SAP NetWeaver® Identity Management Connector Development Kit

Virtual Directory Server Connector Testing Tool

Version 7.2 Rev 1

Preface

The product

The SAP NetWeaver Identity Management Connector Development Kit enables independent software vendors (ISVs) or SAP partners to create an Identity Management connector for their application, and to integrate the application into the Identity Management landscape.

The Connector Development Kit contains information necessary for development of an Identity Management connector – criteria, guidelines, templates, test tool, certification guide, etc.

The reader

This manual is written for people who want to test the implemented Virtual Directory Server (VDS) connector for their application.

Prerequisites

To get the most benefit from this manual, you should have the following knowledge:

- Thorough knowledge of the Virtual Directory Server.
- Java programming skills.
- Basic knowledge of LDAP.
- Knowledge and understanding of the target application.

The following software is required:

- SAP NetWeaver Identity Management Virtual Directory Server version 7.1 SP2 or newer, or version 7.2 or newer installed and licensed.
- A Java development environment. This can be downloaded from http://java.sun.com (version 1.4/1.5).
- Access to the target application.
- Access to the target application Java library (API).
- The implemented Virtual Directory Server connector.
- SAP Log Viewer is recommended but not required.

The manual

This manual gives an overview of the Virtual Directory Server (VDS) Connector Testing Tool. The tool has the capacity to test both connector functionality and performance with a high precision.

A short introduction of the test processes of the testing tool is given in the *Introduction*, where you also can find information about what the testing tool package contains. In the *Overview* section a complete description of the testing tool is given, i.e. its general functioning, operations, properties and test cases.

There are two parts of the configuration: one part directly related to the testing tool and another related to the Virtual Directory Server. Configuration is described in section *Configuring the Virtual Directory Server Connector Testing Tool*.

Related documents

You can find useful information in the following documents:

- *SAP NetWeaver Identity Management Security Guide.*
- *SAP NetWeaver Identity Management Operations Guide.*
- *Help files for the Virtual Directory Server.*
- *Java Help for the Virtual Directory Server Connector API.*
- *SAP NetWeaver Identity Management Connector Development Kit Implementing the Virtual Directory Server Connector*.

Table of contents

Introduction

The Virtual Directory Server (VDS) Connector Testing Tool has been developed to test the Virtual Directory Sever connectors. It has the capacity to test both the connector functionality and the connector performance with high precision. In addition, it is possible to run instructions testing and instructions and expected results testing.

The testing tool generates different test cases used for testing the connector. For functionality testing, the expected results are compared to the ones obtained through the Virtual Directory Server connector. When testing the connector performance there is no comparison with the expected data. Thus, it is recommended to first test the connector functionality to ensure that the performance is measured with a correct connector behavior.

The configuration files are used to create adjusted configurations for every entry, attribute and general parameter, giving the user flexibility. In addition to the configuration files, the Virtual Directory Server needs to be configured to achieve the correct testing tool functioning – a class for pre-processing of data received from the testing tool needs to be created and defined for the server (the Virtual Directory Server).

Note:
The configuration file(s) may contain password in clear text.

At the end of each test process, independently of which one it is, a complete summary is reported. In these summaries final information about what happened while the process was running is given.

Functionality testing

The most important of the test processes is the functionality testing. Two different ways of executing the functionality testing are allowed by the testing tool:

- Random: The test cases are distributed randomly in a fair manner, taking into account the virtual tree structure, the possible legal DNs and the state at the moment.
- Exhaustive: An exhaustive test through the tree structure and the initial configuration is made.

How to set the number of test cases is described in section *Configuring the Virtual Directory Server Connector Testing Tool* on page 15.

The test process runs until it either finds some error, all the desired operations are completed or the maximum time is reached. When the test processes (random or exhaustive) find an error, they stop, give a summary report and indicate why the testing tool has stopped the process (the cause of error). If the testing tool finds error in either the initial configuration or the initial state of the system which does not fit with the specifications of the configuration files, it stops and none of the test processes are executed because without a correct specification of every required property or parameter there is no guarantee for the correct behavior of the testing tool.

If the initial configuration is correct and the test process is executed, then a final summary is created. This final summary indicates whether the test process finished successfully or not, and in the case of failure the reason is provided. If no error during the running of the test process is discovered, but the process stopped because the maximum time was reached, then this is indicated as the cause. Another cause could be that it is not possible to launch all the desired operations. This can occur when deleting more entries than available. Another case is when having no possibility of making modifications in the set of attributes in the system.

A complete log contains the information on each test case that is tried and the final summaries. This way it is possible to follow the path of the testing tool and see how it advances for each generated test case and the obtained results. The log file *log.trc* can be found in the folder *logs* in the installation directory of the VDS Connector Testing Tool, and it can be read with the SAP Log Viewer.

An example of a final summary for the functionality test (in the SAP Log Viewer):

Time	Message	Severity
14:39:44:959	--Connector Testing Tool finished its execution	Info
14:39:44:959		Info
14:39:44:959	----Execution time: 1 secs, 0 milisecs	Info
14:39:44:959		Info
14:39:44:959	------Subtree search: 0	Info
14:39:44:959	------One level search: 59	Info
14:39:44:959	------Base search: 3	Info
14:39:44:959	----Number of searches for fake entries type:	Info
14:39:44:959	------Subtree search: 0	Info
14:39:44:959	------One level search: 11	Info
14:39:44:959	------Base search: 514	Info
14:39:44:959	----Number of searches for entry type: "file"	Info
14:39:44:959	----0 searchs with subtree option	Info
14:39:44:959	----70 searchs with one level option	Info
14:39:44:959	----517 searchs with base option	Info
14:39:44:959	------17 modifications with replace option	Info
14:39:44:959	------7 modifications with delete option	Info
14:39:44:959	------18 modifications with add option	Info
14:39:44:959	----42 modifications	Info
14:39:44:959	----20 deletions	Info
14:39:44:959	----20 additions	Info
14:39:44:959	--Internal LDAP Summary	Info
14:39:44:959		Info
14:39:44:959	--Executed 2 "A_NOT_MATCH_LINKED_ATT" test cases	Info
14:39:44:959	--Executed 2 "MA_ILLEGAL_VALUES" test cases	Info
14:39:44:959	--Executed 11 "D_EXISTING_ENTRY" test cases	Info
14:39:44:959	--Executed 3 "D_NOT_EXISTING_ENTRY" test cases	Info
14:39:44:959	--Executed 1 "A_DUPLICATE_DN" test cases	Info
14:39:44:959	--Executed 1 "MA_DUPLICATE_VALUES" test cases	Info
14:39:44:959	--Executed 1 "A_NOT_ALL_MANDATORY" test cases	Info
14:39:44:959	--Executed 7 "MAA_ADD" test cases	Info
14:39:44:959	--Executed 3 "MFA_ADD" test cases	Info
14:39:44:959	--Executed 12 "MD_NORMAL_CASE" test cases	Info
14:39:44:959	--Executed 3 "D_ILLEGAL_DN_ID" test cases	Info
14:39:44:959	--Executed 1 "A_ILLEGAL_DN_ID" test cases	Info
14:39:44:959	--Executed 4 "MD_NOT_EXISTING_VALUES" test cases	Info
14:39:44:959	--Executed 3 "MA_NORMAL_CASE" test cases	Info
14:39:44:959	--Executed 3 "D_ILLEGAL_DN_VAL" test cases	Info
14:39:44:959	--Executed 10 "MAA_REPLACE" test cases	Info
14:39:44:959	--Executed 2 "MD_MODIFIABLE" test cases	Info
14:39:44:959	--Executed 3 "MA_MODIFIABLE" test cases	Info
14:39:44:959	--Executed 1 "MR_MODIFIABLE" test cases	Info
14:39:44:959	--Executed 2 "MDA_REPLACE" test cases	Info
14:39:44:959	--Executed 1 "MA_MULTIVALUE" test cases	Info
14:39:44:959	--Executed 15 "A_LEGAL_DN" test cases	Info
14:39:44:959	--Executed 2 "MFA_DELETE" test cases	Info
14:39:44:959	--Executed 6 "MR_NORMAL_CASE" test cases	Info
14:39:44:959	--Executed 1 "MR_ILLEGAL_VALUES" test cases	Info
14:39:44:959		Info
14:39:44:959	--20 entry DELETIONS were tried out of 20	Info
14:39:44:959	--20 entry ADDITIONS were tried out of 20	Info
14:39:44:959	--20 attribute MODIFICATIONS with REPLACE optionwere tried out of 20	Info
14:39:44:959	--20 attribute MODIFICATIONS with DELETE optionwere tried out of 20	Info
14:39:44:959	--20 attribute MODIFICATIONS with ADD option were tried out of 20	Info
14:39:44:959		Info
14:39:44:959	--Functionality test was executed successfully	Info
14:39:44:959		Info
14:39:44:959	--############ Final Report for Functionality Testing ############	Info

Performance testing

After running the functionality test on the Virtual Directory Server connector, the next step is the performance testing. The performance test can be run before the functionality testing but it is not recommended since errors might occur and it is not known whether the configuration of the testing tool is correct or not. Neither it will be possible to know if the connector behavior is correct, and hence any functional errors should be discovered before executing the performance testing.

To perform the connector performance testing, the testing tool uses a method which tries to execute the required number of operations to test the performance.

This test process tests each of the standard connector operations. It can be decided how many operations must be done for each operation type, i.e. it can be set how many add-entry operations, delete-entry operations and modifications for each one of the three options ("add", "delete" and "replace"), how many base searches, one-level searches and how many one-level searches from the starting point is to be executed. The execution order is the following:

- add-entry operations
- base searches
- one-level searches
- one-level searches from the starting point
- modifications with Add options
- modifications with Delete option
- modifications with Replace option
- delete-entry operations

For base searches the tool traverses the tree and makes a base search for each entry that it finds. For one-level searches the tool behaves the same as for base searches, but with one-level search scope. One-level searches from starting point traverse from a given starting point with one-level scope.

This test process reports a final summary with the information related to each operation set, with the number of operations done, the total time and the average time spent per operation. If the test process stops caused by an error, the error is reported. The most important part is the final summary report. Here it is only indicated which operation is the one being currently tested, and the errors reported from the Virtual Directory Server in the case that they exist.

