore** *before_count*]
[**-vcontentcount** *content_count*]
[**-voffset** *offset*]
[**-vpagesize** *page_size*]

[**-vvalue** *attribute_value*]

[**obj**] **operations**

[**obj**] **search** *distinguished_name*
[**-bindid** *bid*]
[**-conn** *connection_handle*]
[**-exactaction**]
[**-result** *result_handle*]
[**-allattr** |
-alluserattr |
-attribute *abbreviation* ...]
{**-baseobject** |
-onelevel |
-subtree}
[**-filter** *filter*]
[**-matchedvaluesonly**]
[**-pretty**]
[**-types**]
[**-vafter** *after_count*]
[**-vbefore** *before_count*]
[**-vcontentcount** *content_count*]
[**-voffset** *offset*]
[**-vpagesize** *page_size*]
[**-vsortkey** *sort_keys*]
[**-vtype** {**SUBENTRY** | **SIMPLE** | **VLV**}]
[**-vvalue** *attribute_value*]

[**obj**] **show** *distinguished_name*
[**-bindid** *bid*]
[**-allattr** |
-alluserattr |
-attribute *abbreviation* ...]
[**-pretty**]
[**-types**]

[**obj**] **unbind**
[**-bindid** *bid*]

Purpose

A **metacp** object that manages entries in the directory information tree (DIT) of a Identity store via the Lightweight Directory Access Protocol (LDAP) protocol.

Arguments

distinguished_name

The name of an entry to act on. Supply the complete distinguished name as follows:

distinguished_name

For example:

```
{cn=mueller peter,ou=board,o=pqr ag,c=de}
```

See the String Representation for LDAP Binds chapter for complete information on distinguished name format.

operation

The name of the **obj** operation for which to display help information.

Operations

obj bind

Establishes a binding between **metacp** and an Identity store. The syntax depends upon the protocol type (**-protocol** option) and the security level to be used.

For LDAPv2 binds (**-protocol LDAPv2**) the following syntax must be used:

[**obj**] **bind**
[**-bindid** *bid*]
[**-authentication** *auth_method*]
[**-password** *password*]
[**-user** *username*]
-protocol LDAPv2
[**-server** *ldap_server_name* | **-address** *ldap_server_address*]
[**-ssl**]

For LDAPv3 binds (**-protocol LDAPv3**) the following syntax must be used:

- Anonymous bind over plain LDAPv3 protocol:

  [**obj**] **bind**
  [**-bindid** *bid*]
  -protocol LDAPv3
  [**-server** *ldap_server_name* | **-address** *ldap_server_address*]

- Simple authenticated bind over plain LDAPv3 protocol:

  [**obj**] **bind**
  [**-bindid** *bid*]
  -authentication simple
  -password *password*
  -user *username*
  -protocol LDAPv3
  [**-server** *ldap_server_name* | **-address** *ldap_server_address*]

- Anonymous bind over SSL-protected LDAPv3 protocol (encrypted data transfer):

  [**obj**] **bind**
  [**-bindid** *bid*]
  -protocol LDAPv3
  [**-server** *ldap_server_name* | **-address** *ldap_server_address*]
  [**-ssl**]

- Simple authenticated bind over SSL-protected LDAPv3 protocol (encrypted data transfer):

  [**obj**] **bind**
  [**-bindid** *bid*]
  -authentication simple
  -password *password*]
  -user *username*
  -protocol LDAPv3
  [**-server** *ldap_server_name* | **-address** *ldap_server_address*]
  [**-ssl**]

- SASL-authenticated bind over SSL-protected LDAPv3 protocol (encrypted data transfer and certificate-based client authentication):

  [**obj**] **bind**
  [**-bindid** *bid*]
  -protocol LDAPv3
  [**-server** *ldap_server_name* | **-address** *ldap_server_address*]
  -sasl
  [**-cert7path** *cert8_pathname*]
  [**-key3path** *key3_pathname*]
  -certsubject *nickname*
  -key3password *key3_password*
  -mechanism EXTERNAL

To display the status of currently established binds the following syntax must be used:

[**obj**] **bind -status** [**-bindid** *bid1* [*bid2* ...]]

Options

-*address* ldap_server_address

The real address of a Identity store. Specify *ldap_server_address* in one of the following formats:

- *host*[:_port_]
- *host1*[:*port1*],*host2*[:*port2*][,...]

where *host* or *hostn* is either an IP address or a DNS (domain name server) name and *port* or *portn* is a port number (<32767). **389** is used as the default port number unless the **-ssl** option or the **-sasl** option and the **-mechanism EXTERNAL** options are specified;

in this case, the default port number is **636**.

If more than one real address is specified, an attempt is made to establish a connection using real addresses from left to right.

-authentication **auth_method**

The authentication method to be applied. Supply the **simple** keyword to use simple unprotected authentication.

-bindid **bid**

The name (*bid*) of the bind to be used. If this option is omitted, the default bind is used. (See the Bind Types and Bind IDs section in this chapter for details.)

-cert7path **cert8_pathname**

The full pathname of the **cert8.db** file that contains the certificate database used by the Mozilla ldapssl library. A certificate database typically contains Root certificates, server and user certificates. (See SSL/TLS Certificate Database in the DirX Identity Program Files chapter for details.)

The default value is *install_path***/client/conf/cert8.db**.

-certsubject **nickname**

The nickname that the Mozilla ldapssl library uses to select the correct certificate and key from the **cert8.db** file and the **key3.db** file. Use the Mozilla certutil tool to view the contents of a **cert8.db** file to obtain the nicknames of the client certificates.

This option is mandatory when the **-sasl** option is specified and the value of the **-mechanism** option is **EXTERNAL**.

-key3password **key3_password**

The password that protects the **key3.db** file. Use the Mozilla certutil tool to change the value of this password. The password protects the private keys stored in the **key3.db** file.

This option is mandatory when the **-sasl** option is specified and the value of the **-mechanism** option is **EXTERNAL**.

-key3path **key3_pathname**

The full pathname of the **key3.db** file that contains the keys. (See SSL/TLS Key Database in the DirX Identity Program Files chapter for details.)

The default value is *install_path***/client/conf/key3.db**.

-mechanism **mechanism**

The SASL mechanism to use for client authentication and for the security layer. Supply the **EXTERNAL** keyword (case-sensitive) to use SSL/TLS certificate-based client authentication as the client authentication mechanism and SSL/TLS as the security layer. If a client uses this authentication mechanism, the Identity store authenticates the subject name of the certificate that is specified by the **-certsubject** option.

This option is mandatory when the **-sasl** option is specified.

***-password* password**

The password associated with the user on whose behalf the bind request is being made.

***-protocol* protocol**

The LDAP protocol to be used for the bind. Supply one of the following keywords (case-insensitive):

- **LDAPv2** - LDAPv2 bind
- **LDAPv3** - LDAPv3 bind

Specifying this option overrides the protocol specification field in the directory client configuration file **dirxcl.cfg**.

-sasl

The bind is to be performed using SASL protocol. This option is only supported for LDAP v3 protocol (**-protocol** option is **LDAPv3**).

-ssl

The bind is to be performed using SSL/TLS protocol.

***-server* ldap_server_name**

The symbolic name of a Identity store. **metacp** uses *ldap_server_name* to search the client configuration file **dirxcl.cfg** for an entry that specifies the Identity store´s real address and protocol type.

***-status* [-*bindid* bid1 [bid2 ...]]**

Reports status of specified bind IDs *bidn* or all bind IDs currently in use (including the default bind).This option is mutually exclusive to all other options.

***-user* username**

The name of the user on whose behalf the bind request is being made. Specify the complete distinguished name of the user. See the String Representation for LDAP Binds chapter for a description of distinguished name syntax.

Multiple binds are supported using bind IDs and the concept of a default bind. A bind ID is a string specified in the option **-bindid** associated with an LDAP bind. A bind ID can be used in other **metacp** commands to refer to a specific LDAP bind.

A **bind** operation (or any other **obj** operation) that does not specify a bind ID opens (or refers to) a default bind. There is only one default bind at a time.

Use the **-server** or **-address** options to bind to a specific Identity store. If a Identity store is not specified with the **-server** or **-address** options, the **obj bind** operation searches the directory client configuration file **dirxcl.cfg** for the first line that matches the protocol type specified in the **-protocol** option. The address from this line is used to establish a bind. If the directory client configuration file **dirxcl.cfg** does not contain a Identity store address line, the operation performs a bind to the local Identity store.

Use the **-authentication** option to establish a specific authentication method. If you specify simple authentication, you must supply the **-user** option and the **-password** option. The

password you supply for simple authentication is sent as clear text.

Use the **-ssl** option to establish Secure Socket Layer (SSL)/Transport Layer Security (TLS) for the bind operation when the **-protocol** option specifies **LDAPv2** or **LDAPv3**. When you specify this option, the **obj bind** operation authenticates the Identity store using the certificate information in the file specified in the **DIRX_TRUSTED_CA** environment variable or in the *install_path**/client/conf/cert8.db* certificate database, if the environment variable is not set. See SSL/TLS Certificate Database in the DirX Identity Program Files chapter for more information about SSL/TLS certificate authentication.

Use the **-sasl** option to establish Simple Authentication and Security Layer (SASL) client authentication and SSL/TLS for the bind operation when the **-protocol** option specifies **LDAPv3**. See the recommendations made in the document *Authentication Methods for LDAP* (RFC 2829) and *Lightweight Directory Access Protocol (v3): Extensions for Transport Layer Security* (RFC 2830) for details. The **-sasl** option includes the **-ssl** option.

When an **obj bind** operation is called with a specific bind ID, the operation sets the service controls for the bind to the default values; see the ldapargs modify operation for a list of default values. When an **obj bind** operation is called without a bind ID (a default bind), the operation sets the service controls to the default values when the LDAP protocol to be used changes. For example, the operation sets the service controls to their default values when the last default bind used LDAPv2 protocol and the default bind to be performed uses LDAPv3 protocol.

Example

```
bind -user cn=admin,o=my-company -password dirx \
     -auth simple -protocol LDAPv3 \
     -address hawk.virt.de.com -bindid hawkL3
bind -sasl -mech EXTERNAL -certsubject Admin-Nickname \
     -key3password zorro99 -prot LDAPv3
bind -status
```

The command output is as follows:

```
1) Connection:
Bind-ID: : hawkL3
Protocol: : LDAPv3
Address: : hawk.virt.de.com
User-Name: : cn=admin,o=pqr
Authentication Type: : simple
Status: : bound
2) Connection:
Bind-ID: : (default)
Protocol: : LDAP
```

```
Status: : unbound
```

obj compare

Compares an attribute name and value with the attribute names and values of an entry in the DIT. The syntax is as follows:

[**obj**] **compare** *distinguished_name*
  -attribute *attribute*
  [**-bindid** *bid*]

Options

-attribute **attribute** A required option that specifies the attribute and attribute value to compare. See the String Representation for LDAP Binds chapter for a description of attribute type and value syntax.

-bindid **bid**

The name (*bid*) of the bind to be used. If this option is omitted, the default bind is used. (See the Bind Types and Bind IDs section in this chapter for details.)

Use the **obj compare** operation to determine whether an attribute in the entry specified by *distinguished_name* contains a specific value. If the specified attribute contains the specified value, the operation returns the text **M=TRUE** (meaning Matched=TRUE); if the value is not present, it returns **M=FALSE**.

Example

```
obj compare cn=zapf,ou=asw,o=sni,c=de \
      -attr TN=55029
```

obj create

Creates a new entry in the DIT. The syntax is as follows:

[**obj**] **create** *distinguished_name*
  -attribute *attribute_list*
  [**-bindid** *bid*]

Options

-attribute **attribute_list**

A required option that specifies one or more attributes to be applied to the entry. See the String Representation for LDAP Binds chapter for a complete description of directory service attribute, attribute types, and attribute list formats.

***-bindid* bid**

The name (*bid*) of the bind to be used. If this option is omitted, the default bind is used. (See the Bind Types and Bind IDs section in this chapter for details.)

The **create** operation creates a new entry in the directory information tree. The *distinguished_name* argument is the name of the entry to be created and must be specified as a complete distinguished name. See the String Representation for LDAP Binds chapter for a description of distinguished name syntax.

Use the **-attribute** option to specify the attributes to be applied to the newly created entry. You must specify the **ObjectClass** (**objectClass**) attribute as an *attribute_list* element; specifying additional attributes is optional. See the String Representation for LDAP Binds chapter for a complete list of the supported attribute types and attribute list formats. By default, the **metacp** program creates the operational attributes creation-time, creators-name, structural-object-class, and governing-structure-rule.

Examples

1. The following sample command creates the organization "myCompany".

```
create {o=myCompany,c=us} \
     -attribute objectClass=organization {description=sni usa}
```

2. The following sample command creates the organizational-unit "engineering" underneath the "myCompany" organization with the telephone-number attribute telephoneNumber=+1 964 123.

```
create {ou=engeneering,o=myCompany,c=us} \
     -attribute objectClass=organizationalUnit \
     {description=engineering department} \
     {telephoneNumber=+1 964 123}
```

3. The following sample command creates the organizational-person "hughes" whose telephone number is "423423" within the engineering organization and whose surname is "hughes" and whose given name is "peter".

```
create {cn=hughes,ou=engineering,o=myCompany,c=de} \
     -attribute {objectClass=organizationalPerson;person} \
     {description=software-engineer} \
     {telephoneNumber=+1 964 123 423423} \
     sn=hughes givenName=peter
```

obj delete

Deletes entries from the DIT. The syntax is as follows:

[**obj**] **delete** *distinguished_name*
  [**-bindid** *bid*]

Options

-bindid **bid**

The name (*bid*) of the bind to be used. If this option is omitted, the default bind is used. (See the Bind Types and Bind IDs section in this chapter for details.) The **delete** operation deletes entries from the directory information tree.

You cannot delete entries that still have subordinate entries.

Example

The following example deletes the entry **cn=Miller,ou=engineering,o=sni,c=us**

```
delete cn=miller,ou=engineering,o=sni,c=us
```

obj help

Returns help information about the **obj** object and its operations. The syntax is as follows:

[**obj**] **help** [*operation* | **-verbose**]

Options

-verbose

Displays information about the **obj** object.

Used without an argument or option, the **help** command returns brief information about each **obj** operation. Use the *operation* argument to return a description of the options associated with the operation you specify. Alternatively, you can use the **-verbose** option to return a description of the **obj** object itself.

Example

```
help
```

The output of the sample command is as follows:

```
bind          Binds to the specified directory server.
```

compare	Checks if the object has the specified attribute value.
create	Creates the specified object in the directory.
delete	Removes the specified object from the directory.
list	Lists the children of the specified object.
modify	Modifies the attribute values of an object in the directory.
moddn	Modifies the DN of an object in the directory.
search	Searches for objects in the directory.
show	Reads attributes of an object in the directory.
unbind	Unbinds from the directory server.
help	Displays help text for the 'obj' object and its operations.
operations	Lists the operations that can be performed on the 'obj' object.

obj list

Lists the distinguished names of the children of an entry. The syntax is as follows:

[**obj**] **list** *distinguished_name*
  [**-bindid** *bid*]
  [**-pretty**]

Options

***-bindid* bid**

The name (*bid*) of the bind to be used. If this option is omitted, the default bind is used. (See the Bind Types and Bind IDs section in this chapter for details.)

-pretty

Displays the results of the operation in a tabular format.

The **obj list** operation returns a Tcl list that contains the complete distinguished names of the immediate subordinates of the entry specified in *distinguished_name*. Use the **-pretty** option to return the results of the operation in a tabular, more readable format.

Example

The following sample command lists the distinguished names of the object **ou=ap11,o=sni,c=de**:

```
list ou=ap11,o=sni,c=de -pretty
```

The output of the sample command as follows:

```
1) cn=mueller,ou=ap11,o=sni,c=de
2) cn=hughes,ou=ap11,o=sni,c=de
3) cn=zahn,ou=ap11,o=sni,c=de
4) cn=schmid,ou=ap11,o=sni,c=de
```

obj moddn

Changes the last RDN of an entry or a subtree or moves an entry or subtree to a new superior. The syntax is as follows:

[**obj**] **moddn** *distinguished_name*
  [**-bindid** *bid*]
  -rdn *name_part*
  [**-dontdeleteoldrdn**]
  [**-newsuperior** *new_superior*]

Options

-bindid **bid**

The name (*bid*) of the bind to be used. If this option is omitted, the default bind is used. (See the "Bind Types and Bind IDs" section in this chapter for details.)

-dontdeleteoldrdn

Saves the attribute values of the old RDN that are not present in the new RDN.

-rdn **name_part**

A required option that specifies the new RDN. If the operation moves an object or subtree to a new superior without changing its RDN, the old RDN is supplied for this option. When the last RDN of the object is to be changed **grantRename** permission is required. See the String Representation for LDAP Binds chapter for a description of relative distinguished name syntax.

-newsuperior **new_superior**

Specifies the distinguished name of the new superior of the entry. The new superior must already exist. The new superior must not be the entry to be moved, or the root of the subtree to be moved, or one of its subordinates, or such that the moved object violates any DIT structure rules. If objects subordinate to the moved object violate the active subschema subsequent adjustments must be done to make these objects consistent with the subschema. **grantExport** permission is required for the object being considered with its original name, and **grantImport** permission is required for the object being considered with its new name. See the String Representation for LDAP Binds chapter for a description of distinguished name syntax.

The **moddn** operation changes the last relative distinguished name of an object or subtree or moves an object or subtree to a new superior. The *distinguished_name* argument specifies the entry or subtree to modify. See the String Representation for LDAP Binds chapter for a description of distinguished name syntax. By default, the operation removes

the attributes and attribute values that do not exist in the new RDN.

When the **obj moddn** operation modifies a distinguished name, it automatically updates all references to the complete name that exist within a single Identity store.

Example

In the following sample command, the entry is renamed but the old name (**zahn**) is kept in the new entry. The resulting **cn** attribute has two values: **zahn** and **soeder**, where **soeder** is the naming value of **cn**.

```
obj moddn cn=zahn,ou=staff,o=ibis,c=us -rdn cn=soeder \
     -dontdeleteoldrdn
```

obj modify

Changes the attribute values of an entry. The syntax is as follows:

[**obj**] **modify** *distinguished_name*
[**-bindid** *bid*]
{**-addattr** *attribute_list* |
-changeattr *old_attribute new_attribute* |
-removeattr *attribute_list |*
-replaceattr *attribute_list*}

Options

-*bindid* bid

The name (*bid*) of the bind to be used. If this option is omitted, the default bind is used. (See the Bind Types and Bind IDs section in this chapter for details.)

-*addattr* attribute_list

Adds the attributes and attribute values specified in the *attribute_list* argument to an object. See the String Representation for LDAP Binds chapter for a description of attribute type and value syntax.

-*changeattr* old_attribute new_attribute

Changes the specified existing attribute and attribute values to the specified new attribute and attribute values. See the String Representation for LDAP Binds chapter for a description of attribute type and value syntax.

-*removeattr* attribute_list

Removes the attributes and attribute values specified in the *attribute_list* argument from an object. See the String Representation for LDAP Binds chapter for a description of attribute type and value syntax

***-replaceattr* attribute_list**

Replaces the attributes and attribute values specified in the *attribute_list* argument from an object. See String Representation for LDAP Binds for a description of attribute type and value syntax

The **modify** operation changes attributes and attribute values of entries. The *distinguished_name* argument specifies the entry to modify. See the String Representation for LDAP Binds chapter for a description of distinguished name syntax.

Use the **-addattr** option to add new attributes and their attribute values to an entry or add new attribute values to an existing attribute.

Use the **-changeattr** option to modify values of an existing attribute. You must specify the option name and the desired values for each attribute you want to change. For example, to modify **attr1** and **attr2** for an entry, enter the **-changeattr** option twice as follows: **-changeattr attr1_old attr1_new -changeattr attr2_old attr2_new**

Use the **-removeattr** option to remove existing attribute values or remove existing attributes.

Use the -**replaceattr** option to replace all existing attribute values by the new attribute values.

You may combine the options **addattr, changeattr, removeattr** and **replaceattr** in any order as often as you like.

Examples

In the following sample commands, the **modify** operation performs the following tasks on the entry represented by the distinguished name **cn=miller,ou=sales,o=ins CO,c=us**:

1. Adds the telephone-number attribute type and value:

```
modify {cn=miller,ou=sales,o=ins CO,c=us} \
     -addattr {telephoneNumber=+1 964 123 5678}
```

2. Changes the **telephoneNumber** attribute value:

```
modify {cn=miller,ou=sales,o=ins CO,c=us} \
     -changeattr {telephoneNumber=+1 964 123 5678} \
     {telephoneNumber=+1 964 123 9999}
```

3. Deletes the instance of the **telephoneNumber** attribute assigned the value +1 964 123 9999:

```
modify {cn=miller,ou=sales,o=ins CO,c=us} \
     -removeattr {telephoneNumber=+1 964 123 9999}
```

In the following sample commands, the **modify** command performs the following tasks on the entry represented by the distinguished name

1. Adds two telephone numbers to the object represented by the distinguished name **cn=gunther,ou=sales,o=acme,c=us**:

```
modify cn=gunther,ou=sales,o=acme,c=us \
     -addattr {telephoneNumber=+1 919 555 4545;+1 431 223 4457}
```

2. Changes the initial values of both telephone numbers:

```
modify cn=gunther,ou=sales,o=acme,c=us \
     -changeattr {telephoneNumber=+1 919 555 4545} \
     {telephoneNumber=+1 508 693 9130} \
     -changeattr {telephoneNumber=+1 431 223 4457} \
     {telephoneNumber=+1 508 477 7300}
```

obj nextpage

Returns the next page of a search result if Simple Paging or Virual List View (VLV) is used. The syntax is as follows:

[**obj**] **nextpage**
[**-bindid** *bid*]
[**-conn** *connection_handle*]
[**-exactaction**]
[**-pretty**]
[**-result** *result_handle*]
[**-terminate**]
[**-vafter** *after_count*]
[**-vbefore** *before_count*]
[**-vcontentcount** *content_count*]
[**-voffset** *offset*]
[**-vpagesize** *page_size*]
[**-vvalue** *attribute_value*]

Options

***-bindid* bid**

The name (*bid*) of the bind to be used. If this option is omitted, the default bind is used.

(See the Bind Types and Bind IDs section in this chapter for details.)

-*conn* connection_handle

A handle to the connected directory to search. Specify the name of a connection handle created by a **meta openconn** operation. This is a required option when using this operation to search a connected directory. The connection handle must specify a connected directory of type **LDAP** (a connected directory opened with the **meta openconn -type** option **LDAP**) and it must not specify a format of **LDIF-CHANGE** (a connected directory opened with the **meta openconn -format** option **LDIF-CHANGE**).

-exactaction

Usually distinguished names are defined as CaseIgnore strings that implies that leading and trailing spaces are ignored and several spaces are reduced to one space when comparing distinguished names. There are no problems when modifying such an object and a reduced number of spaces is passed in the DN; the objects are still considered to be the same and the modify operation is performed successfully.

There are directory systems that want to have the exact number of spaces when calling the modify operation. Therefore multiple spaces must not be dropped.

Using the option **-exactaction** guarantees that multiple spaces in DNs of the search result are not dropped. The DNs are provided the same way as they are retrieved from the directory server.

-pretty

Displays the results of the operation in a tabular format.

-*result* result_handle

A name to assign to the result handle returned by the operation.

-terminate

Notifies the directory server that that search result can be discarded because no more pages will be requested by the client any more.

-*vafter* after_count

Specifies the maximum number of objects in the page that should be returned after the target object.

-*vbefore* before_count

Specifies the maximum number of objects in the page that should be returned before the target object.

-*vcontentcount* content_count

Together with **-voffset -vcontentcount** is used to specify the target object in a virtual list view in terms of an offset relative to the beginning of the list and the expected number of entries of the list.

-*voffset* offset

Specifies the target object in a virtual list view in terms of an offset relative to the

beginning of the list.

-*vpagesize* page_size

Specifies the maximum number of objects that should be returned in a page. This option is required for Simple Paging.

-*vvalue* attribute_value

Specifies the way the target object in the virtual list view is calculated. The target object is defined to be first object in the list where the attribute identified by the first attribute type of the sort keys is greater than or equal to *attribute_value*.

In order to avoid huge search results you can either use the **Simple Paging** or the **Virtual List View** (**VLV**) mechanism. **VLV** allows a client to specify that the server returns a contiguous subset of the search result for a given LDAP search with associated sort keys. The subset is defined in terms of offsets into the ordered list or in terms of a greater than or equal comparison value.

obj nextpage is used to return the next page of a search result once the initial search with paging has already been performed.

By default, the **obj nextpage** operation returns results as Tcl lists. Use the **-pretty** option to return formatted results.

When searching a connected directory the operation creates a result handle associated with the results list and assigns the name specified in the **-result** option to the handle. If the **-result** option is not specified the results list is output to the display as a Tcl list (unless the **-pretty** option has been specified). The **-result** option cannot be combined with the **-pretty** option.

If the **-result** option is specified the **-conn** option is required.

Supplying the **-conn** option causes the **obj nextpage** operation to return in the search results list the set of attributes to be synchronized (SSA) specified in the connection handle.

Use the option **-vpagesize** to specify the maximum number of objects that should be returned in a page.

The remaining options **-vbefore**, **-vafter**, **-vvalue**, **-voffset**, and **-vcontentcount** are only relevant for VLV.

The options **-vbefore** and **-vafter** are required if **-vpagesize** is absent. The option **-vbefore** specifies the maximum number of objects before a target object in the page and the option **-vafter** specifies the maximum number of objects after that object in the page.

If the options **-vbefore** and **-vafter** are used additionally one of the following options must be used:

-vvalue *attribute_value*

or

-voffset *offset* [**-vcontentcount** *content_count*]

Use **-vvalue** option if the target object is defined to be the first object in the list where the attribute type is greater or equal to *attribute_value*. The attribute type is taken from the first element of the sort keys. The **-vvalue** option can be used too if **-vpagesize** option is set (instead of **-vbefore** and **-vafter**).

Alternatively, the parameters **-voffset** and optionally **-vcontentcount** can be used for specifying a target object that should be contained in the page. If **-vcontentcount** is absent then its default value **0** is used. Because an LDAP server may not have an accurate estimate of the number of entries in the list and to take into account the cases where the list size changes during the time the user browses through the list, and because the client needs a way to indicate specific list targets beginning and end, offsets within the list are transmitted between the client and the LDAP server as ratios, offset to content count. The LDAP server sends its latest estimate of the number of entries in the list (content count) to the client. Using **-vcontentcount metacp** sends its assumed value of the content count to the server. The LDAP server examines the content count and offset that it receives and computes the corresponding offset within the list, based on its own idea of the content count.

Example:

> *offset* = 60, *content_count* = 100
>
> Suppose that the LDAP servers view of the complete search result conatians 300 objects, then the target object is at position 180.

There are the following special cases when using the **-voffset** and **-vcontentcount** option:

- *offset* is one and *content_count* is non-one: The target object is the first entry in the list.
- Equivalent values of *offset* and *content_count*: The target object is the last entry in the list.

Use the option **-terminate** if you are no longer receiving search pages from the server. That option cannot be combined with any other option. The result of this command is that the LDAP server discards the search result.

1. If the parameters **-conn** and **-result** are not used and the number of result entries is greater than the page size (specified in **-vpagesize**), then the search returns an indication for an incomplete operation as the last element of the search result. For Simple Paging, this is a query reference (cookie) that internally is used for subsequent calls of **obj nextpage**. For VLV, additionally Target-Position and Content-Count are provided. When processing the real entries of the search result an application (TCL script) must take care of that last element and should ignore it.

   Simple Paging:

```
metacp> nextpage -p -vpagesize 3
```

The sample output is as follows:

```
1) o=PQR
2) cn=admin,o=PQR
3) ou=Sales,o=PQR
Incomplete operation:
    Partial-Outcome-Qualifier
        Query-Reference         : \x0b\x00\x00\x00
```

VLV:

```
metacp> nextpage -p -vpagesize 3 -voffset 1
```

The sample output is as follows:

```
1) cn=Abele,ou=Sales,o=PQR
2) cn=admin,o=PQR
3) cn=Tinker,ou=Sales,o=PQR
Incomplete operation:
    Partial-Outcome-Qualifier
        Query-Reference         : \x0c\x00\x00\x00
        Virtual-List-View-Spec.
              Target-Position       : 1
              Content-Count         : 18
```

2. If the parameters **-conn** and **-result** are used then the “incomplete operation” indication is not returned in the search result; it is already dropped.
3. Only one search with simple paging or VLV can be done for a bind connection because the cookie is internally stored for each connection. If another search (with simple paging, VLV, or without paging) is required simultaneously an additional bind connection for that additional search is required.
4. If the last page has been received when using simple paging and you keep on calling the **obj nextpage** operation then an appropriate error code is returned that no more pages are available. (METACP 4515)

Example

1. Terminate paging:

```
nextpage -terminate
```

2. Simple paging with page size 3:

```
nextpage -pretty -vpagesize 3
```

3. VLV with page size 3 starting with the first entry:

```
nextpage -pretty -vpagesize 3 -voffset 1
```

4. VLV with page size 3 and the common name of the first object displayed starts with a **D**:

```
nextpage -pretty -vvalue D -vpagesize 3
```

5. VLV, the common name of the target object starts with a **D**. Three objects before and five objects after this object should be displayed:

```
nextpage -pretty -vvalue D -vbefore 3 -vafter 5
```

obj operations

Returns a list of operations that can be performed on the **obj** object. The syntax is as follows:

[**obj**] **operations**

The list of available operations is in alphabetical order except for **help** and **operations**, which are listed last.

Example

```
obj operations
```

The output of the sample command is as follows:

bind compare create delete list modify moddn search show unbind help operations

obj search

Searches for entries. The syntax is as follows:

[**obj**] **search** *distinguished_name*
[**-bindid** *bid*]
[**-conn** *connection_handle*]
[**-exactaction**]
[**-result** *result_handle*]
[**-allattr** |
-alluserattr |
-attribute *abbreviation* ...]
{**-baseobject** |
-onelevel |
-subtree}
[**-filter** *filter*]
[**-matchedvaluesonly**]
[**-pretty**]
[**-types**]
[**-vafter** *after_count*]
[**-vbefore** *before_count*]
[**-vcontentcount** *content_count*]
[**-voffset** *offset*]
[**-vpagesize** *page_size*]
[**-vsortkey** *sort_keys*]
[**-vtype** {**SUBENTRY** | **SIMPLE** | **VLV**}]
[**-vvalue** *attribute_value*]

Options

-*bindid* bid

The name (*bid*) of the bind to be used. If this option is omitted, the default bind is used. (See the Bind Types and Bind IDs section in this chapter for details.)

-*conn* connection_handle

A handle to the connected directory to search. Specify the name of a connection handle created by a **meta openconn** operation. This is a required option when using this operation to search a connected directory. The connection handle must specify a connected directory of type **LDAP** (a connected directory opened with the **meta openconn -type** option **LDAP**) and it must not specify a format of **LDIF-CHANGE** (a connected directory opened with the **meta openconn -format** option **LDIF-CHANGE**).

Usually all attributes that are defined in the connection handle are passed in the search operation as list of requested attributes. As this list may be very huge (namely when the **meta openconn** operation uses the **-processallattr** option) it makes sense to use the **-allattr** option instead. Therefore the Tcl variable **_md_req_attr_limit** (defined in **init.metacp** with the default value **64**) is used. If the attribute list of the connection handle is beyond that limit then for optimization purposes a search request with the **-allattr** option is generated internally. In that case metacp makes sure that later on in

the synchronization cycle, only the relevant attributes are processed; attributes returned by search but which are not defined in the connection handle are ignored. But you should be aware that operational attributes are not returned with the **-allattr** option. Futhermore the directory server may return an attribute with an alternate LDAP attribute name (if configured). So therefore if you miss an attribute during synchronization keep in mind to change **_md_req_attr_limit** to **-1** so that the list of requested attributes is passed to the server as is.

-exactaction

Usually distinguished names are defined as CaseIgnore strings that implies that leading and trailing spaces are ignored and several spaces are reduced to one space when comparing distinguished names. There are no problems when modifying such an object and a reduced number of spaces is passed in the DN; the objects are still considered to be the same and the modify operation is performed successfully.

There are directory systems that want to have the exact number of spaces when calling the modify operation. Therefore multiple spaces must not be dropped.

Using the option **-exactaction** guarantees that multiple spaces in DNs of the search result are not dropped. The DNs are provided the same way as they are retrieved from the directory server.

-result **result_handle**

A name to assign to the result handle returned by the operation.

-allattr

Shows information about all the attributes of an entry.

-alluserattr

Returns the object's user attributes. For LDAP binds, this option returns the same attributes as the option **-allattr**.

-attribute **abbreviation**

Returns the attributes that correspond to the specified LDAP names or OIDs.

-baseobject

Limits the search scope to the base object; that is, the entry represented by the specified distinguished name.

