EVIDEN

Identity and Access Management

DirX Identity

Connectivity Reference

Version 8.10.12, Edition August 2025

Table of Contents

Preface

This manual is reference for the DirX Identity Connectivity. It describes the different ways DirX Identity accesses target systems. It consists of the following chapters:

- Chapter 1 provides an overview of DirX Identity Connectivity and the connectors, agents, event listeners and triggers that comprise Connectivity.
- Chapter 2 introduces the Identity connectors and provides a detailed description of each connector.
- Chapter 3 introduces the Identity agents and provides a detailed description of each agent.
- Chapter 4 describes event listeners and triggers.

DirX Identity Documentation Set

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 Connectivity Overview

This manual describes the different ways in which DirX Identity accesses target systems. DirX Identity provides:

- **DirX Identity connectors** - software packages that allow event-based workflows to interface directly with the specialized native API of target systems (connected directories).
- **DirX Identity agents** - executables that allow accessing target systems (connected directories) via the specialized native API. Batch workflows use agents to synchronize data between the Identity Store and target systems.
- **Event listeners and triggers** - software packages that generate events and send them to the event manager.

The following tables list the Identity Connectivity components available in each different Connectivity package. Each table lists the component name, its type, the language in which it is written and whether or not it supports realtime (event-driven) processing:

Default Connectivity Package

Connected System	Type	Implementation	Realtime Component?
SPMLv1 Connector	Connector	Java	Yes
SPMLv1Tov2 Connector	Connector	Java	Yes
CSV Connector	Connector	Java	No
LDIF Connector	Connector	Java	No
Identity Domain Connector	Connector	Java	Yes
UNIX PAM	Connector	Java	No
Meta Controller	Agent	C	No

Connectivity Package for Microsoft AD

Connected System	Type	Implementation	Realtime Component?
Microsoft ADS Agent	Agent	C++	No
Microsoft Exchange Agent	Agent	C++	No
ADS (Exchange, Lync) Connector	Connector	Java	Yes
SharePoint Connector	Connector	Java	Yes
Windows Password Listener	Stand-alone	C++	Yes

Connectivity Package for Database Systems

Connected System	Type	Implementation	Realtime Component?
ODBC Agent	Agent	C	No
JDBC Agent	Agent	Java	No
JDBC Connector	Connector	Java	Yes

Connectivity Package for Siemens HiPath

Connected System	Type	Implementation	Realtime Component?
HiPath 4000 Manager/Hicom DMS	Agent	C++ (Tcl)	No

Connectivity Package for Healthcare Systems

Connected System	Type	Implementation	Realtime Component?
medico//s	Connector	Java	Yes

Connectivity Package for Physical Security Systems

Connected System	Type	Implementation	Realtime Component?
SiPass Agent	Agent	C#	No

Connectivity Package for SAP Systems

Connected System	Type	Implementation	Realtime Component?
SAP ECC UM Agent	Agent	Java	No
SAP ECC UM Connector	Connector	Java	Yes
SAP ERP HR Agent	Agent	ABAP	No
SAP NetWeaver UM Agent	Agent	Java	No

Connectivity Package for IBM Systems

Connected System	Type	Implementation	Realtime Component?
IBM Notes Agent	Agent	C++	No
IBM Notes Connector	Connector	C++	Yes
RACF Connector	Connector	Java	Yes

Connectivity Package for Enterprise Single Sign-on Systems

Connected System	Type	Implementation	Realtime Component?
Evidian ESSO Connector	Connector	Java	Yes
Imprivata OneSign Connector	Connector	Java	Yes

Connectivity Package for Cloud Systems

Connected System	Type	Implementation	Realtime Component?
Google Apps Connector	Connector	Java	Yes
Citrix ShareFile Connector	Connector	Java	Yes
Office 365 Connector	Connector	Java	Yes
Salesforce Connector	Connector	Java	Yes

Proxy Connectivity Package

Connected System	Type	Implementation	Realtime Component?
Remote AD Upload Connector	Connector	Java	Yes
Request Workflow Connector	Connector	Java	Yes
OpenICF Connector	Connector	Java	Yes
OpenICF Windows Local Accounts Connector	Connector	Java	Yes

2. Identity Connectors

DirX Identity connectors run in the Identity Integration Framework. Java-based connectors can only run in the Java-based Identity Server, C++- or C-based connectors can only run in the C++-based Identity Server.

See the *DirX Identity Connectivity Administration Guide* to learn more about event-based concepts and applications.

See the *DirX Identity Application Development Guide* for information about corresponding workflow applications.

See the *DirX Identity Integration Framework Guide* to learn about creating your own connectors.

2.1. ADS Connector

The Java-based ADS connector is built with the Identity Java Connector Integration Framework and is derived from the Java-based LDAP connector. It implements only functionality that is not already covered by the LDAP connector. Like all framework based agents, it gets SPML requests from the Identity side and converts them to the appropriate LDAP requests on the Active Directory side and vice versa.

The add-on functionality compared to the LDAP connector is described in the following sections.

2.1.1. Setting a User Password

While setting a user password is supported in the LDAP connector for compatibility reasons, it is a basic feature of the ADS connector because it implements a special Active Directory operation.

If the attribute **unicodePwd** is contained in the attribute list of an SPML Add or Modify request, the ADS connector updates the user's password in Active Directory. Microsoft Active Directory enforces SSL for password changes.

To set up an SSL connection to an Active Directory Server, see the section "LDAP SSL Setup".

2.1.2. Creating a Mailbox-Enabled User

If an SPML Add or Modify request contains the attribute **msExchHomeServerName**, the ADS connector creates a mailbox enabled user by extending the request with the mailbox security descriptor attribute **msExchMailboxSecurityDescriptor**. This descriptor contains default rights. It assigns the user as the owner of his mailbox and gives him the permissions to send and receive mails and the right to modify mailbox attributes.

If the request also contains the attribute **msExchRecipientTypeDetails**, it is assumed that an Exchange mailbox of Exchange Server 2007 version or higher is supposed to be created

or modified. The ADS connector then generates a random globally unique mailbox identifier and extends the request with the **msExchMailboxGuid** attribute set to this generated identifier if the user's **msExchMailboxGuid** attribute is not already set. The **msExchMailboxGuid** attribute of the Active Directory user is the link to the mailbox object in the Exchange Server Mailbox database and should not be overwritten.

All other mail or mailbox enabling attributes are expected to be contained in the SPML request and are passed by the ADS connector to the super class of the LDAP connector. It adds or modifies the object in Active Directory by calling the appropriate LDAP interface functions.

2.1.3. Getting Delta and Deleted Objects

Getting objects that have changed since a previous search is performed with the Active Directory synchronization control DirSync, which is an LDAP server extension control.

When performing a DirSync search, a provider-specific data element (cookie) is passed that identifies the directory state at the time of the previous DirSync search. For the first search, a null cookie is passed and a valid cookie is returned. It is stored on the Identity side and used for the next search request.

The cookie is passed to the ADS connector in the operational attribute **dxm.delta** of an SPML search request. For a full search, the **dxm.delta** attribute must be of dsml value type string (default) no matter what value it contains:

```
<spml:operationalAttributes>
  <dsml:attr name="dxm.delta">
    <dsml:value>FULL</dsml:value>
  </dsml:attr>
</spml:operationalAttributes>
```

For a search of objects changed after the previous search, the dsml value type is base64Binary and the value contains the base64-encoded binary cookie of the previous search:

```
<spml:operationalAttributes>
  <dsml:attr name="dxm.delta">
    <dsml:value
type="xsd:base64Binary">D0Zdo/XFAAAAAIPrhq1S&#xd;MqFCmOL1icEhBSU=</ds
ml:value>
  </dsml:attr>
</spml:operationalAttributes>
```

Delta searches can also be performed together with paged searches (pageSize is set greater than 0 in the operational attributes of the search request).

Deleted objects are automatically included in a delta search. You can also pass any filter in a delta search request. For example, if you want to retrieve only the deleted user objects,you can specify the filter:

```
<spml:filter>
  <dsml:and>
    <dsml:equalityMatch name="objectClass">
      <dsml:value>user</dsml:value>
    </dsml:equalityMatch>
    <dsml:equalityMatch name="isDeleted">
      <dsml:value>TRUE</dsml:value>
    </dsml:equalityMatch>
  </dsml:and>
</spml:filter>
```

2.1.3.1. Handling Range Attributes

You can retrieve Active Directory attributes with more than 1000 values only by performing multiple searches with specified ranges. Use values from 0 to 999 for the first search, values from 1000 to 1999 for the second search and so on.

The ADS connector implements this method transparently for the user. If the attribute **member** is contained in an SPML search request, the ADS connector automatically performs multiple range searches for this attribute. It extends the search result retrieved from the LDAP connector by filling in all values of the member attribute.

Ranging also works with paging (pageSize is configured in the operational attributes of the search request).

In the import direction, the Active Directory LDAP server accepts more than 1000 values and stores them itself in separate range attributes.

2.2. Citrix Share File Connector

The Citrix ShareFile connector implements the Identity Java Connector Integration Framework's DxmConnector interface and connects to a Citrix ShareFile through the HTTP interface. It can be used for real-time workflows in the Java-based (IdS-J) Server. Like all framework-based agents, it gets SPML requests from the Identity side and converts them to the appropriate Citrix ShareFile calls and vice versa. The Citrix ShareFile connectivity is based on HTTP protocol. The connector supports membership stored on groups.

2.2.1. Overview

The connector implements the API methods "add(...)", "modify(...)", "delete(...)" and "search(...)". They represent the corresponding SPML requests "AddRequest", "ModifyRequest", "DeleteRequest" and "SearchRequest".

add(...) - internally uses addUser(...) and addGroup(...)

modify(...) - internally uses modifyUser(...) and modifyGroup(...)

search(...) - internally uses searchUser(...), searchAllUsers(...), searchGroup(...), searchAllGroups(...)

The connector uses the open() method to open a connection to the Citrix ShareFile service. Because communication with the Citrix ShareFile service is through the HTTP protocol (stateless calls), no activities are performed on the close() method.

2.2.2. Limitations

By default, the **isemployee** attribute is set to **true** in the workflow mapping, so the default workflow only creates employees. To change it, create a custom Java mapping for the **isemployee** attribute.

Depending on the type of Citrix subscription, limitations related to the maximum number of employees that can be managed by DirX Identity are possible.

2.2.2.1. DirX Identity Manager Limitations

It is not possible to change a group's name using the GUI in the DirX Identity Manager.

2.2.2.2. Known Issues

Due to a constraint in the DirX Identity workflow engine, the user's e-mail address is used instead of the Citrix ID to identify the user inside an SPML modify or add request when handling membership for a user without an existing account in Citrix. For example:

```
<modifications>
 <modification name="member" operation="add">
   <value type="string">john.doe@dirx.de</value>
 </modification>
</modifications>
```

If there is already a Citrix account for the user, the Citrix ID is used to identify the user inside an SPML modify or add request. For example:

```
<modifications>
 <modification name="member" operation="add">
   <value type="string">7b1fc236-8d5b-44dc-92bc-44b5757c3c4e</value>
 </modification>
</modifications>
```

The Citrix ShareFile API does not allow creating a user with the same e-mail address as an

existing user. Such an attempt fails. The Citrix ShareFile connector returns the ID of the existing user with the specified e-mail address in the error SPML response.

On the other hand, the Citrix ShareFile API allows creating a group with the same name as an existing group. This action is handled by the Citrix ShareFile connector itself. It does not forward this type of request to the Citrix API, but returns an error.

2.2.3. Request and Response Handling

This section describes the supported requests and attributes for the Citrix ShareFile connector. All attributes allowed by the Citrix ShareFile API can be added.

The connector supports the following attributes of users and groups:

Users:

- email
- firstname
- lastname
- company
- isemployee (true or false)
- state (ENABLED or DISABLED)

Groups:

- name
- isShared (true or false)

Membership:

- member - the membership attribute

The connector supports following operational attributes:

- objtype - mandatory, the value can be **user** or **group**. It specifies for all request types (add, modify, delete, search) whether it is a user or group request.

If a user who belongs to two groups should be deleted from one of them, a modify request is performed for the membership change. If a user belongs to only one group and should be deleted from it, a modify request for deleting the user from the group and a modify request for suspending the user is performed.

2.2.3.1. Add Request

In the add request for user or group, the identifier must not be specified. Citrix generates the identifier automatically and returns it in the response.

2.2.3.1.1. Groups

For groups, the **name** attribute is mandatory and represents the name of the group to be created. The following example request creates the group **TestGroup**.

```
<?xml version="1.0" encoding="UTF-8"?>
<addRequest requestID="req-id">
<operationalAttributes>
  <attr name="objType">
    <value type="string">group</value>
  </attr>
</operationalAttributes>
<attributes>
  <attr name="name">
    <value type="string">TestGroup</value>
  </attr>
</attributes>
</addRequest>
```

2.2.3.1.2. Users

The attributes **firstname**, **lastname**, **email** and **isemployee** are mandatory. The **state** attribute is optional. If not present, the Citrix connector creates the user in the ENABLED state.

The following example request creates a user object for **John Doe**. His e-mail address is also specified for the **id** attribute:

```
<?xml version="1.0" encoding="UTF-8"?>
<addRequest requestID="req-id">
<operationalAttributes>
  <attr name="objType">
    <value type="string">user</value>
  </attr>
</operationalAttributes>
<attributes>
   <attr name="lastname">
    <value type="string">Doe</value>
  </attr>
  <attr name="firstname">
    <value type="string">John</value>
  </attr>
  <attr name="email">
```

```
    <value type="string">john.doe@dirx.com</value>
  </attr>
  <attr name="isemployee">
    <value type="string">true</value>
  </attr>
  <attr name="state">
    <value type="string">ENABLED</value>
  </attr>
</attributes>
</addRequest>
```

2.2.3.2. Modify Request

A modify request allows for changing attributes of a user or a group. At the group level, it allows for changing the membership.

The attributes **firstname**, **lastname**, **email**, **company** and **state** can be changed for users. The attributes **name** and **isshared** can be changed for groups.

When changing an attribute, a mandatory parameter in the SPML request is the Citrix ID of the user / group returned by the SPML add response.

In the following example request, the e-mail address of the user John Doe is changed:

```
<modifyRequest requestID="req-id">
<operationalAttributes>
  <attr name="objType">
    <value type="string">user</value>
  </attr>
</operationalAttributes>
<identifier type="urn:oasis:names:tc:SPML:1:0#GUID">
  <id>g9163393-df09-42d9-b19d-d62e9468a83e</id>
</identifier>
<modifications>
  <modification name="id" operation="replace">
    <value type="string">52cc6788-efd0-4647-a269-6784a92a425e</value>
  </modification>
  <modification name="email" operation="replace">
    <value type="string">john.doe@dirx.de</value>
  </modification>
  <modification name="isemployee" operation="replace">
    <value type="string">true</value>
  </modification>
```

```
</modifications>
</modifyRequest>
```

The following example request changes the name of the group **TestGroup** to **TestGroup.changed**:

```
<modifyRequest requestID="modify-01">
<operationalAttributes>
  <attr name="objType">
    <value type="string">group</ value>
  </attr>
</operationalAttributes>
<identifier type="urn:oasis:names:tc:SPML:1:0#GUID">
  <id>g9163393-df09-42d9-b19d-d62e9468a83e</id>
</identifier>
<modifications>
  <modification name="name" operation="replace">
    <value>TestGroup.changed</value>
  </modification>
</modifications>
</modifyRequest>
```

The following example request performs a membership change. It adds a user who does not have a Citrix account to a group. The user's e-mail address is specified in the modification tag under **modifications** → **add**. These fields are both mandatory:

```
<modifyRequest requestID="req-id">
<operationalAttributes>
  <attr name="objType">
    <value type="string">group</value>
  </attr>
</operationalAttributes>
<identifier type="urn:oasis:names:tc:SPML:1:0#GUID">
  <id>g9163393-df09-42d9-b19d-d62e9468a83e</id>
</identifier>
<modifications>
  <modification name="member" operation="add">
    <value type="string">test.user@dirx.com</value>
  </modification>
</modifications>
```

```
</modifyRequest>
```

The following example request performs a membership change. It adds a user who has a Citrix account to a group. The Citrix group ID is specified in the request identifier and the user ID is specified in the modification tag under **modifications** → **add**. These fields are both mandatory:

```
<modifyRequest requestID="req-id">
<operationalAttributes>
  <attr name="objType">
    <value type="string">group</value>
  </attr>
</operationalAttributes>
<identifier type="urn:oasis:names:tc:SPML:1:0#GUID">
  <id>g9163393-df09-42d9-b19d-d62e9468a83e</id>
</identifier>
<modifications>
  <modification name="member" operation="add">
    <value type="string">2fc7f4e9-2d5a-41c4-bde0-818f9373944b</value>
  </modification>
</modifications>
</modifyRequest>
```

2.2.3.3. Delete Request

In a delete request, the identifier is mandatory and represents the ID received from Citrix when creating the user or the group.

The following example request deletes a user:

```
<deleteRequest requestID="delete-01">
<operationalAttributes>
  <attr name="objType">
    <value type="string">user</value>
  </attr>
</operationalAttributes>
<identifier type="urn:oasis:names:tc:SPML:1:0#DN">
  <id>f58688fa-1e36-4514-b506-e0fd0dc07541</id>
</identifier>
</deleteRequest>
```

The following example request deletes a group:

```
<deleteRequest requestID="delete-01">
<operationalAttributes>
  <attr name="objType">
    <value type="string">group</value>
  </attr>
</operationalAttributes>
<identifier type="urn:oasis:names:tc:SPML:1:0#DN">
  <id>g1af88a7-4311-418c-a969-e0759e535f2e</id>
</identifier>
</deleteRequest>
```

2.2.3.4. Search Request

In the search request, the Citrix ShareFile connector supports the standard element **searchBase** and the operational attribute **scope**.

The following table shows valid s**earchBase** / **scope** combinations in a search request:

searchBase / scope	**BASE**	**ONELEVEL**	**SUBTREE**
object ID	OK	INVALID	INVALID
"all"	INVALID	INVALID	OK

In a search request, **searchBase** is a mandatory parameter that specifies the Citrix ID for the user or the group or the **"all"** string to list all users / groups. The attributes contained under the attributes tag are returned with their values in the response if available.

Example request for group:

```
<searchRequest requestID="req-id">
<operationalAttributes>
  <attr name="objType">
    <value type="string">group</value>
  </attr>
  <attr name="scope">
    <value type="string">Base</value>
  </attr>
</operationalAttributes>
<searchBase type="urn:oasis:names:tc:SPML:1:0#DN">
  <id>g80ab8d5-da94-4360-b159-952f2561e5ee</id>
</searchBase>
```

```
<attributes>
  <attribute name="name"/>
  <attribute name="member"/>
</attributes>
</searchRequest>
```

Example request for user:

```
<searchRequest requestID="req-id">
<operationalAttributes>
  <attr name="objType">
    <value type="string">user</value>
  </attr>
  <attr name="scope">
    <value type="string">Base</value>
  </attr>
</operationalAttributes>
<searchBase type="urn:oasis:names:tc:SPML:1:0#DN">
  <id>2fc7f4e9-2d5a-41c4-bde0-818f9373944b</id>
</searchBase>
<attributes>
  <attribute name="lastname"/>
  <attribute name="firstname"/>
  <attribute name="email"/>
  <attribute name="isemployee"/>
  <attribute name="state"/>
</attributes>
</searchRequest>
```

Specify the value **all** in the **searchBase** if you search for all users or all groups.

The following sample request searches for all groups that contain a user with a specific Citrix ID. The Citrix ID of the user is specified in the **filter** tag and the value **all** is specified in the id of the **searchBase** tag:

```
<searchRequest requestID="req-id">
<operationalAttributes>
  <attr name="objType">
    <value type="string">group</value>
  </attr>
  <attr name="scope">
```

```
    <value type="string">subtree</value>
  </attr>
  <attr name="noattrs">
    <value type="string">true</value>
  </attr>
</operationalAttributes>
<searchBase type="urn:oasis:names:tc:SPML:1:0#GenericString">
  <id>all</id>
</searchBase>
<filter>
  <equalityMatch name="member">
    <value type="string">7b1fc236-8d5b-44dc-92bc-44b5757c3c4e</value>
  </equalityMatch>
</filter>
</searchRequest>
```

2.2.4. Configuration

Here is a sample job configuration snippet for the Citrix ShareFile connector:

```
<connector
 role="connector"
 className="net.atos.dirx.dxi.connector.citrix.sharefile.ShareFileCon
nector"
 name="Citrix ShareFile Connector">
 <connection type="CitrixShareFile"
   user="user@mycompany.com"
   password="dirx"
   server="mycompany.sharefile.com">
   <property name="useSystemProxy" value="false"/>
   <property name="proxy" value="myproxy.domain.com:8080"/>
 </connection>
</connector>
```

2.2.4.1. Supported Connection Parameters

The connector supports the following standard properties of the <connection> element of the XML configuration file:

server - name of the Citrix ShareFile server. It is used to build the Citrix ShareFile Access URL in form **https://***server/*.

user - user e-mail address for authentication.

password - password for authentication to the Citrix ShareFile server.

It also supports the following properties:

useSystemProxy - (optional) boolean (default is **false**). If configured, the connector uses the operating system's default proxy. This parameter is supported only on some Microsoft operating systems.

proxy - (optional) string. If configured, the connector uses an HTTP proxy for connections to Citrix ShareFile. The format is *host*:[*port*].

2.3. CSV Connector

The CSV connector implements the Identity Java Connector Integration Framework's DxmConnectorCore, DxmRequestor and DxmContext interfaces and writes and reads CSV files using the SuperCsv classes. Like all framework-based agents, it gets SPML requests from the Identity side by the join engine as part of the workflow engine hosted by the Java-based Server. It converts the SPML requests in order to read from and write to CSV files.

The CSV connector provides the functionality to:

- Add any kind of object - especially user, account or group - to a CSV file.
- Perform searches on a CSV file to import the objects to Identity.

2.3.1. Overview

The connector implements the API methods "add(...)" and "search(...)". They represent the corresponding SPML requests "AddRequest" and "SearchRequest".

2.3.2. Limitations

No handling for binary data and no handling for multi-value attributes is implemented.

2.3.3. Request and Response Handling

This section describes the supported requests and attributes for the CSV connector.

2.3.3.1. AddRequest

In an add request, the identifier is mandatory. The identifier is mapped to the column specified by the connection property namingAttribute. The connection property csvAttributes specifies the columns of the CSV file. The CSV connector writes to the CSV file. The **export_file** property of the connector specifies the CSV file name. If not specified, the file name is read from the framework context variable **ts.***channelName***.env.export_file** where *channelName* is retrieved from the operational attributes of the AddRequest.

Example request:

```
<?xml version="1.0" encoding="UTF-8" ?>
<spml:batchRequest xmlns="urn:oasis:names:tc:SPML:1:0"
xmlns:spml="urn:oasis:names:tc:SPML:1:0"
xmlns:dsml="urn:oasis:names:tc:DSML:2:0:core"
requestID="batch-1"
processing="urn:oasis:names:tc:SPML:1:0#sequential"
execution="urn:oasis:names:tc:SPML:1:0#synchronous"
onError="urn:oasis:names:tc:SPML:1:0#exit">
<spml:addRequest requestID="add-1">
       <spml:identifier type="urn:oasis:names:tc:SPML:1:0#DN">
         <spml:id>cn=Tymchuk Antonio,ou=Product Testing,o=My-
Company,cn=Users,cn=My-Company</spml:id>
      </spml:identifier>
      <spml:attributes>
         <dsml:attr name="c" xmlns=
"urn:oasis:names:tc:DSML:2:0:core">
               <dsml:value>DE</dsml:value>
         </dsml:attr>
         <dsml:attr name="o" xmlns=
"urn:oasis:names:tc:DSML:2:0:core">
            <dsml:value>My-Company</dsml:value>
         </dsml:attr>
         <dsml:attr name="ou"
xmlns="urn:oasis:names:tc:DSML:2:0:core">
            <dsml:value>Product Testing</dsml:value>
         </dsml:attr>
         <dsml:attr name="l" xmlns=
"urn:oasis:names:tc:DSML:2:0:core">
            <dsml:value>My-Company Berlin</dsml:value>
         </dsml:attr>
         <dsml:attr name="employeeNumber"
xmlns="urn:oasis:names:tc:DSML:2:0:core">
            <dsml:value>4873</dsml:value>
         </dsml:attr>
         <dsml:attr name="sn"
xmlns="urn:oasis:names:tc:DSML:2:0:core">
            <dsml:value>Tymchuk</dsml:value>
         </dsml:attr>
         <dsml:attr name="givenName"
xmlns="urn:oasis:names:tc:DSML:2:0:core">
```

```
            <dsml:value>Antonio</dsml:value>
         </dsml:attr>
         <dsml:attr name="cn"
xmlns="urn:oasis:names:tc:DSML:2:0:core">
            <dsml:value>Tymchuk Antonio</dsml:value>
         </dsml:attr>
    </spml:attributes>
</spml:addRequest>
</spml:batchRequest>
```

2.3.3.2. Search Request

In a SPML search request, the CSV connector supports the elements **searchBase** and **filter**, and the operational attributes **scope**, **pageSize**, **noattrs** (if set to FALSE or not existing all attributes are retrieved) and **channelName**.

The join engine sets the operational attribute **channelName** only in a Java server workflow context. **channelName** is used to get the name of the source file for the SearchRequest if no file name was specified in the CSV connector's <connection> **filename** property. The file name is then obtained from the framework context variable **ts.*** ***channelName***.env.import_file*.

If the join engine calls the CSV connector's search method in the context of a workflow running from Identity to the connected system (export mode), the CSV connector returns an empty search result to make the join engine produce an AddRequest resulting in writing a CSV row.

Example request:

```
<?xml version="1.0" encoding="UTF-8" ?>
<spml:searchRequest  xmlns="urn:oasis:names:tc:SPML:1:0"
    xmlns:spml="urn:oasis:names:tc:SPML:1:0"
    xmlns:dsml="urn:oasis:names:tc:DSML:2:0:core"
    requestID="search_01"
    >
<spml:searchBase type = "urn:oasis:names:tc:SPML:1:0#DN">
  <spml:id>cn=users,cn=My-Company</spml:id>
</spml:searchBase>
  <spml:filter>
    <dsml:and>
      <dsml:equalityMatch name="ou">
      <dsml:value>Sales</dsml:value>
      </dsml:equalityMatch>
      <dsml:not>
```

```
      <dsml:present name="assistant" />
      </dsml:not>
    </dsml:and>
  </spml:filter>

<spml:operationalAttributes>
  <dsml:attr name="scope">
     <value>subtree</value>
  </dsml:attr>
  <dsml:attr name="pageSize">
     <value>0</value>
  </dsml:attr>
  <dsml:attr name="channelName">
     <value>users</value>
  </dsml:attr>
</spml:operationalAttributes>
<spml:attributes>
</spml:attributes>
</spml:searchRequest>
```

2.3.4. Configuration

Here is a sample configuration snippet for the CSV connector:

```
<connector
role="connector"
className=" siemens.dxm.connector.framework.csv.CsvConnector "
name="CSV Connector" version="1.00">
<connection type="file"
  filename="filename="C:/MetahubData/data.csv""
  <property name="csvAttributes" value="c, o, ou, DDN, l,
employeeNumber, cn, title, salutation, sn, givenName, description"/>
  <property name=" export_file"
value="C:/MetahubData/mydataConnector.csv"/>
  <property name="separator" value="|"/>
  <property name="hasHeader" value="true"/>
  <property name="namingAttribute" value="DDN"/>
</connection>
</connector>
```

2.3.4.1. Supported Connection Parameters

The following standard properties of the XML configuration file's <connection> element are supported:

filename - (optional); one or more comma-separated file names used as the source file(s) for the search request (import file). In a Java-based workflow context, the framework context variable **ts.***channelName***.env.import_file** specifies the import file name if the **filename** property is not specified.

Non-standard supported properties:

csvAttributes - (optional); the columns of the CSV file. In a Java-based workflow context, the csvAttributes are taken from the mapped attributes.

export_file - (optional); the name of the file to which the CSV records are written. If not specified in a Java-based Server export workflow context, the framework context variable **ts.***channelName***.env.export_file** specifies the file name.

namingAttribute - The attribute that defines the identifier in the CSV file. For searches, this value identifies which column is used for matching the base node part of the filter and is used as the SPML identifier in the result. For Add request, it identifies the column to which the mapped identifier is written. Like the other parameters in a Java-based workflow context, it is retrieved from the framework context.

separator - The attribute that defines the separator of the CSV file. If not specified, a comma is assumed. Like the other parameters in a Java-based workflow context, it is retrieved from the framework context.

hasHeader - The attribute that defines whether the CSV file contains a header line. If not specified, no header line is assumed. Like the other parameters in a Java-based workflow context, it is retrieved from the framework context.

comment - If set, the given value is used to identify a comment line. Lines beginning with this value are skipped during read. This is not part of the CSV specification.

Parameters in a Java-based workflow context:

Here you can specify the parameters for every channel. You can have channels for different subtrees, different object types and so on. At a specific channel, you define the parameters needed for this channel. The parameters are configured as specific attributes. They have the same names as they do in the connection section. The join engine always treats these names as lowercase.

2.4. Evidian ESSO Connector

The Evidian Enterprise Single SignOn (ESSO) connector is a Java-based connector that is built with the Identity Java Connector Integration Framework and uses the Evidian Web API.

The Evidian ESSO connector implements the API methods "add(...)", "modify(...)", "delete(...)"

and "search(...)". These methods represent the corresponding SPML requests "AddRequest", "ModifyRequest", "DeleteRequest" and "SearchRequest".

The connector currently only supports accounts in Evidian ESSO. One account represents the tuple user - application - role.

The SPML identifier consists of the user(DN), application and role: *userDN* **,application=***appname***,role=***rolename*. The role part is optional; if it is omitted, the role " "(empty string) is assumed.

Every account has the fixed attributes **login** and **secret**. An account represents the possibility for the Active Directory user with the given userDN to log in to the given application automatically as the user specified in **login** with the password specified by **secret**. The role is necessary to specify access to the same application as a different login user.

The Evidian ESSO connector offers the following functionality:

- Add an account to Evidian ESSO
- Delete an account from Evidian ESSO
- Modify accounts and profiles
- Search for accounts in the Evidian ESSO system

2.4.1. Prerequisites and Limitations

The Evidian ESSO connector has the following limitations:

- You can only search for accounts of a given Active Directory user.
- Filters and scopes are not supported in searches.

2.4.2. Request and Response Handling

This section describes the supported requests and attributes for the Evidian ESSO connector.

Parameters are handled as extra attributes: every attribute that is not in the list of fixed attributes is treated as a parameter Sample:

```
Id: userdn,application=SAPGUI
Login=testuser
Secret=test
Mandant=1122
```

In this example, **Mandant** is treated as a parameter **Mandant** with the value **1122**.

2.4.2.1. Add Request

The (account) add request creates a new account in Evidian ESSO. The following attributes are supported:

- The complete SPML identifier
- role
- login
- secret

All other attribute names are treated as parameters.

Here is an example request:

```
<spml:addRequest requestID="add-1">
 <spml:identifier type="urn:oasis:names:tc:SPML:1:0#DN">
   <spml:id>cn=Ben Hamm,cn=users,dc=esso,dc=iam,dc=my-it-
solutions,dc=net,application=ANW
ServerAdmin,role=DirXIdentity</spml:id>
 </spml:identifier>
 <spml:attributes>
   <dsml:attr name="role" xmlns="urn:oasis:names:tc:DSML:2:0:core">
<dsml:value></dsml:value>
   </dsml:attr>
   <dsml:attr name="login" xmlns="urn:oasis:names:tc:DSML:2:0:core">
<dsml:value >DomainAdmin</dsml:value>
   </dsml:attr>
   <dsml:attr name="secret" xmlns="urn:oasis:names:tc:DSML:2:0:core">
<dsml:value >dirx</dsml:value>
   </dsml:attr>
 </spml:attributes>
</spml:addRequest>
```

2.4.2.2. Modify Request

The (account) modify request modifies a Evidian ESSO account. The same attributes as in Add Request are supported.

Here is an example request:

```
<!-- Modify login name for user Ben Hamm, Role DirXIdentity and
ServerAdmin application  -->
<spml:modifyRequest requestID="mod-1">
```

```
 <spml:identifier
     type = "urn:oasis:names:tc:SPML:1:0#DN">
     <spml:id>cn=Ben Hamm,cn=users,dc=esso,dc=iam,dc=my-it-
solutions,dc=net,application=ANW
ServerAdmin,role=DirXIdentity</spml:id>
 </spml:identifier>
 <spml:modifications>
     <spml:modification name="login" operation="replace">
           <dsml:value>Taspatch Nik</dsml:value>
     </spml:modification>
  </spml:modifications>
</spml:modifyRequest>
```

2.4.2.3. Delete Request

The delete request is used to delete an account. Here is an example request:

```
<!-- delete ServerAdmin for Ben Hamm with role DirXIdentity -->
  <spml:deleteRequest requestID="del-1">
     <spml:identifier
         type = "urn:oasis:names:tc:SPML:1:0#DN">
         <spml:id>cn=Ben Hamm,cn=users,dc=esso,dc=iam,dc=my-it-
solutions,dc=net,application=ANW
ServerAdmin,role=DirXIdentity</spml:id>
     </spml:identifier>
  </spml:deleteRequest>
```

2.4.2.4. Search Request

The search request is used to retrieve group data such as owner information, members and roles. The search can either be restricted to one specific group or return all groups in the current site. Only searches per user are supported

The base node is the user DN or identifier as in Add Request. The available attributes are:

- userDN
- application
- role
- log
- secret encrypted value

All other names are treated as parameter names.

The search filter is not evaluated. The userDN gives all accounts for this user. The complete identifier filters for application and role of the given user.

Here is an example request:

```
<!-- search entry for ad user Ben Hamm  app Web Center  no role given
-->
<spml:searchRequest xmlns="urn:oasis:names:tc:SPML:1:0"
xmlns:spml="urn:oasis:names:tc:SPML:1:0"
xmlns:dsml="urn:oasis:names:tc:DSML:2:0:core" requestID="search_003">
  <spml:searchBase type="urn:oasis:names:tc:SPML:1:0#DN">
     <spml:id>cn=Ben Hamm,cn=users,dc=esso,dc=iam,dc=my-it-
solutions,dc=net,application=ANW WebCenter</spml:id>
  </spml:searchBase>
  <spml:attributes>
      <dsml:attribute name="userDN"/>
     <dsml:attribute name="application"/>
     <dsml:attribute name="role"/>
     <dsml:attribute name="login"/>
     <dsml:attribute name="secret"/>
  </spml:attributes>
</spml:searchRequest>
```

2.4.3. Configuration

Here is a sample configuration snippet for the Evidian ESSO connector:

```
<connector name="ESSO connector"
           role="connector"
           version="1.02"

className="net.atos.dirx.dxi.connector.evd.esso.EvidianEssoConnector"
>
    <connection name="ESSOconnector"
                url="https://<essohost>:9765/soap"
                user="<Administrator>"
                password="<pw>">
    </connection>
</connector>
```

The Evidian ESSO connector supports the following standard properties of the XML

configuration file's <connection> element:

url (mandatory) - the URL for the Evidian User Access Web Service port - not used.

user (mandatory) - the user name to access the Web service.

password (mandatory) - the password.