An example of a complete performance test process log (in the SAP Log Viewer):

Time	Message	Severity
16:04:14:881	------Total consumed time by performance testing process: 672 milisecs	Info
16:04:14:881		Info
16:04:14:881	--------Subtree search: 0	Info
16:04:14:881	--------One level search: 63	Info
16:04:14:881	--------Base search: 44	Info
16:04:14:881	------Number of searches for fake entries type:	Info
16:04:14:881	------0 searchs with subtree option	Info
16:04:14:881	------63 searchs with one level option	Info
16:04:14:881	------44 searchs with base option	Info
16:04:14:881	--------10 modifications with replace option	Info
16:04:14:881	--------30 modifications with delete option	Info
16:04:14:881	--------17 modifications with add option	Info
16:04:14:881	------57 modifications	Info
16:04:14:881	------10 deletions	Info
16:04:14:881	------10 additions	Info
16:04:14:881	----Internal LDAP Summary	Info
16:04:14:881		Info
16:04:14:881	----Executed 9 "MA_NORMAL_CASE" test cases	Info
16:04:14:881	----Executed 10 "A_LEGAL_DN" test cases	Info
16:04:14:881	----Executed 11 "MAA_" test cases	Info
16:04:14:881	----Executed 11 "MDA_DELETE" test cases	Info
16:04:14:881	----Executed 10 "MR_NORMAL_CASE" test cases	Info
16:04:14:881	----Executed 19 "MD_NORMAL_CASE" test cases	Info
16:04:14:881	----Executed 10 "D_EXISTING_ENTRY" test cases	Info
16:04:14:881		Info
16:04:14:881	----Average time by operation: 6 milisecs	Info
16:04:14:881	----Consumed time: 62 milisecs	Info
16:04:14:881	----10 searches with one level option from the initial starting point out of 10	Info
16:04:14:881		Info
16:04:14:881	----Average time by operation: 1 milisecs	Info
16:04:14:881	----Consumed time: 16 milisecs	Info
16:04:14:881	----10 searches with one level option through the virtual tree out of 10	Info
16:04:14:881		Info
16:04:14:881	----Average time by operation: 1 milisecs	Info
16:04:14:881	----Consumed time: 32 milisecs	Info
16:04:14:881	----30 searches with base option out of 30	Info
16:04:14:881		Info
16:04:14:881	----Average time by operation: 9 milisecs	Info
16:04:14:881	----Consumed time: 94 milisecs	Info
16:04:14:866	----10 entry deletions out of 10	Info
16:04:14:866		Info
16:04:14:866	----Average time by operation: 14 milisecs	Info
16:04:14:866	----Consumed time: 140 milisecs	Info
16:04:14:866	----10 entry additions out of 10	Info
16:04:14:866		Info
16:04:14:866	----Average time by operation: 7 milisecs	Info
16:04:14:866	----Consumed time: 78 milisecs	Info
16:04:14:866	----10 modifications with replace option out of 10	Info
16:04:14:866		Info
16:04:14:866	----Average time by operation: 4 milisecs	Info
16:04:14:866	----Consumed time: 125 milisecs	Info
16:04:14:866	----30 modifications with delete option out of 30	Info
16:04:14:866		Info
16:04:14:866	----Average time by operation: 3 milisecs	Info
16:04:14:866	----Consumed time: 63 milisecs	Info
16:04:14:866	----20 modifications with add option out of 20	Info
16:04:14:866		Info
16:04:14:866	----Performance testing process finished successfully	Info
16:04:14:866		Info
16:04:14:866	----############ Final Report for Performance Testing ############	Info

Instructions testing

This test process executes a set of instructions found in the file *Instructions.instr*, which is located in the folder *instr* in the installation directory of the testing tool.

The test process parses the file and tries to execute every instruction, if the complete parsing was done successfully. For each instruction, the process transforms the instruction in a test case and then the test case is executed. If an instruction fits several test cases then all of them are taken into account. Afterwards, the process inspects the list of allowed test cases for the specific situation and picks one of them. For an instruction to be executed, it must fit with some of the allowed test cases.

An example of a complete instructions test summary (in the SAP Log Viewer):

Time	Message	Severity
16:24:52:556	--------Execution time: 219 milisecs	Info
16:24:52:541		Info
16:24:52:541	----------Subtree search: 0	Info
16:24:52:541	----------One level search: 7	Info
16:24:52:541	----------Base search: 48	Info
16:24:52:541	--------Number of searches for fake entries type:	Info
16:24:52:541	--------0 searchs with subtree option	Info
16:24:52:541	--------7 searchs with one level option	Info
16:24:52:541	--------48 searchs with base option	Info
16:24:52:541	----------4 modifications with replace option	Info
16:24:52:541	----------1 modifications with delete option	Info
16:24:52:541	----------0 modifications with add option	Info
16:24:52:541	--------5 modifications	Info
16:24:52:541	--------6 deletions	Info
16:24:52:541	--------4 additions	Info
16:24:52:541	------Internal LDAP Summary	Info
16:24:52:541		Info
16:24:52:541	------Executed 3 "D_ILLEGAL" test cases	Info
16:24:52:541	------Executed 4 "A_LEGAL_DN" test cases	Info
16:24:52:541	------Executed 4 "MR_NORMAL_CASE" test cases	Info
16:24:52:541	------Executed 1 "MD_NORMAL_CASE" test cases	Info
16:24:52:541	------Executed 3 "D_EXISTING_ENTRY" test cases	Info
16:24:52:541		Info
16:24:52:541	------Instructions testing process was aborted because some error occurred during the process. ERROR: The expected result was not the one in the system when the instruction: "mod_del cn...	Info
16:24:52:541		Info
16:24:52:541	------########## Final Report for Instructions Testing ##########	Info

Instructions and expected results testing

This test process is similar to the instructions testing process. It executes the instructions of the file *InstructionsAndExpectedResults.instr* found in the folder *instr*. There are two differences compared to the instructions testing process:

- A new instruction *expected* is available.
- The correctness of the executed operations is checked only when instruction *expected* is found, i.e. no test case is generated when executing an instruction (the instruction will just be executed) but the instruction *expected*. The instruction is based on a base search, i.e. the instruction *expected* has all the required parameters for a base search and the expected results of the proper search.

An example of a complete instructions and expected results test summary (in the SAP Log Viewer):

Time	Message	Severity
16:43:47:134	--Connector Testing Tool finished its execution	Info
16:43:47:134		Info
16:43:47:134	----Execution time: 16 milisecs	Info
16:43:47:134		Info
16:43:47:134	------Subtree search: 0	Info
16:43:47:134	------One level search: 0	Info
16:43:47:134	------Base search: 1	Info
16:43:47:134	----Number of searches for fake entries type:	Info
16:43:47:134	----0 searchs with subtree option	Info
16:43:47:134	----0 searchs with one level option	Info
16:43:47:134	----1 searchs with base option	Info
16:43:47:134	------0 modifications with replace option	Info
16:43:47:134	------0 modifications with delete option	Info
16:43:47:134	------0 modifications with add option	Info
16:43:47:134	----0 modifications	Info
16:43:47:134	----0 deletions	Info
16:43:47:134	----1 additions	Info
16:43:47:134	--Internal LDAP Summary	Info
16:43:47:134		Info
16:43:47:134		Info
16:43:47:134	--Instructions testing process was aborted because some error occurred during the process. ERROR: ERROR: Unexpected error	Info
16:43:47:134		Info
16:43:47:134	--########## Final Report for Instructions And Expected Results Testing ##########	Info

The package – files and folders

The VDS Connector Testing Tool package contains a set of files and folders which contain the information regarding the configuration settings, the log and the documentation. The files are:

File name	Description
readme.txt	This file contains a short description of each component that forms the complete package.
Z_preprocessingAdd.java	This class must be added to the Virtual Directory Server configuration. For more details see section *Configuring the Virtual Directory Server* on page 28.
VDS_CTT.jar	Running this file starts the testing tool. See section *Using the Virtual Directory Server Connector Testing Tool* on page 34 for more.

The folders in the package are:

Folder name	Description
logs	This folder contains the log file *log.trc*. Its information can be viewed with any text viewer, but to view it in a proper format the SAP Log Viewer must be used.
lib	The jar files needed by *VDS_CTT.jar* are found in this folder: *logging.jar* *mvd.jar* *vdstools.jar* *vdsverifier.jar*

Folder name	Description
doc	This is the folder where the documentation is located. It contains the files providing the information needed to understand the details about the VDS Connector Testing Tool.
conf	The folder where the main configuration files are located: *GeneralConfiguration.conf* – the file that contains the general configuration information. *EntryConfiguration.conf* – the file that contains the attributes configuration information. *AttributeConfiguration.conf* – the file that contains the entries configuration information.
instr	This folder contains the file *Instructions.instr* with the set of instructions for the instructions testing (see section *Instructions testing* on page 5), and the file *InstructionsAndExpectedResults.instr* with the set of instructions for the instructions and expected results testing (see section *Instructions and expected results testing* on page 5).
regex	The folder contains the file *regex.conf*, which must contain any regular expressions except the predefined ones that is used in the configuration files (in the *conf* folder). See section *Regular expressions* on page 37 for more.

Tips and recommendations

An important thing to have in mind is the fact that the testing tool runs the search methods but does not test them explicitly. If these methods do not work properly, then the obtained results might not be the correct ones. However, testing of search methods is possible to some extent, indirectly. When testing for example modification operation, a search method will be used to retrieve the new modified value for comparison purposes (the new value will be compared to the calculated, expected one). If the modified value obtained by the search method is not correct compared to the expected one, this means that either the modification operation failed to modify with the correct value or the search operation failed to retrieve the correct value. It is possible to find out whether the modification operation is causing the error or not, which will indirectly give the information about the search method.

Sometimes it can happen that there are not enough entries to delete, and then the deletion and modification operations can not be executed because of lack of entries. In cases like this, you should only need to increase the number of entries to be added. Another possible solution is to reduce the number of required deletions.

Another thing is related to the number of launched operations. Configuring the tool to execute a given number of add-operations for example, does not necessary mean that the exact same number of entries is added. The tool tries to launch the specified number of operations, but some may fail (may be invalid for example). Illegal add-operations do not add any entries. The same goes for any other type of operation.

It is recommended to get acquainted with the general constraints and the property constraints described in section *Constraints* on page 35 in this document.

Connector Testing Tool Overview

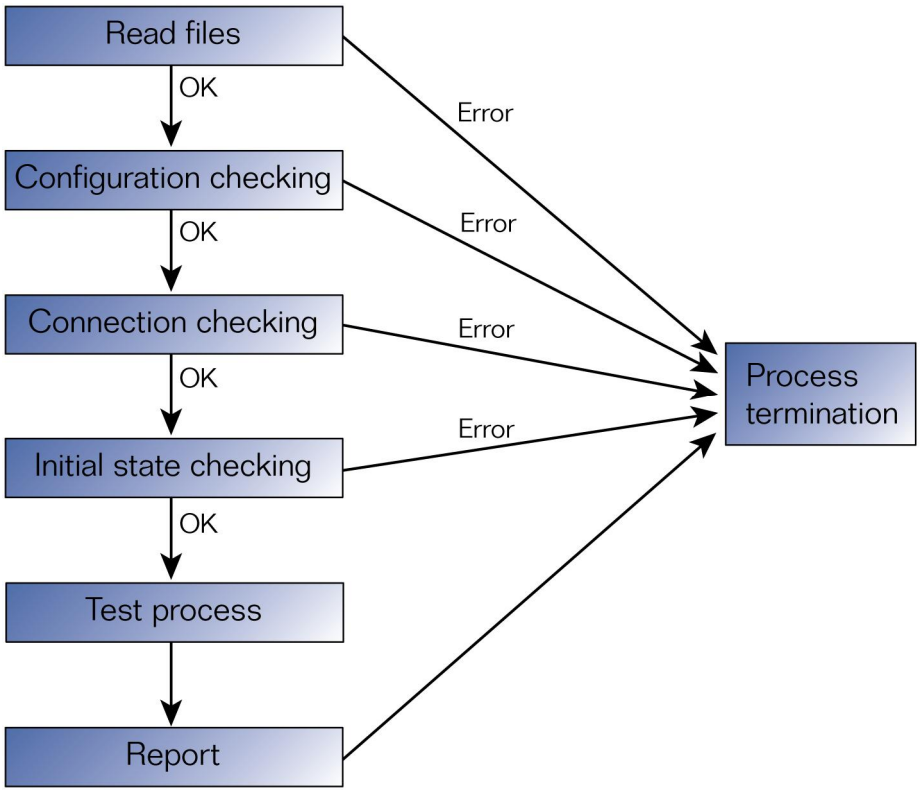
In this section an overview is given over the testing tool general behavior, operations, properties and test cases.