-filter **filter**

Specifies the filter condition for the search. See the String Representation for LDAP Binds chapter for a description of search filters.

-matchedvaluesonly

Specifies that attribute values not matched by the filter are not to be returned.

-onelevel

Limits the search scope to the children of the base object.

-pretty

Displays the results of the operation in a tabular format.

-subtree

Limits the search scope to the subtree below the base object.

-types

Specifies that the results are to contain attribute types but not attribute values.

-vafter **after_count**

Specifies the maximum number of objects in the page that should be returned after the target object.

-vbefore **before_count**

Specifies the maximum number of objects in the page that should be returned before the target object.

-vcontentcount **content_count**

Together with **-voffset -vcontentcount** is used to specify the target object in a virtual list view in terms of an offset relative to the beginning of the list and the expected number of entries of the list.

-voffset **offset**

Specifies the target object in a virtual list view in terms of an offset relative to the beginning of the list.

-vpagesize **page_size**

Specifies the maximum number of objects that should be returned in a page. This option is required for Simple Paging.

-vsortkey **sort_keys**

Specifies a list of attribute types (sort keys) that should be used for paging or virtual list view. This option is required for VLV. Specify *sort_keys* in the following format:

- For simple pageing
  -vsortkey *attr_type* [*attr_type* ...]
- For VLV:
  -vsortkey *attr_type attr_type* ...]

where *attr_type* is the LDAP name of the attribute. For simple paging the first *attr_type* can be prefixed by an exclamation mark (**!**) to specify reverse sort order. For VLV all *attr_type*s can be prefixed by an exclamation mark (**!**). For *attr_type* only indexed attribute types can be specified.

-vtype {SUBENTRY | SIMPLE | VLV}

Specifies the result type. Specify one of the following keywords:

- **SUBENTRY** - Specifies that the search operation returns subentries. This option is only supported for LDAP binds based on DirX Directory.

- **SIMPLE** - Specifies that the search operation uses simple paging mechanism. This option is only supported for LDAP binds not based on DirX Directory.
- **VLV** - Specifies that the search operation uses virtual list view mechanism.

-*vvalue* attribute_value

Specifies how the target object in the virtual list view is calculated. The target object is defined to be first object in the list where the attribute identified by the first attribute type of the sort keys is greater than or equal to *attribute_value*.

The **search** operation searches for entries starting from the specified distinguished name. You must specify one of the search scope options * -baseobject*, **-onelevel**, or **-subtree**.

By default, the operation does not search for attribute information. Use the * -attribute* or **-alluserattr** options to return selected attribute information, or use the **-allattr** option to return information about all attributes.

By default, the **search** operation returns results as Tcl lists. Use the **-pretty** option to return formatted results.

When searching a connected directory the operation creates a result handle associated with the results list and assigns the name specified in the **-result** option to the handle. If the **-result** option is not specified the results list is output to the display as a Tcl list (unless the **-pretty** option has been specified). The **-result** option cannot be combined with the **-pretty** option.

If the **-result** option is specified the **-conn** option is required. If the **-conn** option is specified the **obj search** attribute selection options (**-allattr**, **-alluserattr**, **-attribute**) cannot be specified.

If the -**result** option is specified then the search operation usually expects a complete result returned by the directory server (unless paging or virtual list view mode is turned on). If the directory server returns an incomplete result the synchronization TCL script's behavior is not predictable; for example due to missing entries in the search result, objects are deleted from the directory database. The directory server however may return incomplete results if a size limit or time limit has been encountered (see **ldapargs** command for details) or if referrals to other servers are returned. In the event of an incomplete result the search operation returns an error "Incomplete search result returned." (errCode: METACP 6142). If you want to process such incomplete results you must specify the TCL variable **_allowpartialresult** to prevent the search operation from returning an error. Then the TCL variable **_partialresulttype** is returned and specifies the reason why the partial result has been received. (See the introduction to **metacp** for details about the TCL variables **_allowpartialresult** and **_partialresulttype**.)

Supplying the **-conn** option causes the **obj search** operation to return in the search results list the set of attributes to be synchronized (SSA) specified in the connection handle. To prevent **obj search** from returning SSA attributes you must create a connection handle that specifies only **DDN** in the **-attribute** option, and specify this handle in the **-conn** option. The **obj search** operation then returns only entry names. In this case, you must maintain two connection handles for the connected directory: one for the directory synchronization process, and one for returning entry names without attributes in **obj search** operations.

In order to avoid huge search results you can either use the **Simple Paging** or the **Virtual List View** (**VLV**) mechanism. **VLV** allows a client to specify that the server returns a contiguous subset of the search result for a given LDAP search with associated sort keys. The subset is defined in terms of offsets into the ordered list or in terms or in terms of a greater than or equal comparison value.

The following sections describe how to work with these two mechanisms:

Use the option **-vpagesize** to specify the maximum number of objects that should be returned in a page. (If **-vpagesize** is not used for VLV, alternatively **-vbefore** and **-vafter** must be used.)

Use the option **-vsortkey** to specify the sort criterias.

The remaining options **-vbefore**, **-vafter**, **-vvalue**, **-voffset**, and **-vcontentcount** are only relevant for VLV.

The options **-vbefore** and **-vafter** are required if **-vpagesize** is absent. The option **-vbefore** specifies the maximum number of objects before a target object in the page and the option **-vafter** specifies the maximum number of objects after that object in the page.

If the options **-vbefore** and **-vafter** are used additionally one of the following options must be used:

-vvalue *attribute_value*

or

-voffset *offset* [**-vcontentcount** *content_count*]

Use **-vvalue** option if the target object is defined to be the first object in the list where the attribute type is greater or equal to *attribute_value*. The attribute type is taken from the first element of the sort keys. The **-vvalue** option can be used too if **-vpagesize** option is set (instead of **-vbefore** and **-vafter**).

Alternatively, the parameters **-voffset** and optionally **-vcontentcount** can be used for specifying a target object that should be contained in the page. If **-vcontentcount** is absent then its default value **0** is used. Because an LDAP server may not have an accurate estimate of the number of entries in the list, and to take into account the cases where the list size changes during the time the user browses through the list, and because the client needs a way to indicate specific list targets beginning and end, offsets within the list are transmitted between the client and the LDAP server as ratios, offset to content count. The LDAP server sends its latest estimate of the number of entries in the list (content count) to the client. Using **-vcontentcount metacp** sends its assumed value of the content count to the server. The LDAP server examines the content count and offset that it receives and computes the corresponding offset within the list, based on its own idea of the content count.

Example:

offset = 60, *content_count* = 100

Suppose that the LDAP servers view of the complete search result conatians 300 objects, then the target object is at position 180.

There are the following special cases when using the **-voffset** and **-vcontentcount** option:

- *offset* is one and *content_count* is non-one: The target object is the first entry in the list.
- Equivalent values of *offset* and *content_count*: The target object is the last entry in the list.

1. If the parameters **-conn** and **-result** are not used and the number of result entries is greater than the pagesize (specified in **-vpagesize**), then the search returns an indication for an incomplete operation as the last element of the search result. For Simple Paging, this is a query reference (cookie) that internally is used for subsequent calls of **obj nextpage**. For VLV, additionally Target-Position and Content-Count are provided. When processing the real entries of the search result an application (TCL script) must take care of that last element and should ignore it.

   Simple Paging:

   ```
   metacp> search o=pqr -subtree -pretty -vtype SIMPLE
   -vpagesize 3
   ```

   The sample output is as follows:

   ```
   1) o=PQR
   2) cn=admin,o=PQR
   3) ou=Sales,o=PQR
   Incomplete operation:
       Partial-Outcome-Qualifier
           Query-Reference          : \x0b\x00\x00\x00
   ```

VLV:

```
metacp> search o=pqr -subtree -pretty -vtype VLV
-vpagesize 3 -vsortkey cn -voffset 1
```

The sample output is as follows:

```
1) cn=Abele,ou=Sales,o=PQR
2) cn=admin,o=PQR
```

```
3) cn=Tinker,ou=Sales,o=PQR
Incomplete operation:
    Partial-Outcome-Qualifier
        Query-Reference         : \x0c\x00\x00\x00
        Virtual-List-View-Spec.
              Target-Position       : 1
              Content-Count         : 18
```

1. If the parameters **-conn** and **-result** are used then the “incomplete operation” indication is not returned in the search result; it is already dropped.
2. Only one search with simple paging or VLV can be done for a bind connection, because the cookie is internally stored for each connection. If another search (with simple paging, VLV, or without paging) is required simultaneously an additional bind connection for that additional search is required.

Examples

1. Search without paging

```
obj search o=pqr -subtree -conn ch -result LDAP_rh
```

2. Search with simple paging with page size 3:

```
search o=pqr -subtree -pretty -vtype SIMPLE -vpagesize 3
```

3. Search with VLV with page size 3 starting with the first entry:

```
search o=pqr -subtree -pretty -vtype VLV -vpagesize 3 -vsortkey cn
-voffset 1
```

4. Search with VLV with page size 3 starting with the first entry that common name starts with the letter **D**:

```
search o=pqr -subtree -pretty -vtype VLV -vsortkey cn -vvalue D
-vpagesize 3
```

5. Search with VLV with starting with the first entry (target object) that common name starts with the letter **D**, displaying 3 entries before and 5 entries after the target object:

```
search o=pqr -subtree -pretty -vtype VLV -vsortkey cn -vvalue D
-vbefore 3 -vafter 5
```

obj show

Shows an entry's contents. The syntax is as follows:

[**obj**] **show** *distinguished_name*
[**-bindid** *bid*]
[**-allattr** |
-alluserattr |
-attribute *abbreviation* ...]
[**-pretty**]
[**-types**]

Options

-bindid **bid**

The name (*bid*) of the bind to be used. If this option is omitted, the default bind is used. (See the Bind Types and Bind IDs section in this chapter for details.)

-allattr

Shows information about all the attributes of an entry.

-alluserattr

Returns the object's user attributes. For LDAP binds, this option returns the same attributes as the option **-allattr**.

-attribute **abbreviation**

Returns the attributes that correspond to the specified LDAP names or OIDs.

-pretty

Displays the results of the operation in a tabular format.

-types

Specifies that the results are to contain attribute types, but not attribute values.

Use the *distinguished_name* argument to specify an object to show.

By default, the operation does not return attribute information. Use the **-attribute** or **-alluserattr** options to return selected attribute information, or use the **-allattr** option to return information about all attributes. If you have selected to return attribute information, the **obj show** operation returns attribute types and values by default. Use the **-types** option to limit the results returned to attribute types only.

By default, the results of the **obj show** operation are displayed as a Tcl list. Use the **-pretty** option to return the results in a tabular, more readable format.

Examples

The following sample command shows all attributes associated with the object represented by the distinguished name **cn=Miller,ou=engeneering,o=myCompany,c=de**.

```
show {cn=Miller,ou=engeneering,o=myCompany,c=de} \
    -allattr \
      -pretty
```

The output of the sample command is as follows:

```
1) cn=Miller,ou=engeneering,o=myCompany,c=de
objectClass     : top
: person
: organizationalPerson
commonName : Miller
surname           : Tom
description       : Software-Engineer
```

The following sample command displays a Tcl list that contains all attributes associated with the object represented by the distinguished name **cn=Miller,ou=engeneering,o=myCompany,c=de**.

```
show {cn=Miller,ou=engeneering,o=myCompany,c=de}
     -allattr
```

The output of the sample command is as follows:

```
cn=Miller,ou=engeneering,o=myCompany,c=de
{objectClass=top;person;organizationalPerson} cn=Miller sn=Tom
description=Software-Engineer
```

The following sample command displays the telephone-number attribute associated with the object represented by the distinguished name **cn=Miller,ou=engeneering,o=myCompany,c=de**.

```
show {cn=Miller,ou=engeneering,o=myCompany,c=de}
     -attribute TN -pretty
```

The output of the sample command is as follows:

```
1) cn=Miller,ou=engeneering,o=myCompany,c=de
telephoneNumber : +1 964 123 5678
```

obj unbind

Terminates a binding between **metacp** and a Identity store. The syntax is as follows:

[**obj**] **unbind**
  [**-bindid** *bid*]

Options

-bindid **bid**

The name (*bid*) of the bind to be used. If this option is omitted, the default bind is used. See the Bind Types and Bind IDs section in this chapter for details.

1.1.5. util (metacp)

Synopsis

delete_subtree *base_object*

faddattr *filename*

fcreate *filename*

Purpose

Utilities for **metacp**.

Arguments

base_object

The distinguished name of an entry in the DIT. See the String Representation for LDAP Binds chapter for a description of distinguished name syntax.

filename

The name of the data file.

Operations

delete_subtree

Deletes the subtree under the specified base object. The base object itself is also deleted.

This utility can only be performed for a default bind id. (See the obj bind operation for details.) The syntax is as follows:

delete_subtree *base_object*

Example

```
delete_subtree {ou=sales,o=pqr,c=de}
```

faddattr

Adds attributes and attribute values to existing entries in the DIT in batch mode. The syntax is as follows:

faddattr *filename*

where *filename* is a data file that contains the names and attributes of the entries in the following format:

- Blank lines and lines starting with the # character are ignored.
- All information for an entry must be contained in a single line. Each line is divided into fields separated by the SPACE character. If the SPACE character is part of an attribute, the field should be enclosed in curly braces ({ }).
- The first field in a line contains the distinguished name of the entry and subsequent fields contain attribute information. If recurring values are specified for an attribute, the values must be separated by a semicolon (;) and the field must be enclosed in curly braces \{ }).

Here are some examples of the file format:

```
# Add description
{cn=Smith John,ou=sales,o=pqr,c=de} {description=Sales Manager}
# Add telephone number
cn=Mayer,ou=sales,o=pqr,c=de {telephoneNumber=+49(89)235-42356}
```

Example

```
faddattr datafile
```

fcreate

Creates entries in the DIT in a batch mode. The syntax is as follows:

fcreate *filename*

where *filename* is a data file that contains the names and attributes of the entries in the following format:

- Blank lines and lines starting with the # character are ignored.
- All information for an entry must be contained in a single line. Each line is divided into fields separated by the SPACE character. If the SPACE character is part of an attribute, the field should be enclosed in curly braces (\{ }).
- The first field in a line contains the distinguished name of the entry and subsequent fields contain attribute information. If recurring values are specified for an attribute, the values must be separated by a semicolon (;) and the field must be enclosed in curly braces (\{ }).

Here are some examples of the file format:

```
# Organizational Unit
ou=Services,o=pqr,c=de objectClass=organizationalUnit
# Organizational Persons
{cn=Smith James,ou=Services,o=pqr,c=de} \
{objectClass=organizationalPerson;person;mhsUser} sn=Smith \
telephoneNumber=+49(89)123-456 {facsimileTelephoneNumber=+49(89)123-
789} \
{postalAddress=Services Dpt$Einstein-Ring 4$D-81789 Munich,$Germany}
\
{mhsOraddresses=/G=James/S=Smith/OU2=S41/OU1=MCH1/PRMD=PQR/ADMD=DBP/C
=DE}
cn=Mayer,ou=services,o=pqr,c=de \
{objectClass=organizationalPerson;person;mhsUser} sn==Mayer \
telephoneNumber=+49(89)123-567 {facsimileTelephoneNumber=+49(89)123-
789} \
{postalAddress =Services Dpt$Einstein-Ring 4$D-81789 Munich$Germany}
\
{mhsOraddresses=/G=Erna/S=Mayer/OU2=S41/OU1=MCH1/PRMD=PQR/ADMD=DBP/C=
DE}
cn=Richter,ou=services,o=pqr,c=de \
{objectClass=organizationalPerson;person;mhsUser} sn=Richter \
telephoneNumbe=+49(89)234-678 {facsimileTelephoneNumber=+49(89)234-
6789} \
{postalAddress =Services Dpt$Albert-Ring 6$D-81789 Munich$Germany} \
{mhsOraddresses=/G=Franz/S=Richter/OU2=S12/OU1=MCH1/PRMD=PQR/ADMD=DBP
/C=DE}
```

Example

```
fcreate datafile
```

1.2. metacpdump

Synopsis

metacpdump [**-a**]
[**-c** [**!**] *ComponentNameList*]
[**-h**]
[**-l** *NoOfLookAheadEntries*]
[**-m**]
[**-p**]
[**-s** [**!**] *SubComponentNameList*]
[**-t** [**!**] *ThreadIdentificationList*]
ClientLogFileName ...

Purpose

Displays the contents of a binary DirX Identity client trace log file that are written by metacp.

Arguments

ClientLogFileName

A string that represents the name of a serviceability log file on which to operate.

Options

-a

Suppresses display of additional information available in a directory log file entry, for example, the contents of structured function arguments. (To enable the logging of additional information, set debug level **9** for one or more components in the routing specification file.)

***-c [!]* ComponentNameList**

If the ! character is specified, displays the directory log file entries that are not associated with the components specified in *ComponentNameList*, otherwise, displays the directory log file entries associated with the components in *ComponentNameList*. *ComponentNameList* is a string that contains one or more component names associated with the log file entries. All directory log file entries are currently associated with the **dir** component name. All DirX Identity log file entries are currently associated with the **mdi** component name.

-h

Prints a command usage message.

***-l* NoOfLookAheadEntries**

Is an integer that represents the number of log file entries to look ahead when searching for function entry/exit log file pairs. Merging between a function entry/exit log file entry is not performed if the corresponding entry cannot be found in the specified range. If this option is not specified, **metacpdump** uses the value 512 as its search range.

-m

Suppresses merging of function entry log file entries and function exit log file entries. Invalid or non-existent function input/output and result parameters of a log file entry are represented as a **???** sequence. If this option is specified, **metacpdump** ignores the **-l** option.

-p

Suppresses display of the prolog information for each log file entry.

***-s [!]* SubComponentNameList**

If the **!** character is specified, displays the directory log file entries that are not associated with the subcomponents specified in *SubComponentNameList*, otherwise, displays the log file entries associated with the subcomponents in *SubComponentNameList*. *SubComponentNameList* is a string that contains a single subcomponent name or a comma-separated list of subcomponent names associated with the log file entries.

The **metacpdump** command recognizes the following directory service subcomponent names for the component **mdi**:

Keyword	Meaning
meta	DirX Identity client interface

The **metacpdump** command recognizes the following directory service subcomponent names for the component **dir**:

Keyword	Meaning
adm	Administration
api	Application interface
bth	Bind table handling
icom	Internal thread communication interface
osi	OSI communication
ros	Remote operation service
rpc	RPC interface
sock	Socket interface
sys	System call interface

Keyword	Meaning
util	Utility functions
vthr	Virtual thread interface

***-t [!]* ThreadIdentificationList**

If the **!** character is specified, displays the log file entries not associated with the threads specified in *ThreadIdentificationList*. Otherwise, displays log file entries associated with the threads specified in *ThreadIdentificationList*. *ThreadIdentificationList* is a single integer or a comma-separated list of integers that represent one or more thread Ids. Alternately, it is a single keyword or a comma-separated list of keywords representing thread types. The **metacpdump** command recognizes the following thread types:

Keyword	Meaning
"abort thread"	Handles aborted connections.
MainThread	Initial (main) thread
OsiThread	OSI communication handling thread
OsiIComThread	OSI communication handling thread (supports the primary OSI thread during internal event handling)

Description

The DirX Identity client (**metacp**) subcomponents log important information about their activities and state through an internal serviceability interface. You specify how client trace log messages are to be routed with the file *install_path***/client/conf/dirxlog.cfg**. Each log entry is written as a machine-independent binary record of data defined as the contents of a serviceability prolog structure. The **metacpdump** command displays the contents of this binary file in readable text format. The default location of the client trace log files is *install_path***/client/log**.

Example

The following example is based on a log file **LOG20861.01**. The logging level has been set to **1** for the **meta** subcomponents.

Create a readable log file **log.all** of the whole content:

```
metacpdump LOG20861.01 > log.all
```

The following output is written to the file **log.all**. (For readability a few lines were removed from the result):

```
------------------------------------------------------------------------
-----------
```

```
MainThread 0x00000001 METACP mdi Wed 06/14/00 15:25:15
------------------------------------------------------------------------
-----------
DEBUG1 meta gcimeta.c 1899 25:15:655[2:410]
0 md_t_initialize
(info_block : IN: NULL OUT: 0x2a51d0,
gc_t_in_vect : 0xefffc5a4,
local_strings : TRUE,
out_vect : efffc5b8) = SUCCESSFUL

 DEBUG1 meta gcimeta.c 598 25:18:510[5:461]
1 md_t_readattrconf
(info_block : 0x2a51d0,
gc_t_in_vect : 0xefffc5a4,
local_strings : TRUE,
out_vect : efffc5b8) = SUCCESSFUL
DEBUG1 meta mdt_readattr.c 267 25:19:196[4:705]
2 md_read_attr_config_file
(file_name: "./../../conf/ldifattr.cfg",
ctx_id: 0x2eaaa0,
attr_tab: 0x2eaaf8,
global_info:0x2eab00,
error_line: 2601975) = SUCCESSFUL
DEBUG1 meta mdGetAttrConf.c 803 25:19:735[450]
DEBUG1 meta mdGetAttrConf.c 831 25:22:663[067]
3 md_get_attr_info
(attr_tab: 0x2eaaf8,
abbr: "NEW_RDN") = 0x2ed45c
DEBUG1 meta mdGetAttrConf.c 838 25:22:816[124]
...
...
1 md_t_readattrconf
(info_block : 0x2a51d0,
gc_t_in_vect : 0xefffc5a4,
local_strings : TRUE,
out_vect : efffc5b8) = SUCCESSFUL
DEBUG1 meta mdt_readattr.c 267 25:24:273[1:965]
2 md_read_attr_config_file
(file_name: "./../../conf/x500attr.cfg",
ctx_id: 0x2eeaa8,
attr_tab: 0x2eeb00,
```

```
global_info:0x2eeb08,
error_line: 2601975) = SUCCESSFUL
DEBUG1 meta mdGetAttrConf.c 831 25:24:483[424]
3 md_get_attr_info
(attr_tab: 0x2eeb00,
abbr: "NEW_RDN") = 0x2f0974
DEBUG1 meta mdGetAttrConf.c 838 25:25:132[130]

...
...
4 md_t_supinfo
(info_block : 0x2a51d0,
gc_t_in_vect : 0xefffbdbc,
local_strings : TRUE,
out_vect : efffbdd0) = SUCCESSFUL
DEBUG1 meta gcimeta.c 1247 25:27:784[298]
1 md_t_openconn
(info_block : 0x2a51d0,
gc_t_in_vect : 0xefffc5a4,
local_strings : TRUE,
out_vect : efffc5b8) = SUCCESSFUL
DEBUG1 meta mdt_openconn.c 719 25:29:323[130]
2 md_get_attr_info
(attr_tab: 0x2eaaf8,
abbr: "AR") = 0x2ec74c
DEBUG1 meta mdt_openconn.c 719 25:29:545[168]

...
...
1 md_t_openconn
(info_block : 0x2a51d0,
gc_t_in_vect : 0xefffc5a4,
local_strings : TRUE,
out_vect : efffc5b8) = SUCCESSFUL
DEBUG1 meta mdt_openconn.c 854 25:35:344[032]
2 md_get_attr_info
(attr_tab: 0x2eeb00,
abbr: "DDN") = 0x2f0160

DEBUG1 meta mdt_openconn.c 854 25:35:473[083]
...
```

```
...
2 md_t_search
(info_block: 0x2a51d0,
session: 0x2c3188,
gc_t_in_vect: 0xefffc5a0,
abbrv_result: TRUE,
local_strings: TRUE,
out_vect: efffc5b8) = SUCCESSFUL
DEBUG1 meta gcimeta.c 670 25:39:791[165]
3 md_t_sortresult
(info_block : 0x2a51d0,
gc_t_in_vect : 0xefffc1b4,
local_strings : TRUE,
out_vect : efffc1c8) = SUCCESSFUL
DEBUG1 meta gcimeta.c 2192 25:40:067[1:672]
3 md_t_getentry
(info_block : 0x2a51d0,
gc_t_in_vect : 0xefffc1b4,
local_strings : TRUE,
out_vect : efffc1c8) = SUCCESSFUL
DEBUG1 meta mdt_getentry.c 1378 25:40:161[1:415]
4 md__getentry_generic
(info_block : 0x2a51d0,
source_handle : 0x2e23e8,
handle_name : "rh",
tag_val : "",
search_key : "",
count_matches : FALSE,
ret_tcl : "0",
out_vect : efffc1c8) = SUCCESSFUL
DEBUG1 meta mdt_getentry.c 407 25:40:379[900]
5 md__getrecord_from_handle
(info_block : 0x2a51d0,
source_handle : 0x2e23e8,
handle_name : "rh",
search_key : "",
count_matches : FALSE,
ret_tcl : "0",
out_vect : efffc1c8) = SUCCESSFUL
DEBUG1 meta mdGetEntry.c 1311 25:40:523[171]
6 md_read_entry
```

```
(fp: 0xeefa6ee8,
ctx_id: 0x315128,
global_info: 0x2eab00,
mem_len: 0,
entry: "dn: o=PQR
o: PQR
description: PQR Company
postalAddress: PQR Company$ABC Street 123$D-01234 City$Germany
telephoneNumber: +49 12 345 67 890
objectClass: organization
objectClass: top
createTimestamp: 20000308120944Z
") = SUCCESSFUL
DEBUG1 meta mdGetEntry.c 1343 25:40:976[177]
6 md_split_attr
(entry: "dn: o=PQR
o: PQR
description: PQR Company
postalAddress: PQR Company$ABC Street 123$D-01234 City$Germany
telephoneNumber: +49 12 345 67 890
objectClass: organization
objectClass: top
createTimestamp: 20000308120944Z
",
global_info: ",
attr_s"eq: 0x2f2d44,
is_tagged: TRUE,
attr_table: 0x2eaaf8,
ctx_id: 0x315128,
rec_info: 0x31518c,
last_byte: "") = SUCCESSFUL
DEBUG1 meta gcimeta.c 2046 25:41:860[096]
3 md_t_gethandle
(info_block : 0x2a51d0,
gc_t_in_vect : 0xefffc1b4,
local_strings : TRUE,
out_vect : efffc1c8) = SUCCESSFUL
DEBUG1 meta gcimeta.c 742 25:42:073[052]
3 md_t_findentry
(info_block : 0x2a51d0,
gc_t_in_vect : 0xefffc1b4,
```

```
local_strings : TRUE,
out_vect : efffc1c8) = SUCCESSFUL
DEBUG1 meta gcimeta.c 1682 25:42:336[1:095]
4 md_t_modifyentry
(info_block : 0x2a51d0,
session : 0x2c3188,
gc_t_in_vect : 0xefffc1b4,
local_strings : TRUE,
out_vect : efffc1c8) = SUCCESSFUL
DEBUG1 meta mdUtil.c 3586 25:42:430[179]
5 md_get_attr_info
(attr_tab: 0x2eeb00,
abbr: "DDN") = 0x2f0160

DEBUG1 meta mdt_modifyentry. 553 25:42:790[480]
5 md_x500_modify
(fp_trace: 0xeefa6ed8,
comment_sign: "#",
trace_level: 2,
entry: "dn: o=PQR
o: PQR
description: PQR Company
postalAddress: PQR Company$ABC Street 123$D-01234 City$Germany
telephoneNumber: +49 12 345 67 890
objectClass: organization
objectClass: top
createTimestamp: 20000308120944Z
",
x500_attrconf: 0x2eeb00,
flags: 0x30d17c,
rec_info: 0xefffc0bc,
x500_entry: 0x2a1740,
ctx_id: 0x2e4a88,
sup_info: 0x2e1bbc,
info_block: 0x2a51d0,
session: 0x2c3188,
local_strings: TRUE,
ignore_mod: FALSE,
allow_rename: FALSE,
stat_info: 0x2a51e8,
success: TRUE) = SUCCESSFUL
```

```
DEBUG1 meta gcimeta.c 2192 26:14:309[604]
3 md_t_getentry
(info_block : 0x2a51d0,
gc_t_in_vect : 0xefffc1b4,
local_strings : TRUE,
out_vect : efffc1c8) = SUCCESSFUL
DEBUG1 meta mdt_getentry.c 1378 26:14:438[389]
4 md__getentry_generic
(info_block : 0x2a51d0,
source_handle : 0x2e23e8,
handle_name : "rh",
tag_val : "",
search_key : "",
count_matches : FALSE,
ret_tcl : "1",
out_vect : efffc1c8) = SUCCESSFUL
DEBUG1 meta mdt_getentry.c 407 26:14:532[185]
5 md__getrecord_from_handle
(info_block : 0x2a51d0,
source_handle : 0x2e23e8,
handle_name : "rh",
search_key : "",
count_matches : FALSE,
ret_tcl : "1",
out_vect : efffc1c8) = SUCCESSFUL
DEBUG1 meta mdGetEntry.c 1311 26:14:621[068]
6 md_read_entry
(fp: 0xeefa6ee8,
ctx_id: 0x3247e8,
global_info: 0x2eab00,
mem_len: 0,
entry: "") = MD_EOF
DEBUG1 meta gcimeta.c 742 26:14:969[074]
7 md_t_findentry
(info_block : 0x2a51d0,
gc_t_in_vect : 0xefffbc3c,
local_strings : TRUE,
out_vect : efffbc50) = SUCCESSFUL
DEBUG1 meta gcimeta.c 1320 26:15:130[089]
4 md_t_removeentries
(info_block : 0x2a51d0,
```

```
session : 0x2c3188,
gc_t_in_vect : 0xefffc1b4,
local_strings : TRUE,
out_vect : efffc1c8) = SUCCESSFUL
DEBUG1 meta gcimeta.c 1827 26:15:273[049]
99 md_t_unset
(info_block : 0x2a51d0,
gc_t_in_vect : 0xefffcdcc,
local_strings : TRUE,
out_vect : efffcde0) = SUCCESSFUL
DEBUG1 meta gcimeta.c 1827 26:15:394[108]

...
...
DEBUG1 meta gcimeta.c 1974 26:16:985[224]
99 md_t_shutdown
(info_block : IN: 0x2a51d0 OUT: NULL,
gc_t_in_vect : 0xefffd21c,
local_strings : TRUE,
out_vect : efffd230) = SUCCESSFUL
```

Exit Codes

The **metacpdump** command returns an exit code of **0** on success or a **1** if it encountered an error. The text of the error message is displayed on **stderr**.

See Also

dirxlog.cfg (in DirX Identity Program Files)

1.3. metahubdump

Synopsis

metahubdump [**-a**]
[**-c** [**!**] *ComponentNameList*]
[**-h**]
[**-l** *NoOfLookAheadEntries*]
[**-m**]
[**-p**]
[**-s** [**!**] *SubComponentNameList*]
[**-t** [**!**] *ThreadIdentificationList*]
ServerLogFileName ...

Purpose

Displays the contents of a binary Server (IDS-C) trace log file. Server trace log files are written by the Server (IDS-C) (dxmmsssvr.exe and dxmsvr.exe).

Use **metacpdump** to display client log files that are written by metacp. The command **dxmdumplog** is deprecated because its behaviour depends on the environment variable **DIRX_SVC_EXTINFO**. Use the command **metahubdump** instead.

Refer to the DirX Directory documentation for preparation of log files written by DirX processes.

Arguments

ServerLogFileName

A string that represents the name of a serviceability log file on which to operate.

Options

-a

Suppresses display of additional information available in a directory log file entry, for example, the contents of structured function arguments. (To enable the logging of additional information, set debug level **9** for one or more components in the routing specification file.)

***-c [!]* ComponentNameList**

If the ! character is specified, displays the directory log file entries that are not associated with the components specified in *ComponentNameList*, otherwise, displays the directory log file entries associated with the components in *ComponentNameList*. *ComponentNameList* is a string that contains one or more component names associated with the log file entries. All directory log file entries are currently associated with the **dir** component name. All Server (IDS-C) log file entries are currently associated with the **dxm** component name.

-h

Prints a command usage message.

***-l* NoOfLookAheadEntries**

Is an integer that represents the number of log file entries to look ahead when searching for function entry/exit log file pairs. Merging between a function entry/exit log file entry is not performed if the corresponding entry cannot be found in the specified range. If this option is not specified, **metahubdump** uses the value 512 as its search range.

-m

Suppresses merging of function entry log file entries and function exit log file entries. Invalid or non-existent function input/output and result parameters of a log file entry are

represented as a **???** sequence. If this option is specified, **metahubdump** ignores the **-l** option.

-p

Suppresses display of the prolog information for each log file entry.

***-s [!]* SubComponentNameList**

If the **!** character is specified, displays the directory log file entries that are not associated with the subcomponents specified in *SubComponentNameList*, otherwise, displays the log file entries associated with the subcomponents in *SubComponentNameList*. *SubComponentNameList* is a string that contains a single subcomponent name or a comma-separated list of subcomponent names associated with the log file entries.