2.5. Google Apps Connector

The Google Apps connector implements the Identity Java Connector Integration Framework's **DxmConnector** interface and connects to a Google Apps server through the Google Apps API. It can be used for real-time workflows in the Java-based (IdS-J) Server. Like all framework-based connectors, it gets SPML requests from the DirX Identity side and then converts them to the appropriate Google Apps API calls and vice versa. The Google Apps connectivity is based on HTTP protocol. The connector supports membership stored on the accounts level.

The connector is implemented in the **GoogleAppsConnector** class in the package **net.atos.dirx.dxi.connector.googleapps**.

The connector implements the common methods for the DirX Identity Connector API: add, modify, delete and search.

The operations are simply converted to the Google Apps Admin Directory API requests. The corresponding responses are again translated to SPMLv1 responses.

The Google Apps Admin Directory API is a RESTful service comprised of endpoints that are accessed using standard HTTP requests. The connector uses JavaScript Object Notation (JSON) content types for requests and responses.

The connector communicates using SSL/TLS only.

2.5.1. Prerequisites and Limitations

The connector is based on Admin Directory API version 1.19.0 available at **https://developers.google.com/admin-sdk/directory/v1/libraries**. The connector functionality is limited by the functionality of the API version in use. Compatibility with other API versions is not guaranteed.

To communicate with the Google servers, the connector needs to authenticate using a Service Account Private Key, a Service Account User and a Service Account Email provided by Google on account creation.

The operations are authorized by an OAuth server, so the privileges and scope need to be set in the Google Admin Console; they cannot be modified at the connector level.

The connector supports common Google Apps user objects (common attributes and navigation properties like memberOf, manager and secretary) and Google Apps group objects (common attributes only).

The connector does not support nested group assignment. Nested group assignments cannot be read or written.

2.5.2. Request and Response Handling

This section describes the supported requests and attributes for the Google Apps connector. All attributes allowed by the Google Apps API can be added.

2.5.2.1. Add Request

The (user) add request creates a new user in Google Apps. The following attributes are supported:

- primaryEmail - mandatory, unique
- givenName - mandatory
- familyName - mandatory
- password - mandatory
- suspended
- changePasswordAtNextLogin
- ipWhitelisted
- externalIds - for type "work"
- relations - for type "manager" and "assistant"
- addresses [poBox] - for "primary"
- addresses [extendedAddress] - for "primary"
- addresses [streetAddress] - for "primary"
- addresses [locality] - for "primary"
- addresses [region] - for "primary"
- addresses [postalCode] - for "primary"
- addresses [countryCode] - for "primary"
- organizations [name] - for "primary"
- phones [work]
- phones [work_mobile]
- phones [home]
- orgUnitPath
- includeInGlobalAddressList
- memberOf

Here is an example request:

```
<spml:addRequest  returnData="identifier"
```

```
    requestID="add-user-01" targetID="users"
    xmlns="urn:oasis:names:tc:SPML:1:0"
    xmlns:spml="urn:oasis:names:tc:SPML:1:0"
    xmlns:dsml="urn:oasis:names:tc:DSML:2:0:core">
   <spml:identifier type="urn:oasis:names:tc:SPML:1:0#GenericString">
       <spml:id>1234</spml:id>
   </spml:identifier>
      <spml:attributes>
    <dsml:attrname="userName"><dsml:value> Miller
Tom</dsml:value></dsml:attr>
     <dsml:attr
name="givenName"><dsml:value>Miller</dsml:value></dsml:attr>
     <dsml:attr
name="familyName"><dsml:value>Tom</dsml:value></dsml:attr>
     <dsml:attr
name="password"><dsml:value>password</dsml:value></dsml:attr>
     <dsml:attr
name="state"><dsml:value>ENABLED</dsml:value></dsml:attr>
     <dsml:attr name="primaryMail"><dsml:value>
Miller@domain</dsml:value></dsml:attr>
     <dsml:attr name="phones">
          <dsml:value type="string">

{"value":"0724553207","type":"work_mobile","primary":true}
          </dsml:value>
          <dsml:value type="string">
             {"value":"0724553207","type":"home"}
          </dsml:value>
     </dsml:attr>
     <dsml:attr name="externalIds">
        <dsml:value
type="string">{"value":"123","type":"organization"}</dsml:value>
     </dsml:attr>
     <dsml:attr name="relations">
          <dsml:value type="string">
             {"value":"Razvan","type":"manager"}
          </dsml:value>
          <dsml:value type="string">
             {"value":"Rudi","type":"assistant"}
          </dsml:value>
     </dsml:attr>
```

```
     <dsml:attr name="addresses">
         <dsml:value type="string">

{"type":"work","poBox":"21","extendedAddress":"Griviteinr.77","street
Address":"Grivitei","locality":"Brasov","region":"Brasov",
              "postalCode":"1111","countryCode":"US","primary":true}
      </dsml:value>
    </dsml:attr>
      </spml:attributes>
            <spml:operationalAttributes>
                <spml:attr name="objType">
                    <dsml:value type="string">user</dsml:value>
               </spml:attr>
     </spml:operationalAttributes>
</spml:addRequest>
```

The (group) add request creates a new group in Google Apps. The following attributes are supported:

- email - mandatory, unique
- name
- description

2.5.2.2. Modify Request

In the modify request, the identifier is mandatory. All attributes allowed by the Google Apps API can be modified.

Note that users may experience issues when trying to delete everything under a multiple-valued attribute like "phones" or "addresses". This is a Google API problem: it doesn't allow you to delete everything at once, just one entry at a time.

2.5.2.3. Delete Request

In the delete request, the identifier is mandatory. The delete request does not require additional attributes.

2.5.2.4. Search Request

In the search request, the Google Apps connector supports the standard element **searchBase** and the operational attributes **scope** and **objType**.

To search for all users or groups, the **searchBase** needs to be empty.

2.5.3. Configuration

Here is a sample configuration snippet for the Google Apps connector:

```
<connector
className="net.atos.dirx.dxi.connector.googleapps.GoogleAppsConnector
"
                 name="TS" role="connector">
       <connection user="admin@dirx-interop.com">
           <property name="proxyHost" value="proxy-emea.my-it-
solutions.net"/>
           <property name="proxyPort" value="84"/>
           <property name="domain" value="dirx-interop.com"/>
           <property name="applicationName" value="identity"/>
           <property name="clientId"
value="developerID@developer.gserviceaccount.com"/>
           <property name="clientSecret"
value="###@dxrConnectionLink@dxmClientSecret###"/>
       </connection>
     </connector>
```

2.5.3.1. Supported Connection Parameters

The connector supports the following standard properties of the <connection> element of the XML configuration file:

user - the user identifier to be used for authentication in the format: *name***@***domain*.

It also supports the following properties:

proxyHost: optional. This property provides information about the host name or IP address of the HTTP proxy server. Do not use authenticated proxy servers. If the access to the proxy server requires authentication, deploy another local transparent proxy server that can access to the authenticated one. Use only local proxy server instead.

proxyPort: optional. This property provides information about the port number of the HTTP proxy server. Do not use authenticated proxy servers. See description for proxyHost for more details.

clientId: mandatory. This property provides the Service Account Email Key of your registered remote application, used for authenticating to Google Apps.

clientSecret: mandatory. This property provides the Service Account Private of your registered remote application, used for authenticating to Google Apps.

domain: mandatory. This property is the domain of your registered remote application.

applicationName: mandatory. This property is the name you selected for your application. (The value can be anything you want; the name is used by the Google servers to monitor the source of authentication).

2.5.3.2. Additional Notes

Access to the Google Apps API must be activated using the Google Apps administration web site. (See the online Google Apps documentation.)

The Google Apps Provisioning API has been officially deprecated as of May 15, 2013. It has been replaced by the Admin SDK's Directory API. The Google Apps Provisioning API will continue to work according to the Deprecation Policy.

2.6. Identity Domain Connector

The Identity Domain connector (sometimes called the Service Layer connector) implements the DirX Identity Java Connector Integration Framework's **DxmConnector** interface and connects to an Identity domain.

The Identity Domain connector implements all of the functionality to create, update, delete and search all entries in a DirX Identity domain. It also allows the assignment of roles, groups and permissions to users. The assignments can be attributed, where they can have an end date and one or more role parameters. Rather than communicating on the low-level LDAP protocol, the Identity Domain connector uses the DirX Identity service layer component. This design allows the connector to use the service layer features: the object descriptions, and user resolution.

The Identity Domain connector supports all entry types in a domain: users, roles, groups, business objects, and more. The Identity Domain connector evaluates the object description for the creation of an entry. The object description name can be given explicitly or is determined from other attributes, normally from the type and object class. When an entry is to be created, a client needs to provide only a minimum set of attributes; no DN or naming attribute is required. The connector applies the object description rules for initial values and attributes dependencies.

The Identity Domain connector supports password changes; passwords can be part of both add and modify requests.

The following sections describe how the Identity Domain connector handles requests and responses as well as the connector's configuration.

2.6.1. Prerequisites and Limitations

The Identity Domain connector is contained in the library **dxmSvcLayerConnector.jar**. It is based on the DirX Identity Java Connector Integration Framework and uses the DirX Identity service layer for access to the DirX Identity domain and for searching and maintaining the domain entries. The service layer is contained in the library **dxrServices.jar** and in turn depends on a number of other DirX Identity and external libraries.

The Identity Domain connector supports only one role parameter value per assignment

and role parameter. An assignment can have multiple role parameters, but each parameter can have only one value. If you need more values, then you need to create more assignments. For example, if a user is a member of two projects, create two assignments, one for each project.

2.6.2. Request and Response Handling

This section describes how the connector handles requests and responses.

2.6.2.1. Object Description

The service layer requires an object description name for creating an entry, so the request needs to contain attributes that allow the connector to select an object description. The best way to satisfy this requirement is to pass the object description name in the virtual attribute **odName**. For example, to create a functional user, take the odName **dxrFunctionalUser**. You'll find the object descriptions with their names in the domain configuration tree.

As an alternative, you can provide the LDAP attributes that are sufficient to identify an object description. Normally, these attributes are **objectclass** and optionally **dxrType**. Check the **<mapping>** section of the object descriptions to determine the attributes that are evaluated by the service layer. For a functional user, you only need to provide the **objectclass** value **dxrFunctionalUser**.

If the given values are not sufficient to identify an object description, the connector checks whether the new entry should become a container. If an object class value contains the string **container**, the connector uses the object description name **dxrContainer**.

2.6.2.2. Object Class

Only the object classes that are necessary to identify an object description need to be provided. If the request already contains an object description name, no object classes need to be provided; the service layer takes the missing object classes from the object description.

2.6.2.3. Parent Entry

An entry must be created under another existing entry, the parent. The easiest way to satisfy this requirement is to provide the DN of the parent in the virtual attribute **parentDN**. If the add request contains an identifier DN, the Identity Domain connector extracts the parent DN from this attribute.

If a parent cannot be identified this way, the connector inspects the found object description and its parent descriptions. If the new entry is a user, role or permission, the connector takes the corresponding root entry.

If a parent still cannot be identified, the connector places the new entry under the business objects container.

2.6.2.4. Approval

If the creation of a new entry requires approval, the service layer starts the approval workflow. If the workflow is still in progress, the Identity Domain connector returns the result code PENDING and takes the entry identifier from the workflow's subject identifier. Note that the entry DN may be changed in the remainder of the workflow.

2.6.2.5. Password

The Identity Domain connector expects the password in the **userPassword** attribute. If the user or account is not in approval, it sets the password for the entry using the normal password support. Otherwise, it's the workflow's responsibility to define and set a password for a new user or account. With password support, the passwords are handled in the same way as, for example, with Web Center. Password support sends a password change event with the encrypted password. The workflow **UserPasswordEventManager** will process them in the standard way and update also corresponding accounts.

The connector does not use password support only if the **suppressPwdEvent** option is set to true; it simply stores the password in the entry's user password attribute. In this case, in the object description the property type must be declared as "[B" (binary array).

2.6.2.6. Renaming

A modify request should contain the DN of the entry in its ID and the old DN in the **dxrprimarykeyold** attribute; this configuration should be standard for all Provisioning workflows. This configuration allows the connector to detect a requested move of the entry. In this case, it asks the service layer to perform the rename and then applies other modifications.

2.6.2.7. Delete

Before the Identity Domain connector deletes an entry, it asks the service layer to delete all of its children. Typically, an entry will not be physically deleted, but set to state **TBDEL** (to be deleted).

2.6.2.8. Search

The Identity Domain connector follows the standard rules for search requests:

- It reads an entry, if the operational attributes in the request specify a scope of **base**.
- It performs a paged search, if the operational attribute **pagesize** has a value greater than 0. In the other cases, it performs a normal search, evaluating search base, filter, requested attributes, sizelimit, timelimit, sortattribute and sortorder; except for search base and filter, all of these are operational attributes. If the operational attribute **noattrs** is given with a value of **true**, it doesn't request any attribute.

2.6.2.9. References to other LDAP Entries

References to other LDAP entries in attributes such as **owner** or **dxrLocationLink** must be given as LDAP DNs. Typically, the source databases will not recognize these DNs. Instead, they have a unique value, often something like a primary key. A mapping function needs to

search the entry based on this unique value. An example of how to do this is given in the source for the role mapping function.

2.6.2.10. Privilege Assignments

The Identity Domain connector provides special handling for privilege assignments.

For simple assignments, which do not have an end date or role parameters, it is sufficient to put the DN of the assigned role, permission or group into the respective attribute **dxrRoleLink**, **dxrPermissionLink** or **dxrGroupLink**.

For attributed assignments, the virtual attributes **rolesassigned**, **permissionsassigned** and **groupsassigned** must be used; they can also be used for simple assignments. The content of these attributes is a structured JSON object. Here is an example:

```
{
"privilegeLink": "cn=Project Member,cn=Project Specific,cn=Corporate
Roles,cn=RoleCatalogue,cn=My-Company",
"dxrEndDate": "20181231230000.000Z",
"params": [
   {
       "paramDN": "cn=Project,cn=My-Company,cn=RoleParams,cn=Customer
Extensions,cn=Configuration,cn=My-Company",
       "paramUid": "uid-7f001-cee271-fed935aa6d--7eb4",
       "paramKey": "MoreCustomers",
       "paramValue":
"cn=MoreCustomers,cn=Projects,cn=BusinessObjects,cn=My-Company"
    }
  ]
}
```

The example above contains new lines, tabs and blanks for better reading. Note that before you pass this value to the join engine of a Provisioning workflow, you should trim it; that is, remove all white spaces (blanks and tabs). This would matter if you provided the value in an LDIF or CSV file. Otherwise, the value would not exactly match the value that is returned from the connector and the join engine would always request the connector to modify the entry in LDAP. So, the better way is to have some channel filter component between the source connector and the join engine as in the sample workflow. This filter component should use the provided class PrivilegeAssignedDTO to generate the attribute value. For an example of how to do this, see the sources of the method addAssignmentAttribute(...) in the filter class JdbcRoleAsgFilter; you'll find it in the Additions folder of the product media.

In the request to the connector, the JSON object needs to contain the **privilegeLink** property with the DN of the privilege to assign. This is the minimum content and represents a simple assignment. An end date is expected in the **dxrEndDate** property; no start date is evaluated.

Role parameters are represented in the **params** array. They can have the DN of the parameter, its **dxrUid** and the key and value for the parameter value. A request must contain either the parameter UID or its DN in addition to the parameter value and value key.

The Identity Domain connector supports only one value per role parameter because the sequence of values cannot be determined in JSON. As a result, comparing mapped and actual existing values in the workflow might easily return the wrong result even when logically they are the same.

The Identity Domain connector returns only direct assignments in a search response; it does not return inherited and rule-based assignments. The result also contains assignments in approval. This helps to prevent duplicate request workflows, where an assignment is still in approval and the Provisioning workflow runs again.

The DN values of the privilege, a role parameter or even role parameter values are not usually present in a source database; instead, the source database contains only unique IDs. It's the responsibility of filter and/or mapping functions in the Provisioning workflow to supply these DNs. For more information, see the section "Relational Database User Import Workflow" in the chapter "Using the Source Workflows" in the *DirX Identity Application Development Guide*.

2.6.2.11. Example Add Request

The following add request contains the minimum set of attributes that are necessary for creating a user:

```
<spml:addRequest
    xmlns:spml=urn:oasis:names:tc:SPML:1:0
    xmlns:dsml=urn:oasis:names:tc:DSML:2:0:core
>
    <spml:attributes>
         <spml:attr name=sn>
    <dsml:value>UserWithJustObjectDescriptionName</dsml:value>
         </spml:attr>
         <spml:attr name=givenname>
              <dsml:value>new</dsml:value>
         </spml:attr>
         <spml:attr name=parentDN>
              <dsml:value>cn=Users,cn=My-Company</dsml:value>
         </spml:attr>
         <spml:attr name=odName>
              <dsml:value>dxrUser</dsml:value>
         </spml:attr>
    </spml:attributes>
```

```
</spml:addRequest>
```

2.6.3. Configuration

The Identity Domain connector is configured according to the DirX Identity Java Connector Integration Framework. Its class is **net.atos.dirx.dxi.connector.svclayer.ServiceLayerConnector**.

The connector evaluates the following options:

- **server** - the host name or IP address of the LDAP server.
- **port** - the port number of the LDAP server.
- **user** - the DN of the LDAP user for binding.
- **password** - the password for binding.
- **ssl** - true if TLS is to be used.
- **domain** - the name of the Identity domain, including the prefix **cn=**. If it is missing, the top-level RDN of the user DN is used.
- s**uppressPwdEvent** - whether (**true**) or not (**false**) the connector stores the user password as is in the **userpassword** attribute or sends a password change event to the **UserPasswordEventManager** workflow.

2.7. Imprivata One Sign Connector

The Java-based Imprivata OneSign connector runs inside the Identity Connector Integration Framework. It extends the standard SPML v1 SOAP Connector. It sends SPML SOAP requests over HTTP to the configured Imprivata OneSign endpoint and receives SPML SOAP responses from Imprivata OneSign provisioning service.

The connector supports only specific SPMLv1 requests those are necessary for provisioning of Imprivata OneSign: addRequest, modifyRequest, deleteRequest, searchRequest.

The connector supports basic authentication as well as server-side SSL/TLS authentication. It does not support WS-Security protocols yet.

2.7.1. Prerequisites

The deployment of the connector is the same as for the standard SPMLv1 Connector. See "Prerequisites" in "SPMLv1 Connector" for details.

2.7.2. Configuration

The connector uses mostly the same configuration as the standard SPMLv1 Connector. (See "Configuration" in "SPMLv1 Connector" for details.) Additionally it uses a special configuration parameter:

externalSystemName: mandatory; this property is equal to the name of the configured

Provisioning System Adaptor in the Imprivata OneSign appliance. Set this value in the connector port according to the values configured in the Imprivata OneSign system.

The following is a sample configuration for Imprivata OneSign connector:

```
    <connector
className="siemens.dxm.connector.framework.soap.ImprivataSpmlSoapProx
y" name="TS" role="connector">
     <connection password="{SCRAMBLED}aG5WPw==" port="443"
         server="imprivata" ssl="TRUE" type="Imprivata OneSign"
user="dummy">
         <property name="externalSystemName" value="My-Company"/>
         <property name="includePrefixesForXsdPrimitiveTypes"
value="FALSE"/>
         <property name="timeout" value=""/>
         <property name="path" value="sso/provision/spmlrouter"/>
     </connection>
 </connector>
```

2.8. JDBC Connector

The JDBC connector is the DirX Identity connector that handles the import and export of information into and out of relational databases. It is based on the DirX Identity Connector Integration Framework. The connector implements the **DxmConnectorExtended** interface of the Java Connector Integration Framework.

2.8.1. Overview

The connector implements the API methods "add(...)", "modify(...)", "delete(...)", "search(...)" and "extendedRequest(...)". They represent the corresponding JDBC SQL statements INSERT, UPDATE, DELETE, SELECT and CALL stored Procedure.

2.8.2. Prerequisites

The JDBC connector is contained in

dxmJDBCConnector.jar.

The connector is based on the Java Connector Integration Framework. The framework is contained in the library

dxmConnector.jar.

Depending on the JDBC driver used, the appropriate jar file for the driver is also a prerequisite. For use in a Tcl workflow or standalone, follow the instructions given by the provider of the driver. If you use the driver in the Java-based Server (IdS-J), you need to put

the jar file in the server's *install_path***ids-j-***domain***-S***n****confdb\common\lib*** directory.

2.8.3. Configuration

The agent is composed of multiple sub-units (connectors), each configured within the configuration file. Here is the top-most level structure:

```
<?xml version="1.0" encoding="UTF-8" ?>
<job>
  <connector

className="siemens.dxm.connector.framework.DefaultControllerStandalon
e"
    name="Default Controller" role="controller" version="0.1">
  </connector>
  <connector name="JDBC connector" role="connector"
    className="siemens.dxm.connector.jdbc.JDBCConnector">
<!-- additional JDBC connector material -->
  </connector>
  <connector
className="siemens.dxm.connector.framework.SpmlFileReader"
    name="SPML file reader" role="reader">
    <connection filename="examples\\TestReq.xml" type="SPML" />
  </connector>
  <connector
className="siemens.dxm.connector.framework.SpmlFileWriter"
      name="SPML File writer" role="responseWriter">
    <connection filename="examples\\TestRsp.xml" type="SPML" />
  </connector>
</job>
```

This top-level structure conforms to the generic structure of the DirX Identity Agent Integration Framework, as shown in the following figure:

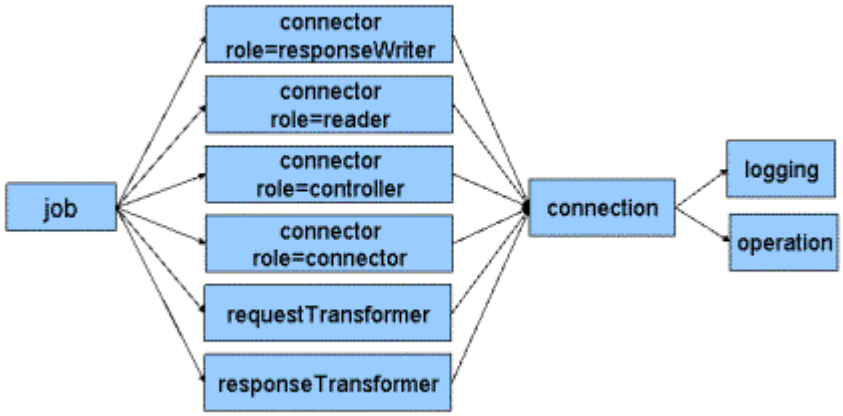


Figure 1. JDBC Connector Top-Level Structure

The readers and writers can also be configured to receive and accept LDIF.

Here, the only variable material (assuming the readers /writers specified here) is:

- The line marked <!-- additional JDBCConnector material -→, which is described in the next sections.
- The connector sections for the roles “reader” and “responseWriter”. They contain the input and output filenames (in bold) and the classes for the reader / writer. Use “siemens.dxm.connector.framework. LdifChangeReader” for LDIF change input format and “siemens.dxm.connector.framework. LdifFileWriter” for LDIF content output.

2.8.3.1. General Notes

This section provides general information about JDBC connector configuration.

2.8.3.1.1. JDBC Connector Element Form of the Connector

Here is an example of the contents of the two elements that form the sub-elements of the connector element, for the connector role within a JDBC connector. It is followed by a brief summary of key components. Further details and explanations are given later.

This is an example of the XML used in:

<!-- additional JDBCConnector material -→

as indicated above.

```
1:<connection
    type="com.microsoft.jdbc.sqlserver.SQLServerDriver"

url="jdbc:microsoft:sqlserver://TIGGER2:1433;databasename=NorthWind"
    user="sa"
    ...
```

```
driverDBType="siemens.dxm.connector.jdbc.SQLServerOverMicrosoftDriver
"
     driver_{property1}="value1"
     driver_{property2}="value2"
     ...
     driver_{propertyN}="valueN"
     >

2:  <jdbc-connection always-follow-references="false">
3:    <tables-and-views>
4:    <table>
5:        <name>employees</name>
6:      </table>
7:      <!-- more tables -->
8:      <view>
9:        <name>employees</name>
10:        <from>employees INNER JOIN Employees AS employeesAsBoss ON
                        employees.ReportsTo =
employeesAsBoss.EmployeeID</from>

11:        <where>employees.ReportsTo =
employeesAsBoss.EmployeeID</where>
12:        <table description="Employees">
13:          <name>employees</name>
14:        </table>
15:        <table>
16:          <name>employeesAsBoss</name>
17:        </table>
18:      </view>
19:    </tables-and-views>
20:    <abbreviation name="order-id">orders.OrderId</abbreviation>
21:    <abbreviation name="boss">employeesAsBoss.FirstName + ' ' +
               employeesAsBoss.LastName AS Boss</abbreviation>
22:    <!-- more abbreviations -->
23:    <relationship from="bossid" referring-to="id" />
24:    <!-- more relationships -->
25:    <functions-and-procedures>
26:      <function name="ADD_FUNC">
27:        <return>
28:          <range exact="0" />
```

```
29:          <range min="1" max="6" /> ]
30:          <!-- more ranges -->
31:        </return>
32:        <argument name="base" in-out="in" preset="2"
dataType="INTEGER" />
33:        <!-- more arguments -->
34:      </function>
35:      <!-- more functions -->
36:      <procedure name="add_proc">
37:        <return name="result">
38:          <range exact="0" />
39:          <range min="1" max="6" /> ]
40:          <!-- more ranges -->
41:        </return>
42:        <argument name="base" in-out="in" preset="2"
dataType="DECIMAL" />
43:        <argument name="addend" in-out="in" preset="5"
dataType="INTEGER" />
44:        <argument name="result" in-out="out" dataType="INTEGER" />
45:        <!-- more arguments -->
46:      </procedure>
47:      <!-- more procedures -->
48:    </functions-and-procedures>
49:  </jdbc-connection>
50:</connection>
51:<logging filename="trace" level="5" logger="JDBCLogger"/>
```

Line 1 - specifies the details of the connection to the database. The attributes starting with the prefix **driver_** are passed directly to the JDBC driver. This action allows additional configuration of user properties of the driver within the connector configuration. For example, a connection attribute **driver_selectTimeout="10"** would be passed to the driver as a user property **selectTimeout** with the value **10**. Check the supported properties of the particular JDBC drivers and see the Java JDBC API for details.

Line 2 - jdbc-connection element - specifies general properties of the JDBC functionality

Line 3 - groups tables and views

Line 8 - table definitions,

Line 8 - view definitions, including table components

Line 20 - abbreviations - normally short-form names for DSML-stype attributes.

Line 23 - referential integrity definitions (called relationships)

Line 25 - groups function-and-procedure definitions

Line 26 - defines all stored functions

Line 36 - defines all stored procedures

Line 51 - <logging> - specifies JDBC connector-specific logging

2.8.3.1.2. Description Attributes

In the interests of legibility, description attributes have been omitted from the example. They are permitted as attributes of elements with the following tags:

- tables-and-views
- table
- view
- abbreviation
- functions-and-procedures
- function
- procedure
- FPReturn
- range
- argument

For descriptions of these elements, see the next sections.

2.8.3.2. Connector Element

The connector element with the attribute values name="JDBC connector" role="connector" is concerned with the detailed configuration of the JDBC connector.

The following sections describe the part of the XML that goes into the line marked <!-- additional JDBCConnector material -→ above. Two tags, **connection** and **logging**, are relevant.

connection

Defines the basic connection of the database and the means needed to support it.

logging

Defines the logging that will be available for the connection and its operation.

2.8.3.3. Connection Element

This section describes the attributes and sub-elements of the JDBC connector connection

element.

2.8.3.3.1. Attributes

Configure the following required attributes:

type

The class of the JDBC driver that you are using to access a database (normally a relational database).

Currently, the following are available:

- For Microsoft SQL-Server 2000: com.microsoft.jdbc.sqlserver.SQLServerDriver
- For Microsoft SQL-Server 2005 and newer: com.microsoft.jdbc.sqlserver.SQLServerDriver
- For MS-Access (JDBC-ODBC Bridge): sun.jdbc.odbc.JdbcOdbcDriver
- For MS-Access: net.ucanaccess.jdbc.UcanaccessDriver
- For Oracle JDBC versions lower than 9.0.1: odbc.jdbc.driver.OracleDriver
- For Oracle JDBC version 9.0.1 and higher: odbc.jdbc.OracleDriver
- For PostgreSQL 9: org.postgresql.Driver

url

The URL of the specific database to be accessed. The form of the URL will be described in the documentation for the JDBC driver.

user

The name of the JDBC user: this user must be empowered to access the data (read and write) within the limits of planned use for the JDBC connector.

password

The user's password. May be omitted if it is just "".

driverDBType

The Java class representing data-type capabilities and conversions for the combination of the selected database and the JDBC driver. If not supplied, a default capability is provided that should handle common eventualities.

Currently, the following are available:

- siemens.dxm.connector.jdbc.AccessOverJdbcOdbcDriver representing sun.jdbc.odbc.JdbcOdbcDriver accessing Microsoft Access databases
- siemens.dxm.connector.jdbc.SQLServerOverMicrosoftDriver representing com.microsoft.jdbc.sqlserver.SQLServerDriver accessing Microsoft SQL Server databases
- siemens.dxm.connector.jdbc.OracleOverOracleDriver representing ORACLE drivers
- siemens.dxm.connector.jdbc.DB2OverIBMDriver representing IBM DB2 drivers

- siemens.dxm.connector.jdbc.PostgreSQLOverJdbcOdbcDriver representing PostgreSQL

2.8.3.3.2. Sub-elements

The following are the sub-elements of connection:

jdbc-connection
Configures the connection to the database.

Property debugfile
Set this property to enable the DriverManager logging. Logging goes to the specified prefix followed by some suffixes.

Example:

```
<property name="debugfile" value="JDBCTrace." />
```

will result in a log file named like:

```
JDBCTrace._71_Wed_Oct_22_17.36.42_CEST_2008.log
```

Property noSPcheck
Set this property to avoid check calls for configured Stored Procedures (SPs) during open. If you don't specify this property or set it to false during open, every configured Stored Procedure is called with default values to check the configuration .(does the SP exist? Have you configured the right arguments? and so on.). Use this property to avoid these calls after successfully testing the Stored Procedure configuration.

Example:

```
<property name="noSPcheck" value="true" />
```

2.8.3.4. JDBC-Connection Element

This section describes the attributes and sub-elements for the JDBC-Connection element.

2.8.3.4.1. Attributes

The JDBC-Connection element has the following attribute:

always-follow-references
Set this to "true" if it is required that the references supplied by relationship elements be followed and the pointing column value be set to null. Doing this may be a good idea to prevent pointers pointing "into thin air" when a pointed-to row is removed. It will not be necessary when all relationships are policed for relational integrity: in that case, the failure to carry out a delete operation triggers the following of references, and trying again to do the original deletion.

2.8.3.4.2. Sub-elements

The JDBC-Connection element has the following sub-elements:

tables-and-views
Provides a general container for table and view specifications.

abbreviation (multiple instances)
At least one of these must be present.

Provides a means to label columns of a particular table. Abbreviations can also represent expressions, such as can be used in an SQL SELECT statement:

SELECT RTRIM(x) AS xWithoutTrailingBlanks FROM ...

However, such expressions cannot in general be used for adds (INSERTs) or modifies (UPDATEs).

See the section "Abbreviation" for details.

relationship (zero or more instances)
A relationship specifies an abbreviation that corresponds to a column in some table that used to point to a row in a table; the two tables can be the same, but usually they are different. The pointed-to table must have a single primary key that is configured as an abbreviation.

functions-and-procedures (optional)
Provides a general container for function and procedure specifications.

2.8.3.5. Logging Element

This element is optional. If absent, the controller log is used, with the level of logging set for it. The JDBC connector log has a default trace level of 3.

This element can contain just a level, which controls the catching of data as described in the following section. Whatever level of information is captured, the data is presented to the controller log, and is then passed to the user depending on the level of setting.

2.8.3.5.1. Attributes

The logging element has the following attributes:

level
Controls the detail that is provided in trace-files. The following values apply:

3 → operation are reported, tagged with the request-id;

5 → Operation SQL is logged (i.e. SQL generated as a direct consequence of a user-requested operation);

7 → Causes calls to main functions in JDBC to be logged.

8 → Causes schema SQL to be logged (i.e. SQL generated as a result of JDBC functions to investigate schema matters);

9 → Causes table/column details to be logged.

For all information to be made available in the system log, the value of 9 or higher must be set for it.

logger

Provides a root filename for the log-file. If this is XXX, a typical output log-file is XXX.000.log.

2.8.3.6. Schema Names

At a more detailed level, several definitions require specification of schema objects, such as tables and columns. For this reason, the nature of names for these objects needs to be introduced. Such names are called "schema names".

There are two forms of names usable within the connector:

- Unquoted names, which are case-insensitive, start with an alphabetic character and can contain alphanumerics or underscores '_' after the first character.
- Double-quoted name, which are case-sensitive, cannot contain double quotes but are otherwise unrestricted.

Not all databases use double quotes for the construction of generalized names: the JDBC connector identifies the native form for its own purposes.

In some elements where the SQL for access to a database is exposed (for example, in the construction of views), the SQL must comply with the standards for the database and double quotes may not be appropriate.

The following is a synopsis of the rules:

Schema-names must be from 1 to 30 bytes long. They can be in one of two forms:

Unquoted

Double-quoted

Unquoted names are case-insensitive. They must begin with an alphabetic character (A-Z, a-z, no more) followed by one or more of:

Alphanumeric (A-Z, a-z, 0-9, no more)

underscore (_)

Spaces and hyphens are not permitted.

Double-quoted names are always enclosed in double quotation marks “...”. Such names can

contain any combination of characters, including spaces.

In referring to a double-quoted name, you must always use double quotation marks whenever referring to the object. Enclosing a name in double quotes enables it to:

Contain spaces

Be case sensitive

Begin with a character other than an alphabetic character, such as a numeric character

Contain characters other than alphanumeric characters and _, $, and #

Be a reserved word

An uppercase unquoted name is taken as identical to the same name with double quotes

2.8.3.7. Table-and-Views Element

This section describes the attributes and sub-elements for the table-and-views element.

2.8.3.7.1. Attributes

Only an optional description attribute is defined.