Test process

The VDS Connector Testing Tool will have a standard behavior independently of the test chosen (functionality, performance, instructions or instructions and expected results). The testing tool goes through the following steps:

- Read files: The testing tool reads the configuration files and any other necessary file to obtain the requirements of the configuration files. In this phase, an initial checking of the read configurations is made.
- Configuration checking: After all files are read, the testing tool checks the configurations trying to find incompatibilities between configurations and other errors.
- Connection checking: In this phase, the testing tool tries to connect to the Virtual Directory Server.
- Initial state checking: This is the phase where the initial state of the system is checked to confirm that this initial state is compatible with the specifications of the configuration files.
- Test process: The tool launches the chosen test process.
- Report: A complete report is written giving detailed information about the test process.

An illustration of the general behavior steps is presented below:



Operations and expected behavior

For the set of operations described in this section it is assumed that every parameter of every operation fits with the specifications of the configuration files.

Add entry

This operation adds a new entry if the entry conforms to the specifications of the configuration files, i.e. if it has a legal *DN* and all its attributes fit with the specifications of the configuration files. For more on specifications of the configuration files see section *The configuration files* on page 15.

Delete entry

Using this operation, an existing entry is deleted using its *DN* in the operation.

Modify entry with ADD option

This operation adds one or more values to an attribute if the values to be added conform to the specifications of the attribute. If the attribute is not multi-value then an addition has no effect. There are exceptions, for example if the attribute allows appending the new values to the existing one, i.e. if the attribute has property *appendable* (see page 11). Then if the attribute has a value "a" and the value "e" is added, then the new value would be "ae".

If the attribute is multi-value but does not allow duplicate values (see page 11) then this needs to be handled as well.

For more on specifications of the configuration files see section *The configuration files* on page 15. The final state of the attribute, after the operation is completed, depends on the properties of the attribute.

This operation can also add attributes to an entry (and not just add values to an attribute as mentioned above). This happens when executing this operation on an attribute that is not part of the entry.

Modify entry with DELETE option

This operation deletes one or more values if the values to be deleted are part of the value set of the attribute. If all the values are deleted (and the new attribute value is NULL) then the attribute is removed, except when the attribute has a property which allows no value, i.e. the attribute has an empty value (see page 11). For such an attribute, this means that if it has a value "a" and the value is deleted, the attribute will have " " as value and not NULL as it normally is the case.

Modify entry with REPLACE option

With this operation, a new value replaces one or more (all) current values of an attribute. If the new value is NULL then the attribute is removed, except when the attribute has a property that allows it to contain no value (see page 11).

Note:
Attributes which have values that form part of the DN must be mandatory and not modifiable. Every attribute must have legal values only. Illegal values must not be accepted as the attribute values.

Base search

This operation lists the attributes of an existing entry. Given a starting point, the operation tries to make base searches for all the entries that exist in the sub-tree under the starting point. More precisely, after the starting point is provided the operation makes a one-level search, and for every entry it finds this way it makes a base search. The time it takes for the one-level search to complete is not taken into account when measuring performance.

One-level search

Given a starting point, this operation tries to make one-level searches under any entry that is found in the sub-tree under the initial starting point, i.e. the procedure is the same as for the base search mentioned above except that for this operation only one-level searches are made.

One-level search under the same starting point

This operation lists all entries under a given starting point and then the operation is repeated under the same given starting point, unlike for the two previously mentioned search operations where the searches are made for entries found under the provided starting point.

Properties

Every attribute has a set of properties. Properties are used to describe the characteristics of an attribute. They define how the attribute changes when an operation is executed on it. Hence, the attribute properties have a direct influence on the appearance of any entry. Attributes have no properties by default, so every property that an attribute must have needs to be assigned.

Properties are listed and described in this section.

linked

If an attribute has this property, its value is used to construct the *DN* value. This property cannot be assigned explicitly. Instead it is assigned automatically when the *DN* specifications are defined.

virtual

An attribute with this property is a virtual attribute, i.e. not actually existing in the real system and as such not listed in the attribute list of any entry in the real system. Instead it is used only to construct the *DN*s where it is desirable to give a value for the *RDN* and the values are not linked to any attribute.

must

Any attribute with this property is a mandatory attribute. It must always be a part of the entry where it is defined.

may

An attribute with this property is optional.

automaticMust

An attribute with this property is automatically added to the entry where it is defined.

automaticMay

An attribute with this property is automatically added as optional for the entry where it is defined.

allowNoValue

An attribute with this property can have an empty value.

appendable

When an attribute with this property is modified with *ADD* option, the value to be added is appended to the end of the current value of the attribute. Having an attribute with a value "a" modified with *ADD* option, where the value for the modification is "e", the new value of the attribute is "ae".

multivalue

This property defines the multi-value attributes. Every attribute not explicitly defined as multi-value is by default a single-value attribute.

allowDuplicate

This parameter represents the list of multi-value attributes allowed to have the same value repeated in the value list.

modifiable

An attribute with this property can be modified.

Test cases

The Virtual Directory Server Connector Testing Tool is able to launch up to 28 different test cases. These test cases test what happens when both legal and illegal operations are executed.

Test cases can be split into two different groups:

- Test cases related to entry specifications.
- Test cases related to attribute specifications.

The test cases related to attributes (18 test cases) are a subset of the test cases related to entries (all 28 test cases available in the testing tool). This leaves a set of 10 test cases which are only entry related (and not attribute related). Test cases are introduced by the identifier that must be used when configuring the testing tool (making use of the configuration files) and are prefixed by the following prefixes:

Prefix	Description
A	Add; adds entry (it has both legal and illegal variants).
D	Delete; deletes entry (it has both legal and illegal variants).
MA	Modify with ADD option; adds one or more values to an attribute (it has both legal and illegal variants).
MAA	Modify attribute with ADD option; adds an attribute to an existing entry (it has only legal variants).
MD	Modify with DELETE option; deletes one or more values of an attribute defined for an existing entry, but does not delete the attribute (it has both legal and illegal variants).
MDA	Modify attribute with DELETE option; deletes an attribute from an existing entry (it has only legal variants).
MFA	Modify fake attribute; modifies the values of a non-existing attribute (it has only legal variants).
MR	Modify with REPLACE option; replaces the set of values of an attribute defined for an existing entry by a new one (it has legal and illegal variants).

Attribute related test cases

The attribute related test cases are:

Test case identifier	Description
MA_DUPLICATE_VALUES	This test case tries to add duplicate values to a multi-value attribute that does not admit duplicate values.
MA_ILLEGAL_VALUES	This test case tries to add illegal values to an attribute.
MA_MODIFIABLE	This test case tries to modify with *ADD* option a read-only (non modifiable) attribute.
MA_MULTIVALUE	This test case tries to add one or more values to an attribute that is not multi-value.
MA_NORMAL_CASE	This test case tries to add one or more values to an attribute in a legal way.
MD_MODIFIABLE	This test case tries to modify with *DELETE* option a read-only (non modifiable) attribute.
MD_NORMAL_CASE	This test case tries to delete one or more existing attribute values.
MD_NOT_EXISTING_VALUES	This test case tries to delete one or more non existing values of an attribute.
MDA_DELETE	This test case tries to delete an attribute making use of modify with *DELETE* option operation.
MDA_REPLACE	The test case tries to delete an attribute making use of modify with *REPLACE* option operation.
MR_ILLEGAL_VALUES	This test case tries to replace one or more attribute values with an illegal value. The set of legal values is given by the specifications of the configuration files.
MR_MODIFIABLE	This test case tries to modify with *REPLACE* option a read-only (non modifiable) attribute.
MR_NORMAL_CASE	This test case tries to delete one or more attribute values in a legal way.
MAA_ADD	The test case tries to add an attribute making use of modify with *ADD* option operation.
MAA_REPLACE	The test case tries to add an attribute making use of modify with *REPLACE* option operation.
MFA_ADD	The test case tries to modify with *ADD* option a non declared attribute.

Test case identifier	Description
MFA_DELETE	This test case tries to modify with *DELETE* option a non declared attribute.
MFA_REPLACE	This test case tries to modify with *REPLACE* option a non declared attribute.

Only entry related test cases

The following only entry related test cases are available:

Test case identifier	Description
A_DUPLICATE_DN	This test case tries to add an entry with the same *DN* as for an existing entry.
A_ILLEGAL_DN_ID	This test case tries to add an entry with an illegal ID in its *DN*. E.g. if the legal specification for the *DN* is for example *cn=<mskeyvalue>,o=idstore*, then an illegal ID would be: *illegalID=<mskeyvalue>,o=idstore*.
A_ILLEGAL_DN_VAL	This test case tries to add an entry with an illegal value in its *DN*. E.g. if the legal specification for the *DN* is for example *cn=<mskeyvalue>,o=idstore*, then an illegal value would be: *cn=illegalValue,o=idstore*.
A_LEGAL_DN	This test case tries to add a legal entry.
A_NOT_ALL_MANDATORY	This test case tries to add an entry without all its mandatory attributes.
A_NOT_MATCH_LINKED_ATT	This test case tries to add an entry with linked attributes which do not match with the *DN*.
D_EXISTING_ENTRY	This test case tries to delete an existing entry.
D_ILLEGAL_DN_ID	This test case tries to delete a non existing entry where the *DN* has an illegal identifier.
D_ILLEGAL_DN_VAL	This test case tries to delete a non existing entry where the *DN* has an illegal value.
D_NOT_EXISTING_ENTRY	This test case tries to delete a non existing entry with legal *DN*.

Configuring the Virtual Directory Server Connector Testing Tool

The VDS Connector Testing Tool uses three configuration files located in the folder *conf* in the installation directory:

- One for the general configuration where the general configuration parameters and properties must be set.
- One for the configuration of entries.
- One for the configuration of attributes.

The Virtual Directory Server must be running and the configuration for the connector must be properly set. A pre-processing class *Z_preprocessingAdd.java* must be added to the Virtual Directory Server configuration to be tested (see section *Configuring the Virtual Directory Server* on page 28).

The configuration files are designed to give flexibility to the user by allowing the user to create adjusted configurations for every entry, attribute and general parameter. The only expected connector behavior is the one described in section *Connector Testing Tool Overview* on page 8. But the constructed connectors to be tested can have any other behavior. The way for adapting each specific behavior to the connector basis must be made making use of the configuration files described in the following subsections.

The configuration files

In this section the configuration files are described. These files contain all the configuration information that the testing tool needs to perform correctly. The following rules apply:

- Spaces at the beginning and at the end of any line are ignored.
- Any line starting with # is considered a comment.
- Any blank line is ignored.

The following configuration files are described:

Name	Location (folder in the installation directory)	Description
GeneralConfiguration.conf	conf	The file contains general information which is necessary for the correct configuration of the testing tool.
EntryConfiguration.conf	conf	The file contains entry specifications.
AttributeConfiguration.conf	conf	The file contains attribute specifications, used to define characteristics of every attribute, and redefine if necessary.
Instructions.instr	instr	The file contains the set of instructions to be executed during the instructions testing. This file needs to be configured only when executing the instruction testing.

Name	Location (folder in the installation directory)	Description
InstructionsAndExpectedResults.instr	instr	The file contains the set of instructions to be executed during the instructions and expected results testing.
Regex.conf	regex	The file contains a regular expression specification. The file needs to be configured only if the regular expressions that are not the predefined ones are to be used. These regular expression specifications, together with the predefined regular expressions, are used to define the set of values of every attribute.