The **metahubdump** command recognizes the following directory service subcomponent names for the component **dxm:**

Keyword	Meaning
agtcl	Agent controller client interface (the interface that the workflow engine uses to start activities)
agtfe	Agent controller front end
agtwr	Agent controller back end
ats	Messaging interface
cdbrt	ConfDB API
ldap	LDAP interactions of ConfDB API with LDAP directory
mss	Server (IDS-C) core
sched	Scheduler
sttcl	Client interface to status tracker (used by agent controller front end and workflow engine for communication with the status tracker)
stt	Status tracker
wfc	Workflow client interface (used by the scheduler to start and control workflows)
wfe	Workflow engine
util	Server (IDS-C) internal utility functions

The **metahubdump** command recognizes the following directory service subcomponent names for the component **dir**:

Keyword	Meaning
icom	Internal thread communication interface
sys	System call interface
vthr	Virtual thread interface

-t *[!]* ThreadIdentificationList

If the **!** character is specified, displays the log file entries not associated with the threads specified in *ThreadIdentificationList*. Otherwise, displays log file entries associated with the threads specified in *ThreadIdentificationList*. *ThreadIdentificationList* is a single integer or a comma-separated list of integers that represent one or more thread Ids. Alternately, it is a single keyword or a comma-separated list of keywords representing thread types. The **metahubdump** command recognizes the following thread types:

Keyword	Meaning
"abort thread"	Handles aborted connections.
MainThread	Initial (main) thread

Description

The Server (IDS-C) (dxmmsssvr.exe and dxmsvr.exe) subcomponents log important information about their activities and state through an internal serviceability interface. You specify how server trace log messages are to be routed with the file *install_path***/server/conf/dirxlog.cfg**. Each log entry is written as a machine-independent binary record of data defined as the contents of a serviceability prolog structure. The **metahubdump** command displays the contents of this binary file in readable text format. The default location of the server trace log files is *install_path***/server/log**.

Example

The following example is based on a log file **LOG20861.01**. The logging level has been set to **1** for the **dxm** subcomponents util, agtwr, wfe.

Create a readable log file **log.all** of the whole content:

```
metahubdump LOG20861.01 > log.all
```

The following output is written to the file **log.all**. (For readability a few lines were removed from the result):

```
------------------------------------------------------------------------
-----------
MainThread 0x0000005a dxmmsssvr.exe dxm Tue 03/20/01 12:57:19
------------------------------------------------------------------------
-----------
-- NOTICE mss dxmmssmain.cpp 244 57:19:70
0 "Dir.X MetaHub Synchronization Server V 6.0A 00 Tue Mar 20 7:03:56
2001 build 18"

-- NOTICE_V mss dxmmssmain.cpp 273 57:19:710
```

```
0 "Cmd line:
Environment:
DIRXMETAHUB_INST_PATH=C:\Program Files\Atos/DirX Identity
DIRXMETAHUB_QUEUE_MGR=QM_kellner01.asw.mchp.owncompany.de
DIRX_LOGCFG_FILE=<not set>"

-- NOTICE util cbasics.cpp 1287 57:19:710
1 loading '"server\conf\dxmmsssvr.ini"' (mode "rt").

-- NOTICE ldap dxmldap.cpp 256 57:19:731
99 Successfully connected to LDAP server "localhost":1389.
-- NOTICE ldap dxmldap.cpp 327 57:19:811
99 Disconnected from LDAP server "localhost":1389.
-- NOTICE ldap dxmldap.cpp 256 57:19:831
99 Successfully connected to LDAP server "localhost":1389.
-- NOTICE mss dxmcmsssvr.cpp 1031 57:19:941
99 registered server "main" ("main", "127.0.0.1")

-- NOTICE ldap dxmldap.cpp 327 57:19:941
99 Disconnected from LDAP server "localhost":1389.
-- NOTICE mss dxmmsssession.cp 816 57:20:031
99 "durable" subscriber initialized (session id: "main", topic:
"Dxm.command.main").

-- NOTICE mss dxmmsssession.cp 816 57:20:111
99 "durable" subscriber initialized (session id: "main", topic:
"Dxm.statusTracker").

-- NOTICE util cthread.cpp 234 57:20:161
99 thread "w_ID:STATUSTRACKER" started.

------------------------------------------------------------------------
-----------
w_ID:STATUSTRACKER 0x00000141 dxmmsssvr.exe dxm Tue 03/20/01 12:57:20
------------------------------------------------------------------------
-----------
-- NOTICE mss dxmcmssworker.cp 132 57:20:161
0 "CmssWorker" started.

------------------------------------------------------------------------
-----------
MainThread 0x0000005a dxmmsssvr.exe dxm Tue 03/20/01 12:57:20
```

```
-------------------------------------------------------------------------
-----------
-- NOTICE util cthread.cpp 234 57:20:161
99 thread "w_ID:SCHEDULER" started.

-------------------------------------------------------------------------
-----------
w_ID:STATUSTRACKER 0x00000141 dxmmsssvr.exe dxm Tue 03/20/01 12:57:20
-------------------------------------------------------------------------
-----------
-- DEBUG1 util cmodule.cpp 191 57:20:171
1 dlopen("C:\Program Files\Atos/DirX Identity\bin\libdxmstt.dll") =
0x1c30000.

-------------------------------------------------------------------------
-----------
w_ID:SCHEDULER 0x000000cc dxmmsssvr.exe dxm Tue 03/20/01 12:57:20
-------------------------------------------------------------------------
-----------
-- NOTICE mss dxmcmssworker.cp 132 57:20:171
0 "CmssWorker" started.

-------------------------------------------------------------------------
-----------
MainThread 0x0000005a dxmmsssvr.exe dxm Tue 03/20/01 12:57:20
-------------------------------------------------------------------------
-----------
-- NOTICE util cthread.cpp 234 57:20:171
99 thread "cleaner thread" started.

-- NOTICE mss dxmcmsssvr.cpp 762 57:20:181
99 starting subscriptions ...

-------------------------------------------------------------------------
-----------
w_ID:SCHEDULER 0x000000cc dxmmsssvr.exe dxm Tue 03/20/01 12:57:20
-------------------------------------------------------------------------
-----------
-- DEBUG1 util cmodule.cpp 191 57:20:181
1 dlopen("C:\Program Files\Atos/DirX Identity\bin\libdxmsdr.dll") =
0x1d60000.
```

```
-- DEBUG1 sched dxmsdrimpl.cpp 782 57:20:181
2 Function parameter:
"InitiatorType:" 0
-------------------------------------------------------------------------
-----------
cleaner thread 0x000000bb dxmmsssvr.exe dxm Tue 03/20/01 12:57:20
-------------------------------------------------------------------------
-----------
-- NOTICE mss dxmcmsscleaner.c 177 57:20:191
0 "CmssCleaner" started.

-------------------------------------------------------------------------
-----------
w_ID:STATUSTRACKER 0x00000141 dxmmsssvr.exe dxm Tue 03/20/01 12:57:20
-------------------------------------------------------------------------
-----------
-- NOTICE ldap dxmldap.cpp 256 57:20:201
2 Successfully connected to LDAP server "localhost":1389.
-------------------------------------------------------------------------
-----------
w_ID:SCHEDULER 0x000000cc dxmmsssvr.exe dxm Tue 03/20/01 12:57:20
-------------------------------------------------------------------------
-----------
-- NOTICE ldap dxmldap.cpp 256 57:20:211
3 Successfully connected to LDAP server "localhost":1389.
-- NOTICE ldap dxmldap.cpp 327 57:20:241
3 Disconnected from LDAP server "localhost":1389.
-- NOTICE sched dxmsdrimpl.cpp 1721 57:20:241
3 New start time of status tracker cleanup computed.
Scheduled next at : "20010320120000Z"
Current Time : "20010320115720Z"
-- NOTICE sched dxmsdrimpl.cpp 1825 57:20:241
3 New start time for synchronizing schedules computed.
Scheduled next at : "20010321000000Z"
Current Time : "20010320115720Z"
-- NOTICE_V sched dxmsdrimpl.cpp 817 57:20:241
2 Scheduler is running.
-------------------------------------------------------------------------
-----------
SUB_7f2570 0x00000129 dxmmsssvr.exe dxm Tue 03/20/01 12:58:17
-------------------------------------------------------------------------
```

```
-----------
-- NOTICE util cthread.cpp 234 58:17:904
0 thread "w_c0671bb1-5ccfb1-e55be8dae0--8000" started.

-------------------------------------------------------------------------
-----------
w_c0671bb1-5ccfb1-e5 0x0000014f dxmmsssvr.exe dxm Tue 03/20/01
12:58:17
-------------------------------------------------------------------------
-----------
-- NOTICE mss dxmcmssworker.cp 132 58:17:904
0 "CmssWorker" started.

-- DEBUG1 util cmodule.cpp 191 58:17:914
1 dlopen("C:\Program Files\Atos/DirX Identity\bin\libdxmwfe.dll") =
0x1ee0000.

-- DEBUG1 wfe dxmwfeimpl.cpp 154 58:17:914
2 Function parameter:
"Workflow Name:"
"cn=BA_PABX2MetaStore_Full,dxmC=Workflows,dxmC=DirXmetahub"
-- DEBUG1 wfe dxmwfeimpl.cpp 155 58:17:914
2 Function parameter:
"InitiatorType:" 1
-- DEBUG1 wfe dxmwfeimpl.cpp 171 58:17:914
2 Function parameter:
"Workflow Start Time defaults to:" "20010320115817Z"
-- NOTICE ldap dxmldap.cpp 256 58:17:934
3 Successfully connected to LDAP server "localhost":1389.
-- NOTICE ldap dxmldap.cpp 327 58:18:064
3 Disconnected from LDAP server "localhost":1389.
-- DEBUG1 wfe dxmwfeimpl.cpp 220 58:18:064
2 Function parameter:
"Polling Time (in ms):" 120000
-- DEBUG1 wfe dxmwfeimpl.cpp 273 58:18:064
2 Constructor for "CWorkflowStub": setting return value 0
("SUCCESSFUL") for acknowledge.
-- DEBUG1 wfe dxmwfeimpl.cpp 299 58:18:064
2 Constructor of class "CWorkflowStub" called,
address of new object is 0x7f7720,
Instance ID is "c0671bb1-5ccfb1-e55be8dae0--8000".
```

```
-- DEBUG1 wfe dxmwfeimpl.cpp 358 58:19:016
2 ENTRY Object (0x7f7720), Function"CworkflowStub::execute".
-- NOTICE wfe dxmwfeimpl.cpp 360 58:19:016
2 START Workflow
Workflow instance ID: "c0671bb1-5ccfb1-e55be8dae0--8000"
Workflow DN:
"cn=BA_PABX2MetaStore_Full,dxmC=Workflows,dxmC=DirXmetahub"
Workflow Name: "BA_PABX2MetaStore_Full"
Workflow Start Time: "20010320115817Z"
-- DEBUG1 wfe dxmwfeimpl.cpp 376 58:19:026
2              Workflow instance ID: "c0671bb1-5ccfb1-e55be8dae0--
8000"
Workflow DN:
"cn=BA_PABX2MetaStore_Full,dxmC=Workflows,dxmC=DirXmetahub"
Workflow Name: "BA_PABX2MetaStore_Full"
Workflow Start Time: "20010320115817Z"
WorkflowState: "open.notRunning.notStarted"
-- DEBUG1 wfe dxmwfeimpl.cpp 401 58:19:036
2 Function parameter:
"Workflow Name:"
"cn=BA_PABX2MetaStore_Full,dxmC=Workflows,dxmC=DirXmetahub"
-- DEBUG1 wfe dxmwfeimpl.cpp 403 58:19:036
2 Function parameter:
"Workflow Activity Context:" "BA_PABX2MetaStore_Full"
-- DEBUG1 wfe dxmwfeimpl.cpp 404 58:19:036
2 Function parameter:
"Operation:" 10
-------------------------------------------------------------------
-----------
SUB_7f2570 0x00000129 dxmmsssvr.exe dxm Tue 03/20/01 12:58:19
-------------------------------------------------------------------
-----------
-- NOTICE util cthread.cpp 234 58:19:066
0 thread "w_127.0.0.1_1541987280_335_3" started.

-------------------------------------------------------------------
-----------
w_127.0.0.1_15419872 0x00000152 dxmmsssvr.exe dxm Tue 03/20/01
12:58:19
-------------------------------------------------------------------
-----------
```

```
-- NOTICE mss dxmcmssworker.cp 132 58:19:066
0 "CmssWorker" started.

-- DEBUG1 util cmodule.cpp 191 58:19:076
1 dlopen("C:\Program Files\Atos/DirX Identity\bin\libdxmagtfe.dll") =
0x2030000.

-- DEBUG1 agtfe dxmfrontend.cpp 360 58:19:076
2 Function: "CFrontEnd::CFrontEnd", Parameters: "MsgIn: JMSType: 1
JMSReplyTo: Dxm.command.main
JMSDestination: Dxm.command.main
DXMVersion: 1.00
DXMObjectType: 2
DXMInitiatorType: 2
DXMInstId: 127.0.0.1_1541987280_335_3
DXMWorkflowName: BA_PABX2MetaStore_Full
DXMWorkflowStartTime: 20010320115817Z
DXMActivityName: BA_HR2MetaStore_Full_ODBCExport
DXMActivityStartTime: 20010320115819Z
DXMWorkflowDN:
cn=BA_PABX2MetaStore_Full,dxmC=Workflows,dxmC=DirXmetahub
DXMActivityDN:
cn=BA_HR2MetaStore_Full_ODBCExport,cn=BA_HR2MetaStore_Full,dxmC=Workf
lows,dxmC=DirXmetahub
DXMJobDN:
cn=BA_HR2MetaStore_Full_ODBCExport,dxmC=Jobs,dxmC=DirXmetahub
"
-- NOTICE ldap dxmldap.cpp 256 58:19:086
3 Successfully connected to LDAP server "localhost":1389.
-- NOTICE ldap dxmldap.cpp 327 58:19:316
3 Disconnected from LDAP server "localhost":1389.
-- DEBUG1 agtfe dxmfrontend.cpp 636 58:19:326
2 Function: "CFrontEnd::CFrontEnd" returns "SUCCESSFUL"
------------------------------------------------------------------------
-----------
w_c0671bb1-5ccfb1-e5 0x0000014f dxmmsssvr.exe dxm Tue 03/20/01
12:58:19
------------------------------------------------------------------------
-----------
-- NOTICE wfe dxmwfeimpl.cpp 996 58:19:416
3 "createInstance"-message acknowledged with return code 0
```

```
("SUCCESSFUL").
Message ID: "127.0.0.1_1541987280_338_4",
Instance ID: "127.0.0.1_1541987280_335_3",
Run Object Name:
"cn=BA_HR2MetaStore_Full_ODBCExport,dxmC=Jobs,dxmC=DirXmetahub"
------------------------------------------------------------------------
-----------
w_127.0.0.1_15419872 0x00000152 dxmmsssvr.exe dxm Tue 03/20/01
12:58:19
------------------------------------------------------------------------
-----------
-- DEBUG1 agtfe dxmfrontend.cpp 736 58:19:456
2 Function: "CFrontEnd::execute", Parameters: "MsgIn: JMSType: 8
JMSReplyTo: Dxm.command.main
JMSDestination: Dxm.command.main
DXMVersion: 1.00
DXMObjectType: 2
DXMOperation: 7
DXMInstId: 127.0.0.1_1541987280_335_3
"
-- DEBUG1 agtwr dxmagtwr.cpp 107 58:19:456
3 Function: "Initialize" called.

-- DEBUG1 agtwr dxmagtwr.cpp 115 58:19:456
3 Pointer of Wrapper: 0x806330
-- DEBUG1 agtwr dxmagtwr.cpp 128 58:19:456
3 Function: "Initialize" returns "SUCCESSFUL".

-- DEBUG1 agtwr dxmagtwr.cpp 155 58:19:456
3 Function: "SetControlData" called.

-- DEBUG1 agtwr dxmagtwr.cpp 158 58:19:456
3 Function parameter:
"Pointer of CWrapper: " 0x806330
-- DEBUG1 agtwr dxmagtwr.cpp 161 58:19:456
3 Function parameter:
"Agent type: " 7
-- DEBUG1 agtwr dxmagtwr.cpp 164 58:19:456
3 Function parameter:
"Job path: " "C:/Program Files/Atos/DirX
Identity/work\BA_PABX2MetaStore_Full\BA_HR2MetaStore_Full_ODBCExport"
```

```
-- DEBUG1 agtwr dxmagtwr.cpp 167 58:19:456
3 Function parameter:
"Executable name: " "ODBCAgentExp"
-- DEBUG1 agtwr dxmagtwr.cpp 171 58:19:456
3 Function parameter:
"Agent delta type: " 1529900236
-- DEBUG1 agtwr dxmagtwr.cpp 174 58:19:456
3 Function parameter:
"Process user name: " ""
-- DEBUG1 agtwr dxmagtwr.cpp 177 58:19:456
3 Function parameter:
"Process domain: " ""
-- DEBUG1 agtwr dxmagtwr.cpp 180 58:19:456
3 Function parameter:
"Expiration time: " 600
-- DEBUG1 agtwr dxmagtwr.cpp 188 58:19:456
3 Function: "SetControlData" returns 0.

-- DEBUG1 agtwr dxmagtwr.cpp 225 58:19:456
4 Function: "Command" called.

-- DEBUG1 agtwr dxmagtwr.cpp 228 58:19:456
4 Function parameter:
"Pointer of CWrapper: " 0x806330
-- DEBUG1 agtwr dxmagtwr.cpp 231 58:19:456
4 Function parameter:
"Control function: " 0x20310eb
-- DEBUG1 agtwr dxmagtwr.cpp 234 58:19:456
4 Function parameter:
"Control function parameter: " 33899544
-- DEBUG1 agtwr dxmagtwr.cpp 237 58:19:456
4 Function parameter:
"Command line: " "-f "C:/Program Files/Atos/DirX
Identity/work\BA_PABX2MetaStore_Full\BA_HR2MetaStore_Full_ODBCExport\
personal.ini" -o "C:/Program Files/Atos/DirX
Identity/work\BA_PABX2MetaStore_Full\BA_HR2MetaStore_Full_ODBCExport\
ODBC-HR-export.txt""
-- DEBUG1 agtwr dxmagtwr.cpp 240 58:19:456
4 Function parameter:
"Config data: " "C:/Program Files/Atos/DirX
Identity/work\BA_PABX2MetaStore_Full\BA_HR2MetaStore_Full_ODBCExport\
```

```
personal.ini"
-- DEBUG1 agtwr dxmagtwr.cpp 246 58:19:456
4 Function parameter:
"In delta data: " ""
-- DEBUG1 agtwr dxmagtwr.cpp 249 58:19:456
4 Function parameter:
"Output data: " "C:/Program Files/Atos/DirX
Identity/work\BA_PABX2MetaStore_Full\BA_HR2MetaStore_Full_ODBCExport\
ODBC-HR-export.txt"
-- DEBUG1 agtwr dxmagtwr.cpp 255 58:19:456
4 Function parameter:
"Trace data: " "C:/Program Files/Atos/DirX
Identity/work\BA_PABX2MetaStore_Full\BA_HR2MetaStore_Full_ODBCExport\
ODBC-HR-trc.txt"
-- NOTICE util cbasics.cpp 1287 58:19:466
5 loading '"C:/Program Files/Atos/DirX
Identity/work\BA_PABX2MetaStore_Full\BA_HR2MetaStore_Full_ODBCExport\
personal.ini"' (mode "r+t").

-- DEBUG1 agtwr dxmwrapper.cpp 1284 58:19:466
6 Function: "CWrapper::ProcessAgentODBC()" called.

-- NOTICE util cbasics.cpp 1670 58:19:466
7 save "C:/Program Files/Atos/DirX
Identity/work\BA_PABX2MetaStore_Full\BA_HR2MetaStore_Full_ODBCExport\
personal.ini" ...

-- DEBUG1 agtwr dxmwrapper.cpp 1616 58:19:486
8 Function: "CWrapper::RunAgentWinNT()" called.

-- DEBUG1 agtwr dxmwrapper.cpp 1764 58:19:486
8 Agent call parameter:
"Executable and CmdLine: " "ODBCAgentExp -f "C:/Program
Files/Atos/DirX
Identity/work\BA_PABX2MetaStore_Full\BA_HR2MetaStore_Full_ODBCExport\
personal.ini" -o "C:/Program Files/Atos/DirX
Identity/work\BA_PABX2MetaStore_Full\BA_HR2MetaStore_Full_ODBCExport\
ODBC-HR-export.txt""
-- DEBUG1 agtwr dxmwrapper.cpp 1766 58:19:486
8 Agent call parameter:
"Current dir: " "C:/Program Files/Atos/DirX
```

```
Identity/work\BA_PABX2MetaStore_Full\BA_HR2MetaStore_Full_ODBCExport"
-- DEBUG1 agtwr dxmwrapper.cpp 2009 58:19:557
8 Function: "CWrapper::RunAgentWinNT()" returns "SUCCESSFUL".

-- NOTICE util cbasics.cpp 1287 58:19:557
8 loading '"C:/Program Files/Atos/DirX
Identity/work\BA_PABX2MetaStore_Full\BA_HR2MetaStore_Full_ODBCExport\
personal.ini"' (mode "r+t").

-- NOTICE util cbasics.cpp 1670 58:19:557
5 save "C:/Program Files/Atos/DirX
Identity/work\BA_PABX2MetaStore_Full\BA_HR2MetaStore_Full_ODBCExport/
ProcessInfo.txt" ...

-- NOTICE util cbasics.cpp 1670 58:19:587
5 save "C:/Program Files/Atos/DirX
Identity/work\BA_PABX2MetaStore_Full\BA_HR2MetaStore_Full_ODBCExport\
personal.ini" ...

-- DEBUG1 agtwr dxmagtwr.cpp 277 58:19:637
4 Function: "Command" returns 0.

-- DEBUG1 agtfe dxmfrontend.cpp 1094 58:19:647
2 Function: "CFrontEnd::execute" returns "SUCCESSFUL"
------------------------------------------------------------------------
-----------
w_c0671bb1-5ccfb1-e5 0x0000014f dxmmsssvr.exe dxm Tue 03/20/01
12:58:19
------------------------------------------------------------------------
-----------
-- NOTICE wfe dxmwfeimpl.cpp 959 58:19:747
3 "execute"-message acknowledged with return code 0 ("SUCCESSFUL").
Message ID: "127.0.0.1_1541987280_338_7",
Instance ID: "127.0.0.1_1541987280_335_3",
Run Object Name:
"cn=BA_HR2MetaStore_Full_ODBCExport,dxmC=Jobs,dxmC=DirXmetahub"
------------------------------------------------------------------------
-----------
SUB_7f2570 0x00000129 dxmmsssvr.exe dxm Tue 03/20/01 12:58:19
------------------------------------------------------------------------
-----------
-- NOTICE util cthread.cpp 234 58:19:867
```

```
0 thread "w_127.0.0.1_1541987280_335_10" started.

------------------------------------------------------------------------
-----------
w_127.0.0.1_15419872 0x00000151 dxmmsssvr.exe dxm Tue 03/20/01
12:58:19
------------------------------------------------------------------------
-----------
-- NOTICE mss dxmcmssworker.cp 132 58:19:867
0 "CmssWorker" started.

-- DEBUG1 agtfe dxmfrontend.cpp 360 58:19:877
1 Function: "CFrontEnd::CFrontEnd", Parameters: "MsgIn: JMSType: 1
JMSReplyTo: Dxm.command.main
JMSDestination: Dxm.command.main
DXMVersion: 1.00
DXMObjectType: 2
DXMInitiatorType: 2
DXMInstId: 127.0.0.1_1541987280_335_10
DXMWorkflowName: BA_PABX2MetaStore_Full
DXMWorkflowStartTime: 20010320115817Z
DXMActivityName: BA_HR2MetaStore_Full_metacp
DXMActivityStartTime: 20010320115819Z
DXMWorkflowDN:
cn=BA_PABX2MetaStore_Full,dxmC=Workflows,dxmC=DirXmetahub
DXMActivityDN:
cn=BA_HR2MetaStore_Full_metacp,cn=BA_HR2MetaStore_Full,dxmC=Workflows
,dxmC=DirXmetahub
DXMJobDN: cn=BA_HR2MetaStore_Full_metacp,dxmC=Jobs,dxmC=DirXmetahub
"
-- NOTICE ldap dxmldap.cpp 256 58:19:887
2 Successfully connected to LDAP server "localhost":1389.
-- NOTICE ldap dxmldap.cpp 327 58:20:508
2 Disconnected from LDAP server "localhost":1389.
-- DEBUG1 agtfe dxmfrontend.cpp 636 58:20:508
1 Function: "CFrontEnd::CFrontEnd" returns "SUCCESSFUL"
------------------------------------------------------------------------
-----------
w_c0671bb1-5ccfb1-e5 0x0000014f dxmmsssvr.exe dxm Tue 03/20/01
12:58:20
------------------------------------------------------------------------
```

```
-----------
-- NOTICE wfe dxmwfeimpl.cpp 996 58:20:538
3 "createInstance"-message acknowledged with return code 0
("SUCCESSFUL").
Message ID: "127.0.0.1_1541987280_337_11",
Instance ID: "127.0.0.1_1541987280_335_10",
Run Object Name:
"cn=BA_HR2MetaStore_Full_metacp,dxmC=Jobs,dxmC=DirXmetahub"
-----------------------------------------------------------------
-----------
w_127.0.0.1_15419872 0x00000151 dxmmsssvr.exe dxm Tue 03/20/01
12:58:20
-----------------------------------------------------------------
-----------
-- DEBUG1 agtfe dxmfrontend.cpp 736 58:20:578
1 Function: "CFrontEnd::execute", Parameters: "MsgIn: JMSType: 8
JMSReplyTo: Dxm.command.main
JMSDestination: Dxm.command.main
DXMVersion: 1.00
DXMObjectType: 2
DXMOperation: 7
DXMInstId: 127.0.0.1_1541987280_335_10
"
-- DEBUG1 agtwr dxmagtwr.cpp 107 58:20:578
2 Function: "Initialize" called.

-- DEBUG1 agtwr dxmagtwr.cpp 115 58:20:578
2 Pointer of Wrapper: 0x81ba90
-- DEBUG1 agtwr dxmagtwr.cpp 128 58:20:578
2 Function: "Initialize" returns "SUCCESSFUL".

-- DEBUG1 agtwr dxmagtwr.cpp 155 58:20:578
2 Function: "SetControlData" called.

-- DEBUG1 agtwr dxmagtwr.cpp 158 58:20:578
2 Function parameter:
"Pointer of CWrapper: " 0x81ba90
-- DEBUG1 agtwr dxmagtwr.cpp 161 58:20:578
2 Function parameter:
"Agent type: " 8
-- DEBUG1 agtwr dxmagtwr.cpp 164 58:20:578
```

```
2 Function parameter:
"Job path: " "C:/Program Files/Atos/DirX
Identity/work\BA_PABX2MetaStore_Full\BA_HR2MetaStore_Full_metacp"
-- DEBUG1 agtwr dxmagtwr.cpp 167 58:20:578
2 Function parameter:
"Executable name: " "metacp"
-- DEBUG1 agtwr dxmagtwr.cpp 171 58:20:578
2 Function parameter:
"Agent delta type: " 18208056
-- DEBUG1 agtwr dxmagtwr.cpp 174 58:20:578
2 Function parameter:
"Process user name: " ""
-- DEBUG1 agtwr dxmagtwr.cpp 177 58:20:578
2 Function parameter:
"Process domain: " ""
-- DEBUG1 agtwr dxmagtwr.cpp 180 58:20:578
2 Function parameter:
"Expiration time: " 600
-- DEBUG1 agtwr dxmagtwr.cpp 188 58:20:578
2 Function: "SetControlData" returns 0.

-- DEBUG1 agtwr dxmagtwr.cpp 225 58:20:578
3 Function: "Command" called.

-- DEBUG1 agtwr dxmagtwr.cpp 228 58:20:578
3 Function parameter:
"Pointer of CWrapper: " 0x81ba90
-- DEBUG1 agtwr dxmagtwr.cpp 231 58:20:578
3 Function parameter:
"Control function: " 0x20310eb
-- DEBUG1 agtwr dxmagtwr.cpp 234 58:20:578
3 Function parameter:
"Control function parameter: " 33899544
-- DEBUG1 agtwr dxmagtwr.cpp 237 58:20:578
3 Function parameter:
"Command line: " ""C:/Program Files/Atos/DirX
Identity/work\BA_PABX2MetaStore_Full\BA_HR2MetaStore_Full_metacp\impo
rt.odbc.delta.tcl""
-- DEBUG1 agtwr dxmagtwr.cpp 240 58:20:578
3 Function parameter:
"Config data: " "C:/Program Files/Atos/DirX
```

```
Identity/work\BA_PABX2MetaStore_Full\BA_HR2MetaStore_Full_metacp\odbc
attr.cfg"
-- DEBUG1 agtwr dxmagtwr.cpp 243 58:20:578
3 Function parameter:
"Input data: " "C:/Program Files/Atos/DirX
Identity/work\BA_PABX2MetaStore_Full\BA_HR2MetaStore_Full_metacp\ODBC
-HR-export.txt"
-- DEBUG1 agtwr dxmagtwr.cpp 246 58:20:578
3 Function parameter:
"In delta data: " "20010320124807Z="
-- DEBUG1 agtwr dxmagtwr.cpp 255 58:20:578
3 Function parameter:
"Trace data: " "C:/Program Files/Atos/DirX
Identity/work\BA_PABX2MetaStore_Full\BA_HR2MetaStore_Full_metacp\meta
dir-import-from-ODBC-HR-trc.txt"
-- DEBUG1 agtwr dxmwrapper.cpp 1459 58:20:578
4 Function: "CWrapper::ProcessAgentMetacp()" called.

-- DEBUG1 agtwr dxmwrapper.cpp 1616 58:20:578
5 Function: "CWrapper::RunAgentWinNT()" called.

-- DEBUG1 agtwr dxmwrapper.cpp 1764 58:20:578
5 Agent call parameter:
"Executable and CmdLine: " "metacp "C:/Program Files/Atos/DirX
Identity/work\BA_PABX2MetaStore_Full\BA_HR2MetaStore_Full_metacp\impo
rt.odbc.delta.tcl""
-- DEBUG1 agtwr dxmwrapper.cpp 1766 58:20:578
5 Agent call parameter:
"Current dir: " "C:/Program Files/Atos/DirX
Identity/work\BA_PABX2MetaStore_Full\BA_HR2MetaStore_Full_metacp"
-- DEBUG1 agtwr dxmwrapper.cpp 2009 58:21:579
5 Function: "CWrapper::RunAgentWinNT()" returns "SUCCESSFUL".

-- DEBUG1 agtwr dxmwrapper.cpp 179 58:21:579
5 Function: "CWrapper::writeOutDeltaDataFromFile()" called.

-- WARNING sys dxmwrapper.cpp 197 58:21:579
5 stat (path: "C:/Program Files/Atos/DirX
Identity/work\BA_PABX2MetaStore_Full\BA_HR2MetaStore_Full_metacp/Delt
aOutputData.txt", buf: 0x215f004) = -1 - errno = ENOENT
buf:
```

```
???
-- NOTICE util cbasics.cpp 1670 58:21:579
5 save "C:/Program Files/Atos/DirX
Identity/work\BA_PABX2MetaStore_Full\BA_HR2MetaStore_Full_metacp/Proc
essInfo.txt" ...

-- DEBUG1 agtwr dxmagtwr.cpp 277 58:21:630
3 Function: "Command" returns 0.

-- DEBUG1 agtfe dxmfrontend.cpp 1094 58:21:630
1 Function: "CFrontEnd::execute" returns "SUCCESSFUL"
------------------------------------------------------------------
-----------
w_c0671bb1-5ccfb1-e5 0x0000014f dxmmsssvr.exe dxm Tue 03/20/01
12:58:21
------------------------------------------------------------------
-----------
-- NOTICE wfe dxmwfeimpl.cpp 959 58:21:710
3 "execute"-message acknowledged with return code 0 ("SUCCESSFUL").
Message ID: "127.0.0.1_1541987280_337_14",
Instance ID: "127.0.0.1_1541987280_335_10",
Run Object Name:
"cn=BA_HR2MetaStore_Full_metacp,dxmC=Jobs,dxmC=DirXmetahub"
-- DEBUG1 wfe dxmwfeimpl.cpp 424 58:21:720
2 Object 0x7f7720, Function "CworkflowStub::execute": setting return
value 0 ("SUCCESSFUL") for acknowledge.
-- NOTICE wfe dxmwfeimpl.cpp 449 58:21:720
2 END Workflow
Workflow instance ID: "c0671bb1-5ccfb1-e55be8dae0--8000"
Workflow DN:
"cn=BA_PABX2MetaStore_Full,dxmC=Workflows,dxmC=DirXmetahub"
Workflow Name: "BA_PABX2MetaStore_Full"
Workflow Start Time: "20010320115817Z"
WorkflowState: "closed.completed.ok"
Return code: 0
-- DEBUG1 wfe dxmwfeimpl.cpp 457 58:21:720
2 EXIT Object 0x7f7720, Function "CworkflowStub::execute", return
value is 0 ("SUCCESSFUL").
-- DEBUG1 wfe dxmwfeimpl.cpp 327 58:22:601
3 Destructor of class "CWorkflowStub" called, address is 0x7f7720.
-- DEBUG1 wfe dxmwfeimpl.cpp 330 58:22:601
```

```
3               Workflow instance ID: "c0671bb1-5ccfb1-e55be8dae0--
8000"
Workflow DN:
"cn=BA_PABX2MetaStore_Full,dxmC=Workflows,dxmC=DirXmetahub"
Workflow Name: "BA_PABX2MetaStore_Full"
Workflow Start Time: "20010320115817Z"
WorkflowState: "closed.completed.ok"
-- NOTICE mss dxmcmssworker.cp 190 58:22:601
0 "CmssWorker" terminated.

-----------------------------------------------------------------
-----------
w_127.0.0.1_15419872 0x00000151 dxmmsssvr.exe dxm Tue 03/20/01
12:58:22
-----------------------------------------------------------------
-----------
-- NOTICE ldap dxmldap.cpp 256 58:22:711
2 Successfully connected to LDAP server "localhost":1389.
-----------------------------------------------------------------
-----------
w_127.0.0.1_15419872 0x00000152 dxmmsssvr.exe dir Tue 03/20/01
12:58:22
-----------------------------------------------------------------
-----------
-- WARNING sys dxmfrontend.cpp 1353 58:22:731
3 stat (path: "C:/Program Files/Atos/DirX
Identity/status\BA_PABX2MetaStore_Full.20010320115817Z", buf:
0x202f958) = -1 - errno = ENOENT
buf:
???
-- WARNING sys dxmfrontend.cpp 1375 58:22:731
3 stat (path: "C:/Program Files/Atos/DirX
Identity/status\BA_PABX2MetaStore_Full.20010320115817Z\BA_HR2MetaStor
e_Full_ODBCExport.20010320115819Z", buf: 0x202f958) = -1 - errno =
ENOENT
buf:
???
-- DEBUG1 agtfe dxmfrontend.cpp 708 58:22:741
3 Function: "CFrontEnd::~CFrontEnd", Parameters: "-"
-- NOTICE mss dxmcmssworker.cp 190 58:22:741
0 "CmssWorker" terminated.
```

```
------------------------------------------------------------------------
-----------
w_127.0.0.1_15419872 0x00000151 dxmmsssvr.exe dxm Tue 03/20/01
12:58:22
------------------------------------------------------------------------
-----------
-- NOTICE ldap dxmldap.cpp 327 58:22:751
2 Disconnected from LDAP server "localhost":1389.
-- WARNING sys dxmfrontend.cpp 1375 58:22:761
2 stat (path: "C:/Program Files/Atos/DirX
Identity/status\BA_PABX2MetaStore_Full.20010320115817Z\BA_HR2MetaStor
e_Full_metacp.20010320115819Z", buf: 0x215f958) = -1 - errno = ENOENT
buf:
???
-- WARNING sys cbasics.cpp 2109 58:22:761
3 fopen (filename: "C:/Program Files/Atos/DirX
Identity/work\BA_PABX2MetaStore_Full\BA_HR2MetaStore_Full_metacp\ODBC
-HR-export.txt", type: "rt") = NULL - errno = ENOENT
-- ERROR util cbasics.cpp 2143 58:22:761
3 file open failed: "dxmCopyFile"
-- DEBUG1 agtfe dxmfrontend.cpp 708 58:22:781
2 Function: "CFrontEnd::~CFrontEnd", Parameters: "-"
-- NOTICE mss dxmcmssworker.cp 190 58:22:781
0 "CmssWorker" terminated.
```

Exit Codes

The **metahubdump** command returns an exit code of **0** on success or a **1** if it encountered an error. The text of the error message is displayed on **stderr**.