2.8.3.7.2. Sub-elements

The table-and-views element has the following sub-elements:

table (1 or more instances)

The database table(s) to which the JDBC connector requires direct access. The first table specified is taken as the *default table*, and will be used when no other table can be deduced as relevant. Note that using an abbreviation does imply an associated table, except when the abbreviation specifies an expression that uses column values from more than one table.

view (0 or more instances)

A view represents a join of tables. It is like a table in most respects, but it is unavailable for adds, modifies or deletes. In some regimes, a view can be specified directly in the view definition; in others (for example, Oracle9i), a view must be pre-configured into the database.

2.8.3.8. Table Element

A table always relates to a specific table in the underlying relational database.

Normally, a table has a defined column that supplies a primary key (usually auto-incrementing, for example, a steadily increasing row-id, but sometimes user-defined); sometimes multiple keys are used (for example, surname and given-name). This configuration provides a unique internal reference (the database may be able to police uniqueness of user-supplied keys). A user of the agent can still specify primary keys independent of the table configuration, provided that this is indeed guaranteed to provide a unique reference. If it does not, the specific rows in the sets that share a key cannot be

modified individually using this key or set of keys.

Abbreviations recorded in "keys" (and primary-keys) are not alternatives, but must normally be used together. In other words, if the abbreviations are "cn" and "gn", then both values should be supplied: cn=fred+gn=jones. Rows can sometimes be accessed by means of a subset of the keys, but sometimes they cannot (depending on whether the result is unique).

For external purposes, specifying a different key is sometimes useful. For example, an entry from an LDAP directory system may be required to have a name corresponding to surname/given-name, while the primary key may be a number that is meaningless outside the database. For this reason, two forms of key are configurable - one for use with the database, and one for external use.

In some systems, the identity of the primary key can be obtained by the JDBC connector by accessing the table metadata. However, this is not always provided as a facility within the JDBC access to the database. If not available from configuration information, the agent assumes that the auto-incrementing columns of the table are the primary keys. If this is not possible, there is an error.

2.8.3.8.1. Attributes

The configured element has the following attributes:

keys
: An optional space-delimited list of abbreviations to be used as names useful to external resources. If absent, configured or evaluated primary keys are used.

primary-keys
: An optional space-limited list of abbreviations that correspond to primary keys as defined within the database. This is not required if (A) the Agent can obtain the information from schema metadata or (B) the table's primary key(s) is/are auto-incrementing, and schema metadata can identify auto-incrementing columns.

The Agent will report an error if it is unable to determine a primary key.

2.8.3.8.2. Sub-elements

The configured element has the following sub-elements:

name (always one)
: This is pure text. It must be unique for all tables, but can be shared with a view. It must comply with the rules for schema names. (See section "Schema Names" for details.)

2.8.3.9. View Element

A view represents a join of two or more tables. Joins can only be accessed by the JDBC connector for search operations. The first table within a view is referred to as the primary table.

In some environments (for example, Oracle9i), a view is a database object that is configured in by the database manager. In this case, "from" and "where" information is unneeded and must not be provided. In others (for example, Microsoft ACCESS or SQL Server), the join is explicitly generated in the calling SQL statement. In this case, "from" and "where" information is needed in exactly the same form as required by a manually-generated SELECT for two or more joined tables.

2.8.3.9.1. Attributes

The view element has the following attributes in addition to an optional description attribute:

keys

An optional space-limited list of abbreviations to be used as names. These must represent columns of the first constituent table. If absent, the primary key(s) of the first table listed are used. (This must exist - see notes on "keys" and "primary-keys" under table-elements.)

2.8.3.9.2. Sub-elements

The view element has the following sub-elements:

name (always one)

This is pure text. It must be unique for all views, but can be shared with its primary table. It must comply with the rules for schema names. (See section "Schema Names" for details.)

from

Gives the FROM text needed for explicit views, in SQL. Mandatory when views are explicit. Absent otherwise.

If present, it must be exactly as the database would use it in a SQL statement. Note particularly that conventions for schema names containing spaces or other characters not permitted in basic schema names *may* apply. In this case, use as would be required for normal SQL when accessing the database with a command-line tool.

where

Gives the WHERE text needed for explicit views, in SQL. Mandatory when views are explicit. Absent otherwise.

If present, it must be exactly as the database would use it in an SQL statement.. Note particularly that conventions for schema names containing spaces or other characters not permitted in basic schema names *may* apply. In this case, use as would be required for the normal SQL that would be used when accessing the database with a command-line tool. For example, double quotes may need to be replaced by square brackets for Access databases.

table

Required to record all the tables specified in the Join. The first table must be the primary table.

In the case where the join is a self-join, the second instance of table must have been declared with a new name in an AS substatement in the WHERE clause used to define the join; this table must also be declared here.

For example:

```
<from>employees INNER JOIN Employees AS employeesAsBoss ON
employees.ReportsTo = employeesAsBoss.EmployeeID</from>
```

requires also:

```
<table> <name>employeesAsBoss</name></table>
```

This permits the table to be used in an abbreviation definition, even though not specified with the main collection of tables.

This table element must contain a subordinate name element, which is text only.

2.8.3.10. Abbreviation

An abbreviation binds a simple externally-accessible name to a column or function defining information from a table or view. Simple abbreviations correspond to a single column. Functional abbreviations represent an expression based zero or more columns, perhaps from multiple tables. Thus the standard LDAP attribute name "sn" (meaning surname) could be bound in an abbreviation to the column:

Employees.LastName

meaning the column LastName in the table Employees.

Column values are only accessible when specified by an abbreviation that has been configured in the configuration file.

Abbreviations are checked when the agent is started, and should normally resolve to a table/column combination (except when an expression is specified).

"Abbreviation" is a term that corresponds closely to a textual DSML attribute descriptor (or an LDAP attribute identifier). Note that the latter two can be used in dotted-integer OID notation (for example, "2.5.5.1"); however, this facility is not supported by the JDBC connector.

DSML specifies a syntax for names; the textual form (as opposed to dotted decimal form is used for abbreviations.

Thus, an abbreviation has a name that must start with an ordinary alphabetic; thereafter, each letter in the name must be:

An alphabetic

A numeric

A hyphen

Abbreviation names are case-insensitive.

An abbreviation is more restrictive than a DSML attribute descriptor in corresponds to a unique column or expression in a database. Thus, in the case given earlier, "sn" cannot be used to represent a LastName column in any other table than Employees. This specific nature is exploited in the agent in that, since the abbreviation definition often identifies the holding table, it is often not necessary to specify the required table explicitly.

The abbreviation maps to a value defined by the text content of the element, initial and trailing spaces being discarded. This value could be:

A column name (when the table-name is unambiguous, because only one table has been declared)

A table and column name;

An SQL value expression identified as a quasi-column by an “... AS name” suffix.

```
<abbreviation name="exp">RTRIM(dbo.accounts.dxrAccountName) AS
exp</abbreviation>
```

Defines that the RTRIM function is used to eliminate trailing blanks of the column dxrAccountName

2.8.3.10.1. Attributes

Abbreviation elements can have the following attributes in addition to an optional description attribute:

format

This attribute is in a form that specifies the inner syntax of a string value (for example, for timestamps). Further details are given under section "Format Codes".

max-size

An optional integer attribute that specifies the maximum size of a textual attribute value. The objective of this is to cause truncation when outgoing data is longer then the specified value. It applies only to string or text attributes.

min-size

An optional integer attribute that specifies the minimum size of a textual attribute value. The objective of this is to cause an error when data intended for the database is shorter then the specified value. It applies only to string or text attributes.

name

The name of the abbreviation. Mandatory, and must be unique for all abbreviations. It must comply with the rules for textual DSML Attribute Descriptors.

Abbreviations have pure text content. This text specifies the definition of the abbreviation in SQL terms. The recommended form for ordinary column-value abbreviations is:

<table-name>.<column-name>

where double quotes are used as necessary.

2.8.3.10.2. Format Codes

The format parameter is available to assist in the parsing of incoming values that are potentially cultural or locale dependent. At present, date-time is affected

Date-Time Formats

The abbreviation format field general date format which can then be used for parsing or string synthesis. Incoming dates are parsed in accordance with the format, and passed to the database (if appropriate) in a form compliant with JDBC standards. Similarly, outgoing dates are used to synthesize information in a defined manner.

Default format is "YYYYMMDDhhmmssZ". If running in lite mode only this format is supported.

ACCESS over the ODBC/JDBC bridge does not apparently support date comparison in predicates.

An example of the format string is:

"dd/MM of YYYY (hh:mm:ss GMT)"

In the string, the following are considered special:

"yy" or "YY" - indicates a two-digit year,

"yyyy" or "YYYY" - indicates a four-digit year,

"M" - indicates a one- or two-digit month,

"MM" - indicates a two-digit month,

"d" or "D" - indicates a one- or two-digit day,

"hh" or "HH" - indicates a two-digit hour,

"mm" - indicates a two-digit minute,

"ss" or "SS" - indicates a two-digit second,

"f", "ff" , ... , "fffffffff" indicates fractions (number of decimal precision) of a TIMESTAMP

They mark the expected position of the indicated field. All other combinations are ignored. Care should be taken when additional characters contain any of the characters used in the special string. Thus, the word "Immediate" will not work as desired, since the two **m**s indicate an expected minute field. Year, month and day fields must be provided, and no field can be provided more than once. Fields can be adjacent, but if "D" "d" or "M" are immediately followed by another field, they are taken as the same as "DD", "dd" or "MM". Future extensions may give non-numerical (locale-dependent) dates.

For TIMESTAMP columns also fractions are allowed. Define the fractions in the format string of the abbreviation. Use **f** for fractions.

fff means 3 fractions (milliseconds format="YYYY-MM-DD hh:mm:ss.fff"). Up to 9 fractions are supported (9 decimal places of precision - nanoseconds).

The input data values must match the format string. Output data values are presented as defined in the format string.

You may use less fractions in format string as defined in the DDL of your column. During add / modify missing fractions are handled as 0.

If you have defined 6 fractions in your database column (microseconds), but you are using a format with **fff** (milliseconds), passing 123 as a fraction will be handled as 123000 microseconds.

2.8.3.10.3. Abbreviations and Data Types

Data types are at the heart of relational databases. Each database will define a range of data types that it is prepared to accept. To store any other kind of value, it would be necessary to map it into an existing data type (for example, a string), but, generally, the rule is that a column (represented by an abbreviation) has a predefined data type, and will only accept values of that data type (with minor variances, such as date-format). There could also be truncation (for example, of places in a floating-point format).

JDBC provides access to all normal data types (and to some unusual ones as well). However, the JDBC connector (while supporting the normal datatypes), does not support every JDBC-supported data type. The following list indicates the datatypes not supported.

Types.ARRAY

Types.BLOB *)

Types.CLOB

Types.DATALINK

Types.DISTINCT

Types.JAVA_OBJECT

Types.OTHER

Types.REF

Types.STRUCT

*) *BLOB:*

> OracleOverOracleDriver supports BLOB in the following way:
>
> BLOB input data must be of type binary.
>
> BLOB data is returned as binary data. The length is limited to 2,147,483,647 bytes(MAXINT). BLOB data exceeding this size is ignored

> (no value for this column is returned). A warning is generated.
>
> As BLOB data is handled as binary data it is transferred in SPML requests/responses as byte array. This may lead to high memory usage.

The following table indicates the standard data types and their standard mappings to Java classes:

ARRAY	BIGINT	BINARY	BIT
BLOB *)	BOOLEAN	CHAR	CLOB
DATALINK	DATE - deprecated	DECIMAL	DISTINCT
DOUBLE	FLOAT	INTEGER	JAVA_OBJECT
LONGVARBINARY	LONGVARCHAR	NUMERIC	OTHER
REAL	REF	SMALLINT	STRUCT
TIME - deprecated	TIMESTAMP	TINYINT	VARBINARY
VARCHAR	NCHAR	NULL	NVARCHAR

To determine whether a particular database requires a particular JDBC data type, an empirical approach usually suffices.

Each database defines its own data type names, but a table of mappings can sometimes be valuable. Here, for example, is a list of all the SQL Server data type names and the corresponding JDBC data type names. As you can see, all data types accessible to the server are supported:

SQL Server Data Type Typename	JDBC Data Type Value
bigint	BIGINT
binary	BINARY
bit	BIT
char	CHAR
datetime	DATETIME
decimal	DECIMAL
float	FLOAT
image	LONGVARBINARY
int	INTEGER
money	DECIMAL
nchar	CHAR NCHAR (2008 and higher)
numeric	NUMERIC
nvarchar	VARCHAR NVARCHAR (2008 and higher)

SQL Server Data Type Typename	JDBC Data Type Value
real	REAL
smalldatetime	TIMESTAMP
smallint	SMALLINT
smallmoney	DECIMAL
sql_variant	VARCHAR
text	LONGVARCHAR
timestamp	BINARY
tinyint	TINYINT
unique-identifier	CHAR
varbinary	VARBINARY
varchar	VARCHAR

2.8.3.11. Relationship Element

A relationship element specifies references from one table to another for which referential integrity enforcement (if implemented) can be handled by nullifying the reference. Use this element field to permit entries to be deleted when entries in other tables affected by referential integrity point to them. It is used in conjunction with the "always-follow-references" flag in the JDBC-Conection element (see section "JDBC-Connection Element"), each time a relevant row is deleted, the Agent attempts to null all configured pointers that would otherwise no longer point to a row.

In order for the JDBC connector to make use of this function:

- The reference that points to the entry to be removed must be nullifiable
- The access control that permits the JDBC connector to nullify the reference must be in force.

2.8.3.11.1. Attributes

The relationship element has the following attributes:

from

An abbreviation that specifies the column in the table that contains a reference; this is the table that is affected by referential integrity. It is never a primary key

referring-to

An abbreviation that specifies the column in the table that supplies the value of the reference. It is always a primary key - in fact, it must correspond to the single primary key for the referenced table.

2.8.3.12. Functions-and-Procedures Element

Functions and procedures are similar, except that a function returns a value and a

procedure does not. Both functions and procedures theoretically have arguments that are IN, OUT, or IN-OUT. However, some regimes may be more restrictive (e.g. to forbid function arguments to be OUT or IN-OUT).

Functions and procedures have distinct names, so that a function and a procedure cannot share a name.

2.8.3.12.1. Attributes

A functions-and-procedures element has an optional description attribute.

2.8.3.12.2. Sub-elements

A functions-and-procedures element has the following sub-elements:

function
: An optional declaration of a stored procedure that returns a value;

procedure
: An optional declaration of a stored procedure that does not return a value;

2.8.3.12.3. Returned Values

The JDBC connector requires that the function or procedure must return an integer result, representing status. This status is then converted in a customized way into either an indication of success, or an indication of failure.

Although this seems restrictive, functions and procedures can be nested inside each other, so that an arbitrary function or procedure (or a whole set of functions/procedures) can be encapsulated in a function or procedure that has the required characteristic.

Stored procedures cannot at present be actually created by the JDBC connector. They must be defined within the database by an appropriate graphic or command-line tool.

2.8.3.13. Function Element

This section describes the attributes and sub-elements of the function element.

2.8.3.13.1. Attributes

Functions can have the following attributes (other than an optional description):

name
: The name of the function, as presented to the JDBC interface. Mandatory, and must be unique for all functions and procedures

2.8.3.13.2. Sub-elements

Functions can have the following sub-elements:

argument

Optional. Defines incoming (and potentially outgoing) data.

return

Mandatory. Controls the processing of the returned information, but does not specify which argument is involved.

2.8.3.14. Procedure Element

This section describes the attributes and sub-elements of the procedure element.

2.8.3.14.1. Attributes

Procedures can have the following attributes (other than an optional description):

name

The name of the procedure, as presented to the JDBC interface. Mandatory, and must be unique for all functions and procedures

2.8.3.14.2. Sub-elements

Procedures can have the following sub-elements:

argument

Mandatory. Defines incoming (and potentially outgoing) data - one must be an OUT or IN-OUT

return

Mandatory. Controls the processing of the returned information, selecting the argument which is to provide the returned value.

2.8.3.15. Argument Element

This section describes the attributes of the argument element.

2.8.3.15.1. Attributes

The argument element for the JDBC connector has the following attributes, as well as an optional description:

in-out

Text of form "IN" "OUT" or "IN-OUT". Default: IN, if not specified.

data-type

Defines the way in which the argument is to be interpreted.

format

This attribute is in a form that specifies the inner syntax of a string value (for example, for timestamps). (See "Format Codes" in "Abbreviation" for details.) This is mandatory for all time-related data types like TIMESTAMP.

2.8.3.16. Return Element

Functions and procedures that are used by the JDBC connector must return an integer value that indicates success, failure, or other outcome. The return elements enable this value to be translated into:

The category of error

Text corresponding to the detailed value.

2.8.3.16.1. Attributes

The return element can have the following attribute:

name
: For a procedure, this must be the configured name of an argument. It is always absent for functions.

2.8.3.16.2. Sub-elements

A return element contains the following sub-elements:

range
: These sub-elements are used to map the returned integer value into a category and a report string.

  The returned value is matched against each range in order of configuration, and the indications provided by the range are taken as the basis for the response made by the JDBC connector.

  There is an error response if no range matches.

2.8.3.17. Range Element

This section describes the attributes of the range element.

2.8.3.17.1. Attributes

The applicability of a specific range to an integer return is determined by:

max + min
: These occur in pairs; if present, the integer return is matched if not exceeding max and not less than min.

exact
: Present if and only if max and min are absent. If present, the integer return is matched if equal to this value.

The translation of a return value is made in terms of the following attributes:

value
: The optional string return for a value within the specified range (default "")

category

The severity of the return: If present, this must take one of the following values (ignoring case):

"OK"

"INFO

"WARNING

"ERROR

The default is "ERROR" if a value attribute is present, otherwise "OK".

2.8.4. Input and Output Data File Formats

The JDBC connector accepts different file formats for input data:

- SPML request
- LDIF change

Similarly, the agent produces the following different output formats:

- SPML response
- LDIF content

The format must be configured in the connector sections, which refer to the reader and the responseWriter.

The following sample configuration snippet defines LDIF-change input and SPML response output:

```
<connector
className="siemens.dxm.connector.framework.LdifChangeReader"
    name="LDIF change file reader" role="reader">
  <connection filename="datain.ldif" type="LDIF change" />
</connector>
<connector className="siemens.dxm.connector.framework.SpmlFileWriter"
     name="SPML File writer" role="responseWriter">
  <connection filename="dataout.xml" type="SPML" />
</connector>
```

For an export, the agent waits for the search request definition in an SPML file. Here is a sample:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Created on Tue Jul 06 10:35:10 BST 2004-->
```

```
<searchRequest requestID="search8"
xmlns="urn:oasis:names:tc:SPML:1:0">
  <searchBase type="urn:oasis:names:tc:SPML:1:0#DN">
    <spml:id>table=employees</spml:id>
  </searchBase>
  <filter>
    <greaterOrEqual name="bd">
      <value>1/1/1963</value>
    </greaterOrEqual>
  </filter>
  <attributes>
    <attribute name="sn"/>
    <attribute name="bd"/>
  /attributes>
</searchRequest>
```

The search results are output in LDIF content format. Here is a sample for the appropriate connector section in the configuration file:

```
<connector className="siemens.dxm.connector.framework.LdifFileWriter"
     name="LDIF File writer" role="responseWriter">
  <connection filename="dataout.ldif" type="LDIF" />
</connector>
```

Even if the input or output is of LDIF format, the agent internally works with SPML. Transformation is done automatically by the LDIF reader and writer.

The following subsections describe the content of the internally-handled input requests and output responses, which are in strict compliance with SPML requirements. You also need to know this format if you include transformation in your job: use a requestTransformer or responseTransformer section in your configuration.

The four main SPML operations are add, modify, delete, and select. The action is based on the supply of an identification, with the exception of add, which can optionally create a new entry based on new contents.

In addition, the extended-request SPML operation is used for function/procedure calls.

In all cases, the agent determines on which table to operate and then applies the supplied SPML information appropriately.

2.8.4.1. Add, Modify, Delete and Search Requests

The four defined operations add, modify, delete and search are extensions of SPMLRequest. Here are some simple examples. The first is for a database that maintains JDBC connector

characteristics:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Created on Mon Jun 21 09:45:07 BST 2004-->
<addRequest requestID="add1" xmlns="urn:oasis:names:tc:SPML:1:0">
  <attributes>
    <attr name="abbname">
      <value>boss</value>
    </attr>
    <attr name="abbmap">
      <value>employeesAsBoss.FirstName &amp; ' ' &amp;
employeesAsBoss.LastName AS Boss</value>
    </attr>
    <attr name="abbprice">
      <value>1</value>
    </attr>
    <attr name="abbimp">
      <value>false</value>
    </attr>
  </attributes>
</addRequest>
```

This is a modify for a personnel-style database:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Created on Mon Jun 21 09:45:11 BST 2004-->
<modifyRequest requestID="modify4"
xmlns="urn:oasis:names:tc:SPML:1:0">
  <identifier type="urn:oasis:names:tc:SPML:1:0#DN">
    <spml:id>sn=davolio</spml:id>
  </identifier>
  <modifications>
    <modification name="id" operation="delete"/>
  </modifications>
</modifyRequest>
```

This relates to the same database as for the add:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Created on Mon Jun 21 09:45:09 BST 2004-->
```

```
<deleteRequest requestID="delete2"
xmlns="urn:oasis:names:tc:SPML:1:0">
  <identifier type="urn:oasis:names:tc:SPML:1:0#DN">
    <spml:id>abbprice=2,table="abbreviation data and price"</spml:id>
  </identifier>
</deleteRequest>
<?xml version="1.0" encoding="UTF-8"?>
 <!-- Created on Mon Jun 21 09:45:13 BST 2004-->
 <searchRequest requestID="search2"
xmlns="urn:oasis:names:tc:SPML:1:0">
   <searchBase type="urn:oasis:names:tc:SPML:1:0#DN">
     <spml:id>table=employees</spml:id>
   </searchBase>
   <attributes>
     <attribute name="id"/>
     <attribute name="gn"/>
     <attribute name="sn"/>
   </attributes>
 </searchRequest>
```

All operations have an optional reqestID. The reqestID is always repeated in any response.

Possible sub-elements for these operations are:

- identifier (applicable to add, modify, delete only)
- searchBase (applicable to search only)
- modifications (applicable to modify only)
- filter (applicable to search only) - specifies the search filter in SPML syntax. ApproximateMatch and ExtensibleMatch are not supported.
- attributes - as types and values (applicable to add only)
- attributes - as a list of types (applicable to search only)
- operationalAttributes - sortAttribute, sortOrder and pageSize are supported for searches
- any - unused
- requested - unused
- execution - unused

The usage of the protocol is as follows:

	add	modify	delete	search
identifier	Optional. If present, points to the name of the new entry, supplying attribute values	Mandatory. Defines the entry to be modified.	Mandatory. Defines the entry to be deleted	
searchBase				Optional. If present, defines what is to be searched. Defaults to definition by attribute selection
modifications		Mandatory. Supplies modifications to be applied		
filter				Optional. Defines the entries of interest.
attributes (types and values)	Optional. Supplies the main attributes to be added			
attributes				Optional. List of attribute descriptors that are to be returned - default all

2.8.4.2. Sorting

Sorting can be specified with the operational attributes sortAttribute and sortOrder. If no sortOrder is given, ASCENDING is assumed. Multiple sortAttributes are supported.

In the following example, sorting is specified first **ASCENDING** for attribute **JOB** and second **DESCENDING** for attribute **ENAME:**

```
<spml:searchBase type="urn:oasis:names:tc:SPML:1:0#DN">
     <spml:id>table=SCOTT.EMP</spml:id>
</spml:searchBase>
<spml:operationalAttributes>
```

```
     <dsml:attr name="sortAttribute">
            <value>JOB</value>
            <value>ENAME</value>
       </dsml:attr>
        <dsml:attr name="sortOrder">
            <value>ASCENDING</value>
            <value>DESCENDING</value>
     </dsml:attr>
</spml:operationalAttributes>
```

2.8.4.3. Paging

Paging can be configured with the operational attribute pageSize. The given pageSize is used for setting the "fetchSize" of the SelectStatement. If paging is configured, the SPML response does not include the whole result set to minimize memory consumption. For database-specific optimization, see the DB /JDBC documentation. (For example, SQL Servers offers responseBuffering=adaptive.)

In the following example, paging is specified with pagesize 5:

```
<spml:searchBase type="urn:oasis:names:tc:SPML:1:0#DN">
         <spml:id>table=SCOTT.EMP</spml:id>
    </spml:searchBase>
    <spml:operationalAttributes>
         <dsml:attr name="pageSize">
              <value>5</value>
         </dsml:attr>
    </spml:operationalAttributes>
```

2.8.4.4. Names within Identifier and Search-base Elements

Names as used by the connector within identifier and search-base elements are always represented in XML as follows:

```
<identifier type="urn:oasis:names:tc:SPML:1:0#DN">
    <spml:id>dn</spml:id>
</identifier>
```

where dn represents the LDAP DN form specified in RFC2253. This is the only SPML-specified type that is supported.

The name used for a search operation is known as the "search-base", and can correspond to a table or to a specific entry (row).

Four DN forms are recognized by the connector:

No.	Description	Signifying	Example(s)
2	Empty string (the "root" name)	Default table	empty or blanks
3	A name comprising a single RDN built out of keys (one or more attributes).	A row matching these values	sn=Doe,gn=John
4	A name comprising a single DN built from the quasi-attributes: *table* and *view*	A table matching the name (including double quotes)	view=employees table="external contractors"
5	A two-RDN name combining keys with *table* or *view*.	A row matching these values	sn=Hodson,table=suppliers gn=Anthony+sn=Hodson,table=suppliers

The quasi-attributes table and view are used with values that are exactly as required for the table or view. For these, values without double-quotes are case-insensitive and are very restricted in the characters that they can use, and values with double-quotes are taken as case-sensitive, and can include spaces and other characters.

These restrictions do not apply to other attributes.

The following table defines the use of the four forms described above by JDBC connector operations:

	add	delete	modify	search
1 - empty	forbidden	forbidden	forbidden	OK (default table implied)
2 - keys	OK	OK	OK	OK
3 - table or view	table is OK view is not OK	Forbidden (this would drop a table or view, which is forbidden).	Forbidden (you can only modify a row, not a table or view).	OK
4 - keys and table	OK	OK	OK	OK

The name itself must be compatible with any other attributes defined within the specific operation (i.e. represent a table, view, row or join of rows), within which the required attributes can be found. The following additional rules exist for identification purposes:

	add	delete	modify	search
1 - empty				
2 - atts	Must not define any existing row	Must define a single row	Must define a single row	May define one or more rows
3 - table or view	Must define a configured table			Must define a configured table or view
4 - atts and table	Must not define any existing row	Must define a single row	Must define a single row	May define one or more rows

2.8.4.5. Add, Modify, Delete, and Search Responses

The following table defines responses with success:

	add	modify	delete	search
result:	"urn:oasis:names:tc:SPML:1:0#success"			
requestID	**returned if supplied**			
identifier	Always returned based on primary key			
attributes	unused			
modifications		unused		
searchResultEntry				Returns names and values for matching entries
operational attributes	unused			
error Message				
any	unused			

The following table defines responses with failure:

	add	modify	delete	search
result:	"urn:oasis:names:tc:SPML:1:0#failure"			

	add	modify	delete	search
requestID	returned if supplied			
error	absent OR "urn:oasis:names:tc:SPML:1:0#unsupportedIdentifierType" OR "urn:oasis:names:tc:SPML:1:0#noSuchIdentifier"			
operational attributes	unused			
error Message	absent OR synthesized using Reports substitutions			
any	unused			

Examples:

```
<AddResponse result="urn:oasis:names:tc:SPML:1:0#success"
requestID="add7" xmlns="urn:oasis:names:tc:SPML:1:0">
  <identifier type="urn:oasis:names:tc:SPML:1:0#DN">
    <id>
      abbid=7,table=\&quot;abbreviation data and price\&quot;
    </id>
  </identifier>
</AddResponse>
<DeleteResponse result="urn:oasis:names:tc:SPML:1:0#success"
requestID="delete1" xmlns="urn:oasis:names:tc:SPML:1:0"/>
<SearchResponse result="urn:oasis:names:tc:SPML:1:0#failure"
requestID="search99" xmlns="urn:oasis:names:tc:SPML:1:0">
  <errorMessage>
    search error:entry not found; dn=sn=davo\+lio+gn=nancy\,
  </errorMessage>
</SearchResponse>
```

2.8.4.6. Stored Functions and Procedures

This section describes the operations for stored functions and procedures.

2.8.4.6.1. extendedRequest Elements

Stored functions and procedures are mapped to extendedRequest SPML operations. The form of these items differs from the other operations described earlier in this document.

The components of an extendedRequest as used by a functions and procedures are illustrated in the following example:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Created on Mon Jun 21 09:45:15 BST 2004-->
<extendedRequest requestID="spin00_3"
xmlns="urn:oasis:names:tc:SPML:1:0">
  <providerIdentifier
providerIDType="urn:oasis:names:tc:SPML:1:0#URN">
    <providerID>SP</providerID>
  </providerIdentifier>
  <operationIdentifier
operationIDType="urn:oasis:names:tc:SPML:1:0#GenericString">
    <operationID>sp_benutzerrolle</operationID>
  </operationIdentifier>
  <attributes>
    <attr name="operation">
      <value>2</value>
    </attr>
    <attr name="rollenname">
      <value>fred</value>
    </attr>
    <attr name="mitarbeiter">
      <value>tom</value>
    </attr>
  </attributes>
</extendedRequest>
```

Items which are in bold represent user-supplied information:

- text value of operationID as the name of the function
- attributes elements
  - value of name attribute
  - text value of value sub-element

The attributes are used to map to the arguments of a pre-stored procedure definition. The names of the arguments are used in the same way that abbreviations are used for normal operations. Normal abbreviations are unused.

The return value can either be success or a failure with diagnostics, or could take a processed value; and the "attributes" element of the extendedResponse (part of the extension of the SPML response) could provide the other return values.

The process for stored functions and procedures is:

- The values supplied as attributes are applied to the appropriate arguments, if present. If absent, NULL values are used.
- On successful return of a function, the returned value is matched to a range provided by conversion. On successful return of a procedure with a defined return, the returned value is similarly handled.
- If the return is empty, or is designated as OK, info, or warning, the extendedResponse indicates success.
- Otherwise, the extendedResponse indicates failure
- All values of OUT or IN/OUT arguments including the returned value are returned as attribute values.

The returned value for the function or procedure must always be an integer.

2.8.4.6.2. extendedResponse Element

An example of a successful response is:

```
<ExtendedResponse result="urn:oasis:names:tc:SPML:1:0#success"
requestID="spin00_1" xmlns="urn:oasis:names:tc:SPML:1:0">
  <attributes>
....<attr name="return-message">
......<ns1:value xmlns:ns1="urn:oasis:names:tc:DSML:2:0:core">
        OK
......</ns1:value>
....</attr>
....<attr name="return-value">
......<ns2:value xmlns:ns2="urn:oasis:names:tc:DSML:2:0:core">
........11
......</ns2:value>
....</attr>
..</attributes>
</ExtendedResponse>
```

Elements of successful responses are given in the following table:

	Stored Functions and Procedures	
result:	"urn:oasis:names:tc:SPML:1:0#success"	
requestID	returned if supplied	
attributes	Contains return values, encoded in terms of attribute names. There will always be at least one of these. For a function, the return value will have a name "function-return-value"	
	"return-message"	present if supplied as a result of range in one of the following forms: OK: *message* INFO: *message* WARNING: *message*
	return-value	always supplied - always an integer
	argument-names ...	argument-values ...
operational attributes	unused	
error Message		
any	unused	

Elements of failed responses are given in the following table:

	Stored Functions and Procedures	
result	"urn:oasis:names:tc:SPML:1:0#failure"	
requestID	returned if supplied	
error	absent	
attributes	Contains return values, encoded in terms of attribute names. Only supplied if the function or procedure successfully executed, but detected an error within its own processing.	
	"return-message"	present if supplied as a result of range in this form Error: *message*

	Stored Functions and Procedures	
	"return-value"	always supplied - always an integer
	argument-names ...	argument-values ...
operational attributes	unused	
error Message	Present if no attributes are available. Synthesized using Reports substitutions	
any	unused	

2.8.5. Error Handling

This section describes JDBC connector error handling, including:

- Generated error log files
- Error-handling procedures

2.8.5.1. Error Log Files (JDBC Connector)

Errors are logged in a system log file provided outside the scope of the JDBC connector.

Errors are also optionally logged in a local log file whose name is derived from the configuration file attribute:

job.connector.logging.filename

For example, with a value "JDBCLogger", the file name may be:

JDBCLogger.000.log

The level of logging is set by the levels set for each of the log files. But note that the information provided by the system log file is no more extensive than that made available by the level set for local logging (whether or not a local log file is provided).

2.8.5.2. Error-Handling Procedures

Configuration errors are normally fatal.

Operation errors usually cause the operation to fail, but do not stop the connector.

Failed operations cause an error response, which carries a single message representing a failure. Logged messages can contain indications of multiple error events.

The language of errors depends on resource files, which change the language of textual messages but do not affect tags that represent the name of XML elements.

2.9. LDAP Connector

The LDAP connector implements the DirX Identity Java Connector Integration Framework's **DxmConnector** interface and connects to an LDAP server through the Netscape LDAP interface. It can be used for Tcl-based workflows in the C++-based Server and realtime workflows in the Java-based (IdS-J) Server. Like all framework-based agents, it gets SPML requests from the Identity side and converts them to the appropriate Netscape LDAP interfaces on the LDAP server side and vice versa.

2.9.1. Overview

The connector implements the API methods "add(...)", "modify(...)", "delete(...)" and "search(...)". They represent the corresponding SPML requests "AddRequest", "ModifyRequest", "DeleteRequest" and "SearchRequest".

2.9.2. Request and Response Handling

This section describes the supported requests and attributes for the LDAP connector.

2.9.2.1. AddRequest

In an add request, the identifier, which is always expected to be a DN, is mandatory. All object classes and all attributes contained in the schema of the LDAP server can be passed in the add request.

Example request:

```
<?xml version="1.0" encoding="UTF-8" ?>
<spml:batchRequest xmlns="urn:oasis:names:tc:SPML:1:0"
  xmlns:spml="urn:oasis:names:tc:SPML:1:0"
  xmlns:dsml="urn:oasis:names:tc:DSML:2:0:core"
 requestID="batch-1"
  processing="urn:oasis:names:tc:SPML:1:0#sequential"
 execution="urn:oasis:names:tc:SPML:1:0#synchronous"
  onError="urn:oasis:names:tc:SPML:1:0#exit">
  <spml:addRequest requestID="add-1">
     <spml:identifier type="urn:oasis:names:tc:SPML:1:0#DN">
        <spml:id>cn=my class,cn=supplier groups,cn=groups,cn=Extranet
Portal,cn=TargetSystems,cn=my-company</spml:id>
     </spml:identifier>
     <spml:attributes>
        <spml:attr name="objectclass"
xmlns="urn:oasis:names:tc:DSML:2:0:core">
            <dsml:value>dxrTargetSystemGroup</dsml:value>
        </spml:attr>
```