General configuration file

This file contains the definition of some basic information needed for the proper functioning of the testing tool. The following contents of this file are described:

- Functionality, performance and instruction parameters
- Functionality and performance parameters
- Performance parameters
- Connection parameters
- Other configuration parameters

Functionality, performance and instruction parameters

This set of parameters contains only one parameter – a maximum allowed time for the chosen test process:

```
time=[INTEGER] #where [INTEGER] indicates the time in seconds
```

Functionality and performance parameters

This set of parameters is common for both the functionality and performance tests. These ones are:

```
#The number of desired attribute modifications with ADD option.
modificationsAdd=[INTEGER]

#The number of desired attribute modifications with DELETE option.
modificationsDel=[INTEGER]

#The number of desired attribute modifications with REPLACE option.
modificationsRep=[INTEGER]

#The number of desired entry additions.
additions=[INTEGER]

#The number of desired entry deletions.
deletions=[INTEGER]
```

Performance parameters

This set of parameters is specific for the performance testing:

```
#The desired number of base searches.
baseSearches=[INTEGER]
```

```
#The desired number of one-level searches.
oneLevelSearches=[INTEGER]

#The desired number of one-level searches from the given starting point.
oneLevelSearchesFromSP=[INTEGER]
```

The starting point is given as a parameter in the general configuration file (see *Connection parameters* below).

Connection parameters

The set of connection parameters contains the following parameters:

```
#The starting point, i.e. where the testing tool starts its work from.
startingPoint=[STRING] #where [STRING] is legal DN, e.g. startingPoint=o=idstore
#or startingPoint=dir=testfbc,dir=testing,root=c

#The server where the VDS configuration to be tested is executed.
server=[STRING] #where [STRING] is the name of the server, e.g. server=localhost,
#server=10.55.164.90 or server=example.com

#The port the VDS is listening on.
port=[INTEGER] #where [INTEGER] is the port number, e.g. port=9389

#This is the user identifier for the configuration that is to be tested.
user=[STRING] #where [STRING] is user ID, e.g. user=admin

#The password for the user defined above.
password=[STRING] #where [STRING] is the user password, e.g. password=password
```

Other configuration parameters

Other existing configuration parameters are:

```
#This parameter defines what type of test is executed: performance,functionality,
#instructions or instructions and expected results.
testType=[STRING] #where [STRING] is type of test, e.g. testType=performance

#The paramter restricts the strength of the test: soft, medium, hard or super.
testingLevel=[STRING] #where [STRING] is the streangth, e.g. testingLevel=soft
```

Note:
The test cases are grouped in the following way:

SOFT	MEDIUM	HARD	SUPER
MA_NORMAL_CASE	MA_MODIFIABLE	MFA_ADD	MA_ILLEGAL_VALUES
MD_NORMAL_CASE	MA_MULTIVALUE	MFA_DELETE	MR_ILLEGAL_VALUES
MR_NORMAL_CASE	MA_DUPLICATE_VALUES	MFA_REPLACE	A_ILLEGAL_DN_VAL
MAA_ADD	MD_MODIFIABLE	A_ILLEGAL_DN_ID	
MDA_DELETE	MD_NOT_EXISTING_VALUES	A_NOT_MATCH_LINKED_ATT	
MDA_REPLACE	MR_MODIFIABLE	A_NOT_ALL_MANDATORY	
A_LEGAL_DN	D_NOT_EXISTING_ENTRY	D_ILLEGAL_DN_ID	
D_EXISTING_ENTRY	D_ILLEGAL_DN_VAL		
MAA_REPLACE	A_DUPLICATE_DN		

Values "soft", "medium", "hard" and "super" are inclusive by strength, i.e. "medium" includes the test cases for both "soft" and "medium", "hard" includes the test cases defined for both "soft", "medium" and "hard", while "super" includes the test cases defined for all abovementioned in addition to itself.

```
#This parameter sets the level on which the collateral effects are tested for:
#soft, medium or hard. E.g. when adding an entry, with "soft" value it is only
#tested if the entry actually is added. With "medium" value it is in addition
#checked for the existence of all the other entries in the system under the
#current starting point. With value "hard" the correctness of all attributes for
```

```
#all the existing entries under the current startng point is checked, in addition
#to the abovementioned.
collateralEffectsChecking=[STRING] #where [STRING] is the level, e.g.
#collateralEffectsChecking=hard

#The parameter indicates whether the DN is compared just by its value or not.
dnEqualsByValues=[STRING] #where [STRING] is "yes" or "no"
```

Note:
Let DN1 and DN2 be two DNs, where DN1: id1_1=val1_1, id1_2=val1_2, …, d1_N=val1_N, and DN2: id2_1=val2_1, id2_2=val2_2, …, id2_M=val2_M.

If "yes" is selected, then these two DNs are equal if, and only if:

- *N=M, and*
- *val1_X=val2_X, for all X=1,2,…,N.*

If "no" is selected, then the two DNs are equal if and only if:

- *N=M, and*
- *val1_X=val2_X, for all X=1,2,…,N, and id1_X=id2_X, for all X=1,2,…,N.*

```
#The parameter holds the seed for the test process.
seed=[INTEGER] #where [INTEGER] is the seed, e.g. seed=47677
```

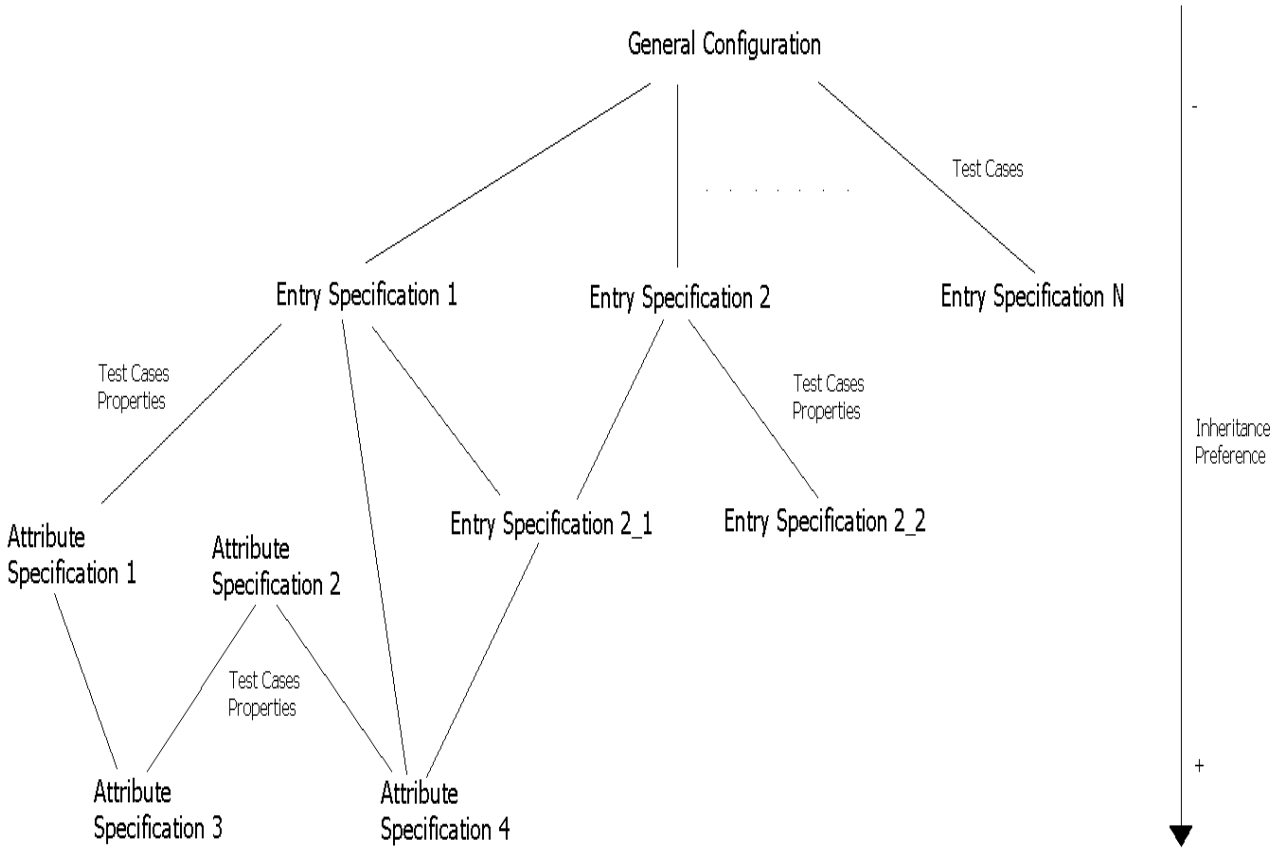
Note:
Changing this seed the test process should create different test cases even with the same initial configuration. It permits creating of the same test set if the same initial configuration and the same seed are kept. Therefore, the test cases can be changed only by modifying the initial configuration or the seed.

A general configuration might look something like this:

```
# ---- Functionality, Performance and Instructions Parameters ----
#
time = 10000
#
# ------------ Functionality and Performance Parameters ----------
#
modificationsAdd = 20
modificationsDel = 30
modificationsRep = 10
additions = 10
deletions = 10
#
# ---------------------- Performance Parameters ------------------
#
baseSearches = 30
oneLevelSearches = 10
oneLevelSearchesFromSP = 10
#
# ----------------------- Connection Parameters ------------------
#
startingPoint = o=idstore
server = localhost
port = 3389
user = admin
password = password
securityOption = null
#
# ------------------- Other configuration parameters -------------
#
testType = instructions
testingLevel = soft
collateralEffectsChecking = hard
dnEqualsByValues = no
seed = 47677
```

Entries configuration file

All entry related specifications are defined in this file. Each entry specification has a set of mandatory parameters and a set of optional parameters. Therefore, an inheritance structure between the general configuration, the entry specifications in the entries configuration file and the attribute specifications in the attributes configuration file exists:



This inheritance structure can be summarized in the following way:

- Each entry specification inherits all the allowed test cases according to the general configuration file (the parameter *testingLevel* in the general configuration file).
- Each entry specification can inherit the test cases and the set of attributes (with their corresponding properties) from other entry specifications. Multi-inheritance is allowed and collisions are resolved by preferences. The parameter where the inherited entry specifications are defined is the parameter *extends* in the entries configuration file.
- Each entry specification can redefine its set of test cases and attributes (with their corresponding properties).
- The entry specifications that are taken into account when constructing the legal entries have the parameter *RDN*.
- Each attribute of every attribute specification has the test cases and parameters that the entry specification where the attribute was defined has.
- Each attribute specification can inherit the test cases, the set of parameters and the set of legal values from other attribute specifications.
- Each attribute specification can redefine its set of parameters and the set of attribute related test cases.

- The parameter *attrs* defines the set of attributes in an attribute specification. For each attribute defined in this parameter, the same properties as the ones defined for the attribute specification are applied.

To understand this inheritance structure completely it might be necessary to read this complete section and the following section *Attributes configuration file* on page 22.