See Also

dirxlog.cfg (in DirX Identity Program Files)

2. Attribute Configuration File Format

An attribute configuration file defines the attributes that are present in a particular connected directory and supplies formatting information that the meta controller (**metacp**) is to use when processing data files associated with the connected directory. The attribute configuration file contains one record for each attribute in the connected directory. An attribute configuration file must exist for each DirX Identity agent that is present in DirX Identity. Additional attribute configuration files may exist to process export and import data files with formats that are different from the formats that the DirX Identity agents handle.

This section describes the format of an attribute configuration file. An attribute configuration file consists of the following fields:

- Attribute definition fields
- Global information fields for parsing connected directory data files

2.1. Attribute Definition Fields

Each attribute defined in an attribute configuration file has a set of attribute definition fields associated with it.The Identity controller uses the information in these fields to:

- Parse the data files provided by the DirX Identity agents, when importing data into the Identity store (a connected directory of type **LDAP**)
- Generate LDAP-style attribute type-and-value syntax when importing data files into the Identity store (a connected directory of type **LDAP**)
- Generate the data files for the DirX Identity agents, when exporting data from Identity store (a connected directory of type **LDAP**)

This section describes the attribute definition fields that can be specified for an attribute.Except the Attribute Length field, all of these fields must be specified for each attribute.

2.1.1. Abbreviation

The abbreviation field specifies the attribute type abbreviation to be used for the attribute. The meta controller uses the specified abbreviation in its attribute mapping operations. The field syntax is:

Abbr:*abbreviation*

For example:

```
Abbr:SN
```

For attribute configuration files that are to be used with LDAP connections to the Identity store, *abbreviation* must be a valid LDAP attribute name. For example,

Abbr:objectClass

See the String Representation for LDAP Binds chapter for more details about valid LDAP attribute names.

2.1.2. Name

The name field specifies a descriptive name for the attribute. The field syntax is:

Name:*name*

For example:

Name:Surname

The purpose of this field is to make the attribute configuration file records easier to read; the meta controller does not use this field in its operations.

2.1.3. Prefix

The prefix field specifies the start tag, if any, that is used in the connected directory data file to identify the start of the attribute's definition. Prefix tags are generally used in directory data files that use a tagged format. The field syntax is:

Prefix: '[*prefix*]'

where *prefix* is one or more characters enclosed in single quotation marks.

For attribute configuration files that are to be used with LDAP connections to the Identity store, *prefix* is the attribute abbreviation followed by an equal sign: *abbreviation**=*. For example,

Prefix:'surname='

If the value of *prefix* is a non-printing character, supply the octal representation that corresponds to the character. For example, to represent a line feed, define the field as:

Prefix:'\012'

If a prefix is undefined for the attribute (because the associated data file is untagged), specify only the single quotation marks ('') in *prefix*. For example:

Prefix:''

The following example defines attributes for a connected directory that uses LDAP prefixes:

```
Abbr:givenName     Name:Given-Name    Prefix:'givenName='
                   Suffix:''          Rec-Sep:';'
                   MRule:CIM
Abbr:initials      Name:Initials      Prefix:'initials='
```

```
                    Suffix:''           Rec-Sep:';'
                    MRule:CIM
Abbr:postalCode     Name:Postal-Code    Prefix:'postalCode='
                    Suffix:''           Rec-Sep:';'
                    MRule:CIM
```

2.1.4. Suffix

The suffix field specifies the end tag, if any, that is used in the connected directory data file to identify the end of the attribute's definition. Suffix tags are generally used in directory data files that use a tagged or an untagged format. For a CSV data file the suffix fields define the separator (e. g. comma or pipe character) and they must be identical for all attributes. The field syntax is:

Suffix: '[*suffix*]'

where *suffix* is one or more characters enclosed in single quotation marks.

If the attribute configuration file is to be used with LDAP connections to the Identity store, specify an empty value for *suffix* (**''**).

If the value of *suffix* is a non-printing character, supply the octal representation that corresponds to the character. For example, to represent a line feed, define the field as:

`Suffix:'\012'`

The following example defines attributes in a connected directory data file that uses a CSV format:

```
Abbr:EXCN   Name:Common-Name          Prefix:''
            Suffix:','                Rec-Sep:''
            Mrule:-
Abbr:EXDSNM Name:Display-Name         Prefix:''
            Suffix:','                Rec-Sep:''
            Mrule:-
Abbr:EXADR  Name:Internet-Address     Prefix:''
            Suffix:','                Rec-Sep:''
            Mrule:-
```

2.1.5. Attribute Length

The attribute length field is an optional field used to process connected directory data files that are formatted as fixed-width tables. The field syntax is:

Attrlen:*max_data_length*

where *max_data_length* is the maximum number of columns occupied by the attribute definition in the table. For example:

`Attrlen:15`

The following example defines attributes in a data file with a fixed-width table format that uses no delimiters between attribute records:

```
Abbr:SN   Name:Surname               Prefix:''
          Suffix:''                  Attrlen:15
          Rec-Sep:''                 Mrule:-
Abbr:GN   Name:Given Name            Prefix:''
          Suffix:''                  Attrlen:10
          Rec-Sep:''                 Mrule:-
Abbr:TN   Name:Telephone Number      Prefix:''
          Suffix:''                  Attrlen:8
          Rec-Sep:''                 Mrule:-
```

2.1.6. Multi-Valued Attribute Separator

The multivalued attribute separator specifies the string, if any, that is used in the connected directory data file to separate multiple attribute values for the attribute. The field syntax is:

Rec-Sep:'[*separator*]**'**

where *separator* is one or more characters enclosed in single quotation marks (' ').

If the value of *separator* is a non-printing character, supply the octal representation that corresponds to the character. For example, to represent a line feed, define the field as:

`Rec-Sep:'\012'`

If a multivalued attribute separator is undefined for the attribute, specify only the single quotation marks (' ') in *separator*. For example:

`Rec-Sep:''`

The semicolon (;) is used as multivalued attribute value separator in an LDAP attribute configuration file.

2.1.7. Matching Rule

The matching rule field defines the matching rule that the **metacp** program is to apply when attempting to detect modifications to the attribute. The field syntax is:

Mrule:*rule-keyword*

Supply one of the following keywords in *rule-keyword*:

- **CIM** - Use Case-Ignore-String matching. With this matching rule, differences in case between source and target attribute values are ignored, for example, GATES is the same as Gates.
- **CEM** - Use Case-Exact-String matching. With this matching rule, differences in case between source and target attribute values are significant. For example, GATES is different than Gates.
- **DN** - Use Distinguished Name matching. With this matching rule, differences in case between source and target attribute values in distinguished names are ignored.
- **OCTET** - Use Octet String matching. With this matching rule, the byte sequences of the source and target attribute values must match.
- **B64** - Use base64 matching. With this matching rule, enclosing apostrophes and padding characters are ignored. For example, 'BAV0aHJlZQ==' is the same as BAV0aHJlZQ.
- The dash character (**-**) - Use to specify that no matching rule is defined.

The **metacp** program uses the matching rule field to determine how to identify differences between the attribute in the connected directory and the attribute in the Identity store. If no matching rule is defined for the attribute, the **meta modifyentry** operation compares the two attributes using CIM. If it detects differences, it sends an **obj modify** operation that specifies **-removeattr** *attribute_type*, **-addattr** *attribute_values* to the Identity store because the server has no matching rule for the attribute. If a matching rule exists, **meta modifyentry** compares the two attributes, and sends an **obj modify** operation that specifies **-removeattr** *old_value* **-addattr** *new_value* to the Identity store if it detects differences.

The matching rule field is only relevant to attribute configuration files that define LDAP attributes.

The following example illustrates the use of the matching rule field for a set of attributes:

```
Abbr:postalCode    Name:Postal-Code   Prefix:'postalCode='
                   Suffix:''          Rec-Sep:';'
                   MRule:CIM
Abbr:owner         Name:Owner         Prefix:'owner='
                   Suffix:''          Rec-Sep:';'
                   MRule:DN
Abbr:title         Name:Title         Prefix:'title='
                   Suffix:''          Rec-Sep:';'
                   MRule:CIM
Abbr:telexNumber   Name:Telex-Number  Prefix:'telexNumber='
                   Suffix:''          Rec-Sep:';'
                   MRule:-
```

2.1.8. Encryption

The encryption field is optional and used to define whether an attribute value needs to be decrypted before the value is sent to the LDAP server. That field is only used in the attribute configuration file describing the attributes of the LDAP directory, e.g. “ldapattr.cfg”.

If set to Y, then the attribute value will be decrypted; if set to N, the attribute value will be taken as it is.

The following example illustrates the use of the encryption field for a set of attributes:

Abbr:postalCode Name:Postal-Code Prefix:'postalCode=' Suffix:'' Rec-Sep:';' MRule:CIM Encryption:N
Abbr:title Name:Title Prefix:'title=' Suffix:'' Rec-Sep:';' MRule:CIM Encryption:Y

Note: This feature is only available, if the meta controller is invoked by the Server (IDS-C) and the Server (IDS-C) provided the environment for performing decryption tasks.

2.2. Global Information Fields

Global information fields for parsing connected directory data files include:

- Record separators
- Field separators
- Continuation line and comment indicators
- Entry and attribute operation code names for LDIF-formatted data files

All global information fields are optional.Specify global information fields at the top of the attribute configuration file, before the attribute definitions.The next sections describe these fields.

Note: Note: LDIF files can contain the string **version:1** at the beginning. DirX Identity determines LDIF files when all these fields are set to these values:

Op-Code-ADD:add
Op-Code-DEL:delete
Op-Code-MOD:modify
Op-Code-MODDN:moddn
Op-Code-MODRDN:modrdn

When all fields are set correctly, the string version:1 is generated as first line into the LDIF file.

2.2.1. Record Separator

The record separator field defines information that the meta controller is to use to distinguish between entries in a connected directory data file. The field syntax is:

Record-Sep:'*separator*'

where *separator* is one or more characters (or the octal representation of one or more characters) enclosed in single quotation marks (**' '**)

The record separator provides the meta controller with a global indicator that it can use to detect the start of the next entry in a data file. The purpose of the indicator is to permit the meta controller to go to the next entry if it is unable to process the current one. If the record separator field is not specified in the attribute configuration file, the meta controller uses '\n' to distinguish between entries.

The following example defines the line feed as the record separator for a data file:

```
Record-Sep:'\012'
```

For XML file format, the record separator must be set to the value **entry**.

2.2.2. Field Separator

The field separator field defines information that the meta controller is to use to distinguish between attribute values in entries within a connected directory data file that uses a tagged format. The field syntax is:

Field-Sep:'*separator*'

where *separator* is one or more characters (or the octal representation of one or more characters) enclosed in single quotation marks (**' '**).

The field separator is used only in tagged data files to skip attributes in the data file that have not been defined in the attribute configuration file. The field separator permits the meta controller to go to the next attribute if it is unable to process the current one. The field separator is not used as an attribute value separator in an untagged (CSV) data file.

The following example defines the comma as the field separator for a data file:

```
Field-Sep:','
```

For XML file format, the field separator must be set to the value newline (**\012**).

2.2.3. Prefix (Base-64)

The prefix Base 64 field defines information that the meta controller is to use to identify a base64-encoded LDIF attribute using the attribute's private prefix followed by this global prefix information. The field syntax is:

Prefix (Base-64):'*separator*'

where *separator* is one or more characters (or the octal representation of one or more characters) enclosed in single quotation marks (**' '**). If this field is not set, the meta controller uses only each attribute's private prefix when it parses the data file.

The following example defines the colon (:) as the character sequence that is combined with an attribute's private prefix:

Prefix (Base-64):':'

2.2.4. Comment

The comment field defines information that the meta controller is to use to identify comment lines in an LDIF-formatted connected directory data file or any other connected directory data file. The field syntax is:

Comment:'*marker*'

where *marker* is one or more characters (or the octal representation of one or more characters) enclosed in single quotation marks (' '). The following example defines the # character as a comment marker:

Comment:'#'

2.2.5. Continuation Line

The continuation line field defines information that the Identity controller is to use to identify continued lines in an LDIF-formatted connected directory data file or any other data file with continuation lines. The field syntax is:

Continuation-Line:'*marker*'

where *marker* is one or more characters (or the octal representation of one or more characters) enclosed in single quotation marks (' '). The continuation line marker is only recognized at the beginning of a line. Consequently, the marker is the indication in a line that this line is a continuation of the preceding line. The following example defines the space as a continuation line marker:

Continuation-Line:' '

2.2.6. Enclosing Sequence

The enclosing sequence field defines a marker that the meta controller is to use identify the start and end of an entry in a connected directory data file. The field syntax is:

Enclosing-Sequence:'*marker*'

where marker is one or more characters (or the octal representation of one or more characters) enclosed in single quotation marks ('). When the meta controller encounters the enclosing sequence marker during its parsing of the data file, it searches for the next occurrence of the enclosing sequence marker and treats everything between the two markers as one entry. The definition of an enclosing sequence marker is useful for parsing untagged data files, especially CSV-formatted files such as Microsoft Exchange data files, where the comma (,) is used as an entry separator but can also be part of an attribute value in an entry.

The following example defines the double-quote (") as an entry boundary marker:

Enclosing-Sequence:'"'

When it encounters two subsequent occurrences of the enclosing sequence marker, the meta controller treats the marker as part of the attribute value. For example, the meta controller parses **"data""data"** as the value **data"data**

2.2.7. Operation Code Field

The operation code field specifies the attribute within an LDIF change file that holds the LDIF change file operation code for an entry in the change file. The field syntax is:

Op-Code Field:*abbreviation*

where *abbreviation* is the attribute type abbreviation for the attribute.

In the following example, the attribute LCHNGTYPE is defined as the LDIF change file operation code attribute:

```
Op-Code Field:LCHNGTYPE
          .
          .
          .
Abbr:LCHNGTYPE     Name:ChangeType    Prefix:'changetype:'
                   Suffix:'\012'      Rec-Sep:''
                   Mrule:-
```

2.2.8. Add Modification Field

The add modification field specifies the attribute in an LDIF change file that represents the "add" attribute modification operation of an LDIF "modify object" change operation. The field syntax is:

ADD-Modification Field:*abbreviation*

where *abbreviation* is the attribute type abbreviation for the attribute.

In the following example, the attribute LADD_OP is defined as the LDIF "modify object, add attribute value" attribute:

```
ADD-Modification Field:LADD_OP
          .
          .
          .
Abbr:LADD_OP       Name:ADD-modification        Prefix:'add:'
                   Suffix:'\012'                Rec-Sep:''
                   Mrule:-
```

2.2.9. Skip Lines

The skip lines field defines information that the meta controller is to use if a data file contains a certain number of lines at the beginning of the file that it should not process; for example, the header line with the attribute names generated by the Microsoft Exchange **admin** program. The number of lines as indicated in Skip Lines is ignored while processing an import data file. The field syntax is:

Skip Lines:*integer*

where *integer* is a value that is greater than or equal to 0 and specifies the number of lines from the beginning of the file that the meta controller is to ignore when processing an import file. For example:

```
Skip Lines:1
```

directs the meta controller to ignore the first line of the import data file.

2.2.10. Replace Modification Field

The replace modification field specifies the attribute in an LDIF change file that represents the "replace" attribute modification operation of an LDIF "modify object" change operation. The field syntax is:

REPLACE-Modification Field:*abbreviation*

where *abbreviation* is the attribute type abbreviation for the attribute.

In the following example, the attribute LMOD_OP is defined as the LDIF "modify object, replace attribute value" attribute:

```
REPLACE-Modification Field:LMOD_OP
          .
          .
          .
Abbr:LMOD_OP  Name:REPLACE-modification   Prefix:'replace:'
              Suffix:'\012'               Rec-Sep:''
              Mrule:-
```

2.2.11. Delete Modification Field

The delete modification field specifies the attribute within an LDIF change file that represents the "delete" attribute modification operation of an "LDIF "modify object" change operation. The field syntax is:

DELETE-Modification Field:*abbreviation*

where *abbreviation* is the attribute type abbreviation for the attribute.

In the following example, the attribute LDEL_OP is defined as the LDIF "modify object, delete attribute or attribute value" attribute:

```
DELETE-Modification Field:LDEL_OP
          .
          .
          .
Abbr:LDEL_OP  Name:DELETE-modification   Prefix:'delete:'
              Suffix:'\012'              Rec-Sep:''
              Mrule:-
```

2.2.12. New RDN Field

The new RDN field specifies the attribute within an LDIF change file that represents the "new RDN" parameter in an LDIF "modify DN" change operation. The field syntax is:

New-RDN Field:*abbreviation*

where *abbreviation* is the attribute type abbreviation for the attribute.

In the following example, the attribute NEW_RDN is defined as the attribute that holds the new RDN in an LDIF "modify DN" change operation:

```
New-RDN Field:NEW_RDN
          .
          .
          .
Abbr:NEW_RDN       Name:New-RDN       Prefix:'newrdn:'
                   Suffix:'\012'      Rec-Sep:''
                   Mrule:-
```

2.2.13. Delete Old RDN Field

The delete old RDN field specifies the attribute within an LDIF change file that represents the "delete old RDN" parameter in an LDIF "modify DN" change operation. The field syntax is:

Delete Old-RDN Field:*abbreviation*

where *abbreviation* is the attribute type abbreviation for the attribute.

In the following example, the attribute DEL_OLD_RDN is defined as the attribute in an LDIF "modify DN" change entry that holds the boolean value that specifies whether to delete the old RDN:

```
Delete Old-RDN Field:DEL_OLD_RDN
          .
          .
          .
Abbr:DEL_OLD_RDN  Name:Delete-Old-RDN  Prefix:'deleteoldrdn:'
                  Suffix:'\012'        Rec-Sep:''
                  Mrule:-
```

2.2.14. New Superior Field

The new superior field specifies the attribute within an LDIF change file that represents the "new superior" parameter in an LDIF "modify DN" change operation. The field syntax is:

New Superior Field:*abbreviation*

where *abbreviation* is the attribute type abbreviation for the attribute.

In the following example, the attribute NEW_SUP is defined as the attribute that holds a new superior in an LDIF "modify DN" change operation.

```
New Superior Field:NEW_SUP
          .
          .
          .
Abbr:NEW_SUP   Name:New-Superior     Prefix:'newsuperior:'
               Suffix:'\012'         Rec-Sep:''
               Mrule:DN
```

2.2.15. Modification Separator

The modification separator field specifies the attribute in an LDIF change file that identifies the end of an attribute modification in an LDIF "modify object" change operation. The field syntax is:

Modification-Sep Field:*abbreviation*

where *abbreviation* is the attribute type abbreviation for the attribute.

In the following example, the attribute EOA is defined as the attribute that holds the "end-of-attribute-modification" marker in an LDIF "modify object" change operation.

```
Modification-Sep Field:EOA
          .
```

```
          .
          .
Abbr:EOA   Name:End-of-Attribute     Prefix:'-'
           Suffix:'\012'             Rec-Sep:''
           Mrule:-
```

2.2.16. Add Op Code

The add op code field specifies the keyword in an LDIF change file that represents an "add object" LDIF change operation. The field syntax is:

Op-Code-ADD:*keyword*

where *keyword* is a string. For example:

```
Op-Code-ADD:add
```

The keyword supplied in the attribute configuration files corresponds to the keyword that is applied as a value to the LDIF Operation Code attribute in the LDIF change file to indicate an "add object" operation. The definition of this keyword in the attribute configuration file permits the meta controller to recognize an LDIF "add object" operation in an LDIF change file.

2.2.17. Delete Op Code

The delete op code field specifies the keyword in an LDIF change file that represents the LDIF "delete object" operation code. The field syntax is:

Op-Code-DEL:*keyword*

where *keyword* is a string. For example:

```
Op-Code-DEL:delete
```

The keyword supplied in the attribute configuration files corresponds to the keyword that is applied as a value to the LDIF Operation Code attribute in the LDIF change file to indicate a "delete object" operation. The definition of this keyword in the attribute configuration file permits the meta controller to recognize an LDIF "delete object" operation in an LDIF change file.

2.2.18. Modify Op Code

The modify op code field specifies the keyword in an LDIF change file that represents the LDIF "modify object" operation code. The field syntax is:

Op-Code-MOD:*keyword*

where *keyword* is a string. For example:

```
Op-Code-MOD:modify
```

The keyword supplied in the attribute configuration files corresponds to the keyword that is applied as a value to the LDIF Operation Code attribute in the LDIF change file to indicate an LDIF "modify object" operation. The definition of this keyword in the attribute configuration file permits the meta controller to recognize an LDIF "modify object" operation in an LDIF change file.

2.2.19. Modify DN Op Code

The modify DN op code field specifies the keyword in an LDIF change file that represents the LDIF "modify DN" operation. The field syntax is:

Op-Code-MODDN:*keyword*

where *keyword* is a string. For example:

```
Op-Code-MODDN:moddn
```

The keyword supplied in the attribute configuration files corresponds to the keyword that is applied as a value to the LDIF Operation Code attribute in the LDIF change file to indicate a "modify DN" operation. The definition of this keyword in the attribute configuration file permits the meta controller to recognize an LDIF "modify DN" operation in an LDIF change file.

2.2.20. Modify RDN Op Code

The modify RDN op code field specifies the keyword in an LDIF change file that represents the LDIF "modify RDN" operation. The field syntax is:

Op-Code-MODRDN:*keyword*

where *keyword* is a string. For example:

```
Op-Code-MODRDN:modrdn
```

The keyword supplied in the attribute configuration files corresponds to the keyword that is applied as a value to the LDIF Operation Code attribute in the LDIF change file to indicate a "modify RDN" operation. The definition of this keyword in the attribute configuration file permits the meta controller to recognize an LDIF "modify RDN" operation in an LDIF change file.

2.2.21. Ignore Empty Value

The ignore empty value field controls whether the meta controller returns empty attributes (attributes with no value) in the results of an **obj search** operation on a connected directory. The field syntax is:

IgnEmptyVal:*n*

where *n* is **Y** to ignore empty attribute values and **N** to return empty attributes.

Set the **IgnEmptyVal** field to **N** when exporting the contents of the Identity store to a connected directory that is a data file with a position-driven (untagged) file format, such as table format or CSV. In this type of data file, the absence of attributes needs to be handled. For example, suppose you are exporting the attributes surname, given-name, and telephone-number to a CSV-formatted data file. Surname and telephone-number have the values **Schmid** and **41496**, and given-name is empty. When **IgnEmptyVal** is set to **N**, the meta controller generates the following data on export:

Schmid,,41496

When exporting to a connected directory that is a data file with a tagged format, you can choose to list the empty attributes with the prefix and no attribute value (**IgnEmptyVal:N**), or to drop empty attributes entirely (**IgnEmptyVal:Y**). For example, if **IgnEmptyVal** is set to **N**, the result is:

```
SN:Schmid
GN:
TN:41496
```

If **IgnEmptyVal** is set to **Y**, the result is:

```
SN:Schmid
TN:41496
```

If you want the meta controller to write only special attributes as empty to the datafile and not all attributes you can do it the following way:

Setting for example the attribute mail to empty (in dependence of whatever, mostly in case of it is empty after mapping) write in your post join mapping:

```
set tgt(mail) [llist ""]
instead of
set tgt(mail) ""
```

3. Directory Data File Formats

Different directories use different data file formats to represent the entries and attributes within them. The meta controller needs to be able to recognize the different directory data file formats so that it can:

- Interpret the source data generated from different connected directories
- Process the source data into different target directory formats

In general, an individual directory's data file format is either:

- Tagged, where each attribute in an entry is tagged by a prefix and optionally by a suffix. Directories that use tagged file formats include X.500 DAP directories, LDAP directories, and Lotus Notes directories.
- Untagged (or "position-driven"), where each attribute in an entry is identified based on its position in the entry. Microsoft Exchange is an example of a directory that uses an untagged data file format.

The meta controller supports both untagged and tagged data file formats, including the LDAP Data Interchange Format (LDIF) tagged file format. This chapter describes the characteristics of tagged and untagged data file formats. It also provides a general discussion of LDIF file format.

3.1. Tagged Data File Format

In a tagged data file, each entry and its attributes are tagged by a prefix and a suffix.The presence of these tags allows the entry to contain an unordered list of attributes.A multiple-value separator can be defined to separate the attribute values of a multivalued attribute.

Each entry can be limited to a single line or split over several lines.If the entry is split over several lines, the description of the data file (that is, the relevant fields in the attribute configuration file) must contain global information that determines the end of an entry.

Here is an example of a tagged file in which:

- The prefixes are "SN=", "GN=", and so on
- The suffixes used include the semicolon (;) and the pipe character (|)
- The record (entry) separator is the line feed (<LF>).

The fields in the attribute configuration file that define this data format are:

```
Record-Sep:'\012'
Abbr:SN  Name:Surname     Prefix:'SN='
         Suffix:';'       Rec-Sep:''
         MRule:-
```

```
Abbr:GN  Name:Given-Name  Prefix:'GN='
         Suffix:'|'       Rec-Sep:''
         MRule:-
Abbr:TN  Name:Telephone-Number  Prefix:'TN='
         Suffix:'|'             Rec-Sep: ''
         MRule:-
Abbr:FTN Name:Fax-Telephone-Number  Prefix:'FAX='
         Suffix:';'                 Rec-Sep: ''
         Mrule:-
Abbr:UP  Name:User-Password     Prefix:'UP='
         Suffix:';'             Rec-Sep: ''
         Mrule:-
```

The data file format looks like:

```
SN=Serling;GN=Rod|TN=44597|FAX=44598;UP=twyl8t;<LF>
SN=Stefano;FAX=44232;UP=xante;<LF>
SN=Fontana;<LF>
```

Here is an example of a tagged file in which:

- The prefixes are "SN=", "GN=", and so on.
- The suffixes used include **";<LF>"** and **"|<LF>"**
- The record (entry) separator is the double line feed (<LF><LF>).

The fields in the attribute configuration file that define this data format are:

```
Record-Sep:'\012\012'
Abbr:SN  Name:Surname        Prefix:'SN='
         Suffix:';\012'      Rec-Sep: ''
         MRule:-
Abbr:GN  Name:Given-Name     Prefix:'GN='
         Suffix:'|\012'      Rec-Sep: ''
         MRule:-
Abbr:TN  Name:Telephone-Number  Prefix:'TN='
         Suffix:'|\012'         Rec-Sep: ''
         MRule:-
Abbr:FTN Name:Fax-Telephone-Number  Prefix:'FAX='
         Suffix:';\012'             Rec-Sep: ''
         MRule:-
Abbr:UP  Name:User-Password   Prefix:'UP='
```

```
         Suffix:';\012'       Rec-Sep: ''
         MRule:-
```

The data file format looks like:

```
SN=Serling;<LF>
GN=Rod|<LF>
TN=44597|<LF>
FAX=44598;<LF>
UP=twyl8t;<LF>
<LF>
SN=Stefano;<LF>
FAX=44232;<LF>
UP=xante;<LF>
<LF>
SN=Fontana;<LF>
<LF>
```

3.2. Untagged Data File Format

In an untagged data file, an individual attribute is identified based on its position in an entry.Attributes are separated by an attribute separator or a field width for the attribute can be defined.The attribute delimiter can differ for each attribute, but typically it is the same for all attributes in the file.A multiple-value separator can be defined to separate the attribute values of a multivalued attribute.Because the attributes are identified by their positions in the entry, attribute separators follow each other for attributes with no value.

The meta controller supports the following untagged file formats:

- Comma-separated value (CSV) format
- Fixed-width table format

CSV format is a special format (the comma is the delimiter between attributes) of a more generic "character-separated" format, in which attributes are delimited by any combination of characters.Here is an example of a character-separated format for the attribute types Surname, Given Name, Telephone Number, Department.The fields in the attribute configuration file that define this format are:

```
Record-Sep:'\012'
Abbr:SN  Name:Surname        Prefix:''
         Suffix:'|'          Rec-Sep:''
         MRule:-
Abbr:GN  Name:Given-Name     Prefix:''
```

```
         Suffix:'|'          Rec-Sep:''
         MRule:-
Abbr:TN  Name:Telephone-Number   Prefix:''
         Suffix:'|'              Rec-Sep:''
         MRule:-
Abbr:DEP  Name:Department     Prefix:''
          Suffix:'|'          Rec-Sep:''
          MRule:-
```

The data format looks like:

```
Fontana|Frank|301-555-2223|NR 1 FE|<LF>
Brown|Murphy||NR 1 AC|<LF>
```

Here is an example of a fixed-width table format for the attribute types Surname, Given Name, Telephone Number, Department. The fields in the attribute configuration file that define this format are:

```
Record-Sep:'\012'
Abbr:SN  Name:Surname        Prefix:''
         Suffix:''           Attrlen:15
         Rec-Sep:''          MRule:-
Abbr:GN  Name:Given-Name     Prefix:''
         Suffix:''           Attrlen:15
         Rec-Sep:''          MRule:-
Abbr:TN  Name:Telephone-Number  Prefix:''
         Suffix:''              Attrlen:12
         Rec-Sep:''             MRule:-
Abbr:DEP Name:Department     Prefix:''
         Suffix:''           Attrlen:10
         Rec-Sep:''          MRule:-
```

The data format looks like:

```
Fontana    Frank      301-555-2223   NR 1 FE      <LF>
Brown      Murphy                    NR 1 AC      <LF>
```

When using the table format, the output of each field is limited by the **AttrLen** component where **AttrLen** defines the maximum length of the field.The field value is composed of the prefix, attribute value(s), optionally multi-value separators, and finally the suffix.In tables

Prefix, **Suffix** and **Rec-Sep** are usually empty (as shown in the example above).If the composed field value exceeds the length specified in **AttrLen** the field value is truncated; a truncated value is indicated by the string "..." at the end of the field.