```
        <spml:attr name="uniqueMember"
xmlns="urn:oasis:names:tc:DSML:2:0:core">
           <dsml:value>cn=my-company</dsml:value>
        </spml:attr>
        <spml:attr name="dxrState"
xmlns="urn:oasis:names:tc:DSML:2:0:core">
           <dsml:value>ENABLED</dsml:value>
        </spml:attr>
    </spml:attributes>
  </spml:addRequest>
</spml:batchRequest>
```

2.9.2.2. ModifyRequest

In a modify request, the identifier is also mandatory. All object classes and their attributes contained in the schema of the LDAP server can be modified.

Example request:

```
<?xml version="1.0" encoding="UTF-8" ?>
<spml:batchRequest xmlns="urn:oasis:names:tc:SPML:1:0"
xmlns:spml="urn:oasis:names:tc:SPML:1:0"
xmlns:dsml="urn:oasis:names:tc:DSML:2:0:core"
requestID="batch-1"
processing="urn:oasis:names:tc:SPML:1:0#sequential"
execution="urn:oasis:names:tc:SPML:1:0#synchronous"
onError="urn:oasis:names:tc:SPML:1:0#exit">
<spml:modifyRequest requestID="mod-2">
<spml:identifier
  type = "urn:oasis:names:tc:SPML:1:0#DN">
  <spml:id>cn=standard class,cn=supplier groups,cn=groups,cn=Extranet
Portal,cn=TargetSystems,cn=my-company</spml:id>
</spml:identifier>
<spml:modifications>
  <spml:modification name="dxrGroupMemberAdd" operation="delete">
     <dsml:value>cn=YYYJimmy Sails
23e,ou=accounts,ou=extranet,o=sample-ts</dsml:value>
  </spml:modification>
  <spml:modification name="dxrGroupMemberAdd" operation="add">
     <dsml:value>cn=XXXJimmy Sails
23e,ou=accounts,ou=extranet,o=sample-ts</dsml:value>
```

```
     <dsml:value>cn=Jimmy Sails 23e,ou=accounts,ou=extranet,o=sample-
ts</dsml:value>
  </spml:modification>
</spml:modifications>
</spml:modifyRequest>
</spml:batchRequest>
```

Rename/Move Functionality

If the operational attributes of the modify request contain the attribute **dxrPrimaryKeyOld** the object in the LDAP system is renamed or moved from the position represented by the DN value of dxrPrimaryKeyOld to the position of the DN value passed with the identifier. If only the RDN part of the DNs are different it is a rename, if other parts of the DNs differ, for example an OU, a move operation is performed.

Usually, the check for the last RDN (rename) is case insensitive. So, you are not able to change, for example, the ou=RedFlag to ou=Redflag. If you want to enable such a rename (case sensitive), you must provide the operational attribute **caseExactRDNComparison** with the value **true** in your modify request. Here is a sample request (ou=RedFlag → ou=Redflag):

```
<spml:modifyRequest requestID="mod-2">
<spml:identifier
    type = "urn:oasis:names:tc:SPML:1:0#DN">
    <spml:id>ou=Redflag,cn=Custom,cn=BusinessObjects,cn=My-
Company</spml:id>
</spml:identifier>
   <spml:operationalAttributes>
       <spml:attr name="dxrPrimaryKeyOld">
           <dsml:value
type="string">ou=RedFlag,cn=Custom,cn=BusinessObjects,cn=My-
Company</dsml:value>
       </spml:attr>
       <spml:attr name="caseExactRDNComparison">
           <dsml:value type="string">true</dsml:value>
       </spml:attr>
   </spml:operationalAttributes>
<spml:modifications>
    <spml:modification name="description" operation="replace">
           <dsml:value>erster modify</dsml:value>
    </spml:modification>
</spml:modifications>
```

```
</spml:modifyRequest>
```

Single Modification Processing

If multiple values are to be deleted or added within a modification request - like in the sample request above - and the request fails with one of the following LDAP error codes in the situations previously described, the LDAP connector by default performs single modifications for each value and then logs the values and operations that failed.

LDAP error codes and situations resulting in single modifications:

- ATTRIBUTE_OR_VALUE_EXISTS (add dn value or value that already exists in DirX; add value that already exists in AD)
- NO_SUCH_ATTRIBUTE (delete dn value or value that does not exist in DirX; delete value that does not exist in AD)
- ENTRY_ALREADY_EXISTS (add dn value that already exists in AD)
- UNWILLING_TO_PERFORM (delete dn value that does not exist in AD)
- NO_SUCH_OBJECT (add dn value for object that does not exists in AD)
  This is the situation when members are tried to be added to a group in Active Directory, but the member objects do not exist yet in the directory.

Single modification processing can be turned off with the connection section property **perform_single_mod**. If not specified it is turned on. Since the ADS connector is derived from the LDAP connector this property can also be specified in an ADS connector's connection section.

LDAP Relaxed Update Control

If an LDAP server supports the LDAP Relaxed Update Control - the DirX LDAP server supports it since version 8.1B - the LDAP connector uses this control when performing modifications. The LDAP server returns **SUCCESS** in the **ATTRIBUTE_OR_VALUE_EXISTS** or **NO_SUCH_ATTRIBUTE** situations described above. In those cases, the LDAP connector no longer performs the single modifications itself because a **SUCCESS** code is returned and the LDAP server performs the operation. In all cases, one of the five error codes shown above is returned. The LDAP connector performs single value modifications and logs the failed values.

2.9.2.3. DeleteRequest

In a delete request, the identifier is also mandatory. Any type of object can be deleted. The delete request does not require additional attributes.

2.9.2.4. SearchRequest

In an SPML search request, the LDAP connector supports the standard elements **searchBase** and **filter** and the operational attributes **scope**, **sizeLimit**, **pageSize**, **pagedTimeLimit**, **sortAttribute**, **sortOrder** and **noattrs** (if set to FALSE or not existing either all attributes or the ones specified are retrieved).

Example request:

```
<?xml version="1.0" encoding="UTF-8" ?>
<spml:searchRequest  xmlns="urn:oasis:names:tc:SPML:1:0"
      xmlns:spml="urn:oasis:names:tc:SPML:1:0"
      xmlns:dsml="urn:oasis:names:tc:DSML:2:0:core"
      requestID="search_01"
      >
 <spml:searchBase type = "urn:oasis:names:tc:SPML:1:0#DN">
    <spml:id>ou=Intranet,o=sample-ts</spml:id>
 </spml:searchBase>
      <spml:filter>
            <dsml:present name="objectClass" />
      </spml:filter>
 <spml:operationalAttributes>
    <dsml:attr name="scope">
       <value>subtree</value>
    </dsml:attr>
    <dsml:attr name="sortAttribute">
       <value>cn</value>
    </dsml:attr>
    <dsml:attr name="sortOrder">
       <value>ASCENDING</value>
    </dsml:attr>
    <dsml:attr name="pageSize">
       <value>0</value>
    </dsml:attr>
 </spml:operationalAttributes>
 <spml:attributes>
    <dsml:attribute name="cn"/>
    <dsml:attribute name="sn"/>
     <dsml:attribute name="givenName"/>
     <dsml:attribute name="company"/>
     <dsml:attribute name="mail"/>
     <dsml:attribute name="st"/>
     <dsml:attribute name="street"/>
     <dsml:attribute name="employeeNumber"/>
     <dsml:attribute name="telephoneNumber"/>
 </spml:attributes>
</spml:searchRequest>
```

2.9.3. Configuration

Here is a sample configuration snippet for the LDAP connector (without SSL connection):

```
<connector
role="connector"
className="siemens.dxm.connector.ldap.LdapConnector"
name="Ldap Connector" version="1.00">
<connection type="LDAP"
user="CN=metatest,CN=Users,DC=mydomain,DC=mchp,DC=mycompany,DC=de"
password="XXXyyy###111"
server="myserver"
port="389"
ssl="false"
<property name="debugfile" value="dbgOut.xml"/>
<property name="perform_single_mod" value="true"/>
<property name="check_password_history" value="false"/>
<mvproperty name=”binaryattributes”>
       <value>customer LDAP attribute type 1</value>
       <value>customer LDAP attribute type n</value>
</mvproperty>
</connection>
</connector>
```

2.9.3.1. Supported Connection Parameters

The LDAP connector supports the following properties from the standard properties of the <connection> element of the XML configuration file:

port - the port number of the LDAP server.

server - the server name or IP address of the LDAP server.

user - the user name in DN form used for the connection.

password - the password for this user used for binding to the LDAP server.

ssl - whether (**true**) or not (**false**) to use server-side or client-side SSL/TLS.

For client-side SSL, the following additional parameters are available:

authentication=“CLIENT_SSL”

keystore – the file name of the key store

keystorepassword – the password for accessing the key store

keystorealias - optionally an alias name of the relevant entry in the key store

truststore - the file name of the truststore

truststorepassword – the password for accessing the trust store

Non-standard supported properties are:

debugfile - (optional); if this property is configured, it outputs each received request and response to this file in SPML format.

perform_single_mod - (optional) whether (true) or not (false) single modifications for multi-value attributes in specific erroneous situations are performed. (See the section "Single Modification Processing" for details.)

check_password_history - (optional) whether (true) or not (false) the connector checks password history. If this property is configured and set to true, the LDAP connector first checks whether the connected LDAP server supports the LDAPAdsPolicyHintsControl. If it does, the LDAP connector uses this control on modify operations with the result that password history is checked for both password reset operations and password change operations.

binaryattributes - (optional); specifies a customer specific list of LDAP attribute types.

2.9.4. LDAP SSL Setup

This section describes LDAP SSL setup.

2.9.4.1. Setting up a Server-side SSL Connection to an LDAP Directory

Use the "keytool" of the Java Runtime Environment to import your certificate into the Cert store.

To set up the SSL connection, see the section in "Core Component → Using LDAP → SSL/TLS" in the online help (not available as PDF documentation).

2.9.4.2. Setting up a Client-side SSL Connection to an LDAP Directory

To set up client-side SSL, you must provide a (file based) keystore containing the client's certificate and private key and a (file-based) truststore containing the related CA certificates.

2.9.4.3. Setting up an SSL Connection to the Active Directory Domain Controller (DC)

Note: setting up this connection is not a trivial task and requires knowledge of the Microsoft Windows Active Directory. The following description contains some information from the Microsoft documentation. If you encounter any problems, please refer to the latest Microsoft documentation.

To establish the SSL connection:

2.9.4.3.1. 1. Install a Certificate Authority on your Windows domain controller

Get the Microsoft documentation about "How to Install/Uninstall a Public Key Certificate Authority for Windows" and perform the steps described. Pay attention to the following issues:

During installation:

- Choose to install an enterprise CA (not a stand-alone CA).
- In the Certificate Authority Identifying Information window, you only need to enter the CA Name field: enter the name of your server, either in fully-qualified form, like **kellner13.iam.mycompany.de**, or just the first part, which is the server name short form. This value acts as the CN part of the DN composed automatically by the tool and shown in a field below. Check whether the composed DN contains the correct CN and DC parts conforming to your environment. A certificate containing this name will be created under a filename also containing this name.
- Check the shared folder field and then specify a shared folder under which all configuration information for the CA is stored. Otherwise, all information - including generated root CA certificate - will be stored in Active Directory.

Possible errors during installation:

- If the error message **Provider could not perform the action since the context was acquired as silent. 0x80090022 (-2146893790)** is logged, the cause is the policy **System Cryptography: Force strong key protection for user keys** stored on the computer under **Control Panel\Administrative Tools\Local Security Policy\Local Policies\Security Options**. If you change the default **User must enter a password each time they use a key** to **User input is not required when new keys are stored and used**, the installation runs successfully.

After installation:

- You will find the root CA certificate under the shared folder you specified. This certificate is computer-related. You can copy it to any place in the network file system and import it to the truststore that is used by your client application. With this certificate, your client (in this case, the LDAP connector) can connect to this computer over SSL. If the client wants to establish an SSL connection with another computer in the Windows domain, this computer must be assigned another certificate, which must then also be imported into your truststore. Certificates can be automatically assigned by setting a group policy in Windows.

2.9.4.3.2. 2. Import the certificate into your truststore

This section first describes how to import the certificate for Java clients into a Java truststore.

If you want to establish an SSL connection to the Active Directory Server with the C++-based ADS Agent, you must import the certificate into the Windows certification store; for example, with the Internet Explorer:

Menu **Tools** → **Internet Options** → **Content** → **Certificates** → **Trusted Root Certification** →

Authorities → Import

You must also set the UseEncryption flag in the AdsAdmin bind profile of your Ads Connected Directory and you must specify the full qualified AD server name in the search base for export or in the ADsPath for import.

For Java clients:

Import the root CA certificate created by the CA installation into the truststore used by the LDAP connector (or your Java client) with the **keytool.exe** tool, which is part of the Java Runtime Environment (JRE). The certificate is to be imported to:

- The truststore under the Java Development Kit (JDK) the LDAP connector runs with if it runs in an Integrated Development Environment (IDE) like Eclipse. For example, **D:\java\lib\security\cacerts**.
- The truststore in the directory *dxi_java_home**/lib/security/cacerts* if it runs in the DirX Identity environment under the Java-based Server.

Setting keytool command line parameters:

Depending on the environment the LDAP connector is supposed to run, change to the directory containing the **cacerts** store and copy the certificate file (and **keytool.exe** if you don't want to specify the complete pathname in the command line) to it. Then call the keytool from the command prompt of the directory with the following parameters:

keytool -keystore *storename* **-import -alias** alias__name_ **-file** *certfile_name*

For example:

```
keytool -keystore cacerts -import -alias jupiter_cert -file
jupiter_certorg.crt.
```

You are prompted for the password of the **cacerts** store, which is by default **changeit**.

To list the certificates in the **cacerts** store, call:

```
keytool -keystore cacerts -list
```

before and after you import your certificate.

To delete an old certificate in the store, call:

```
keytool -keystore cacerts -delete -alias jupiter_cert
```

Setting SSL trace parameters:

Setting the following debug parameter in your javac command line:

```
-Djavax.net.debug=all
```

will trace detailed SSL errors and messages, which helps you determine the reason if the SSL connection does not work. For example, the path of the key store in use is also traced, so you can see whether or not you imported the certificate into the right store.

Specifying a truststore:

If you explicitly specify a truststore in the **javac** command line, for example:

```
-Djavax.net.ssl.trustStore= D:\jdk\jre\lib\security\cacerts
```

this store is used.

2.9.5. Binary Attributes

To map binary attributes correctly between DirX Identity and a connected system (including a file system) specify the **;binary** suffix only for attributes that contain an ASN.1 prefix in their binary data. These are the attributes with either the schema syntax Certificate, like the attribute userCertificate, or with the schema syntax CrossCertPair or CRL or similar. For attributes containing only raw binary data - without an ASN.1 prefix - which are those of schema syntax Octet String, specify the attribute name with the suffix **;raw** in the mapping if it does not belong to the standard LDAP attribute schema. If it belongs to the standard schema - for example jpegPhoto - a suffix is not required but does not do any harm if specified. If you are not sure whether it belongs to the standard LDAP schema, you should specify the **;raw** suffix. This is also true for Active Directory attributes with raw binary data. Consequently, if Active Directory is part of the workflow, those attributes - for example thumbnailPhoto - must also be specified with the **;raw** suffix because the ADS connector is derived from the LDAP connector and inherits the functionality that interprets the suffix. The suffix - if required - must always be specified in both realtime mapping directions.

The LDAP connector knows the following (builtin) list of binary attributes:

"audio",
"authorityrevocationlist",
"cacertificate",
"certificaterevocationlist",
"consumerknowledge",
"crosscertificatepair",
"deltarevocationlist",
"entryaci",
"jpegphoto",
"mhsdeliverableclasses",
"mhsdlarchiveserv",

"mhsdlmembers",
"mhsdlpolicy",
"mhsdlsubscriptionserv",
"mhsoraddreswithcapabilities",
"ntsecurityidentifier",
"photo",
"prescriptiveaci",
"pwdhistory",
"queryoptimizerconfig",
"queryoptimizerstatistic",
"subentryaci",
"supportedalgs",
"usercertificate",
"userpassword",
"userpkcs12",
"usersmimecertificate"

2.9.6. Non-Leaf Objects

The LDAP Connector supports the deletion of non-leaf objects. Even though non-leaf objects are specific to Active Directory this feature is implemented in the LDAP Connector, because it is realized through the LDAP control LDAPAdsDeleteSubtreeControl. The LDAP connector manages all LDAP controls because any other LDAP Server can support them too.

Non-leaf objects in Active Directory are no container objects, like OUs, but objects that are usually expected to be leaf objects, like users. Nevertheless, sometimes these objects are non-leaf objects because they have subentries in certain cases. For example, Active Directory creates subentries for mailbox-enabled users in special situations. Those subentries are only shown by the "Active Directory Users and Computers" tool if the "Users, Contacts, Groups and Computers as containers" setting is checked in the View menu entry.

If such a non-leaf object is to be deleted the LDAP Connector - as parent class of the instantiated ADS Connector class - automatically deletes this object with all its subentries.

2.9.7. 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.

If your DirX Directory installation supports the LDAP session tracking control the various DirX Identity components, like DirX Identity Manager, Web Center, Policy Agent, Provisioning or Request workflows and several more, extend the LDAP audit records with some session tracking related items. The three most important items are:

- SID-Name, which contains the name of the invoking component,
- SID-Info, which contains the DN of the bind user, and
- SID-IP, which contains the IP address of the machine the component runs on.

The LDAP Connector supports session tracking the following way:

In the open method, the LDAP Connector checks if the property for the source component name, also referred to as source application name, is passed in the context. The context class consists of a lot of properties filled by the Connector framework in dependence on the context the LDAP Connector is instantiated from.

If the source application name is set in the context, the LDAP Connector creates an LDAP session tracking control (LDAPSessionIdentifierControl) with the source application name, the bind user DN and the computer name, and appends it to every subsequent LDAP operation (open, add, modify, delete and search). If the DirX LDAP Server supports the LDAP session tracking control it can be found afterwards inside the LDAP audit records.

For example, if a real-time provisioning workflow instantiated the LDAP Connector the source application name has the following format:

> **DXI** {**JoinFromDXI** | **JoinToDXI**} *workflowInstanceID workflowName*,
> for example DXI JoinFromDXI 1495c804a6e$-724d Ident_ADS_Realtime.

2.10. LDIF Connector

The LDIF connector implements the DirX Identity Java Connector Integration Framework's DxmConnectorCore, DxmRequestor and DxmContext interfaces and writes and reads LDIF files using the Netscape LDIF classes. Like all framework-based agents, it gets SPML requests from the Identity side by the join engine as part of the workflow engine hosted by the Java-based Server. It converts the SPML requests in order to read from and write to LDIF files.

The LDIF connector provides the functionality to:

- Add any kind of object - especially user, account or group - to an LDIF content file.
- Perform searches on an LDIF content file to import the objects to Identity.

2.10.1. Overview

The connector implements the API methods "add(...)" and "search(...)". They represent the corresponding SPML requests "AddRequest" and "SearchRequest".

2.10.2. Limitations

It is not currently possible to read and write LDIF change files. Only LDIF content files are supported.

2.10.3. Request and Response Handling

This section describes the supported requests and attributes for the LDIF connector.

2.10.3.1. AddRequest

In an add request, the identifier is mandatory. Any kind of object and attribute can be passed in an add request to the LDIF connector, which writes it as LDIF content record to the file name retrieved from the connector's **export_file** property or if not specified there from the framework context variable **ts.*channelName***.env.export_file*, where *channelName* is retrieved from the operational attributes of the AddRequest. The default channel name is **users**. There might be other channels configured under the LDIF file connected directory in order to read and write other objects than users to an LDIF file.

Example request:

```
<?xml version="1.0" encoding="UTF-8" ?>
<spml:batchRequest xmlns="urn:oasis:names:tc:SPML:1:0"
 xmlns:spml="urn:oasis:names:tc:SPML:1:0"
 xmlns:dsml="urn:oasis:names:tc:DSML:2:0:core"
requestID="batch-1"
 processing="urn:oasis:names:tc:SPML:1:0#sequential"
execution="urn:oasis:names:tc:SPML:1:0#synchronous"
 onError="urn:oasis:names:tc:SPML:1:0#exit">
 <spml:addRequest requestID="add-1">
    <spml:identifier type="urn:oasis:names:tc:SPML:1:0#DN">
       <spml:id>cn=my class,cn=supplier groups,cn=groups,cn=Extranet
Portal,cn=TargetSystems,cn=my-company</spml:id>
    </spml:identifier>
    <spml:attributes>
       <spml:attr name="objectclass"
xmlns="urn:oasis:names:tc:DSML:2:0:core">
           <dsml:value>dxrTargetSystemGroup</dsml:value>
       </spml:attr>
       <spml:attr name="uniqueMember"
xmlns="urn:oasis:names:tc:DSML:2:0:core">
          <dsml:value>cn=my-company</dsml:value>
       </spml:attr>
       <spml:attr name="dxrState"
xmlns="urn:oasis:names:tc:DSML:2:0:core">
          <dsml:value>ENABLED</dsml:value>
       </spml:attr>
    </spml:attributes>
    <spml:operationalAttributes>
       <dsml:attr name="channelName">
          <value>users</value>
       </dsml:attr>
```

```
    </spml:operationalAttributes>
 </spml:addRequest>
</spml:batchRequest>
```

2.10.3.2. Search Request

In an SPML search request, the LDIF connector supports the elements **searchBase** and **filter** and the operational attributes **scope**, **pageSize**, **noattrs** (if set to FALSE or not existing all attributes are retrieved) and **channelName**.

The join engine sets the operational attribute **channelName** only in a Java server workflow context. **channelName** is used to get the name of the source file for the SearchRequest if no file name was specified in the LDIF connector's <connection> **filename** property. The file name is then obtained from the framework context variable **ts.*** ***channelName***.env.import_file*.

If the join engine calls the LDIF connector's search method in the context of a workflow running from Identity to the connected system (export mode), the LDIF connector returns an empty search result (if **contentType** is not specified or set to **LDIF-CONTENT**) to make the join engine produce an AddRequest resulting in writing an LDIF content record.

If the LDIF connector is extended to be able to write LDIF change records, the connection property **contentType** must be set to **LDIF-CHANGE**. This setting makes the LDIF connector return the search result based on the specified import file to the join engine. The join engine then - as usual - calculates the changes compared to the original modify requests and passes the modify requests containing only the changes to the LDIF connector, which writes them as LDIF change records to the specified export file.

Example request:

```
<?xml version="1.0" encoding="UTF-8" ?>
<spml:searchRequest  xmlns="urn:oasis:names:tc:SPML:1:0"
     xmlns:spml="urn:oasis:names:tc:SPML:1:0"
     xmlns:dsml="urn:oasis:names:tc:DSML:2:0:core"
     requestID="search_01"
     >
<spml:searchBase type = "urn:oasis:names:tc:SPML:1:0#DN">
   <spml:id>cn=users,cn=My-Company</spml:id>
</spml:searchBase>
   <spml:filter>
     <dsml:and>
       <dsml:equalityMatch name="ou">
       <dsml:value>Sales</dsml:value>
       </dsml:equalityMatch>
       <dsml:not>
```

```
        <dsml:present name="assistant" />
        </dsml:not>
      </dsml:and>
   </spml:filter>

<spml:operationalAttributes>
   <dsml:attr name="scope">
      <value>subtree</value>
   </dsml:attr>
   <dsml:attr name="pageSize">
      <value>0</value>
   </dsml:attr>
   <dsml:attr name="channelName">
      <value>users</value>
   </dsml:attr>
</spml:operationalAttributes>
<spml:attributes>
</spml:attributes>
</spml:searchRequest>
```

2.10.4. Configuration

Here is a sample configuration snippet for the LDIF connector:

```
<connector
 role="connector"
 className=" siemens.dxm.connector.framework.ldif.LdifConnector "
 name="LDIF Connector" version="1.00">
 <connection type="file"
   filename="import1.ldif,import2.ldif,import3.ldif"
   <property name="binaryAttributes" value="dxmMgrlayout,
dxmContent,dxmCompiled"/>
   <property name=" export_file" value="export.ldif"/>
   <property name="contentType" value="LDIF-CONTENT"/>
   <property name="namingAttribute" value=""/>
 </connection>
</connector>
```

2.10.4.1. Supported Connection Parameters

The following standard properties of the XML configuration file's <connection> element are

supported:

filename - (optional); one or more comma-separated filenames used as the source file for the search request (import file). The wildcards ***** and **?** are supported. For example, **import**.ldif* specifies any file beginning with **import** and ending with **.ldif** and **import?.ldif** specifies any file beginning with **import** followed by the character **0** or **1** and ending with **.ldif**. In a Java-based workflow context, the framework context variable **ts.*** ***channelName***.env.import_file* specifies the import file name if **filename** property is not specified.

Non-standard supported properties include:

binaryAttributes - (optional); if configured, **binaryAttributes** handles the specified attributes as binary ones regarding the correct setting of the appropriate SPML request attribute types.

export_file - (optional); the name of the file the LDIF records are written to. If not specified, in a Java server export workflow context the framework context variable **ts.*** ***channelName***.env.export_file* specifies the file name.

contentType - (optional); The default is **LDIF-CONTENT**. **LDIF-CHANGE** is not yet supported.

namingAttribute - (optional); only relevant for add request handling. If the naming attribute is specified and if the Identifier of the add request is not of type DN (usually the type is DN, which is also the default), but of type OID, the DN value written to the LDIF file is built up by *namingAttribute_value + RDN_of_Identifier*.

2.11. IBM Notes Connector

The IBM Notes connector is a C-based connector that runs in the C-based Server. It handles search and update requests (SPML V1) and therefore is able to export entries from an IBM Notes address book or to import entries into the Notes address book.

For details of the OASIS SPML Service Provisioning Markup Language see http://www.oasis-open.org/committees/provision/docs.

2.11.1. Overview

In DirX Identity, the Java-based Server hosts the Java components, especially the workflow engine that includes the join engine. The join engine issues the search and update requests via the configured connectors: the LDAP connector to the Identity Store and the SOAP connector for requests to the IBM Notes connector. The SOAP connector sends SPML requests to the SPML/SOAP listener of the C++-based Server, which passes them to the Notes connector. Finally, the Notes connector interacts with the Notes server.

The Notes connector has the responsibility to:

- Create (and update) a "Person" document in the Notes address book.
- Register a user if the **dxmLNregisterUser** attribute is set.
- Request a **Rename** at the Notes server if one of the relevant user attributes has

changed: first name, last name, middle initial, unique organizational unit.

- Request a **MoveInHierarchy** operation if the target certifier of the user has been changed.
- Put the user into an appropriate deny group if the user is to be disabled and remove it from the deny group otherwise. The connector must consider that deny groups are limited in size and create a new one if the existing ones reach the limit.
- Indicate that the user is disabled if it is a member of a deny group. As part of a search result entry, the attribute **dxrTSstate** is set to **DISABLED**, if the entry is member of a deny group; otherwise **dxrTSstate** is set to **ENABLED**.
- Create (and update) a “Group” document in the Notes address book.

This functionality is partly provided offline by **adminp**. The following Notes API calls are used:

- Renaming a user:
  ADMINReqRename
- Moving a person:
  ADMINReqMoveUserInHier and
  ADMINReqMoveComplete
- Deleting a person:
  ADMINReqDeleteInNAB
- Registering a person:
  REGNewUser (sets the flag **fREGCreateMailFileUsingAdminp** if the parameter **CreateMailDBNow** is set)

The following sections describe the functionality provided by the Notes connector in details.

2.11.2. Prerequisites and Limitations

The Notes connector supports only IBM Notes server and client versions 7.03 or higher. Earlier versions are no longer supported. Use of additional functionality of the Notes APIs enforces this restriction.

The Notes connector requires the following software packages to be installed on the platform where the C++-based Server is running:

IBM Notes Client V7.03 (or higher)

The operation of the Notes connector is restricted by these limitations:

- Attribute names are handled as CaseExactStrings.
- Search limitations:
- If the SPML identifier (representing the universal IDs) is present, the search filter is ignored.
- If any other search base format is present, it must define the type of document to be

searched for; for example, Type=Person or Type=Group.

- In filters, only matching for equality is supported for attribute values.
- NOT filters are not supported.
- **UniqueOrgUnit**
  The unique organizational unit attribute value that could be part of a person's full name cannot be searched for. As a result, the Notes connector stores the value of the unique organizational unit in a configurable attribute and uses that value, if available, when searching a person document. (See the sections about configuration for details.)
- **Rename** and **MoveInHierarchy**:
  If the parameters of an update operation both result in a **Rename** and a **MoveInHierarchy** operation, then only the **Rename** operation is propagated to the Notes server. The **MoveInHierarchy** operation is only executed in the next update operation if no pending request is present (which is only detected if the Notes real-time workflow is used). The connector itself has no knowledge whether there is a pending rename operation.
- **Item Types**
  The Notes attributes that are supported by the Notes connector must have one of the following item types:
- TEXT
- TEXT_LIST
- NUMBER
- TIME

Other Notes item types (for example, RICH_TEXT) that are not listed above are not supported.

2.11.3. Static Configuration Parameters

Static configuration parameters for the Notes connector are included with the "INI Template" definition that can be viewed in the DirX Identity Manager (Connectivity view) below the object:

Connectivity Configuration data → **Configuration** → **Connector Types** → **Notes**

The connector reads this information only once during the C++-based Server startup.

The Notes connector uses the following configuration information:

2.11.3.1. Connected Directory

AdminReqDB

The Admin Request Database field specifies the name of the Notes Administration Process (adminp) request database that is used when deleting persons.

Example:

```
AdminReqDB=admin4.nsf
```

AdminReqAuthor

The Admin Request Author field specifies the author name of the Notes Administration Process (adminp) request database that is used when deleting persons.

Example:

AdminReqAuthor=*FullName_of_administrator*
AdminReqAuthor=CN=administrator/O=My-Company

AdrBook

The Address Book field specifies the name of the Notes address book.

Example:

AdrBook=names.nsf

GroupMemberLimit

The Group Member Limit field specifies the maximum number of members in a group. When that limit is reached, another group is created and the group name of that group is stored in previous group (nested groups).

UniqueOrgUnitAttrType

IBM Notes doesn't return the **UniqueOrgUnit** attribute when searching with that attribute set. As a result, the Notes connector stores the **UniqueOrgUnit** in an additional configurable attribute **UniqueOrgUnitAttrType** that can be used for searching.

Example:

UniqueOrgUnitAttrType=telexTerminalIdentifier

2.11.3.2. Services

Server

The Server Name field specifies the name of the Notes server in the format:

CN=*server_name***/O=***organization_name*[**/**...]

Make sure that the attribute types in the server name (for example, CN, O, OU) are defined with uppercase letters.

Example:

CN=*my-server***/O=***my-organization*

2.11.3.3. Bind Profile

At least two bind profiles are required:

- A bind profile for the administrator who has the right to add, delete, modify or move persons and groups

- A bind profile that represents an organization or organizational unit - for example, **cert.id** - and is used when
- registering a user
- moving a person
- renaming a person

within that organization or organizational unit.

Furthermore, if a **MoveInHierarchy** operation is called (when Notes users are moved to a different organization or organizational unit), additional bind profiles for each organization or organizational unit are required.

The following fields of the bind profile are used:

User

The **User** field specifies the full pathname of the ID file. This file must be accessible on the machine where the C++-based Server (hosting the Notes connector) is running. Make sure that the pathname matches the pathname that is passed in the connector update requests in the attributes "PathFileCertId" or "PathFileTargetCertId". (Be aware that for Notes real-time workflows, these attributes are set using the values of the Notes profiles in the Notes target system tree. So the pathnames in the bind profiles must match the pathnames that are used in the Notes profiles.)

Password

The **Password** field specifies the password that is related to the ID file.

2.11.3.4. Dynamic Configuration Parameters

The Notes connector evaluates all of the attributes that are sent in the each SPML request. A subset of attributes is set in the organizational unit-specific Notes profiles that are defined in each target system instance.

The available attributes from the Notes profile objects are:

Control Parameters:
CreateIdFile
CreateMailDatabase
CreateMailDBNow

CreateMailFullTextIndex

CreateMailReplicas
CreateNorthAmericanId
SaveIdInAddressBook
SaveIdInFile
SaveInternetPassword
DeleteMailFile

Other Attributes:
CertifierStructure (will be passed as TargetCerfier to the Notes connector)

ClientType
DbQuotaSizeLimit
DbQuotaWarningThreshold

DefaultMailServer (will normally be mapped to the attribute MailServer

LocalAdmin
MailACLManager
MailForwardAddress
MailOwnerAccess

MailServer
MailSystem
MailTemplate
MinPasswordLength

OtherMailServers
PathFileCertId
PathFileCertLog
PathUserId
RegistrationServer
Validity

The Notes connector does not know where these attributes originate because it simply processes the attribute that is passed to it in the SPML request. It is listed here to identify more details about Notes real-time workflows, Notes configuration data and finally the Notes connector.

If you are not using the Notes real-time workflows provided with DirX Identity, make sure that these attributes are passed in the SPML request, if needed.

Be aware, too, that the attribute names are handled as CaseExactStrings.

2.11.4. Attributes at IBM Notes

The following list of attributes is relevant at the target system (IBM Notes) side. Customer projects can synchronize additional attributes provided that the Notes documents in the IBM Notes address book can hold these new attribute types.

ClientType

The **ClientType** field specifies the type of Notes client that the Notes connector is to associate with the registered users it creates during the import process. The syntax is:

ClientType=*number*

where *number* is one of the following values:

- **1** - create registered users of client type "desktop"
- **2** - create registered users of client type "complete"

- **3** - create registered users of client type "mail"

The client types correspond to the different kinds of licenses available for Notes clients.

ComputeWithFormIgnoreErrors

The **ComputeWithFormIgnoreErrors** field specifies the way in which the Notes-API “ComputeWithForm” is called before the Notes document is saved. (“ComputeWithForm” calculates computed fields and evaluates validation formulas defined in the form used by the Notes document.)

The syntax is:

ComputeWithFormIgnoreErrors=*switch*

where *switch* is one of the following values:

- **0** - if you want the function to stop at the first error
- **1** - if you do not want the function to stop executing if a validation error occurs

  If absent, the Notes-API “ComputeWithForm” is not called. This default behavior is compatible with older versions of DirX Identity where this parameter is not configurable.

CreateIdFile

The **CreateIdFile** field controls whether or not Notes connector creates a user ID file for Notes users that it registers during the import process. The syntax is:

CreateIdFile=*switch*

where *switch* is one of the following values:

- **0** - register Notes users, but do not create a user ID file for them
- **1** - register Notes users and create a user ID file for them

If **CreateIDFile** is set to 1, either the **SaveIdInAddressBook** field or the **SaveIdInFile** field (or both) must be set to 1 to specify where the Notes connector is to store the user ID files it creates.

CreateMailDatabase

The **CreateMailDatabase** field controls whether or not the Notes connector creates user mailboxes for Notes users that it registers. The syntax is:

CreateMailDatabase=*switch*

where *switch* is one of the following values:

- **0** - do not create a mailbox
- **1** - create a mailbox

CreateMailDBNow

The **CreateMailDBNow** field controls whether or not the mail file is created during the

registration. The syntax is:

CreateMailDBNow =*number*

where *number* is one of the following values:

- **0** - create mail file later with the administration process
- **1** - create mail file during the registration

CreateMailFullTextIndex

The **CreateMailFullTextIndex** field controls whether or not a full-text index is created when creating the mailbox. The syntax is:

CreateMailFullTextIndex=*number*

where *number* is one of the following values:

- **0** - do not create mail full-text index
- **1** - create mail full-text index

If absent, the mail full-text index is created. (This default behavior is compatible with older versions of DirX Identity where this parameter is not configurable.)

CreateMailReplicas

The **CreateMailReplicas** field controls whether or not the mail replicas should be created with the administration process. The syntax is:

CreateMailReplicas=*number*

where *number* is one of the following values:

- **0** - do not create mail replicas
- **1** - create mail replicas with the administration process

If absent, no mail replicas are created. This default operation is compatible with older versions of DirX Identity where this parameter is not configurable.

CreateNorthAmericanId

The **CreateNorthAmericanId** field controls whether or not the Notes connector creates United States security-encrypted User ID files when registering a new user. The syntax is:

CreateNorthAmericanId=*switch*

where *switch* is one of the following values:

- **0** - do not create U.S.-encrypted user ID files
- **1** - create U.S.-encrypted user ID files

If **CreateNorthAmericanId** is set to 1, the Notes registered user can only be used within the United States.

DbQuotaSizeLimit

The **DbQuotaSizeLimit** field is only used when registering a new user and specifies the size limit of user's mail database. The syntax is:

DbQuotaSizeLimit =*number*

where *number* is the size in MB.

DbQuotaWarningThreshold

The **DbQuotaWarningThreshold** field is only used when registering a new user and specifies the size of a user's mail database at which point a warning about the size of the database is generated. The syntax is:

DbQuotaWarningThreshold =*number*

where *number* is the size in MB.

DeleteMailFile

The **DeleteMailFile** field controls the way the mail files of a person are handled when the person is deleted. The syntax is:

DeleteMailFile=*switch*

where *switch* is one of the following values:

- **0** - don't delete mail file
- **1** - delete the mail file specified in the person record
- **2** - delete mail file specified in person record and all replicas

dxmLNregisterUser

The **dxmLNregisterUser** field controls whether or not the Notes connector registers a user. The syntax is:

dxmLNregisterUser=*switch*

where switch is one of the following values:

- **0** - do not register Notes users
- **1** - register Notes users

InternetAddress

The **InternetAddress** field is only used when registering a new user and specifies the internet mail address of the user. The syntax is:

InternetAddress=*address*

Example:

```
InternetAddress=john@x.com
```

MailACLManager

The **MailACLManager** field is only used when registering a new user and specifies the manager name of the access control list of the mail file. The syntax is:

MailACLManager=*name*

where *name* is the manager name in canonical format. For example:

```
MailACLManager=CN=Administrator/O=MyCompany
```

MailFile

The **MailFile** field is used when registering a new user or when deleting a user with its mail file. It specifies the mail file name including the path relative to the Notes data directory.

Example:

```
MailFile=mail/tom.nsf
```

MailForwardAddress

The **MailForwardAddress** field is only used when registering a new user and specifies the forwarding address of a Domino domain or foreign mail gateway. The syntax is:

MailForwardAddress=*name of the forwarding address*

MailOwnerAccess

The **MailOwnerAccess** field is only used when registering a new user and specifies the mail owner's ACL privileges. The syntax is:

MailOwnerAccess =*number*

where *number* is one of the following values:

- **0** - Manager (default)
- **1** - Designer
- **2** - Editor

MailServer

The **MailServer** field specifies the name of a Notes server on which the Notes connector is to create user mailboxes during the user registration process. Furthermore it's used when deleting a user and its mail must be deleted, too. The syntax is:

MailServer=*server_name*

where *server_name* is the name of a Notes server in the format:

"CN=*server_name***/O=***organization_name*[/...]"

For example:

```
MailServer="CN=Cambridge4/O=Notes/O=IBM"
```

MailSystem

The **MailSystem** field is only used when registering a new user and specifies the type of the mail system. The syntax is:

MailSystem=*number*

where *number* is one of the following values:

- **0** - NOTES (default)
- **1** - CCMAIL
- **2** - VINMAIL
- **99** - NONE

MailTemplate

The **MailTemplate** field is only used when registering a new user and specifies the name of the mail template database. The syntax is:

MailTemplate =*name of the template database*

Example:

```
MailTemplate=mail7.ntf
```

MinPasswordLength

The **MinPasswordLength** field is only used when registering a new user and specifies the minimum number of characters that a user password must have. The syntax is:

MinPasswordLength=*number*

For example:

```
MinPasswordLength=5
```

The Notes connector sets the specified value as an attribute of the registered user entry.

If the value is set to 0 the **SaveIdInAddressBook** field also must be set to 0.

PathFileCertId

The **PathFileCertId** field specifies the pathname to the certificate ID file **cert.id**, which is a binary file that is supplied with the Notes Server installation software. This file contains the certificate that grants the Notes connector the right to create registered users. The syntax is:

PathFileCertId=*pathname*

where *pathname* is the pathname to the certificate ID file. For example:

```
PathFileCertId=a:\cert.id
```

This is a required field if the update operation is to process a **RenameUser** request or if

the **dxmLNregisterUser** field is set to TRUE.

This is a required field that must specify the pathname to the certificate ID file of the source organizational unit if the update operation is to process the **MoveUserInHier** operation.

PathFileCertLog

The **PathFileCertLog** field specifies the pathname to the certifier logging file **certlog.nsf** on the server. This file contains the certifier logging entries of the registered users. The syntax is:

PathFileCertLog=*pathname*

where pathname is the pathname to the certifier logging file. For example:

```
PathFileCertLog=d:\lotus\domino\data\certlog.nsf
```

This is a required field if the **dxmLNregisterUser** field is set to TRUE or if the update operation is to process a **RenameUser** or a **MoveUserInHier** request.

PathFileTargetCertId

The **PathFileTargetCertId** field specifies the pathname to the certificate ID file of a target organizational unit. The file contains the certificate that grants the Notes connector the right to create registered users for the organizational unit. The syntax is:

PathFileTargetCertId=*pathname*

where *pathname* is the pathname to the certificate ID file. For example:

```
PathFileTargetCertId=a:\German.id
```

This is a required field if the update operation is to process a **MoveUserInHier** operation.

PathUserId

The **PathUserId** field specifies the directory in which the Notes connector is to store Notes user IDs created during the user registration process. The syntax is:

PathUserId=*directory*

where *directory* is a directory pathname. For example:

```
PathUserId=e:\notes\data
```

Notes User IDs are binary user certificate files that the Notes connector creates during the registration process if **CreateIdFile** is set to 1. The Notes connector writes these user ID files to the directory specified in the **PathUserId** field if **SaveIdInFile** field is set to 1.

RegistrationServer

The **RegistrationServer** field specifies the name of the Notes registration server that is to register the users in the Notes server address book. The syntax is:

RegistrationServer=*server_name*

where *server_name* is a the name of a Notes server in the format:

"CN=*server_name***/O=***organization_name*[/...]"

For example:

```
RegistrationServer="CN=Cambridge3/O=Notes/O=IBM"
```

SaveIdInAddressBook

The **SaveIdInAddressBook** field controls whether or not the Notes connector saves the user ID files it creates as attachments of the Notes entries for the registered users. The syntax is:

SaveIdInAddressBook=*switch*

where *switch* is one of the following values:

- **0** - do not save user ID files as attachments of the Notes entries for the registered users
- **1** - save user ID files as attachments of the Notes entries for the registered users in the Notes address book

  If **SaveIdInAddressBook** is set to 1, the Notes connector creates the user ID file and stores it as an attachment of the corresponding Person entry for the registered user. If **SaveIdInAddressBook** is set to 1, the registered user must have got a password.

SaveIdInFile

The **SaveIdInFile** field is only used when registering a new user and controls whether or not the Notes connector saves the user ID files it creates in individual files. The syntax is:

SaveIdInFile=*switch*

where *switch* is one of the following values:

- **0** - do not save user ID files in individual files
- **1** - save user ID files in individual files

  If **SaveIdInFile** is set to 1, the Notes connector creates the user ID files and stores them in the directory specified in the **PathUserId** field.

SaveInternetPassword

The **SaveInternetPassword** field is only used when registering a new user and controls whether or not the Notes connector saves the user ID password also for use as an Internet password. The syntax is:

SaveInternetPassword*=switch*

where switch is one of the following values:

- **0** - do not save user ID password also as Internet password
- **1** - save user ID password also as Internet password

  If **SaveInternetPassword** is set to 1, the Notes connector saves the user ID password also in the field for the Internet password.

TargetCertifier

The **TargetCertifier** field specifies the name of the new location when is user is moved.. The syntax is:

TargetCertifier=*name*

where name is a the name of a Notes entity in the format:

"OU=*organizational unit name***/O=/***organization_name*[/...]"

For example:

```
TargetCertifier=/OU=sales/O=my-company
```

Type

The Type field specifies the Notes document type to be extracted from the Notes address book (on Export) or to be created in the Notes address book (on Import). The syntax is:

Type=*document_type*

where *document_type* is a Notes document type.

Example:

```
Type=Person
```

or

```
Type=Group
```

UserIdFile

The **UserIdFile** field is only used when registering a new user and specifies the name of a Notes ID file of a a user. The syntax is:

UserIdFile=*filename*

where *filename* is the name of the user ID file.

Example:

UserIdFile=tom.id

Validity

The **Validity** field defines the lifetime of a certificate in GeneralizedTime syntax. The

syntax is as follows:

Validity=*YYYYMMDDhhmmssZ*

Example:

```
Validity=201012301500000Z
```

Other important attributes

There are many other attributes available in the Notes address book. The important ones include:

User attributes:

FirstName

LastName

MiddleInitial

UniqueOrgUnit

FullName

ShortName

Group attributes:

ListName

All of these attributes are string attributes and define the name of the group or (for user) the combination of FirstName, LastName, MiddleIntitial and UniqueOrgUnit define the user object.

2.11.5. Attributes at Identity Store

The following list of attributes is relevant at the Identity Store side.

dxmLNregistereUser

The attribute **dxmLNregisterUser** is a Boolean attribute and indicates whether a person should be registered in Notes. If set to **FALSE**, only a Notes document is created in the Notes address book.

dxmLNuserRegistered

The attribute **dxmLNuserRegistered** is a Boolean attribute and indicates whether the account has been registered in the Notes server. The attribute is set to **TRUE**, if the **FullName** is present in Notes.

dxmLNuserInAddressBook

The attribute **dxmLNuserInAddressBook** is a Boolean attribute and indicates whether or not the **Type** attribute in the Notes server is set to **Person** or not. **dxmLNuserInAddressBook** is set to **TRUE**, if **TYPE=Person**; it is set to **FALSE**, if type is set to **InactivePerson**. (**Type=InactivePerson** is set to make the user invisible for the Notes

Client).

2.11.6. Feature Details

This section describes Notes connector feature details.

2.11.6.1. General Aspects

This section describes general features of the Notes connector.

2.11.6.1.1. SPMLv1 Identifier

The SPML identifier is mandatory for the following operations:

DeleteRequest

ModifyRequest

It is optional for the following operations:

AddRequest

SearchRequest

When present, it is normally set up as type=value list of the Notes universal IDs. The format is as follows (for example, as part of a Modify request):

```
<spml:identifier type="urn:oasis:names:tc:SPML:1:0#DN">
<spml:id>UniversalIDPart1=<id1>,UniversalIDPart2=<id2>,
   UniversalIDPart3=<id3>,UniversalIDPart4=<id4>
</spml:id>
</spml:identifier>
```

If absent, then the SPML identifier should be set as follows:

```
<spml:identifier type="urn:oasis:names:tc:SPML:1:0#DN">
    <spml:id/>
</spml:identifier>
```

If search requests, the Identifier could also by set using the Notes **Type** attribute, for example

```
<spml:searchBase type="urn:oasis:names:tc:SPML:1:0#DN">
       <spml:id>Type=Person</spml:id>
</spml:identifier>
```

2.11.6.1.2. Deny Groups

If a user is disabled in the Identity Store, the user is put into one of the deny groups that are available in the Notes system. In the SPML update operations, the attribute “dxrTSstate” must be passed with the value set to “DISABLED”. The Notes connector will check all the deny groups and will put the user into one of them if not yet present there. If during an SMPL update operation the value “dxrTSstate=ENABLED” is passed, then the user is dropped from the deny groups. When putting users in the deny group, the Notes connector guarantees that a new deny group is created when the existing ones have reached their capacity limits.

When Notes users are returned in a search result, the attribute “dxrTSstate” is set according to the presence of the users in the deny groups; the value is set to ENABLED if the user is not present in one of the deny groups; otherwise, the value is set to DISABLED.

2.11.6.1.3. Register User

A user will be registered if the dxmLNregisterUser attribute comes along in an ADD or MODIFY request and the Type attribute of a new object (for ADD) or of an existing object (for MODIFY) is PERSON.

The user registration enforces a unique short name; that’s not required by Notes itself, it’s a requirement of the Notes connector.

For registering the user, the following attributes are evaluated:

- ClientType
- CreateIdFile
- CreateMailDatabase
- CreateMailDbNow
- CreateNorthAmericanId
- FirstName
- IdFile (composed of “PathUserId\UserIdFile”)
- InternetAddress
- LastName
- MiddleInitial
- SaveIdInFile
- SaveIdInAddressBook
- SaveInternetPassword
- ShortName
- SMTPHostDomain
- UniqueOrgUnit (derived from the configurable attribute type)

If a mail database should be created, then these attributes are required, too:

DvQuotaSizeLimit

DbQuotaWarningThreshold

MailACLManager

MailFile

MailForwardAddress

MailOwnerAccess

MailServer

MailSystem

MailTemplate

2.11.6.2. Add Request

The Notes connector first checks whether a Notes document is already present in the Notes address book. Therefore for objects of **Type=Group**, it uses the **ListName** for retrieving the object, for objects of **Type=Person** or **Type=InactivePerson**, it uses **FirstName**, **LastName**, **MiddleInitial** and **UniqueOrgUnit**. If **UniqueOrgUnit** is present, it uses the attribute specified in **UniqueOrgUnitAttr** because the attribute **UniqueOrgUnit** is not searchable in Notes address book.

If no such document is found, then the document is created.

If the attribute **dxmLNregisterUser** is set to **TRUE**, the user will be registered. For details, see the section “Register User”.

If the attribute **dxrTSstate** is **DISABLED**, then the user is put into the Deny groups.

2.11.6.3. Add Response

The add response will return the SPML identifier of the new object. It will also return the **FullName** of the new object, if the user was registered before. If available, it will also return the **ShortName** of the user.

2.11.6.4. Delete Request

A user will be deleted if the **TYPE** attribute of the existing user is Person and the user had been registered before. When deleting the user, the **DeleteMailFile** attribute (in the OperationalAttributes section of the SPML Modify request) provides information whether or not to delete the user’s mail database.

If the object is not a registered used, then the Notes document will simply be deleted from the Notes address book.

2.11.6.5. Delete Response

There is no specific information available in the delete response. It either return success or provides the error message.

2.11.6.6. Modify Request

If the object exists in the Notes address book, then the attributes in the Notes document are updated.

If the attribute **dxmLNregisterUser** is present in the attribute list and the value is set to TRUE, then the user will be registered (if not yet registered). For details, see section “Register User” above.

A user will be renamed if the **Type** attribute of the existing user is Person and one of the following attribute changes in a MODIFY request:

- FirstName
- LastName
- MiddleInitial
- UniqueOrgUnit

Keep in mind that the UniqueOrgUnit attribute is not retrievable. Therefore the Notes connector uses the value from the configurable attribute that is defined in UniqueOrgUnitAttrType.

A user will be moved if the **Type** attribute of the existing user is Person and the attribute **PathFileTargetCertId** is present and is different from **PathFileCertId** and the user has not been renamed before. If the user has been renamed, then moving the person is rejected until the user was successfully renamed. It’s the responsibility of the client to send another MODIFY request later on in order to move the person.

2.11.6.7. Modify Response

If available, the modify response will return the **FullName** and the **ShortName** of the object.

2.11.6.8. Search Request

For details on search base, see the section “SPMLv1 Identifier”.

Apart from the search base, an SPML filter can be provided. For limitations on search filters, see the section “Prerequisites and Limitations”.

2.11.6.9. Search Response

The Notes connector will search the relevant objects. It also checks, for every object, whether it is present in one of the Deny groups. If present, it returns the attribute **dxrTSstate** with the value **DISABLED**; otherwise **ENABLED**.

2.12. Microsoft 365 Connector

The Java-based Microsoft 365 connector runs inside the Identity Java Connector Integration Framework. It communicates using the Microsoft Graph API on the common URL https://graph.microsoft.com via common HTTP protocol. The operations are authorized by a dedicated OAuth server available on the common URL

https://login.microsoftonline.com/*TenantID***/oauth2/token**.

The connector is implemented in the class **Office365Connector** in the package **net.atos.dirx.dxi.connector.azure**.

The connector implements the common methods for the DirX Identity Connector API: add, modify, delete, and search.

The operations are simply converted to the Graph API requests. The corresponding responses are again translated to SPMLv1 responses.

The Microsoft Graph API is a Representational State Transfer (REST)-ful service comprised of endpoints that are accessed using standard HTTP requests. The connector uses JavaScript Object Notation (JSON) content types for requests and responses.

The connector communicates using SSL/TLS only.

2.12.1. Prerequisites

The connector is based on Microsoft Graph API version 1.0. The connector functionality is limited by the functionality of the Graph API version in use. The functionality with other Graph API versions cannot be guaranteed.

The connector appends a JSON Web Token (JWT) in the Authorization header of the request. This token is acquired by making a request to the OAuth endpoint and providing valid credentials. The connector supports the use of the OAuth 2.0 service only using a valid symmetric key (Application Secret).

The connector supports common Microsoft 365 user objects (common attributes and navigation properties **memberOf** and **manager**), Microsoft 365 group objects (common attributes only), Microsoft 365 role objects (common attributes only) and Microsoft 365 **subscribedSku** objects (common attributes).

The user navigation properties **memberOf** and **manager** can be written or read only as common **objectId** values (for example, **0ab8ac77-b07a-46ce-a3f6-42a03c5bed6b**).

The connector can handle only one valid license for Microsoft 365 (subscribed sku).

The connector does not support nested group assignment. Nested group assignments cannot be read or written.

2.12.2. Configuration

The connector receives its configuration from the Connector Framework in a format that is specified there and reflects an XML document. Note that DirX Identity Manager presents configuration options in a more convenient way. For example, bind credentials and service addresses are typically collected from appropriate LDAP entries found by selecting the appropriate connected directory and bind profile.

This section discusses the configuration options based on the XML format. These options are either specified attributes in the XML schema of the element <connection> (referred to

as standard properties) or specified as <property> subelements of the <connection> element (referred to as non-standard properties).

The connector evaluates the following standard properties:

server: required. This property provides information about the host name or IP address of the Graph API endpoint. An example is **graph.microsoft.com**.

ssl: required. This value enables SSL/TLS authentication of a Graph API server and secures the communication line.

user: required. This property is the Application ID that identifies DirX Identity as a Graph API client. This ID is also bound to the Application Secret used as a password and is used for OAuth service. An example is **76c6e05b-d989-43a9-ab30-6f6a9a765c71**.

password: required. The password is used as a symmetric key for communication between the Microsoft 365 connector and an OAuth service. An example is **6n2VHFRryIyXlrYCelROtAEJBeiKhhtpwBiX/vp9yO0=**.

type: required. This is the Directory Type, for example, Microsoft 365.

The Microsoft 365 connector evaluates the following non-standard properties beneath the <connection> element:

authEndpoint: required. This property is the full URL of the OAuth service. The URL format is **https://***OAuthServer*/*TenantID*/*OAuthPath*. An example value is **https://login.microsoftonline.com/10ff036f-b6ae-462d-82cb-a7cad3b876c3/oauth2/token.**

path: required. This property provides the path to the latest Graph API Version and is also used for building endpoint URL. Use only valid values in the correct format. The default value is v1.0. An example is **https://graph.microsoft.com/v1.0/groups**.

proxyHost: optional. This property provides information about the host name or IP address of the HTTP proxy server. Do not use authenticated proxy servers. If the access to the proxy server requires authentication, deploy another local transparent proxy server that can access the authenticated one. Use only the local proxy server instead.

proxyPort: optional. This property provides information about the port number of the HTTP proxy server. Do not use authenticated proxy servers. See the description for **proxyHost** for more details.

Here is a sample configuration using some of the properties described here:

```
<connector
className="net.atos.dirx.dxi.connector.azure.Office365Connector"
name="TS" role="connector">
    <connection
password="ZEbMo0Uh/7tbYwUb89HUZ9bLNHABcJ9ELNHfUqsmQCo="
server="graph.microsoft.com" ssl="TRUE" type="Office 365"
```

```
user="def623cc-bac4-41ab-88ca-bea388941262">
         <property name="proxyHost" value="proxy-emea.my-it-
solutions.net"/>
         <property name="proxyPort" value="84"/>
         <property name="authEndpoint"
value="https://login.microsoftonline.com/10ff036f-b6ae-462d-82cb-
a7cad3b876c3/oauth2/token">
         </property>
         <property name="path" value="v1.0"/>
    </connection>
</connector>
```

2.12.3. Creating Azure AD Groups

The following Microsoft Azure AD groups can be created/managed via the Microsoft365 connector using the Microsoft Graph API:

- Microsoft 365 groups (Public, Private, HiddenMembership)
- Security groups

For more information about these groups, see the document:

https://docs.microsoft.com/en-us/graph/api/resources/groups-overview?view=graph-rest-1.0

You can use DirX Identity Manager to create these groups on Microsoft Azure AD. For example:

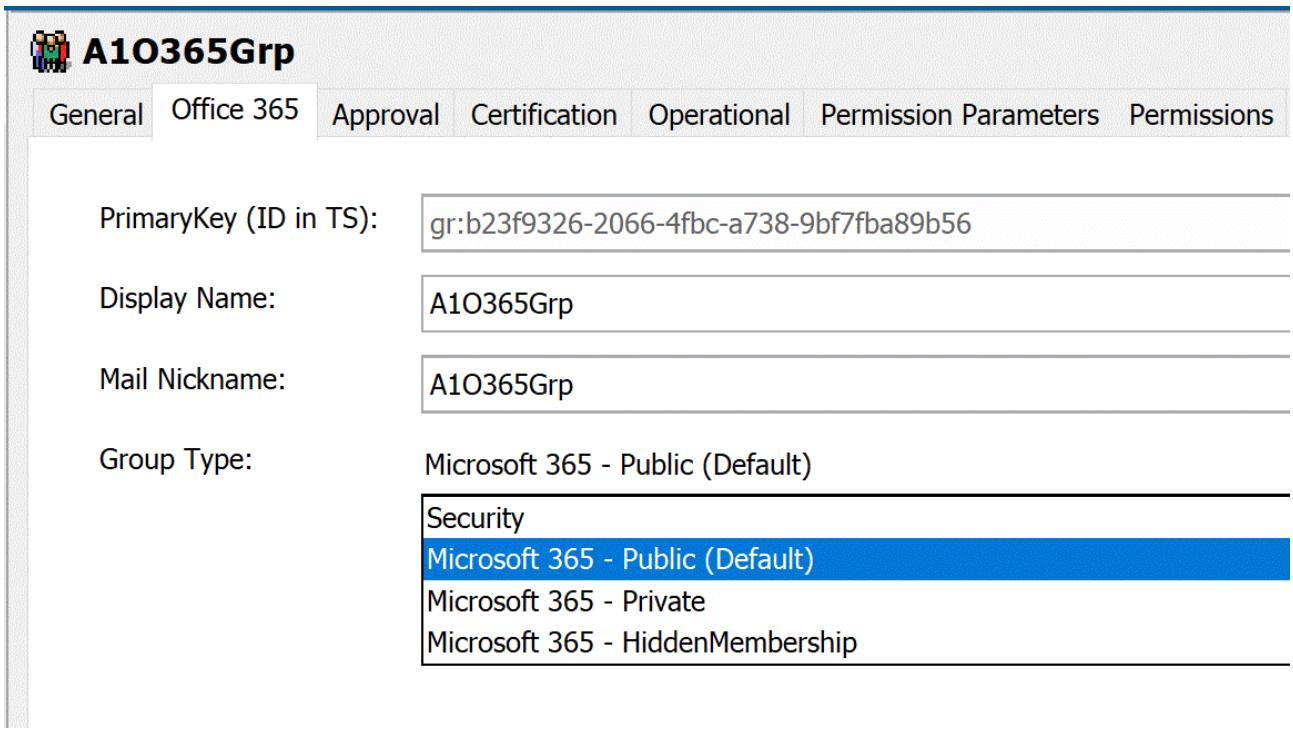


Figure 2. Creating a Group in DirX Identity Manager

Note the following limitations:

- **Group Type** cannot be changed after group creation.
- Do not change **Group Type** on imported groups in edit mode.

Mail-enabled security groups and distribution lists can't be created through the Microsoft Graph API. Microsoft recommends migrating them to Microsoft 365 Groups to achieve the Graph API functionality. For more information, see the document:

https://docs.microsoft.com/en-us/microsoft-365/admin/manage/upgrade-distribution-lists?redirectSourcePath=%252fde-de%252foffice%252f787d7a75-e201-46f3-a242-f698162ff09f&view=o365-worldwide

For details about Microsoft Graph API, refer to the following Microsoft documents:

Graph REST API Reference: https://docs.microsoft.com/en-us/graph/api/overview?view=graph-rest-1.0

Working with users in Microsoft Graph: https://docs.microsoft.com/en-us/graph/api/resources/users?view=graph-rest-1.0

Working with groups in Microsoft Graph: https://docs.microsoft.com/en-us/graph/api/resources/groups-overview?view=graph-rest-1.0

2.12.3.1. Properties Request Body for Creating Groups

The following group resource properties are required when creating a group:

- **displayName** - a string that specifies the name to display in the address book for the group. This property is required.

- **mailEnabled** - a boolean value that is set to true for mail-enabled groups.
- **mailNickname** - a string that specifies the mail alias for the group. Maximum string length is 64 characters.
- **securityEnabled** - a boolean value that is set to true for security-enabled groups, including Microsoft 365 groups.

Groups created using the Microsoft Azure portal always have **securityEnabled** initially set to **true**.

The following optional group resource properties can also be used when creating a group:

- **description** - a string that specifies a description for the group. The maximum string length is 1024 characters.
- **owners** - a string collection that represents the group owners at creation time.
- **members** - a string collection that represents the group members at creation time.
- **visibility** - a string that specifies the visibility of an Microsoft 365 group. Possible values are:
  - **Private** - owner permission is needed to join the group. Non-members cannot view the contents of the group.
  - **Public** - anyone can join the group without needing owner permission. Anyone can view the contents of the group.
  - **HiddenMembership** - owner permission is needed to join the group. Non-members cannot view the contents of the group. Non-members cannot see the members of the group. Administrators (global, company, user, and (helpdesk) can view the membership of the group. The group appears in the global address book (GAL).
  - Empty - interpreted as **Public**.

For more information about the group resource type, see https://docs.microsoft.com/en-us/graph/api/resources/group?view=graph-rest-1.0.

For more information about required and optional properties of a create group request body, see: https://docs.microsoft.com/en-us/graph/api/group-post-groups?view=graph-rest-1.0&tabs=http.

2.12.3.2. groupTypes Property Options

Use the groupTypes property to control the type of group and its membership, as described in the Microsoft document:

https://docs.microsoft.com/en-us/graph/api/group-post-groups?view=graph-rest-1.0&tabs=http

Microsoft 365 Connector **does not support** groupTypes="DynamicMembership".

2.12.3.3. DirX Identity dxrType Values

The following table shows the DirX Identity dxrType values for the supported Azure AD groups:

Group Type	Attribute Values	dxrType Value
Security Group	securityEnabled=true; mailEnabled=false	securityGroup

Microsoft 365 - Public	securityEnabled=false;
	mailEnabled=true;
	groupTypes=Unified;
	visibility=Public
		officeGroupPublic

Microsoft 365 - Private	securityEnabled=false;
	mailEnabled=true;
	groupTypes=Unified;
	visibility=Private
		officeGroupPrivate

Microsoft 365 - HiddenMembership	securityEnabled=false;
	mailEnabled=true;
	groupTypes=Unified;
	visibility=HiddenMembership
		officeGroupHiddenMembership

2.12.3.3.1. Filtering Azure AD Objects

The Microsoft 365 connector supports filtering of accounts, groups, and roles channels via the Microsoft Graph **$filter** query parameter. Use DirX Identity Manager to set the filter in the respective channel. For example, you can define a filter to restrict synchronization to specific AD objects, as shown in the following dialog:

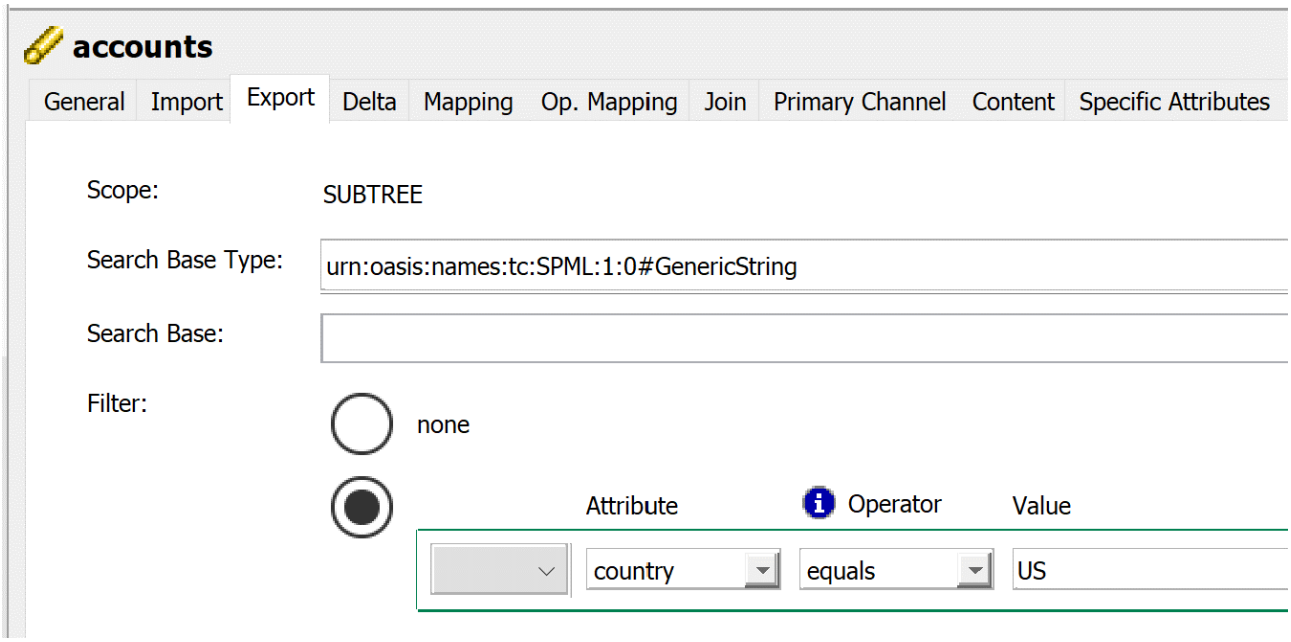


Figure 3. Setting a Filter for an Azure AD Object in DirX Identity Manager

The following limitations apply:

- The Microsoft Graph API for AD objects does not support all filter operators.
- The $filter query parameter does not support all AD object properties.
- Not all AD object properties support filter queries.
- In DirX Identity Manager, not all AD objects own the attributes defined in the attribute configuration file. Although you can select certain operators and attributes in the filter control, not all operators and attributes are supported.

For more information on Microsoft Graph query parameters, see the Microsoft document:

https://docs.microsoft.com/en-us/graph/query-parameters

2.12.3.4. Using the $filter Parameter on User and Group Resources

You can use the $filter query parameter on user and group resource types to retrieve:

- A subset of a collection
- Relationships, like **members**, **memberOf**, **transitiveMembers**, and **transitiveMemberOf**. For example, get all the security groups of which I am a member.

The following example uses the **startswith** $filter query function to find users whose display name starts with the letter "J":

HTTP GET https://graph.microsoft.com/v1.0/users?$filter=startswith(displayName,'J')

The following table shows currently supported and unsupported Microsoft Graph logical operators for Azure AD user and group resources and how they correspond to DirX Identity and DirX DSML operators:

Graph API	Azure AD Resources (User, Group)	DirX Identity Manager	DirX DSML
equals (eq)	$filter=givenName eq 'Max'	equals	<filter><dsml:equalityMatch name="givenName"><dsml:value>Smith</dsml:value>...
in (in)	unsupported	unsupported	unsupported
not equals (ne)	unsupported	not equals	<filter><dsml:not><dsml:equalityMatch name="givenName><dsml:value>Smith</dsml:value>...
greater than (gt)	unsupported	unsupported	unsupported
greater than or equals (ge)	$filter=createdDateTime ge 2020-08-01	is greater than or equal to	<filter><greaterOrEqual name="givenName"><dsml:value>Smith</dsml:value>...
less than (lt)	unsupported	unsupported	unsupported

Graph API	Azure AD Resources (User, Group)	DirX Identity Manager	DirX DSML
less than or equals (le)	$filter= createdDateTime le 2014-08-01	is less than or equal to	<filter><lessOrEqual name="givenName"><dsml:value>Smith</dsml:value>...
unsupported	unsupported	contains	<filter><dsml:substrings name="givenName"><dsml:any>Smith</dsml:any>...
startswith	$filter=startswith(givenName='Max')	begins with	<filter><dsml:substrings name="givenName"><dsml:initial>Smith</dsml:initial>...
unsupported	unsupported	ends with	<filter><dsml:substrings name="givenName"><dsml:final>Smith</dsml:final>...
unsupported	unsupported	is present	<filter><dsml:present name="givenName"/></filter>
and (and)	$filter=startswith(givenName='Max'') and startswith(surName='Smith')	and	<filter><dsml:and>...
or (or)	$filter=startswith(givenName='Max') or startswith(surName'Smith')	or	<filter><dsml:or>...
not (not)	unsupported	not	<filter><dsml:not>...

As shown in the table, the **$filter** operators **ne**, **gt**, **lt** and not are not supported for Azure AD resources. The **contains** string operator is currently not supported on any Microsoft Graph resource. For more information about about query parameters, see the Microsoft documentation at https://docs.microsoft.com/en-us/graph/query-parameters.

Not every Azure AD user and group object property supports filter query. Check the documentation for the resource to see which property is filterable:

User resource: https://docs.microsoft.com/en-us/graph/api/resources/user?view=graph-rest-1.0

Group resource: https://docs.microsoft.com/en-us/graph/api/resources/groups-overview?view=graph-rest-1.0

In these documents, only the properties marked with **Supports $filter** are supported in Microsoft Graph API.

2.12.3.5. Using the $filter Parameter on directoryRole Resources

The **directoryRole** resource type represents an Azure AD directory role (also called an administrator role).

Only the equality match filter operator is supported for this resource type. For example:

...directoryRole?$filter=displayname eq 'Helpdesk Administrator'

For more information on the directoryRole resource type, see the document:

https://docs.microsoft.com/en-us/graph/api/resources/directoryrole?view=graph-rest-1.0

2.12.3.6. Escaping Single Quotes

For requests that use single quotes, if any parameter values also contain single quotes, they must be double escaped; otherwise, the request will fail due to invalid syntax.

In the following example, the string value **let''s meet for lunch?** has the single quote escaped:

HTTP GET https://graph.microsoft.com/v1.0/me/messages?$filter=subject eq 'let''s meet for lunch?'

For more information on encoding query parameters, see the Microsoft document https://docs.microsoft.com/en-us/graph/query-parameters.

2.12.4. Paging

Paging for the Microsoft 365 connector is set to the following defaults in accounts, members, groups, plans and roles channels under the channel's Export tab:

pagedRead=true

timelimit=0

pageSize=100

Do not change these values for the Microsoft 365 connector.

Here is an example:

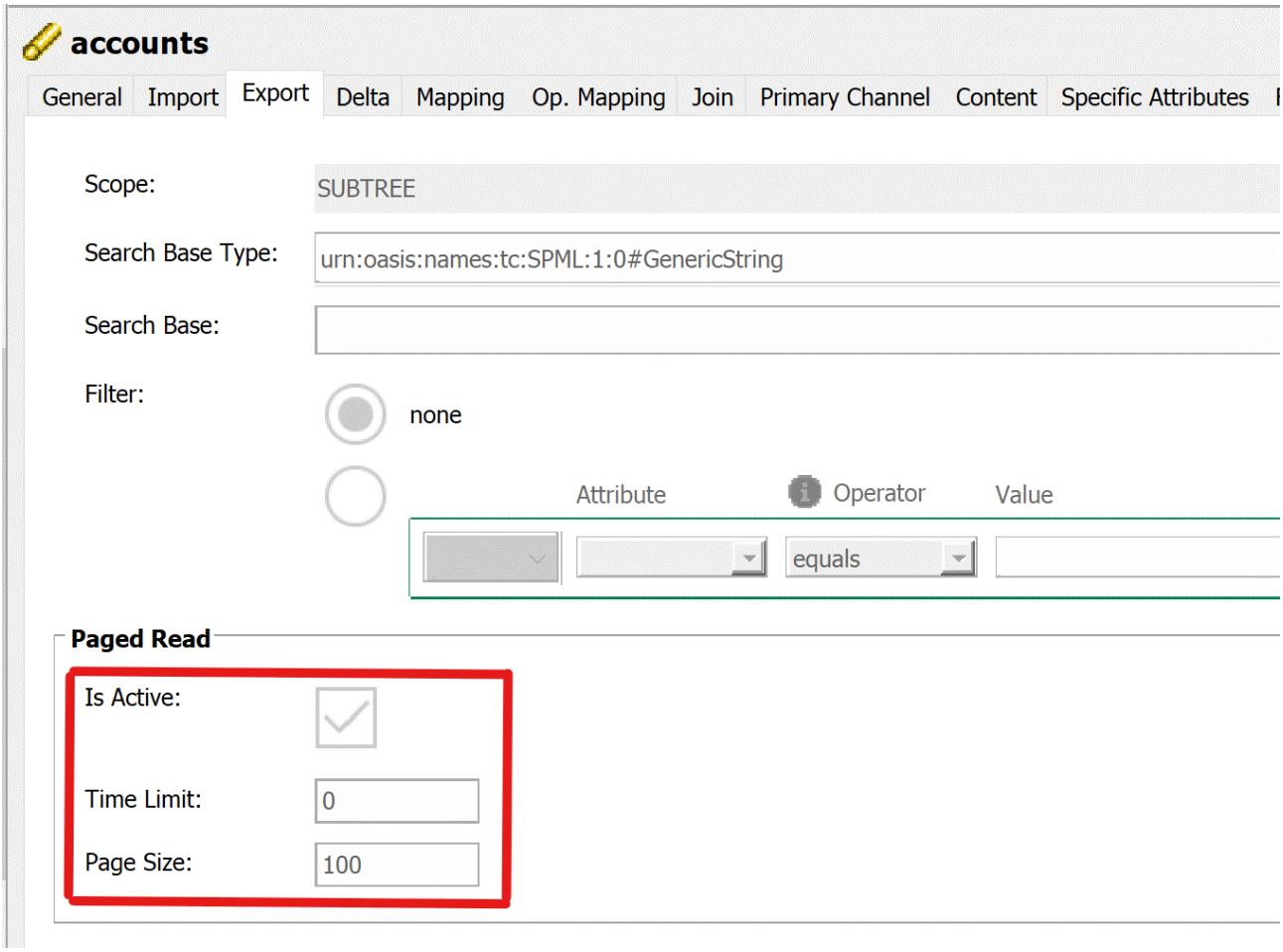


Figure 4. Microsoft 365 Connector Paging Defaults

2.13. OpenICF Connector

The Java-based OpenICF connector runs inside the Identity Java Connector Integration Framework. It communicates with an OpenICF connector server (Java- or .NET-based) using an internal OpenICF protocol. It dynamically converts SPMLv1 requests to OpenICF protocol operations, including automatic conversion of data types.

The connector is implemented in the class **OpenIcfConnector** in the package **net.atos.dirx.dxi.connector.openicf**.

The connector implements the common methods for the DirX Identity Connector API: add, modify, delete and search.

The operations are simply converted to the OpenICF API. The corresponding responses are again translated to SPMLv1 responses.

The connector can dynamically obtain information about required configuration parameters and data schema from a remote OpenICF server and its deployed OpenICF connector bundle.

The connector supports SSL/TLS authentication with the OpenICF server.

2.13.1. Prerequisites

SSL/TLS authentication requires the OpenICF server certificate to be trusted by the JRE used by the connector. Use a certificate issued by a trusted CA or use the Java **keytool** command (**keytool -importcert**) to import the server certificate into the DirX Identity JRE trust store (**cacerts**).

The connector requires the OpenICF connector framework bundle 1.1.1.0 or newer.

2.13.2. Configuration

The connector receives its configuration from the connector framework in a format that is specified there and reflects an XML document. Note that DirX Identity Manager presents configuration options in a more convenient way: bind credentials, SSL flag and service address are typically collected from appropriate LDAP entries found by selecting the appropriate connected directory and bind profile.

This section describes the configuration options based on the XML format. These options are either specified attributes in the XML schema of the element <connection> (referred to as standard properties) or can be specified as <property> sub-elements of the <connection> element (referred to as non-standard properties).

The connector uses two <connection> elements. The first element is related to the OpenICF connector server (type="OpenIcfServer"). The connector evaluates the following standard and non-standard properties for the OpenICF server:

Standard attributes:

server: required. This property provides information about the host name or IP address where an OpenICF connector server (Java- or .NET-based) is deployed. For example, **localhost**.

port: required. This property provides information about the port of an OpenICF connector server. For example, **8759**.

ssl: optional. This property enables SSL/TLS authentication to an OpenICF server and secures the communication line.

password: required; the password is used as a shared secret between OpenICF connector and an OpenICF connector server.

The OpenICF connector evaluates the following non-standard properties beneath the <connection> for the OpenICF server:

timeout: optional. This property provides the timeout in seconds for communication with OpenICF server. The default value is 60 seconds.

bundleName: required. This property provides the name of the OpenICF connector bundle deployed on an OpenICF server that we want to use. For example, **org.forgerock.openicf.connectors.solaris-connector**. An OpenICF connector bundle is fully identified by **bundleName**, **bundleVersion** and **implementationClassName**.

bundleVersion: required. This property provides the version of the OpenICF connector bundle deployed on an OpenICF server that we want to use. For example, **1.1.1.0-**

SNAPSHOT. See the **bundleName** property for more information.

implementationClassName: required. This property provides the fully-qualified name of the main entry class of an OpenICF connector bundle deployed on an OpenICF server that we want to use. For example, **org.identityconnectors.solaris.SolarisConnector**. See the **bundleName** property for more information.

configurationMapping: optional. This master property provides mapping of the standard property names to an OpenICF connector-specific format. For example, **user loginUser** automatically converts the standard configuration property name **user** to OpenICF format **loginUser**. This property allows the use of the standard DirX Identity support mechanism for special cluster workflow handling. The list can contain more values separated by commas. The conversion is valid for the configuration related to OpenICF connector bundle (type="OpenIcfConnector").

The second connection element is related to the OpenICF connector bundle (type="OpenIcfConnector"). Since the configuration of OpenICF connector bundles is for the most part very different for each bundle type, standard properties are not pre-defined. The DirX Identity connector evaluates all of the properties passed to the connection element, converts them to the appropriate type and then sends them as configuration properties to a remote OpenICF connector server. It is necessary to study the documentation for a specific OpenICF connector bundle and to define and deliver all necessary properties properly.

Here is a sample configuration that uses some of the properties described here:

```
<connector
className="net.atos.dirx.dxi.connector.openicf.OpenIcfConnector"
name="TS" role="connector">
<!-- settings for OpenICF server -->
<connection type="OpenIcfServer" server="ALFA" port="8759"
password="{SCRAMBLED}aG5WPw==" ssl="true">
<property name="timeout" value="60"/>
<property name="bundleName"
value="org.forgerock.openicf.connectors.solaris-connector"/>
<property name="bundleVersion" value="1.1.1.0-SNAPSHOT"/>
<property name="implementationClassName"
value="org.identityconnectors.solaris.SolarisConnector"/>
<property name="configurationMapping" value="server host,user
loginUser"/>
</connection>
<!-- settings for OpenICF connector bundle -->
<connection type="OpenIcfConnector" password="{SCRAMBLED}aG5WPw=="
server="someunixhost" user="root" port="22">
<property name="loginShellPrompt" value="#"/>
<property name="connectionType" value="ssh"/>
```

```
<property name="unixMode" value="linux"/>
</connection>
</connector>
```

2.14. OpenICF Windows Local Accounts Connector

The OpenICF Windows Local Accounts connector is implemented as a C#-based OpenICF .NET connector embedded and started by an OpenICF .NET connector server. The OpenICF .NET connector server receives requests from the DirX Identity OpenICF connector, which is a Java-based connector that conforms to the DirX Identity Java Connector Integration Framework and which sends the requests to the OpenICF connector server using an internal OpenICF protocol.

For a description of the Java-based DirX Identity OpenICF connector, see the chapter "OpenICF Connector" in this reference. This chapter also describes how to secure the connection from the Java-based OpenICF connector to the OpenICF connector server with SSL.

2.14.1. Overview

The OpenICF Windows Local Accounts connector is deployed as an OpenICF connector bundle to a .NET-based OpenICF connector server running on any Windows server.

On one side, it implements the OpenICF SPI operations Schema(), Create(), Update(), Delete(), CreateFilterTranslator() and ExecuteQuery() called by the OpenICF connector server that receives the corresponding SPML Add, Modify, Delete and Search requests by the Java-based OpenICF connector.

On the other side, it implements the System.DirectoryServices.AccountManagement API for accessing a Windows local accounts and groups database. The Account Management API is a .NET framework DirectoryServices namespace that provides uniform access and manipulation of user, computer and group security principals for three directory platforms: the Active Directory Domain Services, the local Security Account Manager (SAM) database on every Windows computer and the Active Directory Lightweight Directory Services (AD LDS).

The connector manages user and group objects of a SAM database located on any computer in the Windows network.

The Account Management API can only use the Windows NT LAN Manager (NTLM) protocol for authentication when accessing a SAM database. If user name and password are not provided for authentication, the security context of the calling thread (the account under which the connector server runs) is used for binding. The Account Management API also supports Kerberos or SSL authentication for accessing Active Directory.

The Account Management API does not provide any encryption protocol for the subsequent data transfer when accessing a SAM database. Hence the attributes and values of a create or modify request are not completely encrypted - as could be done by choosing Kerberos when accessing Active Directory. If a password is submitted in such a request it is

always encrypted as stated by Microsoft:

When changing or setting a user's password on a remote SAM DB with the AccountManagement API methods UserPrincipal::ChangePassword(oldPassword, newPassword) or UserPrincipal::SetPassword(newPassword) the function SamrUnicodeChangePasswordUser2 is called behind the scene, which encrypts the new password with a key from the hash of the old password or, in the latter case when no old password is provided, with an internal temporary key.

Moreover there is the possibility to secure the complete RPC/TCP connection, which is the underlying protocol used by the Account Management API by configuring IPSec. IPSec is a computer-wide setting that secures all IP traffic. It is not specific to an individual application like SSL, but it is transparent to applications.

2.14.2. Prerequisites

The OpenICF .NET connector server in the OpenICF Windows Local Accounts connector configuration has the following prerequisites:

- The connector server must be OpenICF .NET connector server version 1.4.0.0 or newer.
- The machine on which the connector server is installed must be running Windows Server 2008, Windows Server 2012, Windows 7 or Windows 8 and must have at least 20 MB of free disk space and 200 MB of available RAM.
- The .NET framework version 4.0 or newer must be installed on the machine where the connector server is installed.

To install the OpenICF .NET connector server:

- Download an OpenICF .NET connector server version 1.4.0.0 or newer from the Internet as a *ConnectorServer*.msi* installation file; for example, **openicf-1.4.0.0-SNAPSHOT-dotnet.msi**.
- Run the *ConnectorServer*.msi* installation file and then follow the wizard's instructions. When it completes, the connector server is now installed as the Windows service **ConnectorServerService**. The default installation location is **C:\Program Files\Identity Connectors\Connector Server**.

To configure the OpenICF .NET connector server:

- Set the key (shared secret key) for the connector server: navigate to the directory where the connector server is installed and then execute the following command in the MS/DOS shell:

  ConnectorServer /setkey *newkey*

  where *newkey* is the value for the connector server key. The same key (= password) must be configured in the OpenICF Windows Local Accounts connector configuration file in the **password** property of the connection section of type **OpenIcfServer** (see the configuration snippet given in the next step).
- Update the connector server configuration file **ConnectorServer.exe.Config**. This file is

located in the connector server installation directory and is an XML-formatted file. The most common items to change in this configuration file are the port number or the trace settings. For the trace settings, you can specify a log file name and a trace level (for example, **All**) in the <trace> section and then find the related logging of all OpenICF connector bundles deployed into the connector server in that log file, for example:

```
    <trace autoflush="true" indentsize="4">
     <listeners>
       <remove name="Default" />
       <!--add name="console" /-->
       <add name="myListener"
type="System.Diagnostics.TextWriterTraceListener"
initializeData="c:\Program Files (x86)\Identity Connectors\Connector
Server\connectorserver.log" traceOutputOptions="DateTime">
         <filter type="System.Diagnostics.EventTypeFilter"
initializeData="All"/>
       </add>
     </listeners>
   </trace>
```

The OpenICF Windows Local Accounts connector has the following prerequisites:

- The WindowsLocalAccounts.Connector-1.4.0.0.zip bundle, which is installed to the DirX Identity subfolder *install_path*\connectors\OpenICF\bundles\dotnet*, must be deployed (that is, unzipped) to the installation folder of the OpenICF .NET connector server. Restarting the OpenICF .NET connector server activates the new bundle..
- On the target Windows machines with the SAM databases to be managed, the Open ICF Windows Local Accounts connector can only access the SAM databases of the target Windows machines if they satisfy the following prerequisites:
- Windows Server 2012 R2 system - the Remote Registry Service must be started. This action is performed by default on this Windows version.
- Windows Server 2008 R2 system - the Remote Registry Service must be started (performed by default).
- Windows 7 Professional Client system - the Remote Registry Service must be started, (not performed by default) and the following Inbound Rules of the Windows Firewall configuration settings must be enabled: **Remote Service Management** (**NP-In**) for the Profile type **Domain**.
- Windows Server 2003 system - the Remote Registry Service must be started (performed by default) and the **File and Printer Sharing** services must be selected in the **Exceptions** tab of the **Windows Firewall** configuration settings.

There may be other Inbound Rules to be enabled or Firewall Exceptions to be set to allow SAM database access in addition to or instead of those described here if they cover the relevant port ranges required for the RPC traffic.

The installation of a .NET Framework is not necessary on a target machine.

If the Remote Registry Service is not started on a target machine, the Windows Local Accounts connector receives the error message "The network path was not found" when trying to perform add, modify, delete or search operations.

2.14.3. Limitations

The following limitations apply:

- Between the machine where the connector service runs and each remote target machine with a local SAM database to be managed, only one RPC connection for each target machine is valid. An additional RPC connection with different credentials than an existing RPC connection to the same target machine may fail due to the well-known Microsoft RPC limitation described under KB106211, KB173199, KB183366, KB824198. As a result, the connections to the target machines should not be built up with changing credentials.
- Due to the preceding limitation, the .NET connector server service must be started under an account with appropriate access rights to the remote target machine's SAM database (it must be a member of the target machine's Administrator group). As a result, no user and password must be specified for the Windows Local Accounts connector. If they are specified, they are passed to the Account Management API authentication method. However, they may not take effect because different credentials for a pre-existing connection to the same machine may already be in effect. If a user name beginning with **dummy** (case insensitive) is specified, the connector does not pass any credentials to the authentication method.
- For search requests only, the EqualityMatch filter for all attributes and the StartsWith filter for the naming attribute, which is the SamAccountName, are supported. The "SearchRequest" section provides an example.

2.14.4. Deployment

This section describes the following deployment scenarios:

- One .NET connector server for all Windows systems in one Windows domain
- One .NET connector server on each Windows target machine
- One .NET connector server for several Windows domains

2.14.4.1. One .NET Connector Server/One Windows Domain

To manage several Windows local systems joined to the same domain, only one .NET connector server needs to be installed on a Windows machine in the domain (see the Windows system requirements for the .NET connector server described in "Prerequisites"). It must be configured so that the .NET connector server service runs under a specific domain account which is added to the Administrator group of each targeted Windows machine in that domain.

On DirX Identity side, one Identity target system relates to one Windows local target system. All Identity target systems in this scenario bind to the same .NET connector server

(specified in the Windows Local Accounts Connector <connection> section of type **OpenIcfServer**), which itself then addresses the request to that Windows target machine that is specified in the Windows Local Accounts Connector <connection> section of type **OpenIcfConnector** (see the example given in the "Configuration" section).

2.14.4.2. One .NET Connector Server per Windows Target Machine

If the Windows target machine is not joined to a domain or for performance or network connectivity reasons, a .NET connector server can also be installed on a Windows target machine itself.

2.14.4.3. One .NET Connector Server/Several Windows Domains

If the domain account under which a .NET connector server service runs is known in other domains through trust relationships and can be added to the Administrator groups of target machines in other domains, those target machines can also be managed by a .NET connector server running in a different domain.

2.14.5. Request and Response Handling

This section describes the SPML requests processed by the Java-based DirX Identity OpenICF connector and the attributes supported by the C#-based OpenICF Windows Local Accounts connector.

Attribute names in uppercase characters with leading and trailing underscore (*) characters are predefined or operational attributes of the OpenICF connector framework. If there are also native attribute names for these attributes - for example, SamAccountName for* NAME *or Members for* ACCOUNTS *or Enabled for* ENABLE__ - both names are supported by the connector and can therefore be used.

In all requests, the operational attribute **objType** with the allowed values **user** or **group** (default = user) specifies whether the operation is performed for users or for groups.

2.14.5.1. AddRequest

The identifier in an add request for a user or a group is mandatory and will become the SamAccountName attribute.

Here is an example add request for a user:

```
<spml:addRequest xmlns="urn:oasis:names:tc:SPML:1:0"
       xmlns:spml="urn:oasis:names:tc:SPML:1:0"
       xmlns:dsml="urn:oasis:names:tc:DSML:2:0:core"
       requestID="add-01"
   <spml:identifier type="urn:oasis:names:tc:SPML:1:0#DN">
       <spml:id>MHummels</spml:id>
   </spml:identifier>
   <spml:operationalAttributes>
```

```
        <dsml:attr name="objType">
            <dsml:value>user</dsml:value>
        </dsml:attr>
    </spml:operationalAttributes>
    <spml:attributes>
        <spml:attr name="Enabled"
xmlns="urn:oasis:names:tc:DSML:2:0:core">
            <dsml:value>false</dsml:value>
        </spml:attr>
        <spml:attr name="__PASSWORD__">
            <dsml:value type="string">Dirx123#</dsml:value>
        </spml:attr>
        <spml:attr name="DisplayName"
xmlns="urn:oasis:names:tc:DSML:2:0:core">
            <dsml:value>Mats Hummels</dsml:value>
        </spml:attr>
    </spml:attributes>
</spml:addRequest>
```

Here is an example add request for a group:

```
<spml:addRequest xmlns="urn:oasis:names:tc:SPML:1:0"
          xmlns:spml="urn:oasis:names:tc:SPML:1:0"
          xmlns:dsml="urn:oasis:names:tc:DSML:2:0:core"
          requestID="add-01"
          >
     <spml:identifier type="urn:oasis:names:tc:SPML:1:0#DN">
          <spml:id>TestGroup2</spml:id>
     </spml:identifier>
     <spml:operationalAttributes>
          <dsml:attr name="objType">
               <dsml:value>group</dsml:value>
          </dsml:attr>
     </spml:operationalAttributes>
     <spml:attributes>
          <spml:attr name="Description"
xmlns="urn:oasis:names:tc:DSML:2:0:core">
              <dsml:value>Created with OpenICF</dsml:value>
          </spml:attr>
          <spml:attr name="Members"
```

```
xmlns="urn:oasis:names:tc:DSML:2:0:core">
              <dsml:value>agerbe66</dsml:value>
         </spml:attr>
    </spml:attributes>
</spml:addRequest>
```

2.14.5.2. ModifyRequest

In a modify request, the identifier is also mandatory and must be set to the SamAccountName value returned in the SPML AddResponse for the object.

Here is an example modify request for a user:

```
<spml:modifyRequest requestID="mod-1: set some attributes">
 <spml:identifier
     type = "urn:oasis:names:tc:SPML:1:0#DN">
     <spml:id>MHummels</spml:id>
 </spml:identifier>
   <spml:modifications>
     <spml:modification name="__GROUPS__" operation="replace">
       <dsml:value>TestGroup1</dsml:value>
     </spml:modification>
     <spml:modification name="DisplayName" operation="replace">
       <dsml:value>Mats Hummels</dsml:value>
     </spml:modification>
     <spml:modification name="Description" operation="replace">
       <dsml:value>Test account for OpenICF Windows Local Accounts
Connector</dsml:value>
     </spml:modification>
     <spml:modification name="Enabled" operation="replace">
       <dsml:value>True</dsml:value>
     </spml:modification>
     <spml:modification name="__CURRENT_PASSWORD__"
operation="replace">
       <dsml:value>Dirx123#</dsml:value>
     </spml:modification>
     <spml:modification name="__PASSWORD__" operation="replace">
       <dsml:value>Dirx456#</dsml:value>
     </spml:modification>
     <spml:modification name="PasswordNeverExpires"
operation="replace">
```

```
        <dsml:value>True</dsml:value>
      </spml:modification>
      <spml:modification name="HomeDirectory" operation="replace">
        <dsml:value>d:\MyTemp</dsml:value>
      </spml:modification>
    </spml:modifications>
</spml:modifyRequest>
```

Here is an example modify request for a group:

```
<spml:modifyRequest requestID="mod-1: add and delete members">
    <spml:identifier
         type = "urn:oasis:names:tc:SPML:1:0#DN">
         <spml:id>TestGroup2</spml:id>
    </spml:identifier>
    <spml:operationalAttributes>
         <dsml:attr name="objType">
              <dsml:value>group</dsml:value>
        </dsml:attr>
    </spml:operationalAttributes>
    <spml:modifications>
        <spml:modification name="Description" operation="replace">
              <dsml:value>Renamed</dsml:value>
       </spml:modification>
      <spml:modification name="Members" operation="add">
            <dsml:value>MHummels</dsml:value>
            <dsml:value>PwlTestUser</dsml:value>
      </spml:modification>
     <spml:modification name="Members" operation="delete">
            <dsml:value>agerbe66</dsml:value>
     </spml:modification>
    </spml:modifications>
</spml:modifyRequest>
```

2.14.5.3. DeleteRequest

In a delete request, the identifier is also mandatory. It must be the SamAccountName of the user or group object. The delete request does not require additional attributes.

2.14.5.4. SearchRequest

In an SPML search request, the OpenICF Windows Local Accounts connector supports the standard element filter, which can be of type EqualityMatch applicable on most attributes or of type Substring Initial applicable only on the naming attribute *NAME* (=SamAccountName). An example of each filter type is shown here. Regarding the attributes section, either all attributes, if none are specified, or the ones specified are retrieved.

Here is an example request that searches for all enabled users with the requested attributes. Note that with the attribute *GROUPS*, you can also retrieve the list of groups of which the user is a member.

```
<spml:searchRequest  xmlns="urn:oasis:names:tc:SPML:1:0"
    xmlns:spml="urn:oasis:names:tc:SPML:1:0"
    xmlns:dsml="urn:oasis:names:tc:DSML:2:0:core"
    requestID="search-user-1">
    <spml:operationalAttributes>
         <dsml:attr name="objType">
              <dsml:value>user</dsml:value>
         </dsml:attr>
    </spml:operationalAttributes>
    <filter>
          <dsml:equalityMatch name="__ENABLE__">
               <dsml:value type="string">true</dsml:value>
          </dsml:equalityMatch>
    </filter>
    <dsml:attributes>
         <dsml:attribute name="__NAME__"/>
         <dsml:attribute name="SamAccountName"/>
         <dsml:attribute name="DisplayName"/>
         <dsml:attribute name="__ENABLE__"/>
         <dsml:attribute name="__LOCK_OUT__"/>
          <dsml:attribute name="__DESCRIPTION__"/>
          <dsml:attribute name="PasswordNeverExpires"/>
          <dsml:attribute name="HomeDirectory"/>
          <dsml:attribute name="__GROUPS__"/>
     </dsml:attributes>
</spml:searchRequest>
```

Here is an example request that searches for all groups with SamAccountName beginning with “Test”:

```
<spml:searchRequest  xmlns="urn:oasis:names:tc:SPML:1:0"
    xmlns:spml="urn:oasis:names:tc:SPML:1:0"
    xmlns:dsml="urn:oasis:names:tc:DSML:2:0:core"
    requestID="search-groups-1">
    <spml:operationalAttributes>
         <dsml:attr name="objType">
              <dsml:value>group</dsml:value>
         </dsml:attr>
    </spml:operationalAttributes>
    <filter>
         <dsml:substrings name="__NAME__">
              <dsml:initial>Test</dsml:initial>
         </dsml:substrings>
    </filter>
    <dsml:attributes>
         <dsml:attribute name="__NAME__"/>
         <dsml:attribute name="SamAccountName"/>
         <dsml:attribute name="Description"/>
         <dsml:attribute name="Members"/>
    </dsml:attributes>
</spml:searchRequest>
```

2.14.6. Configuration

Here is a sample configuration snippet for the OpenICF Windows Local Accounts connector:

```
<connector
  role="connector"
className="net.atos.dirx.dxi.connector.openicf.OpenIcfConnector"
name="ts" version="1.00">
   <connection type="OpenIcfServer"
       server="WindowsServer01"
       port="8759"
       password="pwd for connector server"
    >
       <property name="bundleName"
value="WindowsLocalAccounts.Connector"/>
       <property name="bundleVersion" value="1.4.0.0"/>
       <property name="implementationClassName"
```

```
value="Org.IdentityConnectors.WindowsLocalAccounts.WindowsLocalAccoun
tsConnector"/>
   </connection>
   <!-- Windows LA Connector specific properties -->
   <connection type="OpenIcfConnector">
       <property name="user" value="AdminIcfConnector"/>
       <property name="password" value="pwd local computer admin"/>
       <property name="host" value="TargetSAM01"/>
   </connection>
</connector>
```

The following properties of the OpenICF .NET connector server <connection> section can be specified:

- **port** - the port number of the connector server.
- **server** - the server name or IP address of the connector server.
- **password** - the key value for binding to the connector server. It is the key string value that was applied when running the command **ConnectorServer /setkey** *keyvalue* in the connector server installation folder before starting the server for the first time.
- **bundleName** - the name of the connector bundle running in the connector server.
- **bundleVersion** - the version number of the connector bundle.
- **implementationClassName** - the connector class name where the OpenICF SPI methods are implemented.

The following properties can be specified in the <connection> section of the OpenICF Windows Local Accounts connector:

- **user** - the administrator name of the Windows Local Accounts (SAM) database.
- **password** - the password of the administrator.
- **host** - the name of the computer whose SAM database is to be managed.

2.15. RACF Connector

The Java-based RACF connector extends the Java-based LDAP Connector. It provisions the RACF system through the IBM Tivoli Directory Server for z/OS. See IBM's web page https://www.ibm.com/docs/en/zos/2.5.0?topic=tivoli-directory-server-zos for more information on the Tivoli Directory Server.

The connector evaluates the same configuration properties as the LDAP Connector.

The connector is implemented in the class siemens.dxm.connector.racf.RacfConnector.

It implements the following functional changes compared to the LDAP connector:

- User-group memberships are managed in extra connect entries in RACF.
- User default groups are set by the connector as calculated by the userhook of the workflow's accounts channel.
- Disabling / enabling a RACF user is realized by setting the appropriate values in the attribute "racfAttributes".
- For resetting an existing password, the connector first sets the new password in the attribute racfPassword of the RACF user and then performs an extra bind operation with this user, providing the old and the new password.
- Binding to the RACF system can be certificate-based (SASL bind) in the same way as for the LDAP connector.

2.15.1. Prerequisites

The RACF connector has the following prerequisites:

- The connector accesses a z/OS or OS/390 RACF system via the LDAP protocol. Therefore, a separate IBM Tivoli Directory Server is required per RACF system.
- An LDAP service account must be set up in the RACF database to be able to administer all users and groups. This user needs the RACF authorization "advanced".

2.15.2. Limitations

The RACF connector has the following limitations:

- The connector does not support nested groups. Nested group assignments cannot be read nor written.
- The workflow and the connector do not handle the RACF group member limit for groups that are not default groups.

2.15.3. Limitations of RACF via LDAP (SDBM)

The IBM LDAP access to RACF (via the SDBM backend) imposes some limitations regarding filters, returned attributes and number of returned entries. For details, see the IBM documentation, for example: https://www.ibm.com/docs/en/zos/2.5.0?topic=behavior-sdbm-search-capabilities.

2.15.4. Sample Requests

For sample requests, see the chapter on the LDAP Connector. This chapter contains just a few samples to highlight aspects specific to RACF.

In the RACF Tivoli Directory, users, groups, and connect objects are typically in their own sub-trees:

- Users in profiletype=USER, ...
- Groups in profiletype=GROUP, ...
- Connect objects in profiletype=CONNECT, ...

2.15.4.1. Search Request

The following sample request searches for a single user identified by its racfid and lists the attributes to be returned.

Note that for filtering only a subset of attributes can be used. See the RACF documentation for details.

```
<spml:searchRequest xmlns:dsml="urn:oasis:names:tc:DSML:2:0:core"
                xmlns:spml="urn:oasis:names:tc:SPML:1:0">
     <spml:searchBase
type="urn:oasis:names:tc:SPML:1:0#GenericString">
         <spml:id>profiletype=USER,cn=RACF,o=someNS</spml:id>
     </spml:searchBase>
     <filter>
            <dsml:equalityMatch name="racfid">
                    <dsml:value>UZ00001</dsml:value>
            </dsml:equalityMatch>
     </filter>
     <spml:attributes>
              <dsml:attribute name="racfid"/>
              <dsml:attribute name="racfprogrammername"/>
              <dsml:attribute name="racfattributes"/>
     </spml:attributes>
     <spml:operationalAttributes>
              <spml:attr name="scope">
                      <dsml:value>subtree</dsml:value>
              </spml:attr>
     </spml:operationalAttributes>
</spml:searchRequest>
```

2.15.4.2. Modify Membership and Enable a RACF User

The following request re-enables an account and adds a group membership. Note that the enable / disable is performed via RACF attribute racfAttributes and that it is enough to manage the memberships in the artificial user attribute “member”. The connector performs the appropriate changes in the connect entries in the sub-tree profiletype=CONNECT.

```
<spml:modifyRequest>
     <spml:identifier type = "urn:oasis:names:tc:SPML:1:0#DN">

<spml:id>racfid=UIATES1,profiletype=USER,o=someNS</spml:id>
```

```
     </spml:identifier>
     <spml:modifications>
               <dsml:modification name="racfAttributes"
operation="replace">
                         <dsml:value>RESUME</dsml:value>
               </dsml:modification>
               <dsml:modification name="member" operation="add">

<dsml:value>racfid=GIAMTES1,profiletype=GROUP, o=someNS</dsml:value>
               </dsml:modification>
     </spml:modifications>
</spml:modifyRequest>
```

2.15.4.3. Change a Password

The following request changes the password for a RACF user. Note that the old password must be provided as the operational attribute “currentpassword”.