The contents of this file have the following structure:

```
Entry specification 1
@
Entry specification 2
@
.
.
.
@
Entry specification N
```

Mandatory parameters of an entry specification

There is only one mandatory parameter which every entry specification must have. The parameter is the identifier of the entry specification:

```
ID=[STRING] #where [STRING] is any chain of characters given by the regular
#expression [-0-9A-Za-z_]+
```

Optional parameters

The optional parameters are:

```
#The parameter contains the list of inherited entry specifications.
extends=[STRING] #where [STRING] is the list of entries separated by commas and
#delimited by "[]", e.g. extends=[id1, id2]
```

Note:
To resolve the collision problems, the next policy is followed: the more to the left an element is the higher preference it has (i.e. the elements are listed in the preferred order). So, if an entry specification with extends=[id1,id2] exists, where the entry specification with the identifier id1 has for the test case X the value "yes", and the one with the identifier id2 has the value "no" for the same test case, then the inherited value for the test case is "yes".

```
#The complete DN which identifies an entry in the real system.
RDN=[STRING] #where [STRING] can be defined in the following way:
#dnId1=valRdn,dnId2=val,…,dnIdN=valVirt, where each dnIdX, X=1,2,…,N belongs to
#the set of characters given by the regular expression [-0-9A-Za-z_]+. valRdn can
#be represented by the regular expression (<[-0-9A-Za-z_]+>)+(\?|\*|\+|) and
#valVirt can be represented by the regular expression [-0-9A-Za-z_], e.g.
#RDN=[uid=Jane Bush,o=org] or RDN=[file=file2,dir=dir2,root=c]
```

Note:
A simple example of RDN declaration could be: uid=<firstname><surname>,o=org

where

- *uid: the real RDN for the real system.*
- *<firstname><surname>: two attributes of the entry. Each attribute must have a set of legal values given by a regular expression. Sometimes it is desirable to create entries without linking any attribute to the real RDN. In this case, it is possible to declare the attributes of the real RDN as virtual attributes.*
- *o: represents an identifier of the DN.*
- *org: a constant.*

Using this example, if one entry is created where the attribute firstname has value "Jane" and surname has value "Bush", then an entry with DN: uid=Jane Bush,o=org is created. Space is used to separate every two consecutive attributes values that are part of the DN. So, between Jane and Bush there is a space.

Another RDN declaration is: RDN = [file=<file>,dir=<dir>,root=c]. Some DNs that match with this RDN declaration are (without paying any attention to the values of the linked attributes):*

- *file=file1,root=c*
- *file=file2,dir=dir2,root=c*
- *file=file3,dir=dir1_1,dir=dir1,root=c*

```
#The list of cases allowed to execute for this entry.
testCasesYes=[STRING] #where [STRING] is a list of test case identifiers separated
#by commas and delimited by "{}"

#The list of cases not allowed to execute for this entry.
testCasesNo=[STRING] #where [STRING] is a list of test case identifiers separated
#by commas and delimited by "{}"

#The list of attributes with property "virtual". The attributes with this
#property are never a part of the attribute list for any entry in the real
#system, instead they are just used to construct DNs were giving a value for the
#RDN is desired but the values are not linked to any attribute.
virtual=[STRING] #where [STRING] is the list of attribute names separated by
#commas and delimited by "{}"

#This parameter represents the list of mandatory attributes.
must=[STRING] #where [STRING] is the list of attribute names separated by commas
#and delimited by "{}"

#The list of optional attributes.
may=[STRING] #where [STRING] is the list of attribute names separated by commas
#and delimited by "{}"

#The list of attributes that the connector adds automatically as mandatory.
automaticMust=[STRING] #where [STRING] is the list of attribute names separated by
#commas and delimited by "{}"

#The list of attributes that the connector adds automatically as optional.
automaticMay=[STRING] #where [STRING] is the list of attribute names separated by
#commas and delimited by "{}"

#The list of attributes that can have an empty value (usually attributes with
#empty values are deleted).
allowNoValue=[STRING] #where [STRING] is the list of attribute names separated by
#commas and delimited by "{}"

#The list of attributes with property "appendable". For these attributes, when
#modified with ADD option, the values to be added are appended to the end of the
#current value of the attribute. So, if an attribute has a value "a" and it is
#modified with ADD option where the value for the modification is "e", then the
#new value of the attribute is "ae".
appendable=[STRING] #where [STRING] is the list of attribute names separated by
#commas and delimited by "{}"

#This parameter represents the list of multi-value attributes. Any attribute not
#explicitly declared as multi-value is by default a single-value attribute.
multivalue=[STRING] #where [STRING] is the list of attribute names separated by
#commas and delimited by "{}"

#The list of multi-value attributes that are allowed to have the same value
#repeated in the list (have a duplicate).
allowDuplicate=[STRING] #where [STRING] is the list of attribute names separated
#by commas and delimited by "{}"

#The list of modifiable attributes.
modifiable=[STRING] #where [STRING] is the list of attribute names separated by
#commas and delimited by "{}"
```

An entries configuration file may look something like this:

```
ID = uid
RDN = cn=<cn>,o=idstore
virtual = {cn}
must = {mx_entrytype}
may = {mobile,address,postalcode,displayname,telephone,email,firstname,surname}
automaticMust = {mskey,mskeyvalue}
testCasesNo = {MDA_REPLACE}
```

Attributes configuration file

This file contains all attribute specifications. Each attribute specification is composed by both a set of mandatory parameters and a set of optional parameters. The attribute specifications are created to give a set of legal values for each attribute declared in the complete set of entry specifications in the entries configuration file. It is possible to reassign any property to any attribute, and reassign any test case directly related to attribute modifications.

The contents of this file have the following structure:

```
Attribute specification 1
@
Attribute specification 2
@
.
.
.
@
Attribute specification N
```

Mandatory attributes

There is only one mandatory attribute which is the identifier of the attribute specification:

```
ID=[STRING] #where [STRING] is any chain of characters from the set of characters
#given by the regular expression [-0-9A-Za-z_]+
```

Optional attributes

The optional attributes are:

```
#The parameter contains the list of inherited attribute specifications.
extends=[STRING] #where [STRING] is the list of specifications separated by commas
#and delimited by "[]", e.g. extends=[id1, id2]
```

Note:
To resolve the collision problems, the next policy is followed: the more to the left an element is the higher preference it has (i.e. the elements are listed in the preferred order). So, if an entry specification with extends=[id1,id2] exists, where the entry specification with the identifier id1 has for the test case X the value "yes", and the one with the identifier id2 has the value "no" for the same test case, then the inherited value for the test case is "yes".

```
#This parameter represents the list of multi-value attributes. Any attribute not
#explicitly declared as multi-value is by default a single-value attribute.
attrs=[STRING] #where [STRING] is the list of either attribute names, or
#attribute names joined to an entry specification identifier with the character
#"@" between them, separated by commas and delimited by "{}"
```

Note:
In cases where only the attribute name exists, all the characteristics of this attribute specification are asssigned to all the attributes with the given name, independently of the entry. In cases where "attribute name @ entry specification identifier" exists, the characteristics of this attribute specification are given only to the attribute related to the entry specification identified by the identifier "@".

```
#The parameter representing regular expression identifier. The regular expression
#identifier must be either a predefined one or the one defined in the regular
#expressions configuration file.
legalValues=[STRING] #where [STRING] is the regular expression identifier
```

Note:
For more, see section Regular expressions on page 37.

```
#The list of cases allowed to execute for this attribute.
testCasesYes=[STRING] #where [STRING] is a list of test case identifiers separated
#by commas and delimited by "{}"
```

Note:
The cases that can be a part of this list are:

- *MA_DUPLICATE_VALUES*
- *MA_ILLEGAL_VALUES*
- *MA_MODIFIABLE*
- *MA_MULTIVALUE*
- *MA_NORMAL_CASE*
- *MD_MODIFIABLE*
- *MD_NORMAL_CASE*
- *MD_NOT_EXISTING_VALUES*
- *MDA_DELETE*
- *MDA_REPLACE*
- *MR_ILLEGAL_VALUES*
- *MR_MODIFIABLE*
- *MR_NORMAL_CASE*
- *MAA_ADD*
- *MAA_REPLACE*

```
#The list of cases not allowed for this attribute. The list of available cases is
#the same as for the previous parameter testCasesYes.
testCasesNo=[STRING] #where [STRING] is a list of test case identifiers separated
#by commas and delimited by "{}"

#The list of properties that are assigned to all the attributes listed in the
#parameter attrs.
propertiesYes=[STRING] #where [STRING] is the list of property identifiers
#separated by commas and delimited by "{}"

#This parameter represents the list of properties that are removed from the
#attributes listed in the parameter attrs.
propertiesNo=[STRING] #where [STRING] is the list of property identifiers
#separated by commas and delimited by "{}"
```

An attributes configuration file may look something like this:

```
ID = nameAtts
attrs = {firstname,surname,displayname}
legalValues = name
propertiesYes = {modifiable}
#
@
#
ID = telAtt
attrs = {telephone,mobile}
legalValues = telephone
propertiesYes = {modifiable}
#
@
#
ID = emailAtt
attrs = {email}
legalValues = email
propertiesYes = {modifiable}
#
@
#
ID = addressAtt
attrs = {address}
legalValues = address
propertiesYes = {modifiable}
#
@
#
ID = identAtt
attrs = {mskeyvalue,cn}
legalValues = ident
#
@
#
ID = numAtt
attrs = {mskey}
legalValues = NUMERIC
#
@
#
ID = postalAtt
attrs = {postalcode}
legalValues = postalcode
propertiesYes = {modifiable}
#
@
#
ID = mx_entrytype
attrs = {mx_entrytype}
legalValues = objectclass
```

Regular expressions configuration file

In this file a set of regular expression specifications is defined. Each specification consists of an identifier and a regular expression. The identifier is a string. And the regular expression has to fit with the regular expression definition given in section *Regular expressions* on page 37.

A regular expressions configuration file may look something like this:

```
mytelephone = \(\+[1-9][0-9]{1,2}\) ([0-9]{3} [0-9]{2} [0-9]{3}|[0-9]{3} [0-9]{3} [0-9]{3}|[0-9]{3} [0-9]{3} [0-9]{4})
postalCode = [1-9][0-9]{3,4}
address = (St. |Av. )[A-Z][a-z]{1,8}( [A-Z][a-z]{1,8})?, [1-9][0-9]{0,2}
objectclass = person
name = [A-Z][a-z]{1,8}( [A-Z][a-z]{1,8})?
description = [A-Za-z0-9]{1,100}
type = .pers
#
# ------------------------------------------------------------------------------
# INTEGER       A Java integer
# DOUBLE        A Java double
# BIG_INTEGER   An integer of any length
# BIG_DOUBLE    A double of any length
# CHAR          The set of char given by \.
# ALPHA         A set of alpha characters (including the empty set)
# NUMERIC       A set of digits (including the empty set)
# TELEPHONE     "(\\(\\+[0-9]\\)|\\(00[0-9]+\\))(-[0-9]+)+"
# STRING        "[-a-zA-Z0-9.,øæå\\\\+?*|!'#£\\$¤%&/()=\\{\\}~_:;\\^]*"
# ALPHANUMERIC  A set of alpha characters and digits (including the empty set)
# ------------------------------------------------------------------------------
```

Instructions configuration file

This file contains the set of instructions executed when the instructions test is selected as the test to be run. The instructions are executed in the order that they are found in the file. Each instruction is converted to a test case. If there are several test cases that fit with the instruction then one of them is selected.

There are five different instructions:

- *add DN {attrName1=attrVal1,attrName2=attrVal2,...,attrNameN=attrValN}*
  This instruction is used to add an entry. The instruction is divided into three main parts: The command *add*, the *DN* of the entry to add, and a list of attributes. Each attribute is defined by an attribute name and an attribute value. If several attributes with the same name are added, then one attribute with this name is added but with all the values given for the attribute with this same name. In cases where the attribute has the property *appendable* then the values are appended following an order from left to right.
- *del DN*
  This instruction is used to delete an entry. The entry to be deleted is identified by *DN*.
- *mod_add DN {attrName=attrVal}*
  This instruction is used to modify with *ADD* option the attribute *attrName* of the entry identified by *DN*, and the attribute value to be added is *attrVal*.
- *mod_del DN {attrName=attrVal}*
  This instruction is used to modify with *DELETE* option the attribute *attrName* of the entry identified by *DN*, and the attribute value to be deleted is *attrVal*.
- *mod_rep DN {attrName=attrVal}*
  This instruction is used to modify with *REPLACE* option the attribute *attrName* of the entry identified by *DN*, and the attribute used for the replacement is *attrVal*.