3.3. LDIF Format

LDIF format is a kind of tagged data file format.There are two types of LDIF format:

- LDIF content format
- LDIF change format

LDIF format supports the following features:

- Base-64 encoding
- UTF-8 encoding
- References to external files, in URL format; for example, an attribute type/value pair such as:

  jpegphoto:< file://usr/local/photos/monroe.jpeg
- Alternate record terminators, such as <CR>, <LF>, or <CR><LF>
- Comment lines and continuation lines
- Multiple separators between entries

The next sections briefly describe the LDIF content and change file formats. For a complete description of LDIF formats, see the document entitled "G. Good, The LDAP Data Interchange Format (LDIF) - Technical Specification, RFC 2849".

3.3.1. LDIF Content Format

A data file in LDIF content format contains a list of directory entries and their attributes. Each entry consists of a distinguished name and a list of attributes. Each attribute has a prefix and one or more values. For example:

```
dn: cn=George Costanza, ou=sales, o=NY Yankees, c=us
objectclass: top
objectclass: person
objectclass: organizationalPerson
cn: George Costanza
cn: G. Costanza
cn: Georgie
sn: Costanza
```

3.3.2. LDIF Change Format

A data file in LDIF change format contains a list of directory modifications. Each entry in the change file contains a special LDIF "changetype" attribute that indicates the type of directory modification to be made. There are four types of modification specified in an LDIF change file:

- Add a directory entry
- Delete a directory entry
- Modify one or more attributes of a directory entry
- Modify the distinguished name of a directory entry

These modifications correspond to the set of LDAP operations that modify a directory. The number and type of attributes present in each entry in the change file differs depending upon the value of the **changetype** attribute for the entry. The next sections describe the types of change file entry structures.

3.3.2.1. Add Directory Entry Format

If the value of the "changetype" attribute is "add", the entry contains the distinguished name of the entry to be created and the attribute type and value pairs to be created for the entry. For example:

```
dn: cn=Joe Isuzu, ou=sales, o=Isuzu, c=us
changetype: add
objectclass: top
objectclass: person
objectclass: organizationalPerson
surname: Isuzu
```

3.3.3. Delete Directory Entry Format

If the value of the "changetype" attribute is "delete", the entry contains the distinguished name of the entry to be deleted. For example:

```
dn: cn=Joe Isuzu, ou=sales, o=Isuzu, c=us
changetype: delete
```

3.3.4. Modify Entry Format

If the value of the "changetype" attribute is "modify", the entry contains the distinguished name of the entry to be modified and a list of attributes that represent one or more modifications to be made to attributes of the entry. There are three types of modification operations that can be recorded in an entry of "changetype" modify":

- Add an attribute value (including multiple attribute values)
- Delete an attribute or an attribute value (including multiple attribute values)
- Replace an attribute with other values (including multiple attribute values)

An attribute in the entry identifies the type of modification operation to be performed; its value is the name of the attribute on which to perform the operation. The attribute structure used to represent the modification to be made to the attribute differs depending on the type of modification. The dash (-) is the "end-of-modification" suffix and terminates each modification structure. For example:

```
dn: cn=S. Beckhardt, ou=iris o=lotus c=us
changetype: modify
add: telephonenumber
telephonenumber: 603 222 4344
-
delete: description
description: CFO
-
replace: surname
surname: S. Beckhardt
surname: Beckhardt
-
```

The next sections describe the "add" "delete" and "replace" modification structures.

3.3.4.1. Add Attribute Value Structure

The "add attribute value" structure consists of:

- The "add" modification identifier attribute
- One or more attribute type/value pairs that specify the new values to apply.

For example:

```
add: telephonenumber
telephonenumber: 617 235 4764
telephonenumber: 508 546 6645
-
```

3.3.4.2. Delete Attribute and Delete Attribute Value Structure

The "delete attribute value" consists of:

- The "delete" modification identifier attribute
- One or more attribute type/value pairs that specify the values to be deleted.

For example:

```
delete:description
description:engineer
description:software engineering
-
```

The "delete attribute" operation consists of the "delete" modification identifier whose value is the attribute to be deleted. For example:

```
delete: description
-
```

3.3.4.3. Replace Attribute Value Structure

The "replace attribute value" operation contains:

- The "replace" modification identifier attribute
- A list of attribute values that replace the attribute

For example:

```
replace: surname
surname: Soeder
surname: C. Soeder
-
```

3.3.5. Modify Distinguished Name/Modify Relative Distinguished Name Format

If the value of the "changetype" attribute is "moddn" or "modrdn", the entry consists of:

- The distinguished name of the entry whose name is to be modified
- Information that specifies the new RDN to be applied to the entry
- Information that specifies whether to delete the old RDN; the value of this attribute is either "0" (do not delete) or "1" (delete) (only relevant if "changetype" is "moddn")
- Information that specifies the distinguished name of the entry's new superior (only relevant if "changetype" is "moddn")

For example:

```
dn: cn=richard hustvedt, ou=engr-ma, o=digital, c=us
changetype: moddn
newrdn: cn=r. hustvedt
deleteoldrdn: 0
newsuperior: ou=engr-nh, o=compaq, c=us
```

3.4. Extensible Markup Language (XML) Format

Extensible Markup Language (XML) is a flexible file format.It is described in the XML standard and in various documents.DirX Identity supports two XML formats:

- Directory Services Markup Language (DSML) V1.0
- Flat XML, which is a simple structured format that is similar to LDIF

The next sections briefly describe the XML file formats.

3.4.1. Directory Service Markup Language (DSML V1) Format

A data file in DSML V1 format contains a sections of directory entries and their attributes. Each entry consists of sections of attributes, included into the **<entry>** and **</entry>** tags. The attributes are described in **<attr>** and **<objectclass>**sections. For example:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE dsml SYSTEM 'dsml.dtd'>
<dsml>
    <!-- status error-code="0" msg="Ok" entry-count="1" -->
    <directory-entries>
        <entry>
            <attr name="cn">
                <value>James R. Doran</value>
                <value>Jimmy Doran</value>
            </attr>
            <attr name="telephoneNumber">
                <value>1-914-656-2650</value>
            </attr>
            <attr name="mail">
                <value>jrdoran@us.othercompany.com</value>
            </attr>
            <objectClass>
                <oc-value>person</oc-value>
```

```
                <oc-value>organizationalPerson</oc-value>
                <oc-value>othercompanyPerson</oc-value>
                <oc-value>ePerson</oc-value>
            </objectClass>
        </entry>
        <entry>
            <attr name="cn">
                <value>Harry Hirsch</value>
            </attr>
...
            <attr name="mail">
                <value>hhirsch@owncompany.com</value>
            </attr>
            <objectClass>
                <oc-value>person</oc-value>
                <oc-value>organizationalPerson</oc-value>
                <oc-value>owncompanyPerson</oc-value>
                <oc-value>ePerson</oc-value>
            </objectClass>
        </entry>
    </directory-entries>
</dsml>
```

DirX Identity does not currently support DTDs or DTD sections.

You can find more information about DSML at **http://www.dsml.org**.

3.4.2. Flat XML Format

A data file in flat XML format contains sections of directory entries and lists of their attributes. Each entry consists of a begin tag **<entry>**, the list of attributes and the end tag **</entry>**. Each attribute is described by a begin tag ***<attribute_name>***, the attribute value and an end tag ***</attribute_name>***. For example:

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<!DOCTYPE address SYSTEM "C:\address.dtd">
<address>
    <entry>
        <dn>o=PQR</dn>
        <o>PQR</o>
        <description>PQR Company</description>
        <telephoneNumber>+49 12 345 67 890</telephoneNumber>
```

```
        <objectClass>organization</objectClass>
        <objectClass>top</objectClass>
        <createTimestamp>20000308120944Z</createTimestamp>
    </entry>
    <entry>
        <dn>cn=admin, o=PQR</dn>
...
        <createTimestamp>20000308120947Z</createTimestamp>
    </entry>
</address>
```

4. ChangeLog Data Handling

All directories provide some kind of change log information. Nevertheless, the format is different depending on the vendor. DirX Identity can handle change logs from:

- DirX
- iPlanet
- OID

If any other directory provides the same change log formats as one of these, DirX Identity can handle it.

The next topics describe the data formats and examples of how to handle change log information.

4.1. DirX ChangeLog Format

DirX provides the LDIF agreement mechanism to produce changeLog information in the form of LDIF content and LDIF change files.Compared with the iPlanet and OID mechanism the handling of deletions is more powerful (they keep only the distinguished name of the deleted entry, DirX can keep a configurable amount of additional information).

Please refer to the DirX documentation for any details.

4.2. iPlanet and OID Formats

The directories of iPlanet and OID support the same changeLog format.The following formats are supported:

- Add object
- Delete object
- Modify object
- Modify DN

4.2.1. Add Object Format

Example of a iPlanet or OID add object format:

```
Changenumber=207, cn=changelogentry
objectclass=top
objectclass=changelog
targetdn=cn=Shrivastava Saurabh,ou=metadirectory,o=oracle,dc=com
changetype=Add
operationtime=19991029153546z
servername=lisa
```

```
orclguid=
orclparentguid=000396876854720675358231686944 2
changenumber=207
orclchangeretrycount=0
changes=objectclass:inetOrgPerson
objectclass:OrganizationalPerson
objectclass:Person
objectclass:Top
cn:Shrivastava Saurabh
cn:Shrivastava
cn:Saurabh
cn:Saurabh Shrivastava
cn:sshrivas
employeenumber:22
givenname:Saurabh
sn:Shrivastava
title:Mr.X
mail:sshrivas@us.oracle.com.X
telephonenumber:650-574-9107
postaladdress:1067 Foster city Blvd $Apt B$Foster city CA=94404-X
l:Foster city
orclguid:000396876942251408758371686944 2
creatorsname:cn=orcladmin
createtimestamp:19991029153546z
```

4.2.2. Delete Object Format

Example of a iPlanet or OID delete object format:

```
Changenumber=208, cn=changelogentry
objectclass=top
objectclass=changelog
targetdn=cn=tinker,ou=development,o=pqr,c=us
changetype=Delete
operationtime=20000210133720z
servername=lisa
orclguid=000396876810754063557731686944 2
orclparentguid=
changenumber=208
orclchangeretrycount=0
```

```
changes
```

4.2.3. Modify Object Formats

Example of a iPlanet or OID modify object format to **modify an existing value of an attribute**:

```
Changenumber=209, cn=changelogentry
objectclass=top
objectclass=changelog
targetdn=CN=Morton, OU=Development, O=PQR, C=US
changetype=Modify
operationtime=20000210133841z
servername=lisa
orclguid=000396876810754022757721686944 2
orclparentguid=
changenumber=209
orclchangeretrycount=0
changes=Replace:description^1^20000210133841z^lisa
description:Deep Sea Diver-NEW
```

Example of a iPlanet or OID modify object format **to delete an attribute**:

```
Changenumber=212, cn=changelogentry
objectclass=top
objectclass=changelog
targetdn=cn=digger,ou=sales,o=pqr,c=us
changetype=Modify
operationtime=20000210140635z
servername=lisa
orclguid=000396876812343647458211686944 2
orclparentguid=
changenumber=212
orclchangeretrycount=0
changes=delete:description^2^20000210140635z^lisa
```

Example of a iPlanet or OID modify object format **to add another recurring value to a multi value attribute**:

```
Changenumber=210, cn=changelogentry
```

```
objectclass=top
objectclass=changelog
targetdn=CN=Morton, OU=Development, O=PQR, C=US
changetype=Modify
operationtime=20000210133841z
servername=lisa
orclguid=0003968768107540227577216869442
orclparentguid=
changenumber=210
orclchangeretrycount=0
changes=Replace:postalcode^1^20000210133841z^lisa
postalcode:WV1 9QY
postalcode:second-value
```

Example of a iPlanet or OID modify object format **to delete a value of a multi value attribute**:

```
Changenumber=211, cn=changelogentry
objectclass=top
objectclass=changelog
targetdn=CN=Morton, OU=Development, O=PQR, C=US
changetype=Modify
operationtime=20000210133842z
servername=lisa
orclguid=0003968768107540227577216869442
orclparentguid=
changenumber=211
orclchangeretrycount=0
changes=Replace:telephonenumber^1^20000210133842z^lisa
telephonenumber:+44 1902 777777
telephonenumber:+44 1902 999999
```

4.2.4. Modify DN Format

Example of a iPlanet or OID modify DN format with new parent:

```
Changenumber=214, cn=changelogentry
objectclass=top
objectclass=changelog
targetdn=cn=new-digger,ou=sales,o=pqr,c=us
```

```
changetype=moddn
operationtime=20000210140914z
servername=lisa
orclguid=000396876812343647458211686944 2
orclparentguid=00039687680981954445765168694 42
changenumber=214
orclchangeretrycount=0
changes=newrdn:cn=digger
deleteoldrdn:4294967295
newSupDN:ou=development,o=pqr,c=us
```

5. String Representation for LDAP Binds

This chapter describes the LDAP-style string representations of simple and structured attributes, search filters and distinguished names. The meta controller (**metacp**) uses these string representations in LDAP binds to enter and display directory information. (See Bind Types and Bind IDs in the metacp section for details.)

This chapter provides:

- An overview of the elements and format of simple and structured attributes
- An overview of the elements and format of distinguished names
- An overview of the elements and format of search filters
- A table of reserved characters for attributes
- A description of attribute syntax

Refer to the following documents for additional information about LDAP:

- Wahl, M., Howes, T., and S. Kille, "Lightweight Directory Access Protocol (v3)", RFC 2251, December 1997.
- Wahl, M., Coulbeck, A., Howes, T., and S. Kille, "Lightweight Directory Access Protocol (v3): Attribute Syntax Definitions", RFC 2252, December 1997.
- Kille, S., Wahl, M., and T. Howes, "Lightweight Directory Access Protocol (v3): UTF-8 String Representation of Distinguished Names", RFC 2253, December 1997.
- Howes, T., "A String Representation of LDAP Search Filters", RFC 2254, December 1997.
- Yergeau, F., "UTF-8, a transformation format of Unicode and ISO 10646", RFC 2279, January 1998.

5.1. Simple and Structured Attributes

Simple attributes are in the form:

type=*simple_value*[;_simple_value_ ...]

Structured attributes are in the form:

type=*structured_value*[;_structured_value_ ...]

Each of the elements in simple and structured attributes is described in the following sections.

5.1.1. Attribute Types

For both simple and structured attributes, the *type* parameter identifies the attribute.

Attribute types are identified internally by OID (object identifiers), a unique series of integers separated by the period (**.**) character. For example, **2.5.4.3** is the OID for the

Common-Name attribute type. You can identify attributes by OID, but to make it easier to specify attributes, the **metacp** commands allow you to identify them by LDAP names in command lines.

The LDAP name(s) of an attribute are specified in the directory schema. For example, the LDAP name **c** or **countryName** represents the Country attribute type, the LDAP name **cn** or **commonName** represents the Common-Name attribute type. When specifying LDAP names in **metacp** commands, LDAP names are treated case-insensitive; for example, **cn**, **Cn**, and **CN** are all valid ways to specify the LDAP name for the Common-Name attribute type independent of the exact value for the LDAP name specified in the directory schema.

When you display attributes, **metacp** uses

- the exact LDAP name value in results, when the LDAP name was specified in the request.
- the first exact LDAP name value specified in the system, when specifying options like **-allattr** in requests.
- the object identifier (OID), when the object identifier was specified in the request.

To obtain the LDAP names of attribute types, use **metacp** to read the directory schema as shown in the following example:

```
metacp> bind -protocol LDAPv3
metacp> show cn=LDAPGlobalSchemaSubentry -allattr -pretty
```

5.1.2. Simple Attribute Values

For simple attributes, the *simple_value* parameter is the value assigned to the attribute. *simple_value* can be only a simple value, not another attribute. To enter more than one *simple_value*, separate each with a semicolon as shown in the following examples:

```
{telephoneNumber=+1 964 123 456}
{2.5.4.20=+1 964 123 456}
{telephoneNumber=+1 964 123 456;+1 965 234 543}
```

Simple attribute values are always treated as UTF-8 strings for LDAP binds.

5.1.3. Structured Attribute Values

The LDAP protocol generally handles all attribute values as UTF-8 strings. There is no common rule how to specify structured attribute values. To obtain information about the structured attribute syntaxes that are supported by the Identity store and how they are specified, refer to the String Representations for Structured Attribute Syntaxes section in this chapter and to RFC 2252 (Attribute Syntax Definitions). When displayed, structured attribute values are not broken into subcomponents in pretty mode.

5.1.4. Attribute Lists for Simple and Structured Attributes

Most **metacp** operations allow you to specify more than one simple or structured attribute on the command line. To specify more than one attribute, separate each attribute type and value string with white space.

The following **-attribute** option specifies an objectClass, description and telephoneNumber simple attributes, all separated by a space. Note that the description and telephoneNumber attributes are enclosed in braces (**{ }**) because they contain white space for readability.

```
-attribute objectClass=organizationalUnit
                {description=Engineering Department}
                {telephoneNumber=+1 964 123 4567}
```

5.1.5. Attribute Values in a File

For LDAPv3 binds, attribute values can also be specified in a file or written to a file. The syntax is as follows:

*attribute***[;binary]_FILE**=*filename1*[;*filename2* ...]

where *attribute* specifies the attribute type and *filename* the name of the file containing the attribute value. When specifying multiple attribute values each value is saved in a separate file. *filename1*;*filename2* ... then specify the value files. When reading multiple attribute values **.***number* is appended to *filename* where *number* is **1** to **n** (total number of values). The first value is written to *filename*.

If the keyword **;binary** is specified, the file contains the valid ASN.1 encoded value of the attribute.

For the X.509 attributes Authority Revocation List, CA-Certificate, Certificate Revocation List, Cross-Certificate-Pair, Delta Revocation List, and User Certificate the file always contains the valid ASN.1 encoded value of the attribute regardless of specifying the keyword **;binary**.

Specifying the attribute value in a file is possible for the following **dirxcp** operations:

- **obj create** (**-attribute** option)
- **obj modify** (**-addattr**, **-changeattr**, **-removeattr**, **-replaceattr** options)
- **obj compare** (**-attribute** option)
- **obj show** (**-attribute** option)
- **obj search** (**-attribute** option)

Examples:

- The following example adds two pictures to the jpegPhoto attribute. The files "/tmp/john1.jpg" and "/tmp/john2.jpg" contain the binary representation of the pictures.

```
modify cn=Huber,ou=sales,o=pqr,c=de -addattr \
          {jpegPhoto_FILE=/tmp/john1.jpg;/tmp/john2.jpg}
```

- Compare the jpegPhoto attribute. The file "pict.jpg" contains the value of the jpegPhoto attribute in binary representation.

```
compare cn=zapf,ou=asw,o=sni,c=de \
             -attr jpegPhoto_FILE=pict.jpg
```

- Modify the value of the jpegPhoto attribute. The file "pict_old.jpg" contains the old value and the file "pict_new.jpg" contains the new value.

```
modify cn=Mayer,ou=sales,o=pqr,c=de \
            -changeattr jpegPhoto_FILE=pict_old.jpg \
             jpegPhoto_FILE=pict_new.jpg
```

- Create an entry. The file "cert.cer" contains the ASN.1 encoded value of the user certificate attribute.

```
create cn=Huber,ou=sales,o=pqr,c=de -attr \

{objectClass=person;organizaionalPerson;strongAuthenticationUser}
\
            sn=Huber {userCertificate;binary_FILE=cert.cer}
```

- Show an entry. The ASN.1 encoded value of the user certificate attribute is written to the file "huber.cer".

```
show cn=Huber,ou=sales,o=pqr,c=de -attr \
        {userCertificate;binary_FILE=huber.cer}
```

- Search several objects. The result contains eight objects two of them containing multiple attribute values.

```
search o=pqr,c=de -subtree -attr jpegPhoto_FILE=photo.jpg
```

The output is as follows:

```
{ou=Sales,o=pqr,c=de}
{{cn=Smith John,ou=Sales,o=pqr,c=de}}
{cn=Mayer,ou=Sales,o=pqr,c=de}
{cn=Hohner,ou=Sales,o=pqr,c=de
{jpegPhoto_FILE=photo.jpg;photo.jpg.1;photo.jpg.2}}
{cn=Richter,ou=Sales,o=pqr,c=de}
{cn=Abele,ou=Sales,o=pqr,c=de
{jpegPhoto_FILE=photo.jpg.3;photo.jpg.4}}
{cn=Reichel,ou=Sales,o=pqr,c=de}
{cn=hohner2,ou=Sales,o=pqr,c=de}
```

5.1.6. Binary Attribute Values

The LDAP v3 protocol supports the specification of attribute values in binary format, that is the ASN.1 encoding of the attribute value. To display and specify the binary attribute values on the user interface of **metacp** the Base-64 encoded representation of the attribute value is used. To specify or read attribute values in binary format the syntax is as follows:

{*attribute*;**binary**}	to read the attribute value binary
{*attribute*;**binary=***attribute_value*}	to specify the attribute value binary

where *attribute* specifies the attribute type and *attribute_value* specifies the Base-64 encoding of the binary attribute value when creating or modifying this value.

When the syntax of the attribute is not OCTET STRING null bytes in the value are not allowed.

Specifying the attribute value in binary format is possible for the following **metacp** operations:

- **obj create** (**-attribute** option)
- **obj modify** (**-addattr**, **-changeattr**, **-removeattr** options)
- **obj compare** (**-attribute** option)
- **obj search** (**-attribute** option)
- **obj show** (**-attribute** option)

Examples:

The following example creates the person **cn=TestUser1, ou=Development, o=PQR, c=de** with the attribute MHS-OR-Address (**mhsOraddresses**). The Base-64 encoded value of the MHS-OR-Address represents the value **\{/G=j/S=testUser1/O=PQR/PRMD=pqr/ADMD=/C=de}**.

```
create {cn=TestUser1, ou=Development, o=PQR, c=de} -attr sn=testp \
```

```
{objectClass=organizationalPerson;person;top;mhsUser} \
{mhsOraddresses;binary=MIAwgGGAEwJkZQAAYoATASAAAKKAEwNwcXIAAIMDUFFSpY
CACXRlc3RVc2VyMYEBagAAAAAAAA==}
```

The following example displays the Street Address attribute (**street**) of the person **cn=Digger, ou=Development, o=PQR, c=de** Base-64 encoded:

```
show cn=Digger,ou=Development,o=pqr,c=de -attr {street;binary} -p
```

The output of the sample command as follows:

```
1) cn=Digger,ou=Development,o=PQR,c=de
    street\;binary : MjQgRG91Z2FuIFN0cmVldA==
```

5.2. Distinguished Names

A distinguished name consists of a list of one or more relative distinguished names (RDNs), separated by a comma (**,**).The list of relative distinguished names starts with the last namepart and ends with the first namepart under the root entry.For example:

```
cn=schmid+ou=ap11,ou=dap11,o=dbp,c=de
```

Each RDN consists of one or more naming attributes in the following format:

type=*value*[+*type*=*value*]...

where *type* is an LDAP name or an OID that corresponds to a naming attribute type and *value* is the string representation that corresponds to the attribute syntax assigned to the attribute type. The plus sign (**+**) is used to separate multiple AVAs within one RDN. For example:

```
c=de
```

or

```
2.5.6.2=de
```

or

```
ou=dap11+l=munich
```

When the name of the root entry is specified, the slash (**/**) must be used.

5.3. Search Filters

Use search filter expressions to specify a filter in a **metacp search** operation.A search filter is composed of one or more simple attributes, structured attributes, or distinguished name strings, and search filter operators.Specify a search filter in the following format:

([*logical_operator*](*type matching_operator value*)[(*type matching_operator value*) ...])

where:

logical_operator is one of the following operators:

Operator	Meaning
&	To "logically AND" two specified conditions
|	To "logically OR" two specified conditions
!	To "logically NEGATE" a specified condition

type specifies an LDAP name or an object identifier.

matching_operator is one of the following operators:

Operator	Meaning
=	To specify equality
~=	To specify phonetic matching
>=	To match values that are greater than or equal to a specified value
⇐	To match values that are less than or equal to a specified value

value specifies the attribute value in LDAP syntax. An asterisk (*****) is used to specify substrings or to check for the presence of an attribute.

No SPACE character is permitted between *type* and *matching_operator* and *matching_operator* and *value*.

5.3.1. Search Filter Expression Example

The following sample search filter string

```
(&((cn~=schmid)\
(|(objectClass=organizationalPerson)\
(objectClass=residentialPerson))\
```

```
(!(sn=ronnie))))
```

directs **metacp** to search for names that meet all the following criteria:

- Have an object class attribute value of **Organizational-Person** or **Residential-Person**
- Have a **Common-Name** attribute value that approximately matches **schmid**
- Do not have a Surname attribute value of **ronnie**.

The following search filter string tests for the presence of the **Common-Name** attribute type:

```
(&(c=de)(cn=*))
```

5.4. Reserved Attribute Characters

The following table describes reserved characters used for LDAP binds.(See Bind Types and Bind IDs in the metacp section for details.)

Character	Purpose
\{}	**For attributes and distinguished names:** Encloses the entire attribute (type and value) or distinguished name to indicate that white space is part of an attribute value. For example, **\{cn=Henry Mueller}** or **\{o=SNI AG, c=de}**
;	**For attributes**: Separates multiple values.
\	Escapes a reserved character.

RFC 2252 specifies additional reserved characters that also must be escaped in attribute values.

5.5. Attribute Syntax

The attribute syntax of all attribute types is treated as UTF-8 strings for LDAP binds.

5.5.1. Undefined Types

To specify an attribute type that has not been assigned an LDAP name in the directory schema, use the attribute type OID, for example, **1.2.325.67890.4.2**.

In the output, the **metacp** program returns a string in the form:

oid=attribute_value

For example:

1.2.325.67890.4.2=xyz

indicates that the attribute with the object identifier **1.2.325.67890.4.2** has a value of **xyz**.The value of the attribute type must be specified according to its attribute syntax.

5.6. String Representations for Simple Attribute Syntaxes

The following section describes the string representations for simple attribute syntaxes supported.All other simple attribute syntaxes not described in this section are treated as UTF-8 string.

5.6.1. Attribute Type Syntax

Specify an attribute type as an LDAP name (defined in the meta directory schema) or a dotted notation (for example, 1.2.5.6).

For LDAP names only the following characters are permitted: A to Z, a to z (case ignore), 0 to 9, and - (hyphen).

5.6.2. Bit String Syntax

Specify a bit string as a sequence of 1's and 0's enclosed by the character **'** and the character **B** appended (for example, **'**111101001000010011011O1**'B**).

5.6.3. Boolean Syntax

Specify a boolean as the string **TRUE** or **FALSE.**

5.6.4. Object ID Syntax

Specify an OID as an LDAP name (defined in the directory schema) or a dotted notation. For example, Organizational-Person could be specified as organizationalPerson or 2.5.6.7.

5.6.5. Generalized Time Syntax

Specify generalized time as a simple string. The value of the string is the concatenation of the 8 year-month-day digits (*YYYYMMDD*), plus the six hour-minute-second digits (*HHMMSS*), plus a time zone difference of **Z** (designating GMT), *+HHMM*, or *-HHMM*. The three possible forms, then, are:

- *YYYYMMDDHHMMSS***Z**
- *YYYYMMDDHHMMSS+HHMM*
- *YYYYMMDDHHMMSS-HHMM*

For example, **19970101123000Z** specifies Jan 1, 1997, 12:30:00 GMT.

The only difference between generalized time and UTC time is that the

> year is specified with four digits instead of two.

5.6.6. IA5 String Syntax

Specify an IA5 string using 7-bit ASCII characters, with valid Hex values in the range 20 to 7E (for example, smith@pqr.de).

5.6.7. Integer String Syntax

Specify as an integer in the range 0 to 2**32 -1 (4,294,967,295). For example, 65535.

5.6.8. Numeric String Syntax

Specify as a sequence of digits (0 to 9) and space (for example, 4711 13).

5.6.9. Preferred Delivery Method Syntax

Preferred-delivery-method syntax is a syntax for single-valued attributes that document the order of preference for message delivery methods. Specify this syntax in the following format:

preferredDeliveryMethod=option [**$***option*...]

option is one or more of the following keywords that describe the delivery methods and the order of preference:

- **any** - Any method of delivery
- **mhs** - Message handling system delivery
- **physical** - Physical delivery
- **telex** - Telex delivery
- **teletex** - Teletex delivery
- **g3fax** - G3 FAX delivery
- **g4fax** - G4 FAX delivery
- **ia5** - IA5 terminal delivery
- **videotex** - Videotex delivery
- **telephone** - Telephone delivery

The keywords are specified diminishing order of preference, with the most preferred method first in the list. Separate multiple keywords with a dollar sign (**$**). For example:

preferredDeliveryMethod=mhs$teletex$telephone

5.6.10. Printable String Syntax

Specify a printable string as a sequence of characters. Valid characters are

A to Z,
a to z,
0 to 9,
the space character,

and the special characters:

' (apostrophe),
((left parenthesis),
) (right parenthesis),
+ (plus sign),
, (comma),
- (hyphen),
. (period),
/ (slash),
: (colon),
= (equal sign),
? (question mark).

An example of Printable String Syntax follows:

```
Smith/PQR AG.
```

5.6.11. UTC Time Syntax

Specify UTC time as a simple string.The value of the string is the concatenation of the six year-month-day digits (*YYMMDD*), plus the six hour-minute-second digits (*HHMMSS*), plus a time zone difference of **Z** (designating GMT), *+HHMM*, or *-HHMM*.The three possible forms, then, are:

- *YYMMDDHHMMSS***Z**
- *YYMMDDHHMMSS+HHMM*
- *YYMMDDHHMMSS-HHMM*

For example, **9701011230002** specifies Jan 1, 1997, 12:30:00 GMT.

5.7. String Representations for Structured Attribute Syntaxes

This section describes the following structured attribute syntaxes supported by DirX Identity for LDAP binds.

- Syntaxes for schema attribute types
- Attribute-Type-Description
- Object-Class-Description
- Syntaxes for Message Handling System (MHS) attribute types

- OR-Address
- Syntaxes for miscellaneous attribute types and subcomponents
- Facsimile-Telephone-Number
- Name-And-Optional-UID
- Postal-Address
- Teletex-Terminal-Identifier
- Telex-Number

All attributes with a structured attribute syntax that is not described in this section must be specified in binary format for LDAP binds. (See the Binary Attribute Values section in this chapter for details.)

5.7.1. Attribute-Type-Description

An attribute syntax for directory schema attributes that specify attribute types. The attributeTypes attribute is an example of such attributes in the default directory schema:

Synopsis

attributeTypes=(*attribute_identifier*
[**NAME** [(]'*attribute_type_name*' [...)]]
[**DESC** '*attribute_type_description*']
[**OBSOLETE**]
[**SUP** *derivation*]
[**EQUALITY** *equality_matching_rule*]
[**ORDERING** *ordering_matching_rule*]
[**SUBSTR** *substrings_matching_rule*]
[**SYNTAX** *attribute_syntax* [{*length*}]]
[**SINGLE-VALUE**]
[**COLLECTIVE**]
[**NO-USER-MODIFICATION**]
)

Attribute Type

attributeTypes

The LDAP name or OID that corresponds to a structured attribute with the Attribute-Type-Description syntax. The LDAP Attribute-Types (attributeTypes) operational attribute, which specifies the attribute types used within the schema, has the Attribute-Type-Description attribute syntax. The attributeTypes attribute is multivalued; each value describes one attribute type. The information held in this attribute should be complete and in accordance with the registered definition of each attribute type. The attributeTypes attribute also provides the LDAP names of the attribute types.

Components

attribute_identifer

An object identifier (OID) that corresponds to an attribute type.

***NAME* 'attribute_type_name'**

A string (of up to 1024 characters long) that provides the LDAP name(s) for the attribute type. The values are enclosed in single quotation marks (**'**) and separated by a whitespace character. A list of LDAP names is enclosed in parentheses (**(** ... **)**). For LDAP names only the following characters are permitted: A to Z, a to z (case ignore), 0 to 9, and - (hyphen). A maximum of five LDAP names can be specified.

***DESC* 'attribute_type_description'**

A UTF-8 string (of up to 1024 characters long) that describes the attribute type. The value is enclosed in single quotation marks (**'**).