```
<spml:modifyRequest>
     <spml:identifier type = "urn:oasis:names:tc:SPML:1:0#DN">

<spml:id>racfid=UIATES1,profiletype=USER,o=someNS</spml:id>
     </spml:identifier>
     <spml:modifications>
              <dsml:modification name="racfPassword"
operation="replace">
                       <dsml:value>the-new-password</dsml:value>
              </dsml:modification>
</spml:modifications>
     <spml:operationalAttributes>
              <spml:attr name="currentpassword">
                       <dsml:value>the-old-password</dsml:value>
              </spml:attr>
     </spml:operationalAttributes>
</spml:modifyRequest>
```

2.16. Remote AD Connector

DirX Identity can be deployed as part of the Atos Cloud Service Identity Management as a Service (IDMaaS). Provisioning targets can be in the provider’s (Atos) cloud infrastructure, in a public cloud or on customer premises outside of a cloud; the Remote AD connector is

intended for use in this last scenario.

The Remote AD connector's provisioning components must be able to work without online (LDAP) connection to the DirX Identity configuration database.

A standard framework-based agent job implements the export-to file-action on the customer side. It obtains its configuration from XML files: one for the job and one for the search request with the filter conditions. Importing the file to DirX is performed by an existing LDIF import workflow. The following figure illustrates the Remote AD connector architecture.

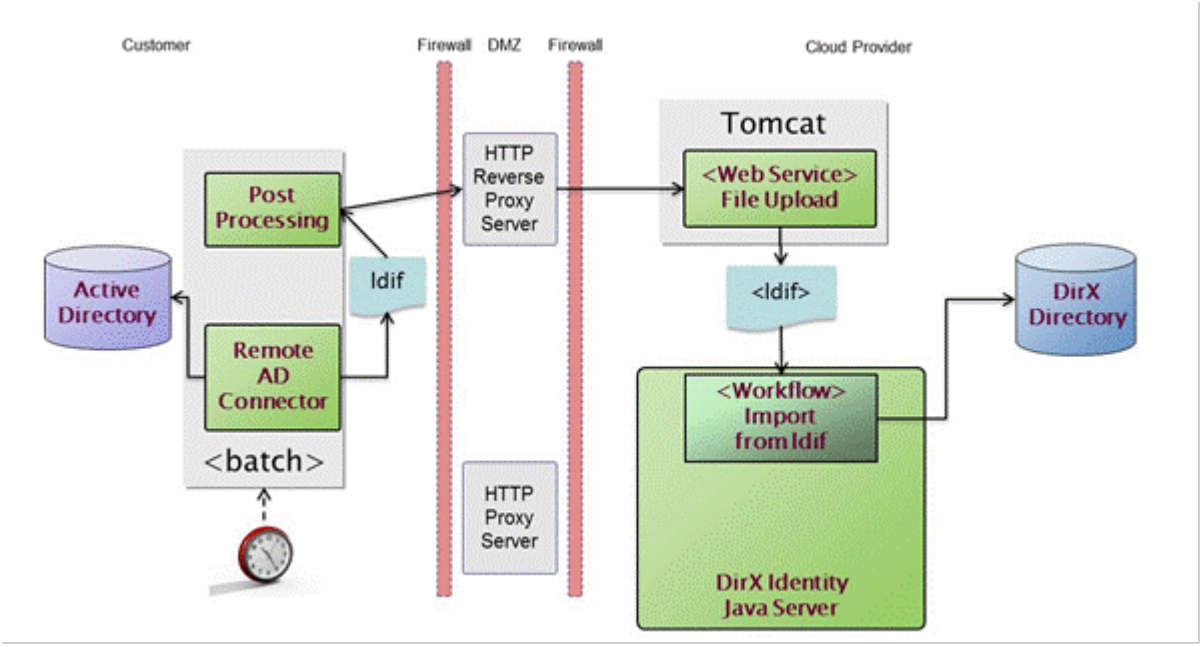


Figure 5. Remote AD Connector Architecture

2.16.1. Security Considerations

Transport connections between the customer and DirX Identity in the cloud should be secured by SSL/TLS.

2.16.2. Requirements and Limitations

To deploy / run the service / client, the following requirements apply:

- Java 7 installed on the customer side
- Apache Tomcat 7 installed on the cloud provider side

It is assumed that inbound traffic into the cloud is routed through a reverse HTTP proxy, which basically translates IP addresses. Outbound traffic will also be routed through an HTTP proxy. As a result, provisioning using native interfaces such as LDAP and JDBC are not allowed and are replaced by HTTP-based SOAP protocol (web service).

Access to the customer premises is expected to be strictly constrained: no remote installation, no reading / writing of (workflow) configuration in LDAP, no messaging.

Authentication to DirX Identity and to Active Directory: authentication for the standard connectors is based on user / password. Authentication via user certificates is not supported.

Encryption of passwords stored in local configuration files: in standard configurations with workflows hosted by a Java-based Server, the passwords used for authentication of the connectors are encrypted on transport. Framework-based standalone jobs as described here lack this feature: they do not know the system PIN for decryption of the system certificate and they do not have access to LDAP, where the private key is usually stored.

2.16.3. Remote AD Agent

The Remote AD agent is a client application for the File Upload Web Service, delivered as jar archives, configuration files and batch file(s). The Remote AD connector can be started by an operating system scheduler.

2.16.3.1. Activities

The agent application performs the following activities:

- Starts the job that connects to Active Directory and searches and exports users to an LDIF file.
- Packs the LDIF file and sends it to the File Upload Web service.
- Deletes the exported LDIF file if the transport is successful.

2.16.3.1.1. The Export-AD-to-File Job

The agent exports users from the Active Directory (AD) domain according to a configurable search request and then writes the resulting entries to an LDIF file. Typically not all Active Directory users need to be imported to DirX. You can filter them according to their organizational unit or other attributes or on group memberships.

The export-AD-to-file job is based on the DirX Identity Connector Integration Framework and is illustrated in the following figure. The default standalone controller reads its configuration from an XML file. It receives a search request from the SPML file reader, forwards it to the AD connector and then passes the search response to the LDIF connector, which writes it to a local LDIF file.

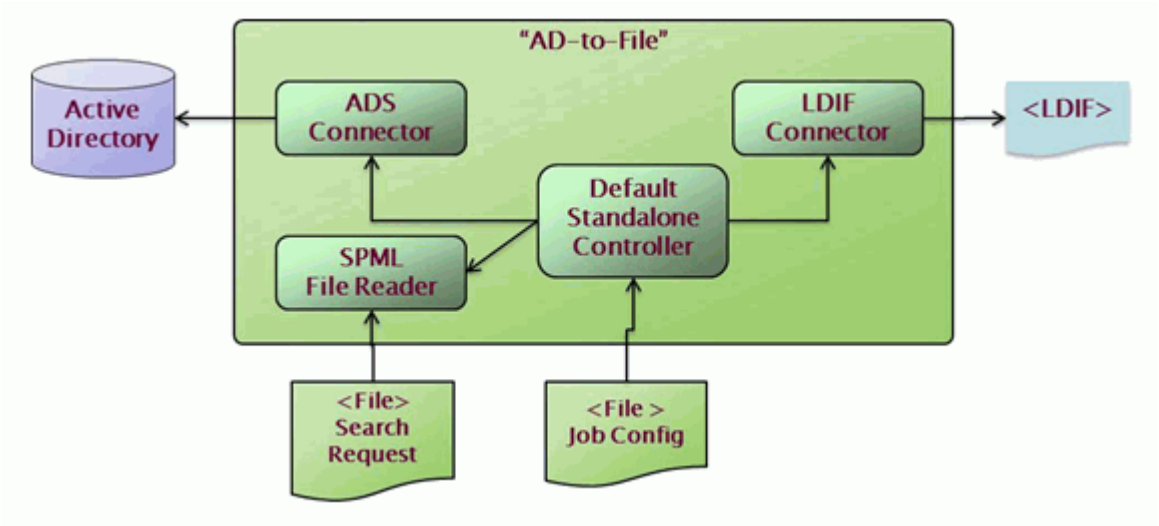


Figure 6. Export-AD-to-File Job

2.16.3.2. Installation

The application requires no installation. To start the agent, run the provided batch file. On the command line, you can specify the trust store path (for example, **cacerts**) to use when connecting to a secure endpoint. See the content of the batch file **remoteADAgent.bat** for details.

2.16.3.3. Configuration

You configure the Remote AD agent with the following files located in the **config** folder:

- The **wsConfig.properties** contains Web service-related parameters.
- The **jobConfig.xml** contains the job configuration of the AD export workflow, especially the AD connectivity parameters.
- The **searchRequest.xml** contains detailed configuration of the search request performed in the AD, especially filtering and returned AD attributes.

The batch file configures logging and trust store location. See the provided files for details.

2.16.4. File Upload Web Service

The File Upload Web service resides in a servlet container (for example, a standalone Tomcat or IdS-J tomcat) on the cloud provider side. Requests to the Web service consist of:

- Customer ID <_xs_:_string_>
- Content of the LDIF file <_xs_:_base64Binary_>

2.16.4.1. Activities

The Web service performs the following activities on the cloud provider side:

- Unpacks the received LDIF file and saves it under the configured folder.
- Invokes a Java-based workflow that corresponds to the received Customer ID. This call is

asynchronous, so there is no check to verify the result. The workflow imports the users from the LDIF file into the Identity store.

- Returns a response message to the client if no problems occurred or generates an error response.

2.16.4.2. Installation

To install the File Upload Web service, locate the **war** file under the **webapps** directory of Apache Tomcat. Tomcat automatically deploys the file. After deployment, adapt the configuration files in the newly created **remoteADFileUpload** folder and then restart Tomcat. The endpoint URL of the Web service is

http(s)://*host*:_port_**/remoteADFileUpload/ProvisionFileUploadService**

Where the *host*, *port* and usage of HTTP(s) depends on the Tomcat configuration, as described in the "Configuration" section.

2.16.4.3. Configuration

You configure the Web service with configuration files located under the **WEB-INF/config** folder. These files include:

- **config.properties** - the default configuration file that is used when a request without a client id is received or if no configuration file for the given customer ID is found.
- *customerID**.config.properties* - customer-specific configuration files. These files are used when an incoming customer ID matches an existing customer configuration file.

The default deployment contains the files **config.properties** and **My-Company.config.properties**. (See the files for more details, search for the CONFIGURE tag.)

2.16.4.3.1. Configuring SSL on Tomcat

The configuration parameters for Tomcat are specified in the file **/conf/server.xml**.

To enable SSL:

- Generate / copy the key store file under **CATALINA_BASE/keys/keystore**.
- Edit the following lines in **server.xml** to enable SSL:

```
<Connector SSLEnabled="true" acceptCount="100" clientAuth="false"
       disableUploadTimeout="true" enableLookups="false"
maxThreads="25"
       port="443" keystoreFile="keys/.keystore" keystorePass="asd123"
       protocol="org.apache.coyote.http11.Http11NioProtocol"
scheme="https"
       secure="true" sslProtocol="TLS" />
```

To enable LDAP authentication and authorization:

- Edit the following Realm element under **<Host/>**:

```
<Realm className="org.apache.catalina.realm.JNDIRealm"
       connectionName="cn=DomainAdmin,cn=My-Company"
       connectionPassword="dirx"
       connectionURL="ldap://localhost:389"
       userPattern="{0}"
       roleBase="cn=Groups,cn=DirXmetaRole,cn=TargetSystems,cn=My-
Company"
       roleSubtree="true"
       roleName="cn"
       roleSearch="(uniqueMember={0})"/>
```

- Disable other realms that are not required.

The meaning of the parameters is as follows:

roleBase
Specifies the entry under which the groups are located in the LDAP directory.

connectionName and **connectionPassword**
Identify a technical user. Tomcat uses this technical user in the bind for the LDAP group search. If an anonymous bind is performed, these parameters are omitted. Currently Tomcat cannot use the authenticated user for performing the search for groups.

2.16.4.3.2. Configuring Authorization Based on Group

To configure a group against which authorization needs to be performed, define the group name under **<security-constraint />** and **<web-app />** elements of the Tomcat **web.xml** file. For example:

```
<auth-constraint>
        <role-name>TSAdmins</role-name>
</auth-constraint>
```

2.17. Request Workflow Connector

The Java-based Request Workflow connector is built with the Identity Java Connector Integration Framework. It sends SOAP requests over HTTP to the configured DirX Identity endpoint and receives SOAP responses from the SOAP service.

The connector is implemented in the class **ReqWfConnector** in the package **com.siemens.dxm.connector.reqwf**.

The connector implements the common methods for the DirX Identity Connector API: add, modify, delete and search.

The add, modify and delete methods create an appropriate request workflow subject type and invoke a new request workflow instance. The search method simply returns success and an empty list of entries.

The connector supports basic authentication as well as server and client-side SSL/TLS authentication. It does not support WS-Security protocols yet.

The main goal of the connector is to create new request workflow instances for delivered SPMLv1-based requests for account and group objects.

2.17.1. Prerequisites

The connector is based on the Identity Java Connector Integration framework. The framework is contained in the library **dxmConnector.jar**.

It uses the common JAX-WS framework for sending and receiving SOAP requests and responses over HTTP. Usage of a JRE 1.7 is required to be able to run the connector.

2.17.2. Configuration

The connector receives its configuration by the connector framework in a format that is specified there and reflects an XML document. Note that Identity Manager presents configuration options in a more convenient manner. Especially bind credentials, SSL flag and service address are typically collected from appropriate LDAP entries found by selecting the appropriate connected directory and bind profile.

The following text discusses the configuration options based upon the XML format. These options are either specified attributes in the XML schema of the element <connection> (referred to as *standard properties*) or can be specified as <property> sub-elements of the <connection> element (referred to as *non-standard properties*).

The connector evaluates the following standard and non-standard properties:

Standard attributes:

server (mandatory)
: this property provides the server part of the endpoint URL.

  Example: **`localhost`**

port (mandatory)
: this property provides the port of the endpoint URL.

  Example: **`4000`**

ssl (optional)
: If no URL is given, this property defines which protocol to use. If **true**, **https** is selected; otherwise the connector sets **http**.

user (mandatory)

the username used for HTTP basic authentication. These credentials are used to authenticate for request workflow creation. Use a DirX Identity user with sufficient access rights; for example, the DomainAdmin.

password (mandatory)

the password used for HTTP basic authentication along with the user property.

The connector evaluates the following non-standard properties beneath the <connection> element:

path (mandatory)

this property provides the path of the URL.

Example: **`workflowService/services/WorkflowService`**

timeout (optional)

the socket read timeout in seconds. The default is **0** seconds, which indicates infinite.

domain (optional)

use this property in an environment with multiple Provisioning domains. Use it to check the connected Provisioning domain name.

Example: **`cn=My-Company`**

primaryWorkflowDN (optional)

this property can be used to specify the DN of the request workflow which will be used for account objects. It will be also used for group objects if no **secondaryWorkflowDN** is configured. If no **primaryWorkflowDN** option is configured, then the "When applicable" section of the active request workflow is evaluated and a suitable request workflow definition is instantiated.

secondaryWorkflowDN (optional)

use this property to specify the DN of the request workflow which will be used solely for group objects. If it is missing, the **primaryWorkflowDN** is used also for group objects. The **secondaryWorkflowDN** option is ignored if no **primaryWorkflowDN** is configured.

Here is a configuration sample using some of the described properties:

```
<connector name="TS" role="connector"
className="com.siemens.dxm.connector.reqwf.ReqWfConnector">
<connection
  type="RequestWorkflow"
  server="localhost"
  port="40000"
  ssl="FALSE"
  user="cn=DomainAdmin,cn=My-Company"
```

```
  password="{SCRAMBLED}aG5WPw==">

  <property name="path"
    value="workflowService/services/WorkflowService"/>
  <property name="domain" value="cn=My-Company"/>
  <property name="primaryWorkflowDN"
    value="cn=Manual Provisioning,cn=Service
    Management,cn=Default,cn=Definitions,cn=wfRoot,cn=My-Company"/>

  <property name="secondaryWorkflowDN" value=""/>

  <property name="timeout" value=""/>
</connection>
</connector>
```

2.18. Salesforce Connector

The Java-based Salesforce connector is built with the Identity Java Connector Integration Framework and uses the REST framework and its APIs.

2.18.1. Overview

The Salesforce connector implements the API methods "add(...)", "modify(...)", "delete(...)" and "search(...)". They represent the corresponding SPML requests"‘AddRequest", "ModifyRequest", "DeleteRequest" and "SearchRequest".

Currently it only supports Account, Contact, User, PermissionSet and Profile objects in Salesforce.

The Salesforce connector offers the following functionality:

- Add an account, a contact or a user to Salesforce
- Delete an account, a contact or a user from Salesforce
- Modify an account, a contact or a user in Salesforce
- Modify a permission set or a profile in Salesforce (attribute **Description** only)
- Search accounts, contacts, users, permission sets and profiles in Salesforce

2.18.2. Prerequisites and Limitations

The Salesforce connector has the following limitations:

- Users cannot be physically deleted in Salesforce. As a result, the Delete operation only sets the **IsActive** attribute to **false** and the customer-specific attribute **StatusInfo__c** to DELETED.

- New profiles can’t be created or deleted using the REST APIs. Only a Modify operation is supported; for example, modifying the **Description** attribute of a profile.

2.18.3. Request and Response Handling

This section describes the supported attributes and requests for the Salesforce connector.

The following sections provide the supported attributes.

2.18.3.1. Supported Account Attributes

- AccountNumber
- AnnualRevenue
- BillingCity
- BillingCity
- BillingCountry
- BillingPostalCode
- BillingState
- BillingStreet
- Description
- Fax
- Id - the identifier of the account; is returned as SPML identifier in the SPML ADD resonse; must be used as SPML identifier in an SPML MODIFY or DELETE request or in SPML SEARCH request if a single object is searched.
- Industry
- IsCustomerPortal - read only
- IsDeleted - read only
- IsPartner - read only
- Name
- OwnerId - reference to a Salesforce user that is defined if the account is used for defining a Customer Portal User.
- Ownership
- Phone
- PhotoUrl
- Rating
- Site
- ShippingCity
- ShippingCountry
- ShippingPostalCode

- ShippingState
- ShippingStreet
- Type
- Website

2.18.3.2. Supported Contact Attributes

- AccountId - reference to a Salesforce account that is defined if the contact is used for defining a Customer Portal User.
- AssistantName
- Birthdate
- Department
- Description
- Email
- Fax
- FirstName
- Id - the identifier of the contact; is returned as SPML identifier in the SPML ADD resonse; must be used as SPML identifier in an SPML MODIFY or DELETE request or in SPML SEARCH request if a single object is searched
- IsDeleted - read only
- HomePhone
- LastName
- LeadSource
- MailingCity
- MailingCountry
- MailingPostalCode
- MailingState
- MailingStreet
- MobilePhone
- Name
- OtherCity
- OtherCountry
- OtherPhone
- OtherPostalCode
- OtherState
- OtherStreet
- OwnerId - reference to a Salesforce user that is defined if the contact is used for defining a Customer Portal User.

- Phone
- PhotoUrl
- Salutation
- Title

2.18.3.3. Supported Permission Set Attributes

- Description
- Id - the identifier of the permission set; is returned as SPML identifier in the SPML ADD resonse; must be used as SPML identifier in an SPML MODIFY or DELETE request or in SPML SEARCH request if a single object is searched.
- LicenseId
- Name
- ProfileId

2.18.3.4. Supported Profile Attributes

- Description
- Id - the identifier of the profile; is returned as SPML identifier in the SPML ADD resonse; must be used as SPML identifier in an SPML MODIFY or DELETE request or in SPML SEARCH request if a single object is searched.
- Name
- UserLicenseId
- UserType

2.18.3.5. Supported User Attributes

- Alias - mandatory in ADD operations.
- City
- CommunityNickName - mandatory in ADD operations; must be unique.
- CompanyName
- Country
- Department
- Division
- Email - mandatory.
- EmailEncodingKey - mandatory in ADD operations.
- EmployeeNumber
- Extension
- Fax
- FirstName

- Id - the identifier of the user; is returned as SPML identifier in the SPML ADD resonse; must be used as SPML identifier in an SPML MODIFY or DELETE request or in SPML SEARCH request if a single object is searched.
- IsActive
- LanguageLocaleKey - mandatory in ADD operations.
- LastName
- LocaleSidKey - mandatory.
- MobilePhone
- Password
- Phone
- ProfileId - mandatory.
- PostalCode
- State
- StatusInfo__c - customer-specific attribute.
- Street
- TimeZoneSidKey - mandatory in ADD operations.
- Title
- Username - mandatory in ADD operations; must be, unique and in the form of an e-mail address (for example, john@acme.com).

2.18.3.6. Operational Attributes

All SPML requests contain a section for operational attributes. In this section, you specify the object type for your SPML request.

In the operational attribute **objtype** you can use the following values:

- **Account** for Salesforce account objects
- **Contact** for Salesforce contact objects
- **PermissionSet** for Salesforce permission set objects
- **Profile** for Salesforce profile objects
- **User** for Salesforce user objects

Here is a sample operational attribute section for handling a Salesforce user:

```
<spml:operationalAttributes>
   <spml:attr name="objtype">
        <dsml:value type="string">User</dsml:value>
   </spml:attr>
</spml:operationalAttributes>
```

Note that for all requests, you must specify the OperationalAttributes section and define the kind of object.

In AddRequest, no Spml-Identifier is set. The Spml-Identifier of the new object that has been created in Salesforce and returned in the AddResponse.

All other requests use the Spml-Identifier in the request. For search operations, the Smpl-Identifier is optional. If you omit it, you should set the **scope** operational attribute to **subtree** to initiate a search with filter.

The following sections describe the operation details.

2.18.3.7. AddRequest

The following example request adds a user object:

```
<spml:addRequest  xmlns="urn:oasis:names:tc:SPML:1:0"
                            xmlns:spml="urn:oasis:names:tc:SPML:1:0"

xmlns:dsml="urn:oasis:names:tc:DSML:2:0:core"
                            requestID="add_01">
<spml:operationalAttributes>
    <spml:attr name="objtype">
         <dsml:value type="string">user</dsml:value>
    </spml:attr>
</spml:operationalAttributes>
<spml:attributes>
    <dsml:attr name="Username">
         <dsml:value>Miller.Tom@My-Company.com</dsml:value>
    </dsml:attr>
    <dsml:attr name="LastName">
         <dsml:value>Miller</dsml:value>
    </dsml:attr>
    <dsml:attr name="FirstName">
         <dsml:value>Tom</ dsml:value>
    </dsml:attr>
    <dsml:attr name="CompanyName">
         < dsml:value>My-Company</dsml:value>
    </dsml:attr>
    <dsml:attr name="Department">
         <dsml:value>Sales</dsml:value>
    </dsml:attr>
    <dsml:attr name="City">
         <dsml:value>Munich</dsml:value>
```

```
    </dsml:attr>
    <dsml:attr name="Country">
         <dsml:value>US</dsml:value>
    </dsml:attr>
    <dsml:attr name="EmployeeNumber">
         <dsml:value>1234</dsml:value>
    </dsml:attr>
    <dsml:attr name="Alias">
         <dsml value>TMill</dsml value>
    </dsml:attr>
    <dsml:attr name="Password">
         <dsml value>dirxdirx1</dsml value>
    </dsml:attr>
    <dsml:attr name="IsActive">
         <dsml value>true</dsml value>
    </dsml:attr>
    <!-- Chatter Free User -->
    <dsml:attr name="ProfileId">
         <dsml:value>00ei0000001QzcCAAS</ dsml:value>
    </dsml:attr>
    <dsml:attr name="EmailEncodingKey">
         <dsml:value>ISO-8859-1</dsml:value>
    </dsml:attr>
    <dsml:attr name="TimeZoneSidKey">
         <dsml:value>America/Los_Angeles</dsml:value>
    </dsml:attr>
    <dsml:attr name="LocaleSidKey">
         <dsml:value>en_US</dsml:value>
    </dsml:attr>
    <dsml:attr name="LanguageLocaleKey">
         <dsml:value>en_US</dsml:value>
    </dsml:attr>
    <dsml:attr name="Email">
         <dsml:value>Miller.Tom@My-Company.com</dsml:value>
    </dsml:attr>
    <dsml:attr name="CommunityNickname">
         <dsml:value>Tom-Miller-1</dsml:value>
    </dsml:attr>
</spml:attributes>
</spml:addRequest>
```

2.18.3.8. ModifyRequest

The (user) modify request modifies a user in Salesforce. The same attributes as in AddRequest are supported.

The (profile) modify request modifies a profile in Salesforce. The only attribute that can be modified is **Description**.

The (permission set) modify request modifies a permission set in Salesforce. The only attribute that can be modified is **Description**.

The following example request modifies a user object:

```
<spml:modifyRequest  xmlns="urn:oasis:names:tc:SPML:1:0"
       xmlns:spml="urn:oasis:names:tc:SPML:1:0"
      xmlns:dsml="urn:oasis:names:tc:DSML:2:0:core"
       requestID="mod_02"
       >
    <spml:identifier
        type="urn:oasis:names:tc:SPML:1:0#DN">
        <spml:id>005i0000003F84dAAC</spml:id>
    </spml:identifier>
   <spml:operationalAttributes>
       <spml:attr name="objtype">
           <dsml:value type="string">user</dsml:value>
       </spml:attr>
   </spml:operationalAttributes>
   <spml:modifications>
<spml:modification name="Title" operation="replace">
   <dsml:value>Dr.</dsml:value>
</spml:modification>
<spml:modification name="City" operation="replace">
   <dsml:value>Munich</dsml:value>
</spml:modification>
    </spml:modifications>
</spml:modifyRequest>
```

2.18.3.9. DeleteRequest

The delete request is used to delete an object from a Salesforce site.

Important: The identifier for each delete request must be set to the group name.

The delete request does not require additional attributes.

The following example request deletes a user object:

```
<spml:deleteRequest  xmlns="urn:oasis:names:tc:SPML:1:0"
                     xmlns:spml="urn:oasis:names:tc:SPML:1:0"
                     xmlns:dsml="urn:oasis:names:tc:DSML:2:0:core"
                     requestID="delete_01">
      <spml:identifier type="urn:oasis:names:tc:SPML:1:0#DN">
            <spml:id>005i0000003FAX9AAO</spml:id>
      </spml:identifier>
      <spml:operationalAttributes>
            <spml:attr name="objtype">
                  <dsml:value type="string">user</dsml:value>
            </spml:attr>
      </spml:operationalAttributes>
</spml:deleteRequest>
```

2.18.3.10. SearchRequest

The search request is used to retrieve either an object by its name (defined in the “searchBase” XML component) or by a filter.

The following example requests search users. The first search request (**search-01**) searches the user object with the name; the second search request (**search-02**) searches the user objects with a filter:

```
<!-- search one user in SalesForce -->
<spml:searchRequest requestID="search-01"
                    xmlns="urn:oasis:names:tc:SPML:1:0"
                    xmlns:spml="urn:oasis:names:tc:SPML:1:0"
                    xmlns:dsml="urn:oasis:names:tc:DSML:2:0:core">
     <spml:searchBase type="urn:oasis:names:tc:SPML:1:0#DN">
           <spml:id>005i0000003FAX9AAO</spml:id>
     </spml:searchBase>
     <spml:operationalAttributes>
           <spml:attr name="objtype">
                 <dsml:value type="string">user</dsml:value>
           </spml:attr>
           <spml:attr name="scope">
                 <dsml:value type="string">base</dsml:value>
           </spml:attr>
     </spml:operationalAttributes>
     <spml:attributes>
```

```
            <dsml:attribute name="Id"/>
            <dsml:attribute name="Username"/>
            <dsml:attribute name="LastName"/>
            <dsml:attribute name="FirstName"/>
            <dsml:attribute name="Name"/>
            <dsml:attribute name="CompanyName"/>
      </spml:attributes>
</spml:searchRequest>

<!—search several users in SalesForce with filter -->
<spml:searchRequest requestID="search-02"
                    xmlns="urn:oasis:names:tc:SPML:1:0"
                    xmlns:spml="urn:oasis:names:tc:SPML:1:0"
                    xmlns:dsml="urn:oasis:names:tc:DSML:2:0:core">
      <spml:searchBase type="urn:oasis:names:tc:SPML:1:0#DN">
            <spml:id></spml:id>
      </spml:searchBase>
      <spml:operationalAttributes>
            <spml:attr name="objtype">
                  <dsml:value type="string">user</dsml:value>
            </spml:attr>
            <spml:attr name="scope">
                  <dsml:value type="string">subtree</dsml:value>
            </spml:attr>
      </spml:operationalAttributes>
      <spml:attributes>
            <dsml:attribute name="Id"/>
            <dsml:attribute name="Username"/>
            <dsml:attribute name="LastName"/>
            <dsml:attribute name="FirstName"/>
            <dsml:attribute name="Name"/>
            <dsml:attribute name="CompanyName"/>
      </spml:attributes>
      <spml:filter>
            <dsml:or>
                  <dsml:equalityMatch name="CompanyName">
                        <dsml:value>Atos</dsml:value>
                  </dsml:equalityMatch>
                  <dsml:equalityMatch name="Alias">
                        <dsml:value>mgoet</dsml:value>
                  </dsml:equalityMatch>
```

```
            </dsml:or>
     </spml:filter>
</spml:searchRequest>
```

2.18.4. Configuration

Here is a sample configuration snippet for the Salesforce connector:

```
<connector
    role="connector"

className="net.atos.dirx.dxi.connector.salesforce.SalesForceConnector
"
         name="ts"
         version="1.00">
    <connection
         type="SalesForce"
          user="<your user name>"
          password="<your password>"
          server="<your Salesforce installation, e.g.
login.salesforce.com>"
          port=""
          ssl="true"
    >
    <property name="debugfile" value="<your output file name>"/>
     <property name="clientId" value=”<your client id>"/>
     <property name="clientSecret" value=”<your client secret>"/>
     <property name="securityToken" value="<your security token>"/>
     <property name="loginPath" value="/services/oauth2/token" />
     <property name="path"  value="e.g. /services/data/v30.0" />
     <property name="proxyHost"  value="<IP address of your HTTP
proxy>" />
     <property name="proxyPort"  value="<<port of your HTTP proxy>"
/>
     </connection>
</connector>
```

The Salesforce connector supports the following standard properties of the XML configuration file’s <connection> element:

server (mandatory) - the Salesforce login site, for example, **login.salesforce.com**.

port - not used.

user (mandatory) - the Salesforce user name of the Salesforce user that has administrative rights.

password (mandatory) - the password of the Salesforce user.

ssl (mandatory) - a flag that should normally be set to **true** because you access the Salesforce installation using HTTPS; for example, **https:/login.salesforce.com**.

Supported non-standard properties include:

clientId (mandatory) - the consumer key of your registered remote application. For details, see the section on the Salesforce workflow in the chapter "Using the Target System (Provisioning) Workflows" in the *DirX Identity Application Development Guide*.

clientSecret (mandatory) - the consumer secret of your registered remote application. For details, see the section on the Salesforce workflow in the chapter "Using the Target System (Provisioning) Workflows" in the *DirX Identity Application Development Guide*.

debugFile (optional) - the name of the file to which all SPML requests and responses are written.

loginPath (mandatory) - the HTTP path for performing OAuth authentication.

path (mandatory) - the HTTP path for performing the requests using the REST API; for example, **./services/data/v30.0**. Note that the path contains the Salesforce REST API version; for example, **30.0**.

proxyHost (optional) - the IP address of your HTTP proxy server.

proxyPort (mandatory) - the port of your **securityToken**.

securityToken (mandatory) - the security token that is assigned to your Salesforce user account when registering the user in Salesforce (the securityToken you will receive as e-mail from Salesforce).

Here are some hints for the handling of path and loginPath:

Using the following snippet:

```
server="login.salesforce.com"
loginPath="/services/oauth2/token"
path="/services/data/v30.0"
```

the Salesforce connector:

- Connects to Salesforce using **https:/login.salesforce.com/="/services/oauth2/token"**.
- Receives an instance URL from Salesforce; for example, **na15.salesforce.com**.

- Sends a search request to Salesforce using **https:/na15.salesforce.com//services/data/v30.0/query**.
- Sends an update request to Salesforce using **https:/na15.salesforce.com//services/data/v30.0/sobjects/Account** or **https:/na15.salesforce.com//services/data/v30.0/sobjects/Contact** or **https:/na15.salesforce.com//services/data/v30.0/sobjects/PermissionSet** or **https:/na15.salesforce.com//services/data/v30.0/sobjects/Profile** or **https:/na15.salesforce.com//services/data/v30.0/sobjects/User**

2.19. SAP ECC UM Connector

The following sections provide information about the SAP ECC UM connector.

Most of the connector and its configuration is described in the section about the SAP ECC UM Agent. This section describes how to extend the connector by defining a filter that can also perform BAPI and RFC calls. Customers can use this filter to extend the capability of the SAP ECC UM connector.

It is assumed that the reader is familiar with SAP's Java Connector (JCo). JCo is SAP's Java middleware and allows SAP customers and partners to build SAP-enabled components and applications in Java easily.

2.19.1. Overview

The Java-based SAP ECC UM connector runs inside the DirX Identity Connector Integration Framework. It converts SPML requests to the appropriate SAP BAPI USER object interface and converts the results and responses of those interfaces back to SPML results and responses.

2.19.2. Request and Response Handling

This section contains an example of a filter that performs a call to the same ECC server as the connector to check if a user exists. This can be used, for example, to change an add request to a modify request. The example only contains the parts to set up a connection and to perform the existence check, not the part to change the SPML request. You can find the source code of the complete example for JCo version 3.0.x on your DVD under

Additions/SampleConnectorFilter/jco3/java

2.19.2.1. Example Filter Implementation for JCo Version 3

As noted in the *DirX Identity Integration Framework Guide*, a filter must implement the interface **ConnectorFilter**. If the filter needs the configuration of the connector, it must also implement the interface **ConnectorFilterConfig**.

This section describes the filter implementation for JCo version 3. JCo version 3 has changed in incompatible ways from previous versions. For example, the connection management has completely changed. You now use a JCoDestination, which just identifies a physical destination to a function call.

The filter requires at least a variable to hold the successor, the connector configuration, and the JCo variable for a destination. You can use a helper class in the UM connector to simplify the process of interpreting the connector configuration for the proper connection parameters and to create a JCoDestination. The class is named **Configuration**. Here are the specifications for these variables:

```
import com.sap.conn.jco.*;
import siemens.dxm.connector.sapUM.Configuration;
public class TestFilterWithConnectorCfg implements ConnectorFilter,
ConnectorFilterConfig  {
    ConnectorFilter successor = null;
    private DxmConnectorConfig connCfg = null;
    private JCoDestination destination;
    private Configuration myConfig;
```

Before the **open()** method can be called, the setConnectorConfiguration() method must be called to provide the connector configuration:

```
public void setConnectorConfiguration(DxmConnectorConfig
connectorConfig) {
    connCfg = connectorConfig;
}
```

In the **open()** method, the filter gets its own configuration and can set up the Configuration variable.

The Configuration helper class includes the following three methods:

- A constructor to create and initialize the instance
- A method to get a JCo destination from the JCo pool
- A method to release the used pool.

You can use the Configuration instance to get one or more destinations objects for the same connection as the connector will use, including the same user and his credentials. The Configuration class does not support the use of a different user and/or connection. If you need this function, you must implement it on your own.

For more information about JCo pooling, read the JCo documentation.