An instructions configuration file may look something like this:

```
add cn=emilio1,o=idstore {mx_entrytype=MX_PERSON,surname=Garcia Tormo}
add cn=emilio2,o=idstore {mx_entrytype=MX_PERSON,address=St. Moholt7050}
mod_rep cn=emilio2,o=idstore {address=St. NewMoholt7051}
mod_add cn=emilio2,o=idstore {firstname=Emilio Jose}
mod_del cn=emilio1,o=idstore {surname=Carceles Patos}
mod_del cn=emilio1,o=idstore {surname=Garcia Tormo}
del cn=emilio1,o=idstore
del cn=emilio,o=idstore
#
#-------------------- LEGAL INSTRUCTION DEFINITION----------------------
#
# add DN {attrName1=attrVal1,attrName2=attrVal2,...,attrNameN=attrValN}
# del DN
# mod_add DN {attrName=attrVal}
# mod_del DN {attrName=attrVal}
# mod_rep DN {attrName=attrVal}
#
#-----------------------------------------------------------------------
```

Instructions and expected results configuration file

The only difference compared to the instructions configuration file is the inclusion of the instruction *expected*. This instruction follows the next formats:

```
expected DN (filter) [searchAttributes]
{[attrName1=vals1][attrName2=vals2]…[attrNameN=valsN]}
```

or

```
expected DN (filter) [searchAttributes] {NOT_FOUND}
```

where:

- DN: the distinguished name for the base search.
- filter: the filter for the base search.
- searchAttributes: attributes that the base search is allowed to return.
- {[attrName1=vals1][attrName2=vals2]…[attrNameN=valsN]}: list of attributes which are the result of the search. Each attribute is identified by its name and by its set of values available when the search is launched. For multi-value attributes the different values are separated by "|".
- {NOT_FOUND}: indicates that the entry identified by the *DN* is not found.

An instructions and expected results configuration file may look something like this:

```
add cn=person1,o=idstore {[mx_entrytype=MX_PERSON]}
expected cn=person1,o=idstore (objectclass=*) [] {[mx_entrytype=MX_PERSON][mskey=*][mskeyvalue=person1]}
#
#
#
add cn=person2,o=idstore {[mx_entrytype=MX_PERSON]}
expected cn=person2,o=idstore (objectclass=*) [] {[mx_entrytype=MX_PERSON][mskey=*][mskeyvalue=person2]}
#
#
#
add cn=person3,o=idstore {[mx_entrytype=MX_PERSON]}
expected cn=person3,o=idstore (objectclass=*) [] {[mx_entrytype=MX_PERSON][mskey=*][mskeyvalue=person3]}
#
#
#
add cn=group1,o=idstore {[mx_entrytype=MX_GROUP]}
expected cn=group1,o=idstore (objectclass=*) [] {[mx_entrytype=MX_GROUP][mskey=*][mskeyvalue=group1]}
#
#
#
add cn=group2,o=idstore {[mx_entrytype=MX_GROUP]}
expected cn=group2,o=idstore (objectclass=*) [] {[mx_entrytype=MX_GROUP][mskey=*][mskeyvalue=group2]}
#
#
#
mod_add cn=group1,o=idstore {[mxmember_mx_person=person1]}
mod_add cn=group1,o=idstore {[mxmember_mx_person=person2]}
mod_add cn=group2,o=idstore {[mxmember_mx_group=group1]}
mod_add cn=group2,o=idstore {[mxmember_mx_person=cn=person3,o=idstore]}
#
#
#
expected cn=person1,o=idstore (objectclass=*) [] {[mx_entrytype=MX_PERSON][mskey=*][mskeyvalue=person1][mxref_mx_group=*]}
expected cn=person2,o=idstore (objectclass=*) [] {[mx_entrytype=MX_PERSON][mskey=*][mskeyvalue=person2][mxref_mx_group=*]}
expected cn=person3,o=idstore (objectclass=*) [] {[mx_entrytype=MX_PERSON][mskey=*][mskeyvalue=person3][mxref_mx_group=*]}
expected cn=group1,o=idstore (objectclass=*) [] {[mx_entrytype=MX_GROUP][mskey=*][mskeyvalue=group1][uniquemember=*][mxref_mx_group=*]}
expected cn=group2,o=idstore (objectclass=*) [] {[mx_entrytype=MX_GROUP][mskey=*][mskeyvalue=group2][uniquemember=*]}
```

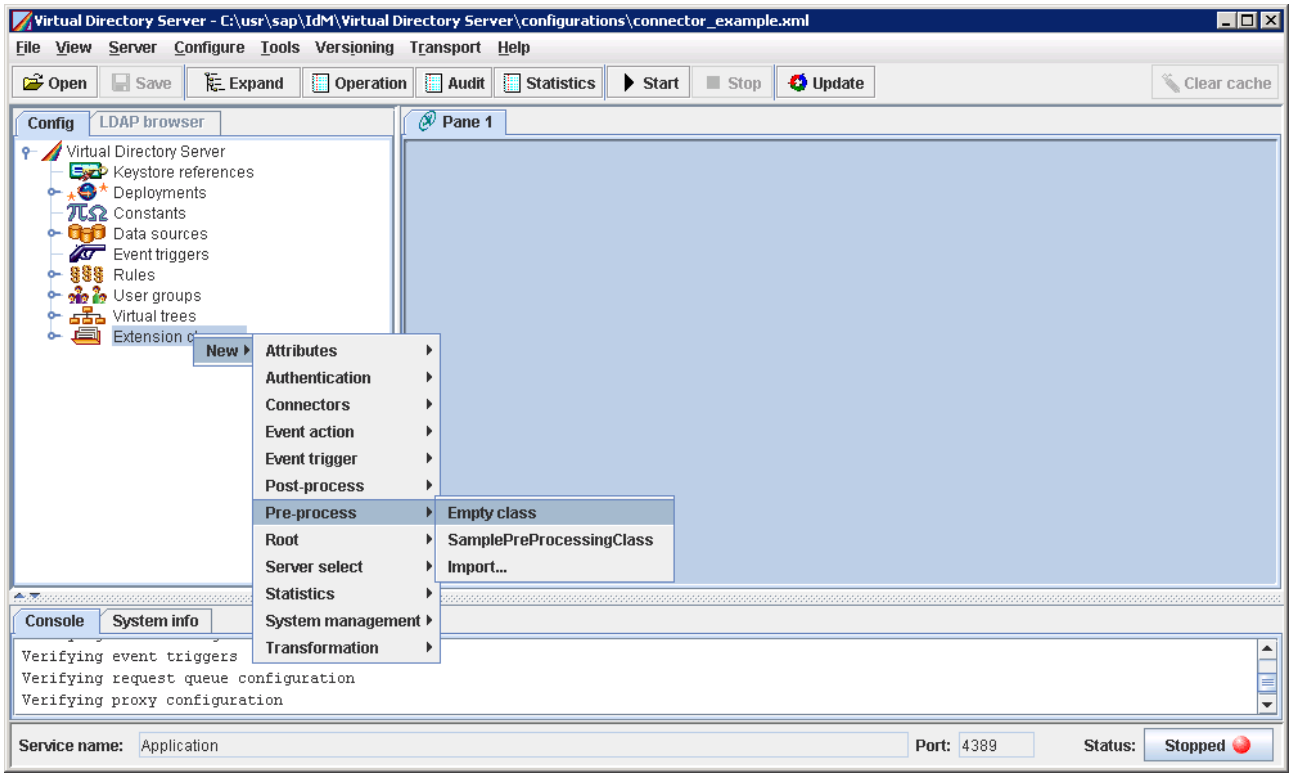
Configuring the Virtual Directory Server

To achieve the correct testing tool behavior, the Virtual Directory Server needs to be configured – a class for pre-processing of data received from the testing tool needs to be created and defined for the server (the Virtual Directory Server).

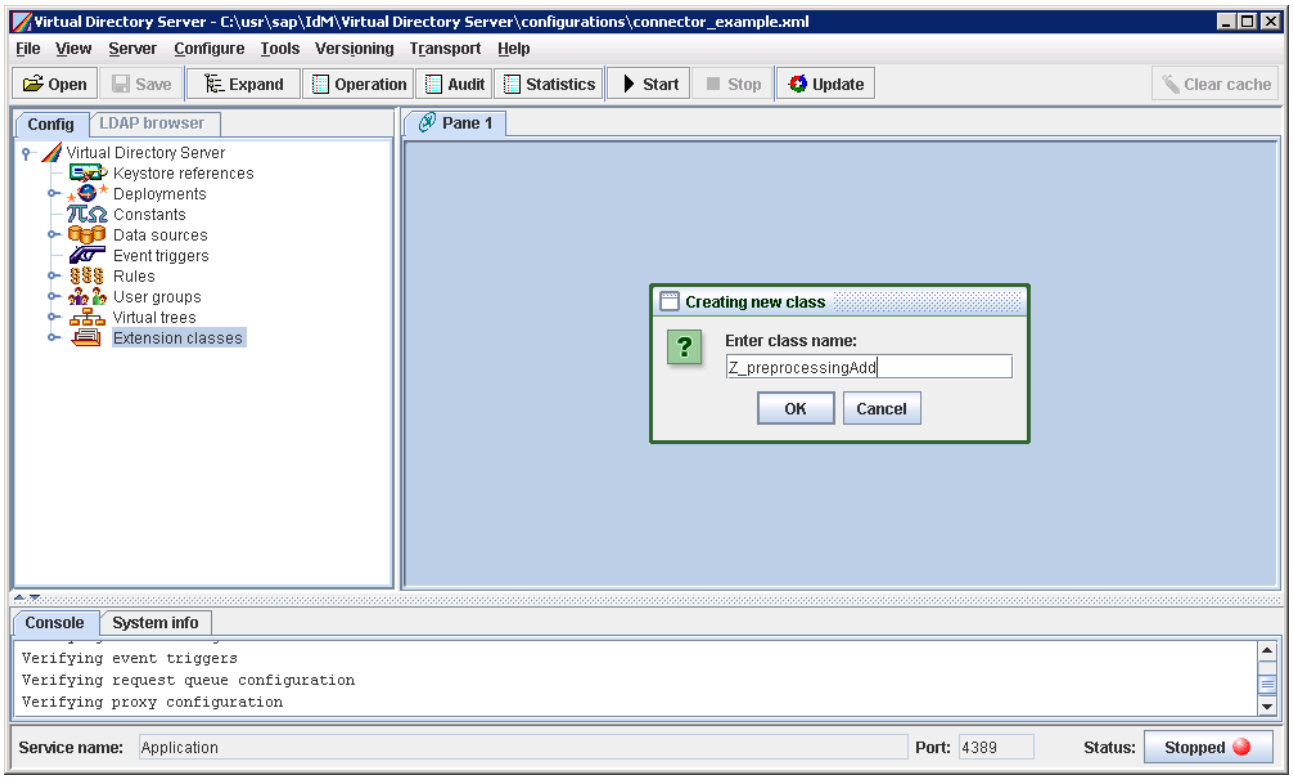
Creating and compiling the pre-processing class