OBSOLETE

The keyword **OBSOLETE** indicates that the attribute type is no longer supported (but its characteristics are maintained). If an attribute type is deleted it is set to **OBSOLETE**, that is this attribute cannot be added to an entry or modified. (The error **Unwilling to Perform** is returned.) The values of obsolete attributes are returned by search and read operations. The default is that the specified attribute type is supported and the keyword **OBSOLETE** is omitted.

***SUP* derivation**

Specifies the attribute type LDAP name or OID that corresponds to the attribute type of which this attribute is a subtype. This component is used only for attributes defined with a supertype.

***EQUALITY* equality_matching_rule**

Specifies the LDAP name or OID of an equality matching rule. This is an optional component, but it should be specified for attributes defined with equality matching rules.

***ORDERING* ordering_matching_rule**

Specifies the LDAP name or OID of an ordering matching rule. This is an optional component, but it should be specified for attributes defined with ordering matching rules.

***SUBSTR* substrings_matching_rule**

Specifies the LDAP name or OID of a substrings matching rule. This is an optional component, but it should be specified for attributes defined with substrings matching rules.

SYNTAX* attribute_syntax *[{ length }]

Specifies the OID of the attribute type syntax for use with LDAP, and an optional indication of the maximum length *length* of a value of this attribute.

SINGLE-VALUE

The keyword **SINGLE-VALUE** indicates that the attribute type is single-valued. The default is that the specified attribute type is multivalued and the keyword **SINGLE-VALUE** is omitted.

COLLECTIVE

The keyword **COLLECTIVE** indicates that the attribute type is a collective attribute. The default is that the specified attribute type is not collective and the keyword **COLLECTIVE** is omitted.

NO-USER-MODIFICATION

The keyword **NO-USER-MODIFICATION** indicates that the attribute type is not modifiable by users. The default is that the specified attribute type is user modifiable and the keyword **NO-USER-MODIFICATION** is omitted.

USAGE **usage**

Specifies how the attribute is to be used. This value is one of the following keywords:

- **userApplications** - For normal user attributes
- **directoryOperation** - For attributes used by the directory server as part of non-distributed operations (for example, timestamps, access control attributes)
- **distributedOperation** - For operational attributes used by several directory servers as part of distributed operations (e.g., knowledge-reference attributes)
- **dSAOperation** - For operational attributes that are used purely locally to the directory server

The default value is **userApplications**.

Description

The LDAP Attribute-Types attribute is provided only to permit an LDAP server to publish the static details of the attributes that it supports within its schema.

The components of this syntax are separated by a whitespace character.

For each attribute type either the SUP or the SYNTAX component must be specified.

Examples

```
attributeTypes=
....
( 2.5.4.20 NAME 'telephoneNumber' EQUALITY telephoneNumberMatch
SYNTAX 1.3.6.1.4.1.1466.115.121.1.44 {32} )
....
```

5.7.2. Object-Class-Description

An attribute syntax for schema attributes that specify object classes. The objectClasses attribute is an example of such attributes in the default directory schema.

Synopsis

objectClasses=(*object_class_identifier*
[**NAME** '*object_class_name*']
[**DESC** '*object_class_description*']
[**OBSOLETE**]
superior_object_class[**$** ...)]]
[*kind*]
attribute_type [**$** ... **)**]]
attribute_type [**$** ...**)**]]**)**

Attribute Type

objectClasses

The LDAP name or OID that corresponds to a structured attribute with the Object-Class-Description syntax. The LDAP Object-Classes (objectClasses) operational attribute is a multi-valued attribute of the schema used to describe which object classes are supported by the directory server. Each value describes one object class. The objectClasses attribute provides the LDAP names of the object classes.

The information specified for this attribute should be complete and in accordance with the registered definition of each object class.

Components

object_class_identifer

An OID that corresponds to an object class.

***NAME* 'object_class_name'**

A string (of up to 1024 characters long) that corresponds to LDAP name given to the object class. Only one LDAP name is provided. It is enclosed in single quotation marks (**'**). For LDAP names only the following characters are permitted: A to Z, a to z (case ignore), 0 to 9, and - (hyphen).

***DESC* 'object_class_description'**

A UTF-8 string (of up to 1024 characters long) that describes the object class. The value is enclosed in single quotation marks (**'**).

OBSOLETE

The keyword **OBSOLETE** indicates that the object class is no longer supported (but its characteristics are maintained). If an object class is deleted it is set to **OBSOLETE**, that is this enrties of this object class cannot be added or modified. (The error **Unwilling to Perform** is returned.) Entries of obsolete object classes are returned by search and read operations. The default is that the specified object class is supported and the keyword **OBSOLETE** is omitted.

***SUP* derivation**

Specifies the LDAP names or OIDs that corresponds to the object classes (if any) that are the superclass for this object class. A list of object classes is enclosed in parentheses and the object classes are separated by a **$** character.

kind

Specifies the kind of object class that the object class registration specifies. The value is one of the following keywords:

- **STRUCTURAL** - Represents a real-world object (for example, device, organization) that is concrete enough to have a place in the DIT
- **AUXILIARY** - Descriptive of real-world objects (usually being applicable to more than one)
- **ABSTRACT** - Represents an abstraction of real-world objects which does not exist in it own right

***MUST* attribute_types**

Specifies one or more LDAP names or OIDs that correspond to attributes that are registered as mandatory for entries of this object class. A list of attributes is enclosed in parentheses and the attribute types are separated by a **$** character.

***MAY* attribute_types**

Specifies one or more LDAP names or OIDs that correspond to attributes that are registered as optional for entries of this object class. A list of attributes is enclosed in parentheses and the attribute types are separated by a **$** character.

Description

The Object-Class-Description permits a LDAP server to publish the static details of the object classes it supports.

The components of this syntax are separated by a whitespace character.

The sequence of the components must be provided as specified in the synopsis section above.

Examples

```
objectClasses=
....
( 2.5.6.16 NAME 'certificationAuthority' SUP top AUXILIARY MUST
(cACertificate $ certificateRevocationList $ authorityRevocationList)
MAY crossCertificatePair )
....
```

5.7.3. OR-Address

An attribute syntax for attributes that specify X.400 originator/recipient (OR) addresses.

Synopsis

ORAddress=[**/CN=***common_name*]
[**/I=***initials*]
[**/Q=***generation_qualifier*]
[**/G=***given_name*]
[**/S=***surname*]
[**/OU4=***organizational_unit_4*]
[**/OU3=***organizational_unit_3*]
[**/OU2=***organizational_unit_2*]
[**/OU1=***organizational_unit_1*]
[**/O=***organization*]
/PRMD=*PRMD*
/ADMD=*ADMD*
/C=*country*

Attribute Type

ORAddress

The LDAP name or OID that corresponds to a structured attribute with the OR-Address syntax. For example, mhsOraddresses is such an attribute. An OR address comprises a number of standard components and a number of optional components defined by the management domain to which the originator/recipient subscribes (know as domain defined components). In most cases only the standard components **C**, **ADMD**, **PRMD**, **O**, **OU1** through **OU4**, **S**, **G**, **Q** and **I** are used.

Components

***CN*=common_name**

The common name in Printable string format.

***I*=initials**

The name initials in Printable string format.

***Q*=generation_qualifier**

The generation qualifier in Printable string format.

***G*=given_name**

The given name in Printable or string format.

***S*=surname**

The surname in Printable string format.

O=*organization*
OU1=*organizational_unit_1*
OU2=*organizational_unit_2*

OU3=*organizational_unit_3*
OU4=*organizational_unit_4*

An identifier of the organization and up to 4 organizational units in Printable string format.

***PRMD*=PRMD**

An identifier of the Private Management Domain in either Printable or Numeric string format.

***ADMD*=ADMD**

An identifier of the Administrative Management Domain in either Printable or Numeric string format.

***C*=country**

An identifier of the country in either Printable or Numeric string format.

Description

Information about X.400 originator/recipient addresses is represented by the OR-Address attribute syntax.

The components of this attribute syntax are separated by a slash (**/**).

Example

```
mhsOraddresses=/G=Irmgard/S=Hohner/OU2=S41/OU1=MCH1/O=PQR/PRMD=PQR
/ADMD=DBP/C=de
```

5.7.4. Facsimile-Telephone-Number

An attribute syntax for attributes that specify Facsimile (FAX) numbers.

Synopsis

FacsimileTelephoneNumber=phone_number
[**$***fax_parameters*]

Attribute Type

FacsimileTelephoneNumber

The LDAP name or OID that corresponds to a structured attribute with the Facsimile-Telephone-Number syntax. For example, facsimileTelephoneNumber is such an attribute.

Components

phone_number

The fax number in Printable string format (of up to 32 characters long).

fax_parameters

The settings for the G3 Fax parameters. Note that setting G3 parameters is hardly ever required. Specify *fax_parameters* in the following format:

fax_parameter[**$***fax_parameters*]

where *fax_parameter* is one of the following identifiers:

- **twoDimensional**
- **fineResolution**
- **unlimitedLength**
- **b4Length**
- **b4Width**
- **a3Width**
- **uncompressed**

Description

FAX numbers are represented with the Facsimile-Telephone-Number attribute syntax. The number consists the country code and number in Printable string format. You can also set G3 non-basic parameters.

The components of this attribute syntax are separated by a **$** character.

Example

```
facsimileTelephoneNumber=+49 89 12345
```

5.7.5. Name-And-Optional-UID

An attribute syntax for attributes that identify objects by a distinguished name and an optional identifier that can remove ambiguity from names that have been re-used.

Synopsis

NameAndOptionalUID=distinguished_name
[**#***BIT_string*]

Attribute Type

NameAndOptionalUID

The LDAP name or OID that corresponds to a structured attribute with the Name-And-Optional-UID syntax. For example, uniqueMember is such an attribute. The

uniqueMember attribute can be used in objects which represent lists of names.

Components

distinguished_name

The distinguished name of the object. (See the Distinguished Names section in this chapter for details.)

Although the **#** character is used as separator for the components of this syntax and it may occur in a string representation of a *distinguished_name*, no additional special quoting is done.

BIT_string

A bit string that uniquely identifies the object. The bit string can be used when an object is removed from the Directory, and another object is subsequently given the same Directory name. The bit string must be chosen for uniqueness, such as a time-stamp (for example). (See the Bit String Syntax section in this chapter for details.)

Description

The identity of objects can be represented with the *Name-And-Optional-UID* attribute syntax. The attribute consists of the object distinguished name and an optional bit string that distinguishes between objects with the same distinguished name. The association between a user of the Directory and a name when the UID is present can be established by strong authentication using Version 2 (or later) certificates.

The components of this attribute syntax are separated by a **#** character.

Examples

```
uniqueMember=o=sni,c=DE#'100'B
```

5.7.6. Postal-Address

A structured attribute syntax for postal addresses.

Synopsis

PostalAddress=postal-address-string1
[**$***postal-address-string2*]
[**$***postal-address-string3*]
[**$***postal-address-string4*]
[**$***postal-address-string5*]
[**$***postal-address-string6*]

Attribute Type

PostalAddress

The abbreviation that corresponds to a structured attribute with the Postal-Address syntax. For example, postalAddress is such an attribute.

Components

postal-address-stringn

A UTF-8 string (of up to 30 characters long) that provides a line of a postal address.

Backslashes (\) and **$** characters, if they occur in a component, are escaped by using an additional backslash, for example the string **A\$** represents the value **A$** in a component.

Description

Postal addresses are represented with the Postal-Address attribute syntax. The attribute consists of up to 6 address lines in UTF-8 string format. Each address line is limited to 30 characters in (in accordance with ITU-T recommendation F.401). One address line is required; the remaining 5 are optional.

The components of this attribute syntax are separated by a **$** character.

Example

```
postalAddress=PQR AG$Sales Dpt$Einstein-Ring 4$D-81789 Munich$Germany
```

5.7.7. Teletex-Terminal-Identifier

An attribute syntax for attributes that specify a Teletex terminal.

Synopsis

TeletexTerminalIdentifier=teletex-terminal
[**$***teletex-non-basic-parameters*]

Attribute Type

TeletexTerminalIdentifier

The LDAP name or OID that corresponds to a structured attribute with the Teletex-Terminal-Identifier syntax. For example, teletexTerminalIdentifier is such an attribute.

Components

teletex-terminal

A string (of up to 1024 characters long) that identifies the terminal. Specify the *teletex-terminal* parameter using the Printable String syntax. See the Simple Attribute Syntax section in this chapter for a description of Printable String syntax representation.

teletex-non-basic-parameters

A component that sets non-basic parameters for various teletex options. Specify *teletex-non-basic-parameters* in the following format:

[**$control:***control_character_sets*]
[**$graphic:***graphic_character_sets*]
[**$misc:***miscellaneous_capabilities*]
[**$page:***page_formats*]
[**$private:***private_use*]

where:

control:*control_character_sets* specifies a string that defines the control character sets to use.

graphic:*graphic_character_sets* specifies a string that defines the graphic character sets to use.

misc:*miscellaneous_capabilities* specifies a string that defines miscellaneous capabilities to use.

page:*page_formats* specifies a string that defines the page formats to use.

private:*private_use* specifies a string that defines user-defined Teletex parameters.

Description

Teletex terminals and their parameters are represented by the Teletex-Terminal-Identifier attribute syntax. The information consists of a printable string that identifies the terminal and optional non-basic, advanced parameters that control terminal parameters such as character sets, page formats and other capabilities.

The components of this attribute syntax are separated by a **$** character.

Example

```
{teletexTerminalIdentifier=PQR AG Teletex center$page:letter}
```

5.7.8. Telex-Number

An attribute syntax for attributes that specify Telex numbers.

Synopsis

TelexNumber=telex_number
[**$***country_code*]
[**$***answer_back*]}

Attribute Type

TelexNumber

The LDAP name or OID that corresponds to a structured attribute with the Telex-Number syntax. For example, telexNumber is such an attribute.

Components

telex_number

The telex number in Printable string format (of up to 14 characters long).

country_code

The county code in Printable string format (of up to 4 characters long).

answer_back

The short textual string (of up to 8 characters long), in Printable string format, with which the telex station responds when required to indicate its identity. (For example, when the telex station is connected to.)

Description

Telex numbers are represented with the Telex-Number attribute syntax. The number consists of the country code, Telex number, and answer-back code.

The components of this attribute syntax are separated by a **$** character.

Example

```
{telexNumber=24344$046$GAMEX B}
```

6. DirX Identity Program Files

The command-line programs **metacp**, **metacpdump** and **metahubdump** use the following files:

- The DirX Identity client log and trace message configuration files (*install_path***/client/conf/dirxlog.***). The logging configuration files control the trace and exception messages that **metacp** logs. The **metacpdump** program is used to interpret the trace message section of the generated log files. The default location of the generated log files is *install_path***/client/log**.
- The Directory client configuration file (**dirxcl.cfg**). The **metacp** program uses this file to locate Identity stores during a bind operation.
- The Identity controller (**metacp**) SSL/TLS certificate database (**cert8.db**). The **metacp** program uses this file to determine whether it can trust certificates sent from directory server when a bind operation specifies the use of SSL/TLS protocol. This file also contains the user certificates sent from directory clients when a bind operation specifies the use of certificate-based client authentication (SASL EXTERNAL).
- The Identity controller (**metacp**) SSL/TLS key database (**key3.db**). The **metacp** program uses this file to access its private key when a bind operation specifies the use of certificate-based client authentication (SASL EXTERNAL).
- IDMS configuration and key material files. The IDMS configuration and key material files contain the parameters for SSL/TLS initialization when the DirX DSA, LDAP server or **metacp** program use the encrypted variant of the X.500 protocols over IDM (IDMS) and the necessary key material files.

This chapter describes these files.

6.1. Logging Configuration Files for metacp

dirxlog.on
dirxlog.off
dirxlog.cfg

Purpose

Logging configuration files define the messages to be logged and how and where the logs should be written.Two types of messages can be logged:

- **Exception Messages** - ASCII format messages that are human-readable. The directory service always logs exception messages for both servers and clients.
- **Trace Messages** - Binary format messages that must be read by the **metacpdump** command. Trace logging can be turned on and off explicitly.

Description

DirX Identity client logging (for example, **metacp**) is controlled by the specifications contained in the *install_path***/client/conf/dirxlog.cfg** file. When DirX Identity is installed, the

following configuration files specifying default values are installed for client logging:

*install_path***/client/conf/dirxlog.cfg**

(enables only client exception logging)

*install_path***/client/conf/dirxlog.on**

(enables default client trace and exception logging)

*install_path***/client/conf/dirxlog.off**

(enables only client exception logging)

The client log files are written to the directory *install_path***/client/log**.

Enabling Logging

- Enable client logging:

  Before starting the application, copy the configuration file *install_path***/client/conf/dirxlog.on** to the file *install_path***/client/conf/dirxlog.cfg**.
- Disable client logging:

  Before starting the application, copy the configuration file *install_path***/client/conf/dirxlog.off** to the file *install_path***/client/conf/dirxlog.cfg**.

The **.on** and **.off** files for DirX Identity client logging are installed in *install_path***/client/conf**.

You can modify the **.on** and **.off** files to define your desired logging parameters, and you can create additional files that function in a manner similar to the **.on** and **.off** files and manually maintain the copy from **dirxlog.cfg** to these files.

You can override the default file name for the logging configuration file by setting the **DIRX_LOGCFG_FILE** environment variable to the full pathname of the file. This is NOT recommended because **metacp**, the Server (IDS-C) and the DirX direxctory server will read the same logging configuration file if they are restarted.

Use the logging configuration **.on** and **.off** files to perform the following tasks:

For trace logging:

Turn on client trace logging and specify the traces to be logged and the destinations for the log files

Turn off client trace logging for all or selected directory subcomponents

For exception logging:

Override the default exceptions logged for clients and the default destination for the log file

Turn client exception logging off for all or selected directory subcomponents

File Formats

All logging configuration files consist of two sections: one that defines trace logging and one that defines exception logging. The following subsections describe the format of the trace logging and exception logging sections.

The Trace Logging Section

The trace logging section defines the component and subcomponents for which to capture trace messages and the debug level of the traces to log. It consists of a line in the format:

component:*subcomponent*.*level*,[...]:
process.*max_num_files*.*max_num_entries*:_file_name_

where:

component

Is the name of the component to log. All directory log file entries are currently associated with the **dir** component name. All DirX Identity log file entries are currently associated with the **mdi** component name.

subcomponent

Is the name of the subcomponent to log and the debug level of the traces to capture. The valid values for *subcomponent* are listed in the table that follows.

You can use a wildcard character (*****) to specify all subcomponents. In this case, however, the debug levels you specify with *level* are used for all subcomponents.

The **mdi** component supports the following DirX Identity subcomponent names:

Value	Meaning
meta	Dir Identity client interface

The **dir** component supports the following directory subcomponent names:

Value	Meaning
adm	Administration
api	Application interface
bth	Bind table handling
ctx	Context-specific memory interface
icom	Internal thread communication interface
osi	OSI communication

Value	Meaning
ros	Remote operation service
sock	Socket interface
sys	System call interface
util	Utility functions
vthr	Virtual thread interface

level

Specifies the debug level of the traces to log. *Level* is an integer or list of integers ranging from **0** through **9.** You can specify debug levels:

- As a range. For example, **1-5** indicates debug levels 1 through 5 inclusive
- As individual levels separated by a period. For example, **1.3.4.5** indicates levels 1, 3, 4, and 5
- As a combination of ranges and individual levels. For example **1.3-5** indicates level 1, 3, 4, 5.

The integers **0** and **9** have special meanings. Level **0** disables logging for the subcomponent. (Not listing the subcomponent at all has the same effect.) Level **9** causes the structures in a trace function to also be evaluated for each log level specified; that is, the contents of the interface parameters defined as structures are also logged. (Note that the structure logging can create extremely large log files.) You can turn on level **9** for each valid value for *level*.

Levels **1-8** log the contents of the other parameters. Level **1** logs the top-level functions, while the higher log levels determine the depth of the internal functions that are logged. For example, level **6** traces the function call *foo* with parameters. If levels **6** and **9** are specified, function call *foo* is traced with parameters and with the structures of those parameters.

You can specify multiple *subcomponent*.**level*** **entries, separating each with a comma and using a backslash (\) to continue the line. Without the backslash, the *NEWLINE** character terminates the line.

There are the following level values for the component **mdi**:

Sub-component	Valid value for level	Usage
meta	1	Interface functions of the DirX Identity client interfaces
	2	Internal interface functions of the DirX Identity client interfaces

There are the following level values for the component **dir**:

Sub-component	Valid value for level	Usage
adm	1	**metacp** (Administration) translator functions
	2	**metacp** internal functions for abandon operation
	4	LDAP client functions, such as BIND, SEARCH, CREATE etc.
	5	LDAP functions for extracting names/attributes from a search result
	6	LDAP functions for freeing internally used LDAP memory
	7	OCL functions (OSS Convenience library, string parsing)
	8	DAM functions (dir abbreviation module, that is functions for reading and writing the abbreviation file and validation of abbreviations)
api	1-2	Interface functions and internal functions of application interface
bth	1	Bind table handling functions (binding entries handling)
	2	Operation entries handling
	3	Subscriber handling
	4	lock / unlock functions
	5	resume_info handling
	6	bind_waiter handling
ctx	1	CTX package (context-specific memory) interface functions related to context creation/deletion
	2	CTX package interface functions related to context memory allocation/release
	3-6	CTX package internal functions
osi	1	Incoming/outgoing event-handling functions
	3	TP routines

Sub-component	Valid value for level	Usage
ros	1	RTROS interface functions
	3	RTROS internal functions
	4	CMX interface functions
	6	AMS compiler logging
	7-8	AMS (NDS) library functions
sock	1	Socket interface functions (connection establishment)
	2	Data transfer
	3	Event generation
	4	Name handling and utility functions
sys	1	System calls (2)
	2	System library functions (3C)
	3	System library functions (3C) related to memory allocation
	4	WIN32 library functions
util	1	Utility functions
vthr	1	VT package (Virtual thread) interface functions related to threads creation/termination
	2-3	VT package interface functions related to threads controlling
	4	VT package interface functions related to mutexes
	5	VT package interface functions related to condition variables
	6	VT package interface functions related to thread specific data
	7	VT package internal functions

process

Indicates how to store the log entries. Valid values for *process* are:

DISCARD - Do not write any log entries.

BINFILE - Write entries to a binary file. You must supply the name of the file. (See the *file_name* parameter description.) BINFILE must be used for the **dir** component

TEXTFILE - Write entries to a human-readable text file. You must supply the name of the file. (See the *file_name* parameter description.)

STDOUT - Write entries in human-readable text to standard output.

STDERR - Write the entries in human-readable text to standard error.

GOESTO:*severity*...-Write the entries to same destination as the messages of the specified component. You can specify multiple severities by separating each one with a comma. *severity* can be:

- **FATAL** - Fatal errors
- **ERROR** - Non-fatal errors
- **WARNING** - Warnings
- **NOTICE** - Informational notices
- **NOTICE VERBOSE** - Verbose informational notices

max_num_files

The maximum number of log files to create (a number from 01 to 99 inclusive). When the first log file reaches its capacity (specified by *max_num_entries*), a second file is written and so on until the number of files specified by *max_num_files* is reached. When the maximum number of files is reached, the next file overwrites the first file, and the next the second file, and so on. If multiple files are written, the sequence number of the file (a number in the range 01 to the number specified by *max_num_files*) is appended to the file name.

max_num_entries

The maximum number of entries to write to the file.

file_name

The full pathname of the file in which to store the entries. You can use the **%s** variable to insert the pathname of the base directory in which DirX Identity is installed in the file name. (See the description of **$DIRXMETAHUB_INST_PATH** in the DirX Identity Environment Variables chapter for details.) You can use the **%d** variable to insert the ID of the currently running process in the file name. For example, **LOG%d** creates a file named **LOG** appended with the process ID. You must supply *file_name* only if you supply a *process* parameter of BINFILE, TEXTFILE, and FILE.

Sample Trace Section Lines

The following sample line uses the wildcard character to specify logging for all subcomponents. It captures debug messages of levels 1-5 and level 9. The **%s** variable is used to insert the base directory in which DirX Identity is installed, for example C:/Program Files/Atos/DirX Identity in the file name. The **%d** variable is used to store the log messages in a file named LOG*process_id*:

```
dir:*.9.1-5:BINFILE.2.200:%s/client/log/LOG%d
```

The following sample specifies the logging of level 1 messages from the **adm** and **api** subcomponents and of level 2 messages from the **sys**, **vthr**, and **util** components:

```
dir:adm.1,api.1,\
sys.2,vthr.2, util.2:\
BINFILE.2.200:C:/Program Files/Atos/DirX Identity/client/log/LOG%d
```

In both of the sample lines above, two binary log files are created. The first 200 log entries are written to the file **C:/Program Files/Atos/DirX Identity/client/log/LOG*_process_id_**.01* and the second 200 log entries are written to the file **C:/Program Files/Atos/DirX Identity/client/log/LOG*_process_id_**.02*. Then **LOG*_process_id_**.01* is overwritten with the next 200 log entries, then **LOG*_process_id_**.02* is overwritten and so on.

Depending on the logging configuration, the binary log files can become very big. It is recommended to configure only the log levels needed, for instance:

```
ros.4.9    in order to trace all DAP, DSP or DISP-PDUS
vthr.1    in order to trace the interthread communication
```

The Exception Logging Section

The exception logging section of the logging configuration files defines the severity of exceptions to log and how and where to log them. It consists of lines in the following format:

severity:*process*.*max_num_files*.*max_num_entries*:*file_name*

where:

severity
: Specifies the severity of the exception to store. Valid values are:

  FATAL - Fatal errors

  ERROR - Non-fatal errors

  WARNING - Warnings

  NOTICE - Informational notices

  NOTICE VERBOSE - Verbose informational notices

process
: Indicates how to handle the log entries for the specified severity. Valid values, which can be used with any of the severities listed above, are:

  DISCARD - Do not write any log entries.

BINFILE - Write entries to a binary file. You must supply the name of the file. (See the *file_name* parameter description.) BINFILE must be used for the **dir** component

TEXTFILE - Write entries to a human-readable text file. You must supply the name of the file. (See the *file_name* parameter description.)

STDOUT - Write entries in human-readable text to standard output.

STDERR - Write the entries in human-readable text to standard error.

GOESTO:*severity_or_component*...-Write the entries to same destination as the entries of the specified severity or component. You can specify multiple severities or components by separating each one with a comma.

max_num_files

The maximum number of log files to create (a number from 01 to 99 inclusive). See the *max_num_files* parameter in "The Trace Logging Section" for more details.

max_num_entries

The maximum number of entries to write to the file.

file_name

The full pathname of the file in which to store the entries. See the *file_name* parameter in "The Trace Logging Section" for more details.

Sample Exception Lines

The following example specifies that:

- Exceptions with severity of FATAL are stored in two places. The first is a text file with up to 100 entries named */usr/tmp* **/EXC***process_id*. Only one such file is created. When the file reaches 100 entries it is overwritten. The second is not shown in the example. It is the file where the traces for the **dir** components are stored.
- Exceptions with a severity of ERROR are stored in three places. The first is a text file with up to 100 entries named */usr/tmp***/USR***process_id*. Only one such file is created. When the file reaches 100 entries it is overwritten. The second is the same place where exceptions with a severity of FATAL are stored. The third is the same place where traces for the **dir** component are stored.
- Messages with a severity of NOTICE are stored in the same place as messages with a severity of ERROR.

```
FATAL:TEXTFILE.1.100:/usr/tmp/EXC%d;GOESTO:dir
ERROR:TEXTFILE.1.100:/usr/tmp/USR%d;GOESTO:FATAL
NOTICE:GOESTO:ERROR
```

Examples

In the examples that follow, lines beginning with the # character are comment lines used

to differentiate the trace logging section from the exception logging section.

The following is a sample **dirxlog.on** file.

```
# Trace Logging
dir:sys.1,sock.1.9,vthr.1,ctx.0,\
ros.1,\
bth.1,\
util.1-2:BINFILE.6.2000:/usr/tmp/LOG%d
#
# Exception Logging
FATAL:TEXTFILE.1.100:/usr/tmp/EXC%d;GOESTO:dir
ERROR:TEXTFILE.1.100:/usr/tmp/USR%d;GOESTO:FATAL
NOTICE:GOESTO:ERROR
WARNING:GOESTO:FATAL
NOTICE_VERBOSE:GOESTO:FATAL
```

The following is an example of a **dirxlog.off** file. Note that the first line of the file turns trace logging off for the **dir** component. (Exception logging is not disabled.) In addition, the dash is used as a placeholder and indicates an empty field.

```
# Trace Logging
dir::DISCARD:-
#
# Exception Logging
FATAL:TEXTFILE.1.100:/usr/tmp/EXC%d
ERROR:TEXTFILE.1.100:/usr/tmp/USR%d;GOESTO:FATAL
NOTICE:GOESTO:ERROR
WARNING:GOESTO:FATAL
NOTICE_VERBOSE:DISCARD:-
```

See Also

metacpdump

6.2. Directory Client Configuration File

dirxcl.cfg

Purpose

The directory client configuration file **dirxcl.cfg**, used by **metacp**, defines the Identity stores that are available to the meta controller.

Description

The default location for the **dirxcl.cfg** file is:

install_path/**client/conf**

You can override the default location by setting the **DIRX_CLCFG_FILE** environment variable to the full path name of the file.

In the directory client configuration file, comment lines are lines that begin with a # character in the first column and are ignored.

Other than comments, the file contains two types of lines, as follows:

- A mandatory line for the address of the meta controller
- Optional Identity store (LDAP server) address lines.

The meta controller address line starts with the word **self** (case insensitive), followed by the address of the meta controller in valid PSAP address format.

A Identity store address line is specified in the following format:

name address protocol

where:

name
is the symbolic name of the Identity store. This name is the name to be specified in the **obj bind** command in the **-server** option. (See the obj bind operation for details.)

address
is the real address of the Identity store in one of the following formats:

- *host*[:*port*]
- *host1*[:*port1*],*host2*[:*port2*][,...]

where *host* or *hostn* is either an IP address or a DNS (domain name server) name and *port* or *portn* is a port number (<32767). **389** is used as the default port number.

If more than one real address is specified, an attempt is made to establish a connection using real addresses from left to right.

protocol
is the protocol to be used for the LDAP bind. Supply one of the following keywords (case-insensitive):

- **LDAPv2** - LDAP version 2
- **LDAPv3** - LDAP version 3

The protocol specification can be overridden by **-protocol** option in the **obj bind** command. (See the obj bind operation for details.)

The first Identity store address line identifies the default Identity store.

Anything following and separated from the protocol specification in a Identity store address line with a space is ignored.

If no Identity store address line is specified, the server address must be specified in the **obj bind** command in the -**address** option for an LDAP server. (See the obj bind operation for details.)

If **LDAPv2** or **LDAPv3** is specified in the **-protocol** option of an **obj bind** operation and no LDAP server address line is specified in the client configuration file **dirxcl.cfg** a bind to the local Identity store is established. (See the obj bind operation for details.)

If you use environment variables to override a **dirxcl.cfg** file entry and an error occurs (for example, if **DIRX_CLCFG_FILE** is set to a nonexistent file or is incorrectly specified), the meta controller reports the error, but makes no attempt to return to the overridden default. If errors occur, consult the exception log file for a description of the error.

Sample dirxcl.cfg File

```
# any line starting with '#' is a comment and is ignored
# any line starting with 'self' (case insignificant) contains
# the psap of the meta controller - there must be 1 such line
# further lines contain Identity store names and addresses;
# the first is the default
# the next line is the psap of the meta controller itself - can be
first, # last or anywhere
Self TS=Client1,NA='TCP/IP!internet=139.23.81.32+port=2555'
# LDAPv3 Server
hawk hawk.virt.de.com LDAPv3
# LDAPv2 Server
eagle eagle.virt.de.com,hawk@virt.de.com LDAPv2
```

6.3. SSL/TLS Certificate Database

cert8.db

Purpose

The SSL/TLS certificate database, used by **metacp**, defines the server and certificate authority (CA) certificates that the meta controller is to trust when authenticating a server during a bind operation with SSL/TLS enabled.It also contains the client certificate that the meta controller is to use when authenticating clients during a bind operation with SASL EXTERNAL enabled.

Description

The default location for the **cert8.db** certificate database is:

install_path/**client/conf**

You can override the default location and file name by setting the **DIRX_TRUSTED_CA** environment variable to the full pathname of the file.

When an LDAP client connects to an LDAP server over SSL/TLS, the LDAP server authenticates itself by sending its certificate to the LDAP client. The LDAP client needs to determine whether the certificate authority (CA) that issued the certificate is trusted.

The LDAP client (**metacp**) uses the Mozilla SDK to implement SSL/TLS support. The security protocols and versions supported by the SDK are SSL V2 and SSL V3. The Identity store supports SSL V3 and TLS V1. Consequently, if **metacp** connects to an LDAPv3 server, the two programs will agree on SSL V3 during the negotiation of the security protocol to use.

The Mozilla SDK API requires a certificate database to hold the CA certificate. The prerequisites for connecting over SSL are as follows:

1. The meta controller has access to a Mozilla certificate database. This database must be the **cert8.db** database file. Note that previous and later versions of these applications use different file formats for the certificate database. Attempting to use a different version of the certificate database will result in certificate database errors. The LDAP API function called by **metacp** uses this certificate database to determine if it can trust the certificate sent from the server.
2. The certificate database that you are using can contain:
    - The certificate of the certificate authority (CA) that issued the LDAP server's certificate
    - The certificates of all the CAs in the hierarchy, if the certificate authorities (CAs) are organized in a hierarchy
    - The certificate of the LDAP server
    - User certificates
3. The CA certificate is marked as "trusted" in the certificate database.

When **metacp** sends an initial request to the secure LDAP server, the LDAP server sends its certificate back to **metacp**. **metacp** then determines which CA issued the LDAP server's certificate and searches the certificate database (**cert8.db**) for the certificate of that CA.

If **metacp** cannot find a trusted certificate of the server or the issuer it refuses to connect to the server. It also refuses to connect if the LDAP server's certificate has expired or if the certificate extension restricts the usage to client usage only. The latter applies if the LDAP server sends a browser certificate instead a server certificate.

The DirX Identity installation contains an example **cert8.db** file containing the CA certificate of the "Test CA" that issued all test keymaterial. It includes also the user certificate for **cn=mayer,ou=sales,o=my-company** for the purpose of testing SASL binds.

Certificate Administration

metacp clients use the Mozilla LDAP SDK to implement the LDAP stub and the SSL Security support.In order to manipulate the certificate databases read by **metacp** you need to use the utility program certutil provided as part of the SDK.

You may download the sources and binaries from the Mozilla web site

https://developer.mozilla.org/en-US/docs/Mozilla/Projects/NSS

For a complete documentation on the certutil command line tool see

https://developer.mozilla.org/en-US/docs/Mozilla/Projects/NSS/tools/NSS_Tools_certutil

6.4. SSL/TLS Key Database

key3.db

Purpose

The key database, used by **metacp** when a SASL bind operation with the EXTERNAL mechanism is performed, contains the data required to access the private key that corresponds to the client certificate.

Description

The default location for the **key3.db** key database is:

install_path/**client/conf**

You can override the default location and file name by setting the **DIRX_KEY3DB_FILE** environment variable to the full pathname of the file.

When an LDAP client (in this case, **metacp**) binds to an LDAP server (in this case, the Identity store) using the ldap_sasl mechanism (see the obj bind operation for details), **metacp** accesses

- the **cert8.db** certificate database in order to retrieve the client's user certificate
- the **key3.db** key database in order to retrieve the client's private key.

Use the Mozilla command line tool certutil to add Client PSEs to the **cert8.db** database and to the **key3.db** key database files. **Note:** The newer certutil tool targets per default **cert9.db** and **key4.db** files.Use the option **-d** to work with **cert8.db** and **key3.db** files.See the given documentation link above.

The DirX Identity installation contains an example **key3.db** file containing a private key for testing purposes for the user with the nickname "mayer".It is protected with the passphrase "dirxdirx".See the description of the **-sasl** option of the **metacp** operation **obj bind**.

6.5. IDMS Configuration and Key Material Files

idmssl.cfg
*own_keymaterial***.pem**
*password_file***.pwd**
*trustedCA***.pem**

Purpose

The encrypted IDM (IDMS) configuration file **idmssl.cfg** specifies the parameters for the SSL initialization that is needed to perform the X.500 protocols over IDMS, the encrypted variant of IDM.The ***.pem** and **.pwd** files contain the necessary key material.

Keep in mind that in DirX Identity installations, **metacp** normally uses LDAP protocol. The information about **idmssl.cfg** is only needed in the very rare case when a customer still runs workflows or any other Tcl scripts using DAP protocol (over IDMS rather than OSI). For more details (for example, on how to create the appropriate ***.pem** files), please refer to the DirX documentation.

Description

Each DirX process that uses IDMS has its own configuration file (**idmssl.cfg**) located in a specific configuration (/**conf**) folder:

- For the DSA (**dirxdsa** process), the file is located in *install_path***/server/conf/idmssl.cfg**. (This information is only given here for completeness.)
- For the LDAP server (**dirxldapv3** process), the file is located in *install_path***/ldap/conf/idmssl.cfg**. (This information is only given here for completeness.)
- For the DUA client **metacp**, the file is located in *install_path***/client/conf/idmssl.cfg** (and its input should be derived from *install_path***/conf/idmssl.cfg.template**).

If IDMS is enabled, the processes read the file at startup. To enable IDMS, the DirX process's own PSAP address must contain a DNS component with an SSLPORT subcomponent value greater than 0. The **DIRX_OWN_PSAP** environment variable specifies the DSA process's own PSAP address, whereas the LDAP server and the DUA client (**metacp**) get their own PSAP addresses from the SELF entry in the **dirxldap.cfg** or **dirxcl.cfg** configuration files.

The settings are used for the lifetime of the process; that is, a restart is required for changes to the configuration file to take effect.

The **idmssl.cfg** file contains the pathnames of the key material files used by the SSL/TLS library and configuration options for the security protocol, ciphers, wait time and logging level.

In the configuration file, comment lines begin with the hashtag character (#) in the first column and are ignored. The format of all other lines is:

keyword value

The following parameters (*keyword*) must be specified:

idm_ssl_own_pse_file

Specifies the fully-qualified name of the running process's own PEM file. This PEM file must contain the following items:

- The private key and the public key certificate representing the Personal Security Environment (PSE) of the running process.
- The issuers' Certificate Authority (CA) certificates (chain of CA certificates of all intermediate CAs and the root CA).

The file must be accessible and readable for the started DirX process.

DirX Identity installs an example file **IDMPSE.pem** under *install_path**/conf*. This file contains the demo PSE for **/O=My-Company/OU=DirX-Example/OU=DirX8.2B/CN=dirxIDM** issued by the testCA **/O=My-Company/OU=DirX-Example/CN=test-CA**. Note that these files match the content of the sample files provided by the DirX installation.

idm_ssl_pwd_file

Specifies the fully-qualified name of the file that contains the password for accessing the private key contained in the own PEM file. (See the **idm_ssl_own_pse_file** parameter for details.) The password must be the only content of this file. When creating the file, the password must be specified in plain ASCII format. When IDMS reads the password for the first time, it encrypts it and reuses the encrypted format in subsequent starts. (Similar to the handling of the LDAP server's SSL password; see the description of the LDAP PKCS12 Password File attribute.) The file must be accessible and readable for the started process.

DirX Identity installs the file IDMPSE.pwd under *install_path***/conf**. This file contains the clear text password **dirxdirx**, which is suitable for accessing the private key contained in the own PEM file. Note that these files match the content of the sample files provided by the Dir.X installation.

idm_ssl_trusted_ca_cert_file

Specifies the fully-qualified name of the PEM file of all trusted CAs. This PEM file contains the certificates of all CAs trusted to issue valid server certificates. In client mode, the certificate sent by the server is checked against the certificates contained in this file. The file must be accessible and readable for the started process.

DirX Identity installs an example file **testCA.pem** under *install_path***/conf**. This file contains the CA certificate of the CA **/O=My-Company/OU=DirX-Example/CN=test-CA** as the trusted CA.

idm_ssl_protocol

Specifies the SSL protocol that IDM accepts for incoming and outgoing connections. The value must be one of **SSLv3**, **TLSv1**, **TLSv11**, **TLSv12** or **ALL** (for all protocols). Both sides must accept the same protocol to establish an SSL connection. SSLv2 is considered to be unsafe and is therefore not supported.

idm_ssl_ciphers

Specifies the list of ciphers accepted for encryption. IDMS accepts all cipher names and shortcuts that OpenSSL supports.

To get a list of cipher names, see the OpenSSL documentation or start an OpenSSL-shell and perform a **ciphers** command to get a list of cipher names or cipher groups. In the following example, the list of RSA ciphers names and groups are read:

```
OpenSSL> ciphers RSA
```

The command output is as follows:

```
AES256-GCM-SHA384:AES256-SHA256:AES256-SHA:CAMELLIA256-SHA:DES-
CBC3-SHA:DES-CBC3-MD5:AES128-GCM-SHA256:AES128-SHA256:AES128-
SHA:SEED-SHA:CAMELLIA128-SHA:IDEA-CBC-SHA:IDEA-CBC-MD5:RC2-CBC-
MD5:RC4-SHA:RC4-MD5:RC4-MD5:DES-CBC-SHA:DES-CBC-MD5:EXP-DES-CBC-
SHA:EXP-RC2-CBC-MD5:EXP-RC2-CBC-MD5:EXP-RC4-MD5:EXP-RC4-MD5:NULL-
SHA256:NULL-SHA:NULL-MD5
```

The cipher names can also be meta-names like RSA, MEDIUM or HIGH as supported by OpenSSL.

idm_ssl_io_timeout

Specifies the maximum time in seconds that IDMS is to wait while performing SSL_accept() to complete the initial SSL handshake. If this time period expires, the handshake is aborted and the TCP connection is disconnected.

If IDMS acts as a server (the DSA process), the initial SSL handshake requires both sides to exchange some messages (read/write) in order to establish an SSL connection. If the client does not send data within **idm_ssl_io_timeout** seconds, IDM will abort further data exchange and assume that the connection is broken.

While waiting for data, the corresponding IDM worker thread is blocked and does not service other requests. As a result, it is recommended not to configure this parameter in minutes or hours.

As IDMS is currently only used between DirX components that can be assumed to behave in a friendly manner, a timeout may indicate serious network problems or a third-party attack.

idm_ssl_logging

Specifies whether or IDM generates SSL logging. Use this option only for debugging. Specify one of the following values:

- **0**—Logging is turned off.
- **1**—Logging of SSL function calls (level low).

- **2**—Logging of SSL function calls and select and poll events (level medium).
- **3**—Logging of SSL function calls, select and poll events, and input / output data (level high).

If SSL logging is enabled, a log file is generated under *install_path**/tmp*. The name of the log file is **idmssl**pid***_id**.txt*, where *pid* is the process identifier of the running process and id is either **d** (DSA), **l** (LDAP server, lower case l) or **c** (**metacp**).

Keep in mind that changing the logging status requires a restart of the process. A logging file grows endlessly. You can use logging, for example, to track down IDM SSL connection problems.

Although individual configuration is possible, the default IDMS configuration file that is installed with DirX provides the configuration settings shown below for all DirX processes using IDMS. All entities use the same key material: ($DIR identifies the DirX Identity installation directory).

```
########################################
# SSL Configuration file for IDM protocol
# each line contains a token and a value
########################################
# the pathname of the file containing the own private key and
certificate chain
# in PEM format
idm_ssl_own_pse_file $DIR/conf/IdmPSE.pem
# the pathname of the password file for accessing the private key
idm_ssl_pwd_file $DIR/conf/IdmPSE.pwd
# the pathname of the PEM file that contains trusted CA
certificates
# the file may contain multiple CA certificates in PEM format.
# In client mode, these CA certs are used to verify the certificate
received
# from the server.
idm_ssl_trusted_ca_cert_file $DIR/conf/testCA.pem
# the security protocol to be used - one of: SSLv3, TLSv1, TLSv11,
TLSv12
idm_ssl_protocol TLSv12
# the ciphers to use (names must be compatible with OpenSSL naming
schema)
idm_ssl_ciphers RSA
# max wait time in seconds in SSL I/O
idm_ssl_io_timeout 10
# SSL log level (0=off,1=low)
```

```
#  0 = off
#  1 = low    (SSL function calls)
#  2 = medium ( == low + select/poll eventing )
#  3 = high   ( == medium + I/O data )
idm_ssl_logging 0
########################################
```

7. DirX Identity Environment Variables

This chapter is a table that describes the environment variables used by DirX Identity programs and processes. The table includes the variable name and description, the commands and processes that use the variable, and the variable default value, which is used if the variable is not set.

Name	Description	Used by...	Default
DIRX_CLCFG_FILE	Supplies the name of the directory client configuration file.	The **metacp** command	**$DIRXMETAHUB_INST_PATH/client/conf/dirxcl.cfg**
DIRX_CTX_LIMIT	Specifies the maximum memory size of the metacp process in MB.	metacp	Unlimited
DIRX_DEL_TIME	Specifies the time threshold (in hours) for trace and exception message file deletion. Any trace or exception file (**LOG***, **USR***, **EXC*** in the trace file directories), that was created more than **DIRX_DEL_TIME** before process startup will be deleted.	All processes	No trace file is deleted.
DIRX_LOG_DATASIZE	Supplies the maximum length of data to be logged in a log record.	All processes	1024
DIRX_LOGCFG_FILE	Supplies the name of the log configuration file.	All processes	**$DIRXMETAHUB_INST_PATH/client/conf/dirxlog.cfg**
DIRX_MAX_THREADS	Specifies the maximum number of parallel threads. The maximum value is **512**.	All processes	**128** This value must be adapted according your system configuration.
DIRXMETAHUB_INST_PATH / DIRXIDENTITY_INST_PATH	Supplies the pathname of the base directory in which DirX Identity is installed.	All processes	**C:\Program Files\DirX\Identity** (on Windows)

Name	Description	Used by...	Default
DIRX_SVC_EXTINFO	Specifies the information that is required to evaluate log files that have been generated by non-DirX client or server components such as **metacp**. Specify the value of the environment variable in the following format: *library_name1*, <*function_table_name1* [,*message_table_name1*] |-,message_table_name1> *[:library_name2*, <*function_table_name2* [,*message_table_name2*] |-,message_table_name2>] ... *[:library_nameN*, <*function_table_nameN* [,*message_table_nameN*] |-,message_table_nameN**>**] where: *library_name* Specifies the name of a dynamic link or shared library with the extension **.dll**, **.so**, **.sl** that contains an additional function or message table. *funtion_table_name* Specifies the name of the function table that contains the function pointers of additional encoding-/decoding-functions (XDR-functions).	The **metacpdump** and **metahubdump** command	None. The environment variable is set appropriately in the commands.

Name	Description	Used by...	Default
	message_table_name Specifies the name of the message table that contains all additional serviceability messages. Example: libdirxapi_md,metadir_ids,mdi_msg_table		
DIRX_TRUSTED_CA	Specifies the full pathname of the file that contains the server certificates or CA certificates used for SSL binds. (See the description of SSL/TLS Certificate Database the DirX Identity Program Files chapter.)	The **metacp obj bind** command	When this environment variable is not set, the file **$DIRXMETAHUB_INST_PATH/client/conf/cert8.db** is used.
DIRX_SET_TLS_LEVEL_MIN	Specifies the minimum version of the TLS protocol. The valid values are "1.0", "1.1", "1.2" and "1.3".	**metacp**, IdS-C server	1.0
DIRX_SET_TLS_LEVEL_MAX	Specifies the maximum version of the TLS protocol. The valid values are "1.0", "1.1", "1.2" and "1.3".	**metacp**, IdS-C server	1.3
DIRX_KEY3DB_FILE	Specifies the full pathname of the file that contains the private keys used for SASL-authenticated binds with the EXTERNAL mechanism. (See the description of SSL/TLS Certificate Database the DirX Identity Program Files chapter.)	The **metacp obj bind** command	When this environment variable is not set, the file **$DIRXMETAHUB_INST_PATH/client/conf/key3.db** is used
DXI_JAVA_HOME	Specifies *dxi_java_home* in native notation. For Windows platforms, this variable is only defined after execution of a command like this (including quotes): **call "%DIRXIDENTITY_INST_PATH%\setdxienv.bat"**	Java and keytool processes of DirX Identity. Exceptions (see below)	None. The variable is set appropriately in the environment (Unix platforms) in the file *install_path***/setdxienv.bat** (Windows platforms), respectively.

The environment variable DXI_JAVA_HOME is not used for these applications:

- Java-based Server for Windows and Linux. For these platforms, the Java server uses the configured properties **vm** and **dxi.java.home.bin** (Windows only) in the file **idmsvc.ini** in *install_path**/ids-j-***technical-domain-name***-S**number**/bin* which is created during configuration of a Java-based Server.
- Web Applications deployed into Tomcat. The JRE used by these files depends on the Tomcat configuration and can also be the JRE for DirX Identity.

TLS Support

metacp is linked with NSS 3.42 (Network Security Service) that supports TLS1.3. (Note that SSL3.0 has been disabled.).

The accepted range of TLS versions is set to TLS1.0 up to TLS1.3 by default. This range can be restricted with the environment variables DIRX_SET_TLS_LEVEL_MIN and DIRX_SET_TLS_LEVEL_MAX. See the previous table for descriptions of these variables.

8. Directory Synchronization Templates

DirX Identity provides the following directory synchronization templates in the **Samples\metacp** directory of the DirX Identity installation:

- Full import to the Identity store from an LDIF content file (initial and subsequent; demos 1, 2, and 3)
- Delta import to the Identity store from an LDIF content file (demo 4)
- Export from the Identity store to an LDIF content file (demo 5)
- Import to the Identity store from a Microsoft Exchange data file in Microsoft Exchange "admin" format (demo 6)
- Export from the Identity store to a Microsoft Exchange data file in Microsoft Exchange "admin" format (demo 7)
- Import to the Identity store from a DirX Identity Microsoft Exchange agent data file (demo 8)
- Export from the Identity store to a DirX Identity Microsoft Exchange agent data file (demo 9)
- Import to the Identity store from a Lotus Notes data file (initial, subsequent, delta; demos 10, 11, and 12)
- Export from the Identity store to Lotus Notes data file format (demo 13)
- Conversion of Exchange format to LDIF content format (demo 16)

This chapter describes the main features of the synchronization templates for importing from an LDIF content file and exporting to an LDIF content file. (See LDIF Content Format for a description of the file format.) See the file **I-metacpReadme.txt** in the **Samples\metacp** directory for a detailed description of how the demos can be applied and what the setup needs to be. See the chapters on the Lotus Notes and Microsoft Exchange Identity agents in the *DirX Identity Agent Reference* for a description of Lotus Notes and Exchange import and export data file formats.

This chapter also describes how to read and write XML data files.

8.1. Import from an LDIF Content File

This example synchronization template imports the contents of an LDIF-formatted file (LDIF content format) into the Identity store.You can use it to:

- Perform an initial import to an empty Identity store from an import data file.The import data file is the only source for import and all of its entries are imported.
- Perform a subsequent import to a non-empty Identity store.The control logic in the synchronization profile performs a search in the Identity store to get a list of the existing entries and uses this list to perform dynamic delta recognition on an entry basis.Entries that exist in the Identity store but which are not supplied in the import data file are deleted-the logic assumes that the import data file provides 100% of the Identity store contents and the Identity store is cleared of unwanted entries.The synchronization

profile performs a second level of delta recognition for entries that exist in the Identity store and in the import data file.All attributes to be synchronized are compared between both entries and the Identity store entry is modified accordingly.

The example synchronization template runs under the following assumptions:

- The Identity store setup is the sample PQR database starting at /O=PQR
- The PQR database contains no Organizational-Person (ORP) entries
- The collective attribute subentry does not define any collective attributes
- The meta controller uses the default LDAP attribute names supplied with DirX Identity

8.1.1. Files

This synchronization template uses the following files and Tcl scripts:

Attribute Configuration Files	**Description**
l-ldifattr.cfg	LDIF attribute information (source)
ldapattr.cfg	LDAP attribute information (target)
Data Files	**Description**
ldif.data	LDIF file for initial import into the Identity store
ldif.data1	LDIF file for a subsequent import into the Identity store
Synchronization Profile Scripts	**Description**
common.tcl	Control logic section. Contains the utility functions that are common to all of the synchronization templates.
import.tcl	Control logic section. Imports a file into the DirX server using one global search.
map.l-import.ldif.tcl	Mapping section. Maps attribute values from LDIF syntax to LDAP syntax for importing into the Identity store.
l-import.ldif.tcl	Variable section. The user-callable script to run the initial and subsequent imports. The script calls import.tcl.
test.import.tcl	Control logic section. Performs only the data mapping and does not interact with the Identity store (for updates). This Tcl script is called if the profile switch **test_mapping_only** is set to TRUE in l-import.ldif.tcl. The script writes each input data entry plus the Tcl variable arrays (one for the entry with its attributes and one for the generated LDAP entry with its attributes) to the trace file. This output allows you to check the correctness of your attribute mappings.

8.1.2. Synchronization Profile Structure

The following sections describe the contents and design of this example synchronization profile.

Variable Section Settings

The Variable section of the example synchronization profile (see the file l-import.ldif.tcl) provides the following information:

- A set of profile switches that configure the operation of the synchronization profile
- The names of the mapping procedures defined in the mapping section

The following table describes the profile switches.

Name	Description
home_dir	The home directory in which the synchronization profile Tcl scripts are located
debug_trace	Controls the output of the source and target Tcl variables. Possible values are: 0-Do not output 1-Output to standard out (the display) 2-Output to the meta controller trace file
trace_file	The name of the meta controller trace file
trace_level	The trace level to be used. Possible values are: 1-Perform an error trace (only trace failed operations) 2-Perform a full trace (trace all operations) 3-Perform a full trace only on operations that are actually sent to the Identity store
init_mode	Runtime execution mode. Possible values are:* REAL* - All update operations are sent to the Identity store (actual database synchronization is performed)* TRIAL* - No updates are made to the Identity store (actual database synchronization is not performed). Use TRIAL mode to examine whether the synchronization, if run in REAL mode, operates as expected and does not perform any unwanted updates
superior_info	The parameters for the automatic creation of superior entries
agent_attr_file	The name of the Identity agent attribute configuration file
x500_attr_file	The name of the Identity store attribute configuration file
agent_data_file	The name of the import data file (generated by the Identity agent export procedure)
agent_attr_list	The attributes to be synchronized from the source connected directory
conn_type	The type of the connection to the Identity store (LDAP)

Name	Description
x500_attr_list	The attributes to be synchronized in the Identity store
file_format	The import data file format. Possible values are **TAGGED** and **UNTAGGED**.
process_all_attr	Controls whether or not all attributes are processed. Possible values are **TRUE** (process all attributes) and **FALSE** (process those attributes in ldap_attr_list).
object_class_abbr	The type of object class abbreviations (LDAP) used in the control logic section.
user_name	User name parameter for the **obj bind** operation
user_pwd	User password parameter for the **obj bind** operation
dsa_name	Name of the Identity store as defined in the directory client configuration file (**dirxcl.cfg**).
server_address	Identity store address (LDAP server DNS name or IP address and port number) for the **obj bind** operation.
protocol	The protocol to access the Identity store. Possible values are **LDAPv2** and **LDAPv3**. See the description of the obj bind operation for details.
base_obj	Search base parameter for the **obj search** operation
subset	Search scope parameter for the **obj search** operation
filter	Search filter parameter for the **obj search** operation
sort_key	Sort key parameter for the **meta sortresult** operation. Possible values are **ASC** and **DESC**. See the **meta sortresult** description in the meta object section for details.
sort_order	Sort order parameter for the **meta sortresult** operation. Possible value is **DDN**. See the **meta sortresult** description in the meta object section for details.
perform_read	Controls whether the meta controller is to read all attributes from the Identity store during the initial search (FALSE) or with an additional read operation (TRUE)
remove_objects	Controls whether the meta controller is to remove entries in the Identity store returned in a search result that do not exist in the source data file. Possible values are TRUE (remove entries) or FALSE (do not remove entries).
marking_attr	This switch is only relevant when "remove_objects" is FALSE: Instead of removing the entry the specified attribute type is added to the entry (using the value specified in "marking_value") to mark it as deleted.
marking_value	This switch is only relevant when "remove_objects" is FALSE: Specifies the attribute value to be used for the attribute type specified in "marking_attr".

Name	Description
keep_objects	Specifies the objects that should never be removed from the Identity store, even if not provided in the data file. The template profile switch specifies the name of the administrator on whose behalf the directory bind is performed; this list can be extended
test_mapping_only	Controls whether the meta controller performs the data mapping operations but no other operations. Possible values are **TRUE** (test mapping only) and **FALSE** (run entire synchronization profile).
test_max_entries	Controls how many entries are to be mapped, if **test_mapping_only** is **TRUE**. Supply the number of entries to be mapped. If all entries are to be mapped, supply -1.
_localcode	The code set in which the data to be processed is encoded. Possible values are **PC850** or **LATIN-1**.

Mapping Section Procedures

For LDAP access to the Identity store, the main purpose of the mapping section of the example synchronization profile (see the file map.l-import.ldif.tcl) is to assign attribute values from LDIF records to attribute values in directory objects. This is a simple one-to-one mapping.

Control Logic

The control logic supplied in the example synchronization profile has the following characteristics:

- Synchronizes from a single source only (the import file is the entry owner)
- Moves all of the entries from the import file into the Identity store
- Creates superior entries automatically with the parameters specified in the profile switch in the Variable section
- Processes import entries individually
- Performs dynamic delta recognition on the subsequent import. The delta recognition is performed in two phases:
  - Matched entries: For matched entries, performs the modify operation on the target entry in the Identity store according to the target attribute synchronization flags set in the profile switch in the Variable section
  - The set of entries supplied in the import file: compares the complete import file with the entries in the Identity store, removes any entries not supplied in the import file, and adds entries available in the import data file that do not exist in the Identity store.
- No ordering of the import file is necessary

8.1.3. Customizing This Synchronization Template

To customize this synchronization template:

- Ensure that its control logic matches your import synchronization needs
- Determine whether you want automatic creation of superior entries. If you do, supply the information for it in the appropriate profile switch in the Variable section.
- Develop attribute configuration files for the source and target directory (1 each). You can use the attribute configuration files supplied with DirX Identity as templates, or create your own.
- Choose the specific set of attributes that you want to synchronize, for the source directory and for the target directory
- Decide on your data file names and supply them in the appropriate profile switch in the Variable section
- Supply your credentials for binding to the Identity store in the appropriate profile switch in the Variable section
- Choose the file format (tagged or untagged) that represents the data to be imported. Depending on your selection, you'll need to have additional information in the attribute configuration file (for a tagged file) or additional information supplied in the set of synchronized attributes (for an untagged file).
- Select the search parameters to be used to get an appropriate search result that is used for matching the entries.
- Select the entries that should never be deleted.
- Decide whether entries should be deleted or whether such entries simply should be marked as deleted.
- Select to search with all relevant attributes, or to search with just the distinguished name and perform another read operation for the attributes whenever they are needed (for example, before a modify operation can be performed).
- Create the mapping procedures by:
  - defining the mapping for each attributes
  - writing Tcl code to perform the mapping

Use the mapping procedure scripts supplied with DirX Identity as templates.

8.2. Export to an LDIF Content File

This example synchronization template exports the contents of the Identity store to an LDIF-formatted file.The example synchronization template runs under the following assumptions:

- The Identity store setup is the sample PQR database starting at /O=PQR
- The PQR database contains entries
- The meta controller uses the default LDAP attribute names supplied with DirX Identity

8.2.1. Files

This synchronization template uses the following files and Tcl scripts:

Attribute Configuration Files	Description
l-ldifattr.cfg	LDIF attribute information (target)
ldapattr.cfg	LDAP attribute information (source)
Data Files	**Description**
export.data	LDIF file created by the export process
Synchronization Profile Scripts	**Description**
common.tcl	Control logic section. Contains utility functions that are common to all of the synchronization templates.
export.tcl	Control logic. Exports data from the Identity store using one global search.
map.l-export.ldif.tcl	Mapping section. Maps LDAP entries to LDIF syntax.
l-export.ldif.tcl	Variable section. This is the user-callable Tcl script to run the export operation. The script calls export.tcl

8.2.2. Synchronization Profile Structure

The following sections describe the contents and design of this example synchronization profile.

Variable Section Settings

The Variable section of the example synchronization profile (see the file l-export.ldif.tcl) provides the following information:

- Profile switches that configure the operation of the synchronization profile
- The names of the mapping procedures defined in the mapping section

The following table describes the profile switches.

Name	Description
home_dir	The home directory in which the synchronization profile Tcl scripts are located
debug_trace	Controls the output of source and target Tcl variables. Possible values are:* 0* - Do not output **1** - Output to standard out (the display)* 2* - Output to the meta controller trace file
trace_file	The name of the meta controller trace file

Name	Description
trace_level	The trace level to be used. Possible values are:* 1*-Perform an error trace (only trace failed operations)* 2*-Perform a full trace (trace all operations)* 3*-Perform a full trace only on operations that are actually performed
agent_attr_file	The name of the Identity agent attribute configuration file.
x500_attr_file	The name of the Identity store attribute configuration file.
dump_file	The name of the export data file to be created or to be appended (depends on the **file_mode** switch).
file_attr_list	The attributes to be synchronized in the export data file
file_format	The data format to use for the generated export data file. Possible values are **TAGGED** and **UNTAGGED**.
file_mode	The mode to write to the export data file (overwrite or append).
conn_type	The type of the connection to the Identity store (LDAP).
x500_attr_list	The attributes to be synchronized from the Identity store.
user_name	User name parameter for the **obj bind** operation.
user_pwd	User password parameter for the **obj bind** operation.
dsa_name	The name of the Identity store as defined in the directory client configuration file (**dirxcl.cfg**).
server_address	Identity store address (LDAP server DNS name or IP address and port number) for the **obj bind** operation.
protocol	The protocol to access the server for the **obj bind** operation. Possible values are **LDAPv2** and **LDAPv3**. See the obj bind operation.
base_obj	Search base parameter for the **obj search** operation.
subset	Search scope parameter for the **obj search** operation.
filter	Search filter parameter for the **obj search** operation
sort_key	Sort key parameter for the **meta sortresult** operation. Possible values as **ASC** and **DESC**. See the description of the **meta sortresult** operation in the meta object section.
sort_order	Sort order parameter for the **meta sortresult** operation. Possible value is **DDN**. See the description of the **meta sortresult** operation in the meta object section.
sorted_list	Controls whether the export data file is sorted (**TRUE**) or not (**FALSE**)
perform_read	Controls whether the meta controller is to read all attributes from the Identity store during the initial search (**FALSE**) or with an additional read operation (**TRUE**)
_localcode	The code set in which the data to be processed is encoded. Possible values are **PC850** or **LATIN-1**.

Mapping Section Procedures

For LDAP access to the Identity store, the main purpose of the mapping section of the example synchronization profile (see the file map.l-export.ldif.tcl) is to assign attribute values from objects to attribute values in LDIF records. This is a simple one-to-one mapping.

Control Logic

The control logic supplied in the example synchronization profile has the following characteristics:

- Exports all entries in a Identity store subtree (subject to the specified search filter) to a flat file
- Processes Identity store entries individually
- Sorts the entries in the generated export data file (if the **sorted_list** profile switch is set to TRUE)

8.2.3. Customizing This Synchronization Template

- Ensure that its control logic matches your export synchronization needs
- Develop attribute configuration files for the source and target directory (one each). You can use the attribute configuration files supplied with DirX Identity as templates, or create your own.
- Choose the specific set of attributes that you want to synchronize, for the source directory and for the target directory
- Decide on the name of the export data file to be generated and supply it in the appropriate profile switch in the Variable section
- Supply your credentials for binding to the Identity store in the appropriate profile switch in the Variable section
- Select the search parameters that will generate an appropriate search result
- Choose the file format (tagged or untagged) that represents the data to be exported. Depending on your selection, you'll need to have additional information in the attribute configuration file (for a tagged file) or additional information supplied in the set of synchronized attributes (for an untagged file).
- Create the mapping procedures by:
  - defining the mapping for each attributes
  - writing Tcl code to perform the mapping

  Use the mapping procedure scripts supplied with DirX Identity as templates.
- Decide whether you want the generated export file to be sorted or not, and set the **sorted_list** profile switch in the Variable section appropriately

9. Data Format Handling Procedures

This section discusses procedures necessary to handle typical data formats:

- XML Formats
- ChangeLog Formats

9.1. Handling XML Files

This section describes how to handle XML data files in DirX Identity.Specifically, it describes how to:

- Read an XML data file
- Write an XML data file

See also the XML data format section.

9.1.1. Reading an XML Data File

To read an XML data file, perform the following steps:

- Create the attribute configuration handle *ah* by passing the name of an attribute configuration file to the **meta readattrconf** function.
- For flat XML, create a connection handle *ch* with a **meta openconn** function of the form:

  meta openconn -conn ch -type FILE -format FLAT-XML -attrconf ah -filemode READ ...
- For DSML: create a connection handle *ch* with a **meta openconn** function of the form:

  meta openconn -conn ch -type FILE -format DSML -attrconf ah -filemode READ ...
- Use the meta getentry function in a loop for reading the file, for example:

  meta getentry -source ch -target eh

The Tcl variable array **eh** is available in the same way as usual and can be used for mapping purposes.

Keep in mind that the enclosing tokens in an XML data file are **<xml_data.xml>** and **</xml_data.xml>**, for example

```
<?xml version='1.0' encoding='ISO-8859-1'?>
<!-- Example -->
<xml_data.xml>
          ...
</xml_data.xml>
```

> Using different tags in the data file results in a warning “Invalid record” in the trace file at run time.