To create the pre-processing class, do the following:

1. Select "Extension classes" in the virtual tree in the Virtual Directory Server console and choose **New/Pre-process/Empty class** from the context menu:



The "Creating new class" dialog box is displayed:

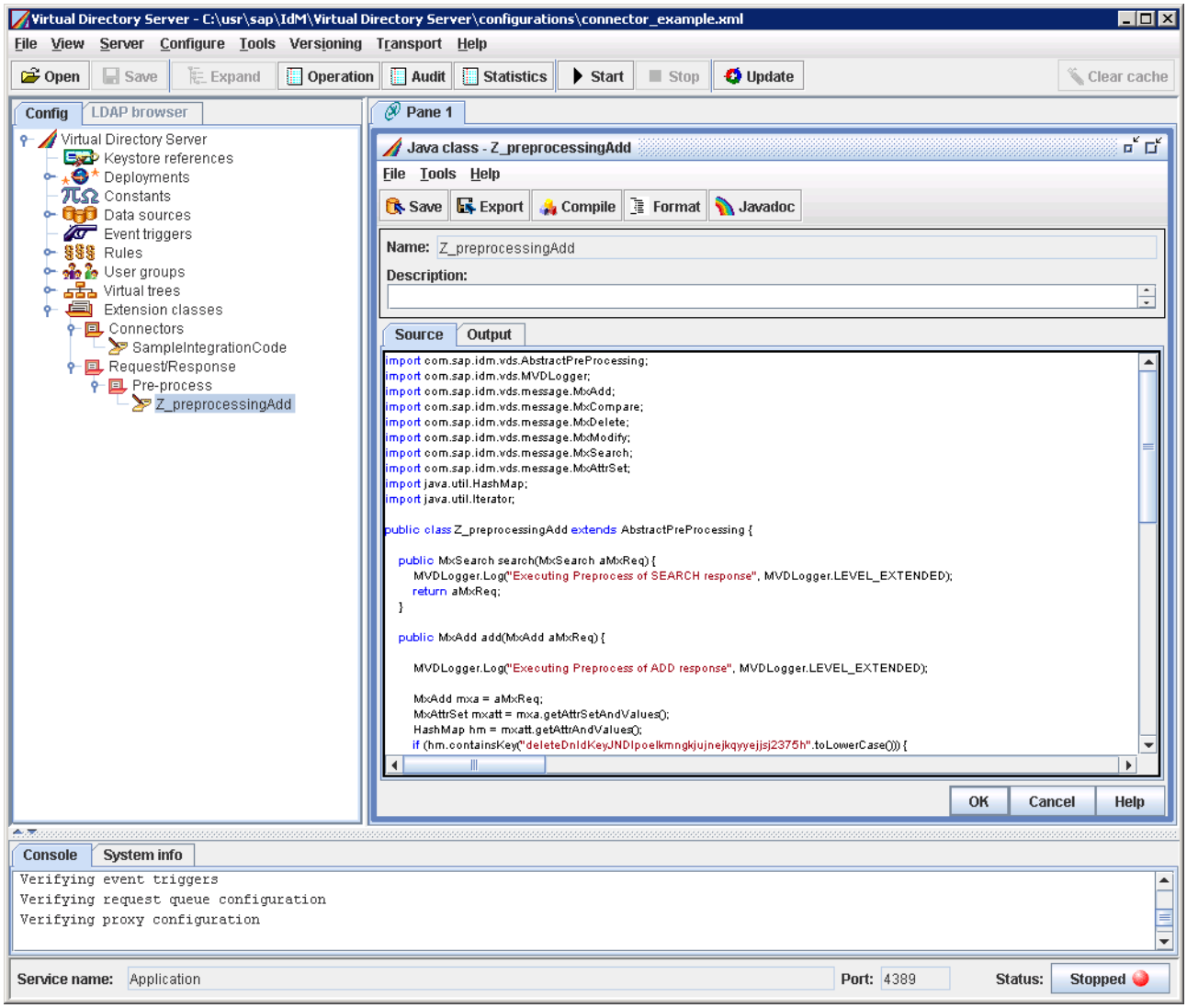


Define name for the class as shown above. The chosen name should be "Z_preprocessingAdd".

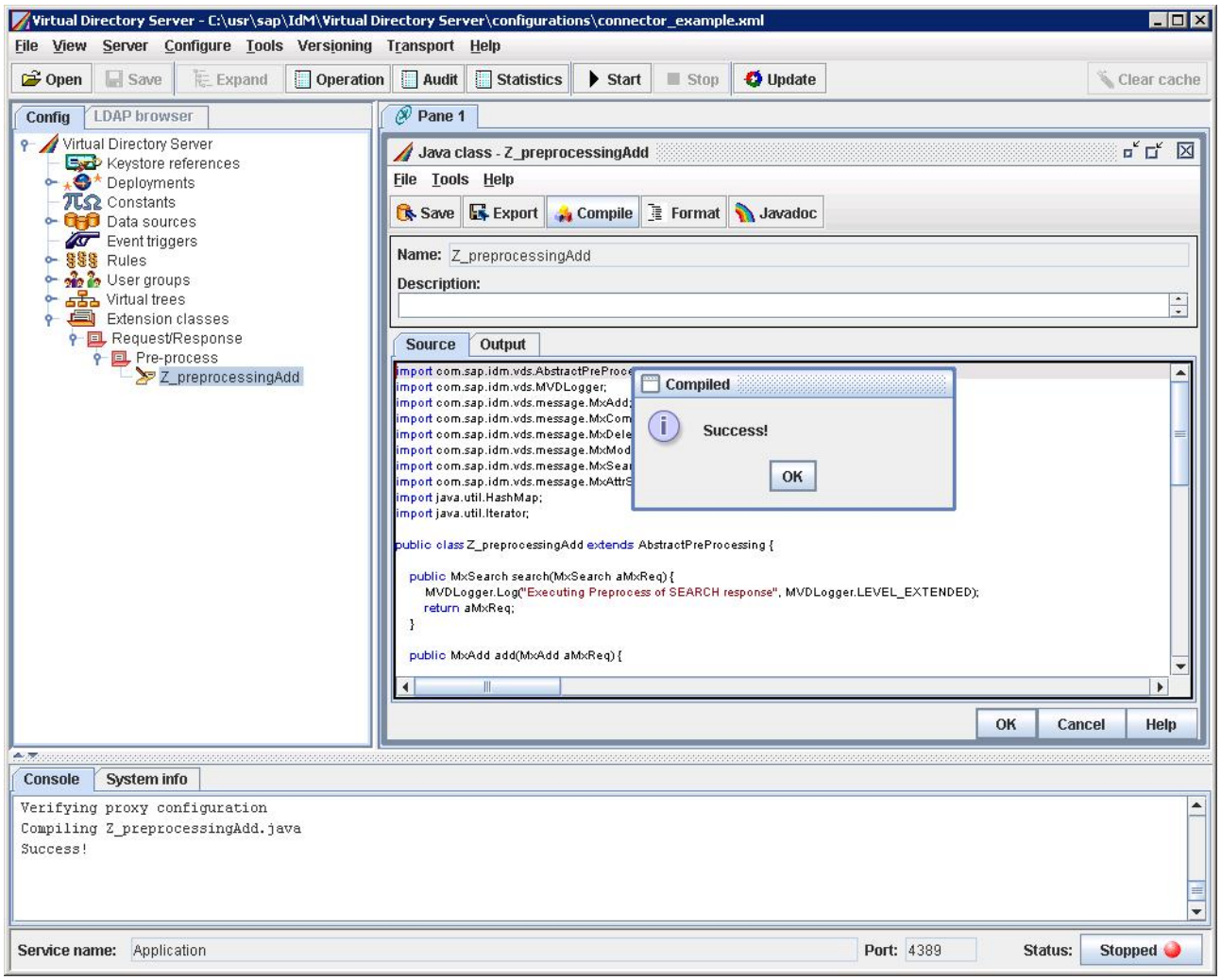
2. Choose "OK".

   The current content of the pre-processing class needs to be substituted by the content of the file *Z_preprocessingAdd.java* located in the testing tool package (*<Connector Development Kit install directory>\VDS Connector Testing Tool*).

3. Copy the content of the file *Z_preprocessingAdd.java* and paste it in the created class in order to substitute the current content of the pre-processing class:



4. Choose the "Compile" button to compile the pre-processing class.

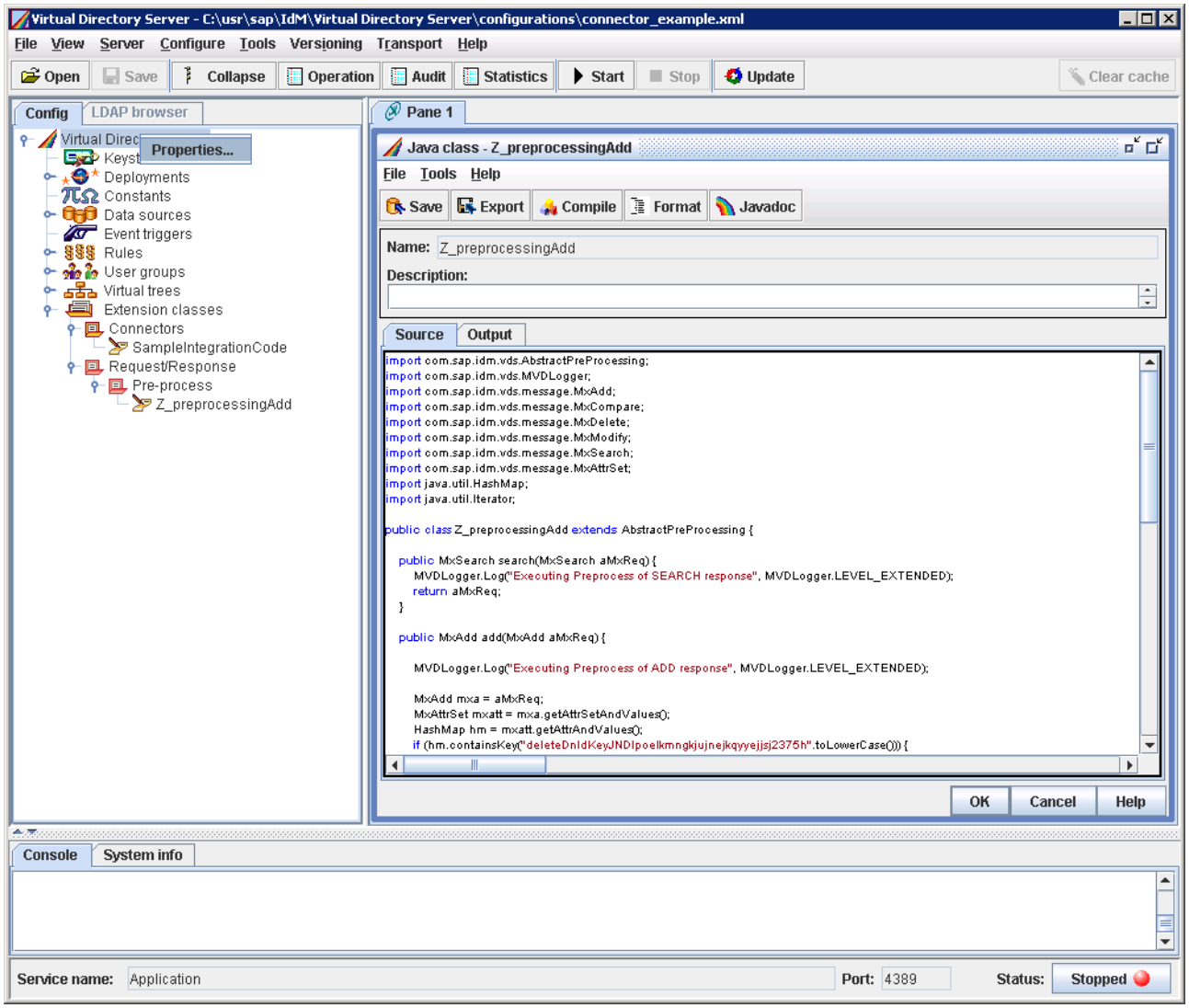


If everything was done properly, a dialog box indicating the compilation success is displayed.