9.1.2. Writing an XML Data File

To write an XML data file, perform the following steps:

Create an attribute configuration handle *ah* by passing the name of an attribute configuration file to the **meta readattrconf** function.

- For Flat XML: create a connection handle ch with a **meta openconn** function of the form:

  meta openconn -conn ch -type file -format FLAT-XML -attrconf ah -filemode WRITE ...
- For DSML: create a connection handle ch with a **meta openconn** function of the form

  meta openconn -conn ch -type file -format DSML -attrconf ah -filemode WRITE ...
- Use the **meta gethandle** function in to create a handle for the output file, for example:

  meta gethandle -conn ch -entry eh

The Tcl variable array *eh* is available in the same way as usual and can be filled for mapping purposes.To write the data XML file from a directory search result:

- Write the header using the **meta writetext** function
- Call the **meta getentry** function on the directory search result in a loop:
  - Perform the mapping of the directory entry to the entry represented by the handle *eh* obtained with **meta gethandle**
  - Write entry *eh* using the **meta writerecord** function
- Write the footer using the meta writetext function.

9.2. Handling ChangeLogs

The ChangeLogs are read from the iPlanet or OID directory by calling **obj search**.For each object in the search result a **getentry** is called and a TCL variable array with the result is filled (see also changeLog data formats).So for the first example listed above, there might exist TCL variables such as

```
rh_ldap(DDN) = Changenumber=207, cn=changelogentry
rh_ldap(objectClass) = {top changelog}
rh_ldap(targetdn) = {cn=Shrivastava
Saurabh,ou=metadirectory,o=oracle,dc=com}
rh_ldap(changetype) = Add
rh_ldap(operationtime) = 19991029153546z
...
```

```
rh_ldap(changenumber) = 207
rh_ldap(orclchangeretrycount) = 0
rh_ldap(changes) = objectclass:inetOrgPerson
          objectclass:OrganizationalPerson
          objectclass:Person
          objectclass:Top
          cn:Shrivastava Saurabh
          cn:Shrivastava
          cn:Saurabh
          cn:Saurabh Shrivastava
          cn:sshrivas
          employeenumber:22
          givenname:Saurabh
          sn:Shrivastava
          title:Mr.X
          mail:sshrivas@us.oracle.com.X
          telephonenumber:650-574-9107
          postaladdress:1067 Foster city Blvd $Apt B$Foster city
CA=94404-X
          l:Foster city
          orclguid:000396876942251408758371686944 2
          creatorsname:cn=orcladmin
          createtimestamp:19991029153546z
```

The relevant fields that are of further interest are highlighted.

In order to handle the iPlanet and OID-ChangeLogs, the following operation can be used:

```
meta getchangelog
          -name <DN>
          -changetype <operation change type>
          -changes <list of changes>
          -source <handle to pseudo LDIF-file>
          -target <result handle>
```

metacp internally can parse LDIF-Change_Files. As that parser should also be used for handling of the ChangeLogs, an entry from the Search-Result (which represents the ChangeLogs) needs to be mapped to the correct LDIF syntax. The only items of interest are **targetdn**, **changetype**, **changes**.

Therefore the three interface parameters will be concatenated in memory and an appropriate prefix (if necessary) is assigned to each of them. Such a byte stream can easily

be handled by the LDIF parser. The only difference is that the parser gets its values from memory and not from file.

The command **getchangelog** will fill a TCL variable array (named by ***result handle*** of parameter **-target**) the same way as if **getentry** would have read an LDIF change file.

Example:

After having called 'getentry' for the first entry from the search result, the 'getchangelog' will be called as follows:

```
getchangelog
          -name [lindex $rh_ldap(targetdn) 0]
          -changetype [lindex $rh_ldap(changetype) 0]
          -changes [lindex $rh_ldap(changes) 0]
          -source file_ch
           -target rh
```

The first entry from the examples listed above will be mapped to the following byte stream:

```
dn: cn=Shrivastava Saurabh,ou=metadirectory,o=oracle,dc=com
changetype: Add
objectclass:inetOrgPerson
objectclass:Person
objectclass:Top
cn:Shrivastava Saurabh
cn:Shrivastava
....
```

The following TCL variables will be set:

```
rh(DDN) = {cn=Shrivastava Saurabh,ou=metadirectory,o=oracle,dc=com}
rh(_ldif_opcode) = Add
rh(objectClass) = {inetOrgPerson Person Top}
rh(cn) = {{Shrivastava Saurabh} Shrivastava}
```

The parameter **$rh_ldap(changenumber)** holds the last change number. So the last changenumber is always available and can easily be used in the next synchronisation cycle, when the **obj search** command needs to be called with the following filter:

```
-filter (&(objectclass=changeLogEnty)(changenumber>='LAST-CHANGE-
```

```
NUMBER'))
```

9.2.1. ChangeLog Sample Code

In the following sample code, only the relevant functions are listed.

The sample code doesn't include error handling etc. (e.g. if **obj search** fails). Furthermore, it doesn't list the functions that need to be called first in order to get the relevant handles that are passed to **openconn**, e.g. **ah** or **ldap_ah** etc.

```
#
# open a pseudo LDIF data file (That is just a 'dummy' open for the
routine 'getchangelog'. so that this routine
# gets a correct source handle 'file_ch')
#
set cmd "meta openconn  -type     FILE \
        -file          \"$agent_data_file\" \
        -mode     READ \
        -format   TAGGED \
        -attrconf ah       \ # attribute handle generated by
'readattrconf'
                           \ # when reading 'l-ldifattr.cfg'
        -conn     file_ch"
if {$agent_attr_list != ""} then {
append cmd " -attribute {$agent_attr_list}"
}
exec_cmd $cmd

#
# create a connection handle "change_ch" that requests all the
attributes that need to be read from the
# LDAP server in order to get the changelog entry
# The parameter 'change_attr_list' contains all the attributes that
are read (by 'obj search') from the Changelog
# entry: 'changes', 'targetdn', 'changenumber', 'changetype', 'DDN'
#
set cmd "meta openconn -type     LDAP \
        -attrconf ldap_ah       \ # attribute handle generated by
'readattrconf'
                                \ # when reading 'l-ldifattr.cfg'
        -conn     change_ch"
if {$change_attr_list != ""} then {
```

```
append cmd " -attribute {$change_attr_list}"
if {$process_all_attr == "TRUE"} then {
append cmd " -processallattr"
}
}
exec_cmd $cmd

#
# retrieve the relevant objects from the LDAP server
# (The parameter filter is set as follows:
# set filter {(&(objectclass=changelogentry) (server=$host_name)
(changenumber>=$change_number)
(!(modifiersname=$subscriber_dn)))}
# with 'change_number' being the relevant change number that has been
processed up so far)
#
set cmd "obj search {$base_obj} \
         $subset \
         -conn       change_ch \
         -result     ldap_res"
if {$filter != ""} then {
append cmd " -filter \"$filter\""
}
set status [catch { eval $cmd } result]

set result 0
while {$result == 0} {
#
# extract an entry from the search result
#
set result [exec_cmd "meta getentry -source ldap_res \
                                    -target rh_ldap"]
if {$result == 0} then {
         set max_change_number [lindex $rh_ldap(changenumber) 0]
         # pass the retrieved attributes of the changelog entry to
'getchangelog'; the TCL variable array named
         # by the '-target' parameter contains all the changes.
         exec_cmd "meta getchangelog     \
              -name     {[lindex $rh_ldap(targetdn) 0]}     \
              -changetype {[lindex $rh_ldap(changetype) 0]} \
              -changes    {[lindex $rh_ldap(changes) 0]}      \
```

```
              -source     file_ch        \ # handle to
                                         \ # pseudo LDIF
                                         \ #  file
              -target     rh"
         #
         # now the change log entry is available in the Tcl-Variable
"rh"
         #
         ...
}
}
```

Appendix A: Country Codes

This appendix lists countries and their 2-letter ISO 3166 country code. This code is used in the Country Name simple syntax.

Country	Code
AFGHANISTAN	AF
ALBANIA	AL
ALGERIA	DZ
AMERICAN SAMOA	AS
ANDORRA	AD
ANGOLA	AO
ANGUILLA	AI
ANTARCTICA	AQ
ANTIGUA AND BARBUDA	AG
ARGENTINA	AR
ARMENIA	AM
ARUBA	AW
AUSTRALIA	AU
AUSTRIA	AT
AZERBAIJAN	AZ
BAHAMAS	BS
BAHRAIN	BH
BANGLADESH	BD
BARBADOS	BB
BELARUS	BY
BELGIUM	BE
BELIZE	BZ
BENIN	BJ
BERMUDA	BM
BHUTAN	BT
BOLIVIA	BO
BOSNIA AND HERZEGOVINA	BA
BOTSWANA	BW
BOUVET ISLAND	BV
BRAZIL	BR

Country	Code
BRITISH INDIAN OCEAN TERRITORY	IO
BRUNEI DARUSSALAM	BN
BULGARIA	BG
BURKINA FASO	BF
BURUNDI	BI
CAMBODIA	KH
CAMEROON	CM
CANADA	CA
CAPE VERDE	CV
CAYMAN ISLANDS	KY
CENTRAL AFRICAN REPUBLIC	CF
CHAD	TD
CHILE	CL
CHINA	CN
CHRISTMAS ISLAND	CX
COCOS (KEELING) ISLANDS	CC
COLOMBIA	CO
COMOROS	KM
CONGO	CG
CONGO, THE DEMOCRATIC REPUBLIC OF THE	CD
COOK ISLANDS	CK
COSTA RICA	CR
COTE D'IVOIRE	CI
CROATIA	HR
CUBA	CU
CYPRUS	CY
CZECH REPUBLIC	CZ
DENMARK	DK
DJIBOUTI	DJ
DOMINICA	DM
DOMINICAN REPUBLIC	DO
ECUADOR	EC
EGYPT	EG
EL SALVADOR	SV

Country	Code
EQUATORIAL GUINEA	GQ
ERITREA	ER
ESTONIA	EE
ETHIOPIA	ET
FALKLAND ISLANDS (MALVINAS)	FK
FAROE ISLANDS	FO
FIJI	FJ
FINLAND	FI
FRANCE	FR
FRENCH GUIANA	GF
FRENCH POLYNESIA	PF
FRENCH SOUTHERN TERRITORIES	TF
GABON	GA
GAMBIA	GM
GEORGIA	GE
GERMANY	DE
GHANA	GH
GIBRALTAR	GI
GREECE	GR
GREENLAND	GL
GRENADA	GD
GUADELOUPE	GP
GUAM	GU
GUATEMALA	GT
GUINEA	GN
GUINEA-BISSAU	GW
GUYANA	GY
HAITI	HT
HEARD ISLAND AND MCDONALD ISLANDS	HM
HOLY SEE (VATICAN CITY STATE)	VA
HONDURAS	HN
HONG KONG	HK
HUNGARY	HU
ICELAND	IS

Country	Code
INDIA	IN
INDONESIA	ID
IRAN, ISLAMIC REPUBLIC OF	IR
IRAQ	IQ
IRELAND	IE
ISRAEL	IL
ITALY	IT
JAMAICA	JM
JAPAN	JP
JORDAN	JO
KAZAKHSTAN	KZ
KENYA	KE
KIRIBATI	KI
KOREA, DEMOCRATIC PEOPLE'S REPUBLIC OF	KP
KOREA, REPUBLIC OF	KR
KUWAIT	KW
KYRGYZSTAN	KG
LAO PEOPLE'S DEMOCRATIC REPUBLIC	LA
LATVIA	LV
LEBANON	LB
LESOTHO	LS
LIBERIA	LR
LIBYAN ARAB JAMAHIRIYA	LY
LIECHTENSTEIN	LI
LITHUANIA	LT
LUXEMBOURG	LU
MACAO	MO
MACEDONIA, THE FORMER YUGOSLAV REPUBLIC OF	MK
MADAGASCAR	MG
MALAWI	MW
MALAYSIA	MY
MALDIVES	MV
MALI	ML
MALTA	MT

Country	Code
MARSHALL ISLANDS	MH
MARTINIQUE	MQ
MAURITANIA	MR
MAURITIUS	MU
MAYOTTE	YT
MEXICO	MX
MICRONESIA, FEDERATED STATES OF	FM
MOLDOVA, REPUBLIC OF	MD
MONACO	MC
MONGOLIA	MN
MONTSERRAT	MS
MOROCCO	MA
MOZAMBIQUE	MZ
MYANMAR	MM
NAMIBIA	NA
NAURU	NR
NEPAL	NP
NETHERLANDS	NL
NETHERLANDS ANTILLES	AN
NEW CALEDONIA	NC
NEW ZEALAND	NZ
NICARAGUA	NI
NIGER	NE
NIGERIA	NG
NIUE	NU
NORFOLK ISLAND	NF
NORTHERN MARIANA ISLANDS	MP
NORWAY	NO
OMAN	OM
PAKISTAN	PK
PALAU	PW
PALESTINIAN TERRITORY, OCCUPIED	PS
PANAMA	PA
PAPUA NEW GUINEA	PG

Country	Code
PARAGUAY	PY
PERU	PE
PHILIPPINES	PH
PITCAIRN	PN
POLAND	PL
PORTUGAL	PT
PUERTO RICO	PR
QATAR	QA
REUNION	RE
ROMANIA	RO
RUSSIAN FEDERATION	RU
RWANDA	RW
SAINT HELENA	SH
SAINT KITTS AND NEVIS	KN
SAINT LUCIA	LC
SAINT PIERRE AND MIQUELON	PM
SAINT VINCENT AND THE GRENADINES	VC
SAMOA	WS
SAN MARINO	SM
SAO TOME AND PRINCIPE	ST
SAUDI ARABIA	SA
SENEGAL	SN
SERBIA AND MONTENEGRO	CS
SEYCHELLES	SC
SIERRA LEONE	SL
SINGAPORE	SG
SLOVAKIA	SK
SLOVENIA	SI
SOLOMON ISLANDS	SB
SOMALIA	SO
SOUTH AFRICA	ZA
SOUTH GEORGIA AND THE SOUTH SANDWICH ISLANDS	GS
SPAIN	ES
SRI LANKA	LK

Country	Code
SUDAN	SD
SURINAME	SR
SVALBARD AND JAN MAYEN	SJ
SWAZILAND	SZ
SWEDEN	SE
SWITZERLAND	CH
SYRIAN ARAB REPUBLIC	SY
TAIWAN, PROVINCE OF CHINA	TW
TAJIKISTAN	TJ
TANZANIA, UNITED REPUBLIC OF	TZ
THAILAND	TH
TIMOR-LESTE	TL
TOGO	TG
TOKELAU	TK
TONGA	TO
TRINIDAD AND TOBAGO	TT
TUNISIA	TN
TURKEY	TR
TURKMENISTAN	TM
TURKS AND CAICOS ISLANDS	TC
TUVALU	TV
UGANDA	UG
UKRAINE	UA
UNITED ARAB EMIRATES	AE
UNITED KINGDOM	GB
UNITED STATES	US
UNITED STATES MINOR OUTLYING ISLANDS	UM
URUGUAY	UY
UZBEKISTAN	UZ
VANUATU	VU
VENEZUELA	VE
VIET NAM	VN
VIRGIN ISLANDS, BRITISH	VG
VIRGIN ISLANDS, U.S.	VI

Country	Code
WALLIS AND FUTUNA	WF
WESTERN SAHARA	EH
YEMEN	YE
ZAMBIA	ZM
ZIMBABWE	ZW

Appendix B: Code Conversion

This chapter describes the features DirX Identity provides to perform code conversion. These features are based on the built-in capabilities of the Tcl 8.3 scripting language.

This chapter provides:

- Basic usage of the code conversion features
- Expert Usage of the code conversion features
- A systematic overview of the Tcl features for code conversion
- Character sets

Please refer to the original Tcl documentation for more details.

B.1. Basic Usage

DirX Identity supports all Tcl capabilities described in the character sets section below.

The main functionality of the meta controller is controlled via the Tcl variable **_localcode** (as it was in previous versions).This chapter describes the common usage of this variable and the related behavior to

- transfer information between LDAP and utf-8 coded files
- transfer information between LDAP and files coded in another character set
- control DirX Identity interactively from a terminal window

More detailed information (especially when you intend to program with Tcl) is provided in the following chapters.

Transfer of information between LDAP and utf-8 coded files

Set the **_localcode** variable to **utf-8**.

This allows either to read utf-8 coded files to LDAP or to write from LDAP to utf-8 coded files.

This mode is the fastest mode available. If you need high performance synchronizations, you should keep all data always in utf-8 format.

The setting of the _localcode variable influences the code conversion for all files (i.e. you cannot handle different character sets for different files at the same time). If you want to handle files with different character sets please refer to expert usage section.

Transfer of information between LDAP and files into another codeset

Set the **_localcode** variable to the required character set (for example Latin1).

You can evaluate the available character sets with the command **encoding names**. It is also possible to extend the available character sets with additional ones. See the original Tcl 8.3 documentation.

This allows either to read files of that character set to LDAP or to write from LDAP to files in that character set.

Note that the setting of the _localcode variable influences the code conversion for all files (i.e. you cannot handle different character sets for different files at the same time). If you want to handle files with different character sets please refer to expert usage section.

The DirX Identity default applications are delivered with _localcode set to Latin1 to obtain compatibility to previous versions.

Interactive operation from a terminal window

In this mode, you can define operations for DirX Identity interactively in a terminal window. The necessary code conversion is done automatically.

Note: for compatibility reasons you can set the **_localcode** variable to **PC850** if you work with a DOS box. This setting is ignored. Tcl converts the terminal input and output automatically to utf-8 format.

You can define **Unicode** characters in strings in the form "\udddd" (for example "\u592a" represents a Chinese glyph and "\u0041" represents the glyph "A"). To input **utf-8** characters you have to use the method for Unicode characters.

For **single byte characters** you can use the representation "\xdd" where Tcl assumes the first byte to be zero (for example "\x41" represents the glyph "A"). In this case Tcl has the behaviour that the \x mode does only end when a character not equal a to f is input. Therefore you must input all following characters in the \x mode if they are in the range a to f.

Examples:

\xfc\x62\x65l results in the German word **übel** (b and e are in the range a to f, l is not).

f\xfcndig results in the German word **fündig** (n is not in the range a to f).

Output conversion to the screen can result in the message 'Not convertible' if utf-8 characters are contained which cannot be represented with the terminal character set.

You can redirect the output to a file where you can define any character set you want (use the fconfigure *channel* -encoding *encoding* command for this purpose - this is described in the next chapters in detail).

Output conversion to a file can result in the message 'Not convertible' if utf-8 characters are contained which cannot be represented with the defined encoding set.

If you want to use the source command with command files not in the format of the terminal window, you have to read the full file content into a variable and evaluate it afterwards (see the Tcl documentation for details).

B.2. Expert Usage

This chapter provides detailed knowledge of the internal structure of the meta controller, which is necessary to understand its behavior and to program specific code conversions in Tcl scripts directly.

Metacp Architecture

To understand the capabilities of the code conversion mechanisms the internal architecture of metacp has to be understood (see next figure).

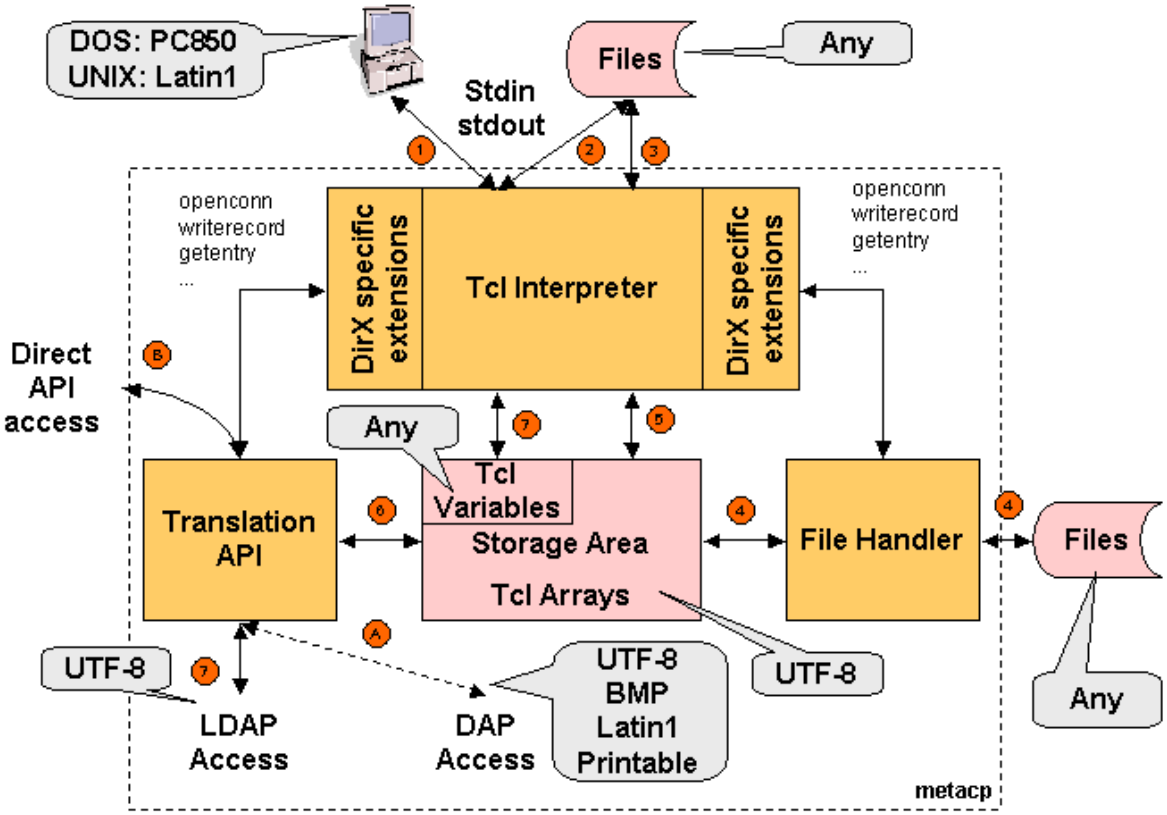

Figure 1. meta controller architecture for code conversions

Storage Area (Tcl variables and Tcl arrays)

Contains all Tcl variables and the Tcl arrays that are used for intermediate storage of the entries to be synchronized. Tcl assumes all internal strings to be in utf-8 format.

Tcl Interpreter

All Tcl functionality is available plus DirX specific extensions (for example to access the translation API or the File Handler). It handles the stdin (per default the keyboard) and the stdout (per default the screen) channels (1). These channels can be redirected to files (2).

It also contains commands to handle file input and output via other channels (3).

Via Tcl commands the storage area can be accessed directly. Variables and arrays can be processed and converted.

Translation API

Gets orders via its API functions and translates it to LDAP or DAP specific calls. Returns the results back to the API functions (for example search results). It is called by the Tcl extensions or from external programs (for example DirXmanage).

File Handler

Reads and writes files (for example LDIF, CSV or XML files) as specified in the attribute configuration to and from the storage area. It is called from the Tcl extensions. It is able to convert data from or to the internal used character set utf-8 to any of the provided character sets.

Code conversions

Code conversions are controlled by:

- The metacp specific Tcl variable **_localcode** that can be used to set a global character set for all file handling (a default value). As value you can choose any of the provided character sets. For compatibility reasons the values UTF8, LATIN1 and PC850 are still supported.
- The metacp specific settings of the **-encoding** parameter of the **meta openconn** and **meta readattrconf** commands for individual setting of character sets for specific files.
- The Tcl specific settings of the system encoding (**encoding system**).
- The Tcl specific settings of the different channels (stdin, stdout or individual channel definitions). These settings can be set or retrieved via the **fconfigure** command.

The next paragraphs describe typical applications of these settings.

Transfer of information between LDAP and utf-8 coded files

Set the **_localcode** variable to **utf-8**.

When reading files (4), the File Handler reads the file into the storage area of the Tcl arrays while no conversion is done. Internal Tcl routines can now work on these arrays (for example mapping functions (5)). The Tcl arrays can be accessed by the Translation API (6) to move the data to the LDAP API (7) and from there to the LDAP server (not shown in the picture). Because the data is already in utf-8 format, no conversion is performed.

Note: This mode is the fastest mode available. If you need high performance synchronizations, you should keep all data always in utf-8 format.

When writing files, the data is retrieved from the LDAP server 7) - in utf-8 format) via the Translation API into the storage area (6). Internal Tcl routines can now work on these arrays (for example mapping functions - (5. The File Handler can access the Tcl arrays and write it to files ((4) - no conversion is performed).

Transfer of information between LDAP and files codes in another character set

Set the **_localcode** variable to the required character set (for example **Latin1**).

When reading files, the File Handler reads the file (4) into the storage area of the Tcl arrays while a conversion from the defined character set to utf-8 is done. Internal Tcl routines can now work on these arrays ((5) - for example mapping functions). The Tcl arrays can be accessed by the Translation API (6) to move the data to the LDAP server (7). Because the data is already in utf-8 format, no conversion is performed.

When writing files, the data is retrieved from the LDAP server ((7) in utf-8 format) via the Translation API into the storage area (6). Internal Tcl routines can now work on these arrays ((5) - for example mapping functions). The File Handler can access the Tcl arrays (4) and write it to files while converting it from utf-8 to the required character set.

Converting files from one character set to another

If you want to convert a file coded in character set A to a file in character set B you have to set the individual **-encoding** parameters for the **meta openconn** and **meta readattrconf** commands.

For example:

- Open the tagged source file **mysourcefile.data** with character set **iso-8859-6** with the command:

  ```
  meta openconn -type FILE \
  -file mysourcefile.data \
  -mode READ \
  -format TAGGED \
  -encoding iso-8859-6 \
  -attrconf ah \
  -conn ch
  ```

- You can read the data from the file to the metacp storage area. It is automatically converted from iso-8859-6 to utf-8 characters.
- Perform all necessary handling of the Tcl variables and fields (all in utf-8 format) like mapping routines and other issues.
- Open the untagged target file **mytargetfile.data** with character set **unicode** with the command:

```
meta openconn -type FILE \
-file msexch.data \
-mode WRITE \
-format NON-TAGGED \
-encoding unicode \
-attribute {o ou sn givenName telephoneNumber facsimileTelephoneNumber} \
-attrconf ah \
-conn ch
```

- Write the information to the output file. The utf-8 characters are automatically converted to Unicode characters (in this example).
- Close the files as usual.

Interactive operation from a terminal window

For compatibility reasons you can set the _localcode variable to PC850 in a DOS box (this is no longer required because Tcl converts the terminal input and output automatically from and to utf-8 format).

You can use the escape sequences described in the next chapter to input Unicode or Hex characters in your input strings.

The commands, which are input at the terminal window, are sent to the Translation API and processed there. For example you can enter a search request to the LDAP server. After conversion from the terminal character set (1), all parameters are handled in utf-8 format and transferred to the LDAP server (7). The search result is returned also in utf-8 format (7) and converted to the terminal character set (1). Output conversion to the screen can result in the message 'Not convertible' if utf-8 characters are contained which cannot be represented with the terminal character set.

You can redirect the output to a file (2) where you can define any character set you want (use the **fconfigure** ***channel*** -**encoding** ***encoding*** command for this purpose - details see next chapter).

If you want to use the source command with command files not in the format of the terminal window, you have to read the file content into a variable and evaluate it afterwards (see the Tcl documentation for details). A sample routine **dxm_source** to perform this task is contained in the DirX Identity Connectivity Configuration under Configuration → Tcl → Other Scripts → Common Script.

Directly programmed code conversions

If you intend to create files directly (3) from the content in the storage area 5) or (7, you can open channels to those files and set the character set with the **fconfigure -encoding** command for each of them (see the next chapter for details).

You can also access the storage area via Tcl commands and perform character set conversions directly in memory 5) or (7.Be aware not to send variables other than utf-8 coded ones to the operating system (stdin, stdout).This will lead to strange results.

External access of the Translation API

External programs can access the translation API directly (B).In this case automatic code conversions based on a special interface switch (_localstrings - not used by metacp) can be performed.If _localstrings is set to true, no conversion is performed (the strings are assumed to be in utf-8 format), otherwise a transformation from Latin1 to utf-8 is done.

The DAP functionality at the Translation API is only available for compatibility reasons.

B.3. Tcl Features

Tcl has changed the internal representation of characters to utf-8/Unicode, i.e. it assumes to be all strings in this format when conversions have to be done to the operating system.

Unicode is a two byte (16 bit) representation used at the operating / user interface level.

utf-8 is a representation of Unicode, which is identical for all characters from 00 to 7F to the ASCII character set.All other Unicode characters are represented by a 1- to 3-byte sequence with the most significant bit of the first character set.Tcl handles these character sequences as one character when calculating strings (for example in the length command).

Therefore, utf-8 is fully compatible to all scripts and strings that only use the ASCII character set.

Defining Unicode Characters

You can define **Unicode** characters in strings in the form "\udddd" (for example "\u592a" represents a Chinese glyph and "\u0041" represents the glyph "A"). To input **utf-8** characters you have to use the method for Unicode characters.

For **single byte characters** you can use the representation "\xdd" where Tcl assumes the first byte to be zero (for example "\x41" represents the glyph "A"). In this case Tcl has the behaviour that the \x mode does only end when a character not equal a to f is input. Therefore you must input all following characters in the \x mode if they are in the range a to f.

Examples:

\xfc\x62\x65l results in the German word **übel** (b and e are in the range a to f, l is not).

f\xfcndig results in the German word **fündig** (n is not in the range a to f).

Tcl has built in functionality for about 50 common character encodings. You can display the available encodings with the command:

encoding names

Important encodings are:

- **ascii** - pure ascii (single-byte)
- **utf-8** - the internal character set of Tcl (multi-byte)

- **unicode** - the Unicode character set (two-byte)
- **cp850** - the PC DOS character set for DOS shells (single-byte)
- **iso8859-1** - the Latin-1 (Windows) character set (single-byte)

Channel Based Conversion

You can define a character encoding for each input / output channel:

fconfigure *channelid* -encoding *encoding*

Example:

```
set fd [open $file r]
fconfigure $fd -encoding shiftjis
```

Tcl now converts automatically all shiftjis coded characters (shiftjis is a widely used Japanese character set) from the file to the internal utf-8 format.

The Tcl source command always reads files using the system encoding.

You can check the encoding for a specific channel with the command:

fconfigure channelid

Example:

```
fconfigure stdin
fconfigure $fd
```

in a DOS box the encoding is set to cp850 (which is the PC850 character set).

System Encoding

The system encoding is the character encoding used by the operating system. Tcl automatically handles conversions between utf-8 and the system encoding when interacting with the operating system.

Tcl usually can determine a reasonable default system encoding based on the platform and locale settings. If not, it uses ISO8859-1 (Latin1) as default setting.

You can check the actual system encoding with

encoding system

You can redefine the system encoding with

encoding system *encoding*

This is not recommended because the interaction of the system could not work correctly. Use the fconfigure channelid -encoding encoding command instead.

String Conversion

Strings can be converted with the functions:

encoding convertfrom
encoding convertto

Example:

```
set ha [encoding convertfrom utf-8 "\xc3\xbc"]
```

On a DOS terminal window this should result in the echoed output "ü".

B.4. Character Sets

Tcl 8.3 supports a variety of different encodings. (see install_path\lib\tcl8.3\encoding (on Windows) or install_path/lib/tcl8.3/encoding (on UNIX)).The encodings listed in these encoding files can be used in the _localcode variable by dropping the file suffix ".enc".

Example:

> Encoding file name: cp850.enc
> _localcode must be set to "cp850".

The following character sets or code sets are available:

ascii

big5

cp437
cp737
cp775
cp850 (also PC850 for use in _localcode variable)
cp852
cp855
cp857
cp860
cp861
cp862
cp863
cp864
cp865
cp866

cp869
cp874
cp932
cp936
cp949
cp950
cp1250
cp1251
cp1252
cp1253
cp1254
cp1255
cp1256
cp1257
cp1258

dingbats

euc-cn
euc-jp
euc-kr

gb12345
gb1988
gb2312

identity

iso2022
iso2022-jp
iso2022-kr
iso8859-1 (also Latin1 for use in _localcode variable)
iso8859-2
iso8859-3
iso8859-4
iso8859-5
iso8859-6
iso8859-7
iso8859-8
iso8859-9

jis0201
jis0208
jis0212

koi8-r

ksc5601

macCentEuro
macCroatian

macCyrillic
macDingbats
macGreek
macIceland
macJapan
macRoman
macRomania
macThai
macTurkish
macUkraine

shiftjis

symbol

unicode

utf-8 (also UTF8 for use in _localcode variable)

For compatibility reasons the values **UTF8**, **LATIN1** and **PC850** are also supported for the **_localcode** variable.

You can extend the available character sets by writing your own character conversion tables. See the original Tcl 8.3 documentation for details.

DirX Product Suite

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

DirX Identity

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

DirX Directory

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

DirX Access

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

DirX Audit

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

For more information: support.dirx.solutions/about

Legal remarks

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