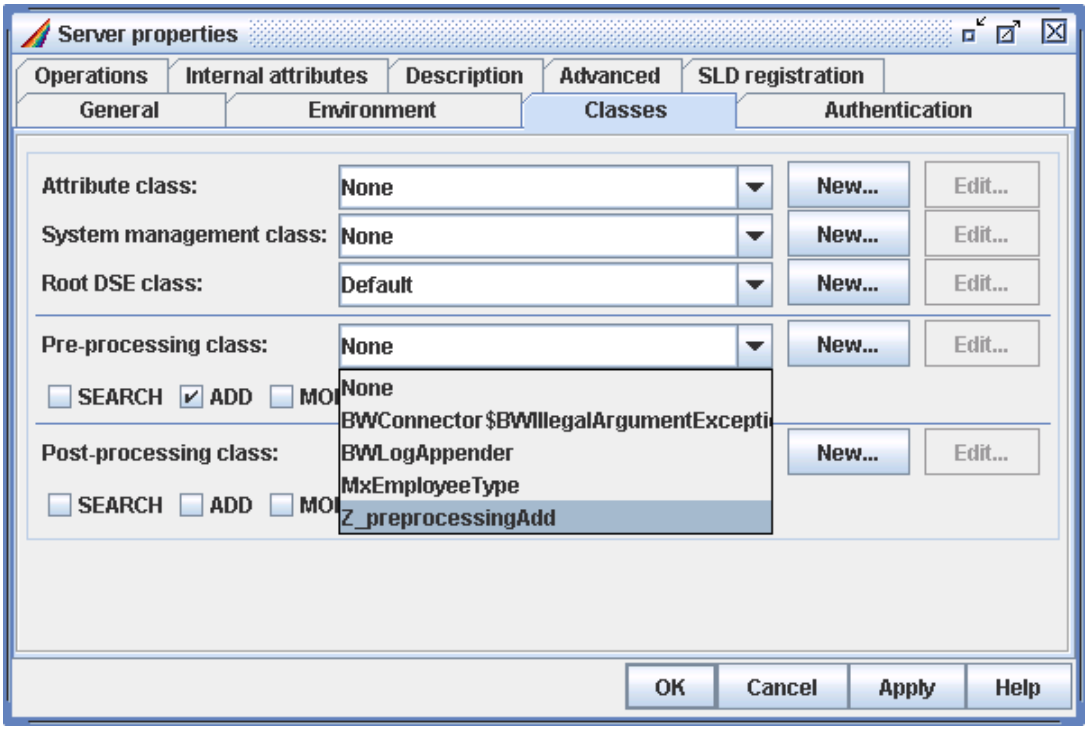
Defining the pre-processing class for the Virtual Directory Server

Now that the pre-processing class is created and successfully compiled, it needs to be referenced from the Virtual Directory Server:

1. Select the "Virtual Directory Server" node in the console tree.



2. Choose "Properties" from the context menu. The "Server properties" dialog box is displayed:



In the "Classes" tab do the following:

Select "ADD" in the "Pre-processing class" section.

Select "Z_preprocessingAdd" as the pre-processing class.

3. Choose "OK" to confirm and to close the dialog box.

The Virtual Directory Server and the pre-processing class are now configured, and the testing tool is ready for use.

Using the Virtual Directory Server Connector Testing Tool

When the configuration is completed, the Testing Tool can be run by running the file *VDS CTT.jar*. By running the testing tool this way, it will be possible to see the log information only by using the file *log.trc*.

If it is desirable to see the log information in real time, without using the SAP Log Viewer, then the testing tool should be started from the command prompt by executing the following command line: *java -jar "VDS CTT.jar"*. Using this option, the log file has the same contents as when using the first option but it is possible to see the information in real time in the console. It is also possible to transfer it to another file.

Constraints

In this section an important set of constraints is presented – both the general constraints and the property constraints.

General constraints

The general constraints are:

- Neither an entry specification nor an attribute specification can inherit from themselves.
- Both an entry specification and an attribute specification can only inherit from a previously declared specification. This means that if an entry specification "A" wants to inherit the entry specification "B", then "B" must be declared previously in the configuration file.
- Two entry specifications cannot have the same *RDN* declaration.
- When *RDN* is declared in an entry specification it must follow the pattern given by all the other *RDN* declarations with common parts. E.g. if there are two entry specifications with *RDN* declarations, where the *RDN* declaration of one of them is *id2=Y*,id1=X*, and the *RDN* declaration of the another one is an *RDN* which follows from the previous one, then it must have exactly *id2=Y*,id1=X* as the right-most part of its *RDN* declaration. The *RDN* could be *id3=Z,id2=Y*,id1=X*.
- If several *RDN* declarations are used to construct a longer *RDN* declaration, then each RDN declaration component needs to be constructed by taking the longest *RDN* declaration (with common parts) as initial reference. In the previous point, the *RDN* declaration *id2=Y*,id1=X* is part of *id3=Z,id2=Y*,id1=X*. That is the reason why *id2=Y*,id1=X* uses the symbol "*" and not "+" (which is the one that probably makes more sense to use), since at least for this specific declaration *id2* it is necessary to construct a legal entry with this *RDN*. If the left-most part of the declaration has the symbol "*" it is internally changed to "+", and if it is "?" then it is internally removed.
- The right-most part of an *RDN* declaration can neither have the symbol "?" nor "*".
- In the entry specifications all 28 available test cases offered by the testing tool are assignable. Any test case declared or inherited is assigned to all the attributes declared in the entry. However, each attribute can further define or reassign its properties using the attribute specifications of the attributes configuration file where no limitations exist for the properties to be defined but there are some limitations for the test cases (see section Test cases on page 12).
- In a specification a parameter can not be repeated.
- The order of the parameters for any specification is not relevant but the order of the parameters of the general configuration file is.
- Any attribute defined in an entry specification must have a definition in an attribute specification and the opposite.
- Two entry specifications can not have the same identifier. The same for attribute specifications.
- Every attribute must have one of the next properties: *must*, *may*, *automaticMust*, or *automaticMay*.

Property constraints

In the next lines the constraints between the properties are listed. The constraints are considered to be reciprocal, i.e. if A is B, then B must be A.

Property name	**Description**
Virtual	An attribute with property *virtual* must have the *linked* property.
Linked	An attribute with linked property can not have the properties *modifiable, multivalue, automaticMust* and *automaticMay*.
Must	An attribute with *must* property can not have the properties *may, automaticMust* and *automaticMay*.
May	An attribute with *may* property can not have the properties *automaticMust* and *automaticMay*.
automaticMust	An attribute with property *automaticMust* can not have the properties *automaticMay* and *modifiable*.
automaticMay	An attribute with property *automaticMay* can not have the *modifiable* property.
allowsNoValue	No restriction (but remember the reciprocity).
Appendable	An attribute with *appendable* property can not have the *multivalue* property.
Multivalue	No restriction (but remember the reciprocity).
allowsDuplicate	An attribute with *alowsDuplicate* property must have the *multivalue* property.
Modifiable	No restriction (but remember the reciprocity).

Regular expressions

For the definition of legal *DNs* and the legal attribute values, regular expressions are used. In this section the legal syntax for regular expressions and how they must be used for the *DNs*, and the attribute values definition is described. But first of all let's see what a regular expression is.

What is a regular expression?

A regular expression (RE) is an expression which describes a set of strings without numerating its elements. There is a set of basic operators to construct the REs, which are:

- Alternation ("|")
  This character is used to separate alternatives. For example, "red|green" means either "red" or "green".
- Quantification
  A quantifier expresses the frequency with which a regular expression can occur. The most common quantifiers are:
  - "*": Indicates that the preceded RE can occur zero, one or more times. For example, "12*3" may mean "13", "123", "1223", "12223", etc.
  - "+": Indicates that the preceded RE can occur one or more times. So, "he+llo" may mean "hello", "heello", "heeello", etc.
  - "?": This indicates one or zero repetitions. So, for example "(un)?necessary" means either "unnecessary" or "necessary".
- Grouping ("("")")
  These are the characters used to group. An example of its use: "(fa|mo)ther" means either "father" or "mother".

Accepted regular expressions

The following are the legal regular expressions that can be used in the configuration files:

Regular expression	Description
X?	X zero or one time.
X+	X one or more times.
X*	X zero or more times.
X {n}	X n times, where n is a positive integer (including zero).
X {n,}	X at least n times.
X {n, m}	X at least n times but no more than m times, where n and m are positive integers (including zero).

Regular expression	Description
[]	Represents a set of characters. Only one character out of those in the set is selected each time it is required. Some special conditions must be taken into account when defining these sets. Let's see some examples: • [abc] "a" or "b" or "c" • [^abc] Neither "a" nor "b" nor "c" The character "^" is only admitted as the first character ("\"). The characters "^", "{", "}" and "$" must always be preceded by the escape character. Except when "^" is the first character. To include the character "]" as part of the set, it must be preceded by the escape character. • [a-z] Any letter from "a" to "z". • [a-zA-Z0-9] Any letter of the English vocabulary or any digit from "0" to "9". The interval must always be ascendant, i.e. it is not legal with the interval z-a. Two consecutive ampersands ("&") are illegal. The character "-" must be either the first or the last one if it is not used to express an interval.
X|Y	X or Y.
()	Parenthesis is used to envelop a regular expression. For (X|Y)+ the following match the regular expression: X, XYYYX, YY.
\.	Any character.
\d	Any digit.
\D	Any character, but not digits.
\a	Any alpha character.
\A	Any character but not an alpha character.
\w	Any alpha character or digit.
\W	Any character but neither alpha character nor digit.
\s	Any space character.
\S	Any character but not a space character.

To include any special characters as part of a character set, it must be preceded by the escape character. The set of characters that must follow this rule are: "[", "]", "(", ")", "*", "+", "?", "^", "|", "{", "}", "$" and "\". Therefore there is a set of predefined regular expressions, which are:

Predefined regular expression	**Description**
INTEGER	A 32-bits integer.
DOUBLE	A 64-bits double.
BIG_INGEGER	An integer of any length.
BIG_DOUBLE	A double of any length.
CHAR	A character from the character set given by "\.".
ALPHA	A set of alpha characters (including the empty set).
NUMERIC	A set of digits (including the empty set).
TELEPHONE	"(\\(\\+[0-9]\\)|\\(00[0-9]+\\))(-[0-9]+)+"
STRING	"[-a-zA-Z0-9.,øæå\\\\+?*|!'#£\\ $¤%&/()=\\{\\}~_:;\\^]*"
ALPHANUMERIC	A set of alpha characters and digits (including the empty set).

If a value of a regular expression coincides with a pre-defined regular expression, then it is possible for example the definition of the string "INTEGER" to be defined like this: "[I]NTEGER".