```
public void open(DxmFilterConfig config, Context context) throws
DxmConnectorException {
    if (connCfg != null) {
        try {
```

```
            myConfig = new Configuration(connCfg, "myPool");
            destination = myConfig.getDestinationFromPool();
        } catch (JCoException e) {
            throw new DxmConnectorException("Open Filter " + name +"
failed:" , e);
        }
    } else {
    throw new DxmConnectorException(name + ": Connector configuration
not set");
    }
}
```

The **close()** method releases the pool and calls the close() method of its successor:

```
public void close() {
    try {
        if (destination != null) {
            myConfig.releasePool();
        }
    } catch(Exception e) {
        System.err.println(…)
    }
    successor.close();
}
```

The main work is done in the **doFilter()** method. In this example, only add requests are filtered, and no responses:

```
public SpmlResponse doFilter(SpmlRequest request) throws
DxmConnectorException {
    if (request instanceof AddRequest ) {
        request = doCallECC((AddRequest) request);
    }
    return successor.doFilter(request);
}
```

The **doCallECC()** method performs the simple BAPI call **BAPI_USER_EXISTENCE_CHECK** to check if a user exists. First, it extracts the identifier from the add request:

```
private SpmlRequest doCallECC(AddRequest request)  throws
```

```
DxmConnectorException {
    Identifier id  = request.getIdentifier();
    if (id.getType() != null && id.getType().toString
().equalsIgnoreCase(IdentifierType.VALUE_2.toString())) {
    IdentifierChoice idchoice = id.getIdentifierChoice();
    String username = idchoice.getId();
    try {
```

The destination is already set up in the **open()** method. It uses the destination to get the ECC repository to set up a function. A JCo function has an **execute()** method which uses a destination parameter:

```
// get a repository from the destination
JCoRepository repository = destination.getRepository();
// using BAPI if user exists
JCoFunctionTemplate fctExistenceTempl = (JCoFunctionTemplate)
repository.getFunctionTemplate("BAPI_USER_EXISTENCE_CHECK");
JCoFunction fctExistence = fctExistenceTempl.getFunction();
// get the import parameter list
JCoParameterList plImp = fctExistence.getImportParameterList();
// set parameter, here the user name
plImp.setValue("USERNAME", username);
// execute BAPI EXISTENCE_CHECK
fctExistence.execute(destination);
// Check return structure
JCoStructure returnSt = fctExistence.getExportParameterList
().getStructure("RETURN");
```

If the returned message type is informational and the message number is 88, the user exists; if it is 124, the user does not exist. Any other type or number is a failure.

For the full example, see the source code on the DVD.

2.19.3. Configuration

To use a filter, you must specify a <port> element. The <port> element combines a list of <filter> elements with one <connector>. It specifies the port to a target system and is required when SPML requests from the controller to a connector must be filtered.

The following snippet shows an example from a configuration file. The filter can have its own parameters (like “myAttribute”):

```
<port connector="SAP UM Agent" mode="inout">
```

```
    <filter className="yourFilterClass" name="filterName">
        <property name="myAttribute" value="myValue" />
    </filter>
    <connector role="connector"
className="siemens.dxm.connector.sapUM.sapUMuser" name="SAP UM Agent"
version="2.00">
        <connection type="SAP_R3_UM"
                user="theUser"
                password="password"
                server="server-address">
            <property name="logonVariant" value="0" />
            <property name="client" value="nnn" />
            <property name="systemID" value="nn" />
            …
        </connection>
    </connector>
</port>
```

2.20. SharePoint Connector

The Java-based SharePoint connector is built with the DirX Identity Connector Integration Framework and can be used for validation workflows in the C++-based Server and real-time workflows in the IdS-J-Server. Like all framework-based agents, it gets SPML requests from the Identity side and converts them to the appropriate SOAP requests on the SharePoint side and vice versa.

The SharePoint connector offers the following functionality:

- Add a group to a SharePoint site.
- Modify group information including members and roles.
- Delete a group from a SharePoint site.
- Perform searches on SharePoint sites to retrieve group information including members and roles.

2.20.1. Overview

The SharePoint connector implements the API methods "add(...)", "modify(...)", "delete(...)" and "search(...)". They represent the corresponding SPML requests"‘AddRequest", "ModifyRequest", "DeleteRequest" and "SearchRequest".

The connector uses the standard SharePoint Web Service methods for UserGroup handling (http://yourserver/_vti_bin/UserGroup.asmx).These Windows SharePoint Web Services (WSS 3.0) have not changed since SharePoint Server 2010. Although they are no longer maintained, they are still supported.

The SOAP requests and responses are handled via the Axis framework 1.4.

A connection is always made to one specific site on the SharePoint server.

2.20.2. Limitations

This section describes SharePoint connector limitations and restrictions.

Groups

You cannot change the description or the default user login name in a modify request.

Please note that at this time all operations are group based.

Users

You cannot create, modify or delete users in SharePoint. You can only assign users that already exist in the active directory or the SharePoint site to a group or remove them from a group.

Roles

You cannot create, modify or delete roles in SharePoint. You can only assign roles that already exist in the SharePoint site to a group or remove them from a group.

Permissions

You cannot assign permissions for SharePoint Webs or Lists.

2.20.3. Request and Response Handling

This section describes the supported requests and attributes for the SharePoint connector.

2.20.3.1. AddRequest

The add request creates a new group in the SharePoint site. The following attributes are supported:

- objectClass (mandatory)
  Must be "Group".
- groupName (mandatory)
  Name for the new group.
- ownerType
  The type of the group owner (either "User" or "Group").
  If no ownerType is passed in the request the value is set to "User" and the bind account is used as owner for the new group.
- ownerIdentifier
  The name of the group owner (either a valid username in the active directory or another group in the same SharePoint site).
  If no ownerIdentifier is passed in the request the value is set to the username of the

bind account.

- defaultUserLoginName
  A valid username in the active directory.
  If no defaultUserLoginName is passed in the request the value is set to the username of the bind account.
- description
  A short description for the new group.
- member
  A list of valid usernames in the active directory.
- role
  A list of valid roles in the SharePoint site.

Example Request:

```
<?xml version="1.0" encoding="UTF-8" ?>
<batchRequest xmlns="urn:oasis:names:tc:SPML:1:0"
xmlns:spml="urn:oasis:names:tc:SPML:1:0"
xmlns:dsml="urn:oasis:names:tc:DSML:2:0:core"
  requestID="batch-1"
processing="urn:oasis:names:tc:SPML:1:0#sequential"
  execution="urn:oasis:names:tc:SPML:1:0#synchronous"
onError="urn:oasis:names:tc:SPML:1:0#exit">
  <spml:addRequest requestID="add-1">
<spml:identifier type="urn:oasis:names:tc:SPML:1:0#GenericString">
<spml:id>NewGroupName</spml:id>
</spml:identifier>
<spml:attributes>
<spml:attr name="objectclass"
xmlns="urn:oasis:names:tc:DSML:2:0:core">
                    <dsml:value>Group</dsml:value>
</spml:attr>
<spml:attr name="groupName" xmlns="urn:oasis:names:tc:DSML:2:0:core">
 <dsml:value>NewGroupName</dsml:value>
</spml:attr>
<spml:attr name="ownerType" xmlns="urn:oasis:names:tc:DSML:2:0:core">
 <dsml:value>User</dsml:value>
</spml:attr>
<spml:attr name="ownerIdentifier"
xmlns="urn:oasis:names:tc:DSML:2:0:core">
 <dsml:value>MOSS2007\ossadm</dsml:value>
</spml:attr>
<spml:attr name="defaultUserLoginName"
```

```
xmlns="urn:oasis:names:tc:DSML:2:0:core">
 <dsml:value>MOSS2007\ossadm</dsml:value>
</spml:attr>
<spml:attr name="description"
xmlns="urn:oasis:names:tc:DSML:2:0:core">
 <dsml:value>Description for the new group</dsml:value>
</spml:attr>
<spml:attr name="member" xmlns="urn:oasis:names:tc:DSML:2:0:core">
 <dsml:value>moss2007\username1</dsml:value>
 <dsml:value>moss2007\username2</dsml:value>
</spml:attr>
<spml:attr name="role" xmlns="urn:oasis:names:tc:DSML:2:0:core">
 <dsml:value>Full Control</dsml:value>
</spml:attr>
</spml:attributes>
</spml:addRequest>
</batchRequest>
```

2.20.3.2. ModifyRequest

The modify request can be used to change group information, to add and remove group members and to add and remove roles.

Important: The identifier for each modification request must be set to the group name. If the request modifies the group name then this is the old group name.

The following attributes are supported:

- groupName (add/replace)
  A new name for the group.
- ownerType (add/replace)
  The type of the new group owner (either "User" or "Group").
- ownerIdentifier (add/replace)
  The name of the group owner (either a valid username in the active directory or another group in the same SharePoint site).
- member (add/remove)
  A list of valid usernames in the active directory.
  No error is raised if an add modification is performed for a username that is already a member of the group.
  No error is raised if a delete modification is performed for a username that is not a member of the group.
- role (add/remove)
  A list of valid roles in the SharePoint site.
  No error is raised if an add modification is performed for a role that is already assigned

to the group.
No error is raised if a delete modification is performed for a role that is not assigned to the group.

- The properties "defaultUserLoginName" and "description" cannot be modified and are therefore ignored.

Example Request:

```
<?xml version="1.0" encoding="UTF-8" ?>
<batchRequest xmlns="urn:oasis:names:tc:SPML:1:0"
xmlns:spml="urn:oasis:names:tc:SPML:1:0"
xmlns:dsml="urn:oasis:names:tc:DSML:2:0:core"
 requestID="batch-1"
processing="urn:oasis:names:tc:SPML:1:0#sequential"
 execution="urn:oasis:names:tc:SPML:1:0#synchronous"
onError="urn:oasis:names:tc:SPML:1:0#exit">
  <spml:modifyRequest requestID="mod-1">
 <spml:identifier
     type = "urn:oasis:names:tc:SPML:1:0#GenericString">
     <spml:id>GroupName</spml:id>
 </spml:identifier>
 <spml:modifications>
     <spml:modification name="groupName" operation="replace">
         <dsml:value>NewGroupName</dsml:value>
     </spml:modification>
     <spml:modification name="ownerIdentifier" operation="replace">
         <dsml:value>Human Resources Owners</dsml:value>
     </spml:modification>
     <spml:modification name="ownerType" operation="replace">
         <dsml:value>Group</dsml:value>
     </spml:modification>
 <spml:modifications>
     <spml:modification name="member" operation="add">
         <dsml:value>moss2007\username3</dsml:value>
         <dsml:value>moss2007\username4</dsml:value>
     </spml:modification>
     <spml:modification name="role" operation="add">
         <dsml:value>Design</dsml:value>
         <dsml:value>Contribute</dsml:value>
     </spml:modification>
 </spml:modifications>
 </spml:modifications>
```

```
</spml:modifyRequest>
</batchRequest>
```

2.20.3.3. DeleteRequest

The delete request is used to delete a group from a SharePoint site.

Important: The identifier for each delete request must be set to the group name.

The delete request does not require additional attributes.

Example Request:

```
<?xml version="1.0" encoding="UTF-8" ?>
<batchRequest xmlns="urn:oasis:names:tc:SPML:1:0"
xmlns:spml="urn:oasis:names:tc:SPML:1:0"
xmlns:dsml="urn:oasis:names:tc:DSML:2:0:core"
     requestID="batch-1"
processing="urn:oasis:names:tc:SPML:1:0#sequential"
     execution="urn:oasis:names:tc:SPML:1:0#synchronous"
onError="urn:oasis:names:tc:SPML:1:0#exit">
  <spml:deleteRequest requestID="del-1">
<spml:identifier
  type="urn:oasis:names:tc:SPML:1:0#GenericString">
  <spml:id>GroupName</spml:id>
</spml:identifier>
 <spml:attributes>
  <spml:attr name="objectclass"
xmlns="urn:oasis:names:tc:DSML:2:0:core">
               <dsml:value>Group</dsml:value>
  </spml:attr>
 </spml:attributes>
</spml:deleteRequest>
</batchRequest>
```

2.20.3.4. SearchRequest

The search request is used to retrieve group data such as owner information, members and roles. The search can either be restricted to one specific group or return all groups in the current site.

There are two ways to filter on one specific group:

- Define a filter with an equality match on the attribute "groupName".

- Limit the search scope to "base" and set the request identifier to the group name.

If the search is limited to one group name and the group cannot be found in the current site, then the search return the error code **NO_SUCH_OBJECT**.

Supported attributes for the search result include:

- objectClass
  Always returns "Group".
- groupName
- ownerType
  The type of the group owner (either "User" or "Group").
- ownerIdentifier
  The name of the group owner (either an active directory user or a group in the same site).
- description
- member
  The user names of all group members.
- role
  A list of all role names assigned to the group.
- site
  Returns the name of the current site.

Example Request:

```
<?xml version="1.0" encoding="utf-8"?>
<spml:searchRequest requestID="search-01"
xmlns:dsml="urn:oasis:names:tc:DSML:2:0:core"
xmlns:spml="urn:oasis:names:tc:SPML:1:0">
<spml:filter>
 <dsml:and>
   <dsml:equalityMatch name="groupName">
     <dsml:value type="string">SearchGroupName</dsml:value>
   </dsml:equalityMatch>
   <dsml:equalityMatch name="objectclass">
     <dsml:value type="string">Group</dsml:value>
   </dsml:equalityMatch>
 </dsml:and>
</spml:filter>
<spml:attributes>
 <dsml:attribute name="groupName"/>
 <dsml:attribute name="ownerIdentifier"/>
 <dsml:attribute name="ownerType"/>
```

```
 <dsml:attribute name="description"/>
 <dsml:attribute name="member"/>
 <dsml:attribute name="role"/>
 <dsml:attribute name="objectclass"/>
 <dsml:attribute name="site"/>
</spml:attributes>
</spml:searchRequest>
```

2.20.4. Configuration

Here is a sample configuration snippet for the SharePoint connector:

```
<connector
role="connector"
className=" siemens.dxm.connector.sharepoint.SharePointConnector"
name="ts" version="1.00">
<connection type="Sharepoint"
  server="sharepoint-2016-01"
  port="80"
  user="domain-01\admin"
  password="!xxxYYY123"
  ssl="false"
  <property name="endpoint" value="http://sharepoint-2016-
01/sites/DXI_TestSiteCollection/SiteA"/>
  <property name="searchGroupsFromSiteCollection" value="false"/>
  <property name="debugfile" value="dbgOut.xml"/>
</connection>
</connector>
```

2.20.4.1. Supported Connection Parameters

The SharePoint connector supports the following standard properties of the XML configuration file's <connection> element:

server - (mandatory); the SharePoint server name.

port - (mandatory); the SharePoint server port number listening for HTTP requests.

user - (mandatory); the bind user in <domain>\<name> syntax with admin rights in SharePoint.

password - (mandatory); user password.

ssl - (optional); flag whether to connect over SSL to the SharePoint server or not. Default is false.

Supported non-standard properties include:

endpoint - (mandatory); the base site from where the group search starts.

searchGroupsFromSiteCollection - (optional); if set to true, all groups from the complete Site collection – also those from sites parallel to the specified base site- are searched for. If set to false (the default), only the groups from the specified (sub)site are searched for.

debugFile - (optional); if specified, all SPML requests and responses are written to the configured file.

2.21. SPMLv1 Connector

The Java-based SPML v1 SOAP connector runs inside the DirX Identity Connector Integration Framework. It sends SPML SOAP requests over HTTP to the configured endpoint and receives SPML SOAP responses from a SOAP service.

There are two flavors of the connector identified by their class name in the package *siemens.dxm.connector.framework.soap*:

- **DxaSpmlSoapProxy**: produces SPML requests according the latest OASIS SPMLv1 standard (http://www.oasis-open.org/committees/download.php/4138/os-pstc-spml-schema-1.0.xsd).
- **SpmlSoapProxy**: produces SPML requests according a more recent draft of SPMLv1 that is not considered the official release any more (http://www.oasis-open.org/committees/download.php/2396/cs-pstc-spml-schema-1.0.xsd).

The connector supports all SPMLv1 requests and the corresponding methods of the connector API: addRequest, modifyRequest, deleteRequest, searchRequest, extendedRequest, cancelRequest, batchRequest.

The connector supports basic authentication as well as server and client-side SSL/TLS authentication. It does not support WS-Security protocols yet.

The connector does not support connection pooling. It uses the Axis "maintainSession" feature together with a configurable time-out in order to hold connection between consecutive requests.

2.21.1. Prerequisites

The connector is part of the Identity Java Connector Integration Framework and uses Apache Axis1 V1.4 for sending and receiving SOAP requests and responses over HTTP. As a result, the following libraries need to be in the classpath, which are delivered together with the framework:

axis.jar

saaj-api.jar and saaj-impl.jar

jaxrpc-api.jar and jaxrpc-ri.jar

saaj-api.jar and saaj-impl.jar

commons-discovery.jar

commons-logging.jar

2.21.2. Configuration

The connector receives its configuration by the connector framework in a format that is specified there and reflects an XML document. Note that Identity Manager presents configuration options in a more convenient manner. Especially bind credentials, SSL flag and service address are typically collected from appropriate LDAP entries found by selecting the appropriate connected directory and bind profile.

The following discusses the configuration options based upon the XML format. These options are either specified attributes in the XML schema of the element <connection> (referred to as *standard properties*) or may be specified as <property> sub-elements of the <connection> element (referred to as *non-standard properties*).

The connector evaluates the following standard and non-standard properties:

Standard attributes:

url

(optional) The endpoint where to send the request;

for example, **http://localhost:8080/spml/spmlservice**.

You either need to specify the SOAP endpoint completely in this **url** parameter or provide the parts in the attributes **server**, **port**, **ssl** and the non-standard property **path**.

A protocol selector of "https" requests SSL/TLS protocol. In this case, you must ensure that the certificate of the addressed Web server is imported in the trust store of the Java runtime. See the JDK documentation (**keytool**) for details.

server

(optional) If no **url** is given, this property provides the server part of the url. In the above sample, this would be "localhost".

port

(optional) If no **url** is given, this property provides the port of the url. In the above sample, this would be "8080".

ssl

(optional) If no **url** is given, this property tells the connector which protocol to use. If **true**, **https** is selected. Otherwise, the connector sets **http**. In the above sample, a missing **ssl** property or the value **false** would apply.

user

(optional) the user name used for HTTP basic authentication.

password

(optional) the password used for HTTP basic authentication.

trustStore

(optional) the path to the trust store file, which contains the certificate of the server to be used for SSL/TLS server-side authentication.

trustStorePassword

(optional) the password that is required to read the certificate from the trust store.

keyStore

(optional) the path to the key store file that contains the private key or certificate to be used for SSL/TLS client authentication.

keyStorePassword

(optional) the password that is needed to read the key from the key store.

keyStoreAlias

(optional) the alias name to identify the private key in the key store.

The SOAP connector evaluates the following non-standard properties beneath the <connection> element:

maintainSession

(optional) boolean (true / false); if set to "true" (the default), maintains the HTTP session to the target Web service between consecutive requests and thereby saves performance.

path

(optional) If no url is given, this property provides the path of the url. In the above sample, this would be "spml/spmlservice".

timeout

(optional) The socket timeout in seconds. Default is 60.

includePrefixesForXsdPrimitiveTypes

(optional) boolean (true / false); if set to "false" (the default), the DSML value types are not declared with full XML name. Only the XML attribute **type** is declared and a common string is used as its value.

httpHeaders

(optional) multi-value string in the format **"**httpHeaderName httpHeaderValue***"* (for example **"X-Requested-With XMLHttpRequest"**); if set, each SOAP request sent over HTTP will contain additionally these custom HTTP headers. Note that a custom HTTP header name and its value must be separated by a space.

Here is a configuration sample using the **url** property to denote the SOAP endpoint using the SPMLv1 compliant connector implementation:

```
<connector    role="connector"
className="siemens.dxm.connector.framework.soap.DxaSpmlSoapProxy"
        name="SPML connector">
    <connection type="SOAP"

url="http://localhost:8080/spmlsoapservice/services/SpmlSoapService"
    />
</connector>
```

The following is an alternative with the non-compliant SPML connector class using the properties server, port, path and ssl to denote the SOAP endpoint.

```
    <connector  role="connector"
className="siemens.dxm.connector.framework.soap.SpmlSoapProxy"
name="SoapConnector">
<connection type="SOAP"
   server="localhost" port="8080" ssl="false"
>
   <property name="path"
value="spmlsoapservice/services/SpmlSoapService"/>
</connection>
</connector>
```

2.22. SPMLV1ToV2 Connector

The SpmlV1ToV2 SOAP connector implements the Identity Java Connector Integration Framework's **DxmConnector** interface and connects to SPMLv2 Web services. For details of the OASIS SPML Service Provisioning Markup Language, see http://www.oasis-open.org/committees/provision/docs.

2.22.1. Overview

The connector implements the API methods "add(...)", '"modify(...)" "delete(...)"and "search(...)". They represent the corresponding SPMLv1 requests AddRequest, ModifyRequest, DeleteRequest and SearchRequest. It transforms each of them to the corresponding SPMLv2 request and sends them to the configured URL of an SPMLv2 service provider. The connector transforms the received SPMLv2 response into the corresponding SPMLv1 response and returns it at the interface.

The connector supports the SPML2-DSML profile.

The following sections describe how the connector handles the requests and responses, the connector's configuration, and the extension points to adapt to non-standard capabilities of the SPMLv2 service.

2.22.2. Prerequisites

The SPMLv2 connector is contained in

dxmSpmlV1ToV2Connector.jar.

The connector is based on the Identity Java Connector Integration Framework and uses Apache Axis1 V1.4 for sending and receiving SOAP requests and responses over HTTP. The framework is contained in the library

dxmConnector.jar.

The following libraries are required for SOAP handling with Axis1 and are delivered together with the framework:

axis.jar

saaj-api.jar and saaj-impl.jar

jaxrpc-api.jar and jaxrpc-ri.jar

saaj-api.jar and saaj-impl.jar

commons-discovery.jar

commons-logging.jar

SPMLv2 classes are contained in

com.siemens.dxm.provisioning.jar

2.22.3. Request and Response Handling

This section describes how the connector transforms SPMLv1 to / from SPMLv2 requests and responses.

2.22.3.1. General Aspects

This section describes general aspects of the connector's request and response handling operations.

2.22.3.1.1. SPMLv1 Identifier

SPMLv1 supports several identifier types, especially the DN type. The SpmlV1ToV2 connector produces the type "GenericString" in responses, especially in search result entries. It ignores the type in SPMLv1 requests.

2.22.3.2. AddRequest

In an addRequest, the identifier is optional, both in SPMLv1 and v2. The connector inserts a

psoID into the SPMLv2 addRequest, if the SPMLv1 addRequest has an identifier.

It inserts a targetID into the psoID, if one is configured or passed as an operational attribute. The operational attribute overrides the configuration value.

The connector puts all attributes of the SPMLv1 request into the data section of the SPMLv2 request that are neither configured a reference, password or other capability attribute. In case reference or capability attributes are configured the corresponding handlers are invoked just after this. So they can update the SPMLv2 request and especially insert the capability sections into the request.

The connector puts the returned attributes and capabilities from the SPMLv2 response into the SPMLv1 addResponse.

If a password attribute name is configured or passed as an operational attribute and a value is contained in the SPMLv1 attributes, the connector sends a setPasswordRequest after the addRequest. It delegates generation of the request to the password handler. If either the addRequest or the setPasswordRequest fails, the connector returns an error result code in the SPMLv1 addResponse to the controller.

2.22.3.3. ModifyRequest

The connector transforms the mandatory SPMLv1 identifier into the SPMLv2 psoID and adds the targetID, if one is available as an operational attribute or in the configuration.

The connector transforms each SPMLv1 modification to an SPMLv2-DSML modification for all attributes of the SPMLv1 request that are neither configured as a reference, password or other capability attribute. As prescribed by the SPML2-DSML profile, the modify operation occurs two times in the SPMLv2 request as can be seen in the following sample:

```
<modification modificationMode="replace">
  <dsml:modification name="description"
  operation="replace">
     <dsml:value>role1 for tests modified</dsml:value>
  </dsml:modification>
</modification>
```

In case reference or capability attributes are configured the corresponding handlers are invoked just after this. So they can update the SPMLv2 request and especially insert the capability sections into the request.

The SPMLv1 modifyResponse only contains the identifier and no modifications.

If a password attribute name configured or passed as an operational attribute and a modification is contained in the SPMLv1 request, the connector sends a setPasswordRequest after the modifyRequest. It delegates generation of the request to the password handler. If either the modifyRequest or the setPasswordRequest fails, the connector returns an error result code in the SPMLv1 modifyResponse to the controller.

2.22.3.4. DeleteRequest

The connector simply transforms the mandatory SPMLv1 identifier into an SPMLv2 psoID and includes the targetID. No handlers are called.

2.22.3.5. SearchRequest

If the SPMLv1 operational attribute “scope” is set to “base”, the connector issues an SPMLv2 lookupRequest, otherwise a searchRequest.

2.22.3.5.1. Processing a lookupRequest

The SPMLv1 search base is transformed to the SPMLv2 psoID and the targetID added.

The connector invokes the reference and capability handler so that they are able to insert their capabilities into the SPMLv2 request.

The connector is responsible for putting the attributes of the SPMLv2 response that are not configured as reference or capability attribute into the SPMLv1 search result entry, while the reference and capability handler are responsible for their corresponding capabilities.

2.22.3.5.2. Processing a searchRequest

The SPMLv1 search base is transformed to the SPMLv2 psoID and the targetID added.

If the SPMLv1 requested attributes, the connector requests “everything” as return data. Each capability or reference attribute of the requested attributes is put as “<includeDataForCapability.../>” into the SPMLv2 searchRequest.

The operational SPMLv1 attribute “sizelimit” is taken as the SPMLv2 “maxSelect”. The connector strips off all reference or capability attributes from the filter. It’s the responsibility of the corresponding capability handlers to evaluate them. The reference handler as an example could insert “<hasReference>” elements into the SPMLv2 request.

The connector invokes the reference and capability handler so that they are able to insert their capabilities into the SPMLv2 request.

The connector is responsible for putting the attributes of each SPMLv2 response PSO that are not configured as reference of capability attribute into the SPMLv1 search result entries, while the reference and capability handler are responsible for their corresponding capabilities.

If the SPMLv2 service provider does not return all PSO’s in one searchResponse, the connector issues the necessary iterateRequest’s and includes their PSO’s into the SPMLv1 response.

2.22.4. Configuration

The connector is configured according the Identity Java Connector Integration Framework. It evaluates connector options and connection class (or XML element).

Here is a complete XML configuration sample:

```
<connector role="connector"
    className =
"com.siemens.dxm.connector.spmlv1tov2.SpmlV1ToV2Connector"
    name="ts">
   <connection type="spmlv2"
        url="http://localhost:8088/spml/spmlservice"
        user="cn=DomainAdmin,cn=my-company"
        password="dirx" >
   </connection>
   <property name="targetID" value="roles"/>
   <property name="referenceAttributes"
value="dxrPermissionLink, dxrRoleLink"/>
   <property name="referenceHandler"
value="com.siemens.dxm.connector.spmlv1tov2.handler.SimpleReferenceHa
ndler"/>
   <property name="passwordAttribute"
value="userPassword"/>
   <property name="capabilityAttributes"
value="dxrRoleParams"/>
   <property name="capabilityHandler"
value="com.siemens.dxm.connector.spmlv1tov2.handler.RoleParamHandler"
/>
</connector>
```

2.22.4.1. Connection Options

The connection options specify the connection parameters to reach the remote SPMLv2 web service over a SOAP/HTTP transport. It evaluates the following parameters: First the XML standard attributes of the <connection> element:

url: optional. The endpoint to which to send the request; for example, “http://localhost:8080/spml/spmlservice”.

You must specify the SOAP endpoint completely in the **url** parameter or provide the parts in the **server**, **port** and **ssl** standard properties and in the non-standard property **path**.

Note that a protocol selector of "https" requests SSL/TLS protocol. In this case, you must ensure that the certificate of the addressed Web server is imported in the trust store of the Java runtime. See the JDK documentation (**keytool**) for details.

server: optional. If no URL is given, this property provides the server part of the URL. In the sample above, this is "localhost".

port: optional. If no URL is given, this property provides the port of the URL. In the sample

above, this is "8080".

ssl: optional. If no URL is given, this property tells the connector which protocol to use. If **true**, **https** is selected; otherwise, the connector sets **http**. In the sample above, a missing **ssl** property or the value **false** would apply.

user: optional; the user name used for HTTP basic authentication.

password: optional; the password used for HTTP basic authentication.

trustStore: optional; the path to the trust store file, which contains the certificate of the server to be used for SSL/TLS server-side authentication.

trustStorePassword: optional; the password that is required to read the certificate from the trust store.

keyStore: optional; the path to the key store file that contains the private key or certificate to be used for SSL/TLS client authentication.

keyStorePassword: optional; the password that is needed to read the key from the key store.

keyStoreAlias: optional; the alias name to identify the private key in the key store.

The SOAP connector evaluates the following non-standard properties beneath the <connection> element:

path: optional. If no URL is given, this property provides the path or suffix of the URL. In the above sample, this would be "spml/spmlservice".

maintainSession: optional, boolean (true / false); if set to **true** (the default), the SOAP connector maintains the HTTP session to the target Web service between consecutive requests and thus increases performance.

httpHeaders: optional, multi-value string in the format **"**httpHeaderName httpHeaderValue*"*** (for example **"X-Requested-With XMLHttpRequest"**); if set, each SOAP request sent over HTTP will also contain these custom HTTP headers. Note that a custom HTTP header name and its value must be separated by a space.

The following configuration snippet of the <connection> element shows a configuration where the URL is determined from the parts “ssl”, “server”, “port” and “path”:

```
<connection type="spmlv2"
ssl="false"
server="localhost"
port="8088"
user="cn=DomainAdmin,cn=my-company"
password="dirx">
<property name="path"
```

```
value="spml/spmlservice"/>
</connection>
```

2.22.4.2. Connector Options

The non-standard XML options beneath the <connector> element specify the connector's class name and determine the handling of SPMLv2 options and capabilities. They are all optional and specified in XML <property „„/> elements.

Note that all these options can be overridden per request. See the following section on operational attributes for details.

targetID: The target identifier to be used in all PSO identifiers in SPMLv2 requests. This value must be set according the "listTargets" response of the SPMLv2 service.

referenceAttributes: A comma separated list of attribute names. These attributes in an SPMLv1 request are expected to be handled as references in SPMLv2 requests and responses.

The connector does not pass them as normal attributes or modifications to the SPMLv2 service; instead, it passes them to the configured reference handler.

referenceHandler: The class name of a reference handler.

A reference handler has to implement the interface "com.siemens.dxm.connector.spmlv1tov2.api.Spmlv2ReferenceHandler". It is expected to take the reference attributes from SPMLv1 requests and insert them as reference capabilities into SPMLv2 requests and transform reference capabilities from SPMLv2 responses into attributes in SPMLv1 responses.

If reference attributes are configured, but no reference handler, the connector takes its default reference handler implementation with the class name "com.siemens.dxm.connector.spmlv1tov2.handler.SimpleReferenceHandler".

For more details on reference handling, see the section on reference handlers.

passwordAttribute: The name of the attribute, which contains the password.

The connector does not include the password as normal attribute or modification into the SPMLv2 request; instead it delegates password handling to the configured password handler.

passwordHandler: The class name of a password handler. A comma-separated list of attribute names. These attributes in an SPMLv1 request are expected to be handled as references in SPMLv2 requests and responses.

A password handler must implement the interface "com.siemens.dxm.connector.spmlv1tov2.api.Spmlv2PasswordHandler". It is expected to take the password attribute out of SPMLv1 requests and insert it as password capability into SPMLv2 requests.

The connector does not expect passwords to be contained in responses.

If a password attribute is configured, but no password handler is configured, the connector takes its default password handler implementation with the class name “com.siemens.dxm.connector.spmlv1tov2.handler.DefaultPasswordHandler”.

For more details on reference handling, see the section on password handlers.

capabilityAttributes: A comma-separated list of attribute names. These attributes in an SPMLv1 request are expected to be handled as capabilities in SPMLv2 requests and responses.

The connector does not pass them as normal attributes or modifications to the SPMLv2 service; instead, it passes them to the configured capability handler.

Separating reference and password capabilities from other ones, allows a customer to re-use the default implementation for the SPMLv2 specified reference and password capabilities and only provide an implementation for proprietary capabilities.

capabilityHandler: The class name of a capability handler.

A capability handler must implement the interface “com.siemens.dxm.connector.spmlv1tov2.api.Spmlv2CapabilityHandler”. It is expected to take the capability attributes (excluding reference and password attributes) from SPMLv1 requests and insert them as capabilities into SPMLv2 requests and transform capabilities from SPMLv2 responses into attributes in SPMLv1 responses.

The SpmlV1toV2 connector does not provide any default implementation for a general capability handler, but instead provides a sample that can you can use as a starting point for your own implementation.

For more details on reference handling, see the section "Custom Capabilities".

2.22.4.3. Overriding Connector Options per Request

An SPMLv2 service can support a number of entity types at the same time. Often they are identified by different target identifiers in their PSO and support different capabilities. As an example, a user typically has other references than a group or role.

If a particular connector configuration is valid for more than one entity type, it may be necessary to support different capabilities per type. For these scenarios, the SpmlV1ToV2 connector allows you to override the overall capability options per request.

SPMLV1 requests may contain operational attributes. The connector evaluates all operational attributes with a name matching one of the connector attributes “targetID”, “referenceAttributes”, “referenceHandler”, “passwordAttribute”, “passwordHandler”, “capablityAttributes” or “capabilityHandler”. If it finds one, the attribute value overrides the one from the configuration while processing this request. If the next request does not contain this operational attribute, the value of the connector configuration holds again.

2.22.5. Custom Capabilities

SPMLv2 defines few mandatory operations and allows each provider to define and implement its own custom capabilities. Some important capabilities are already part of the SPMLv2 specification, especially the search, reference and password capabilities.

The SpmlV1ToV2 connector provides default implementations for these capabilities out of the box since they are considered to be widely distributed. To allow for custom extensions, the connector also supports interfaces for appropriate capability handlers:

Spmlv2HandlerOptions

This is a basic interface for al handlers. It allows you to pass configuration options to the handler.

Spmlv2PasswordHandler

This is the interface for a password handler. The handler is expected to produce a SPMLv2 setPassword request.

Spmlv2ReferenceHandler

This is the interface that a reference handler must implement. The handler is expected to add the capabilities into SPMLv2 requests, take capabilities from a SPMLv2 response and insert them as attributes into the corresponding SPMLv1 response.

Spmlv2CapabilityHandler

This interface is intended for all types of capability handlers. The handler is expected to add the capabilities into SPMLv2 requests, take capabilities from a SPMLv2 response and insert them as attributes into the corresponding SPMLv1 response. The connector passes the SPMLv1 and v2 requests and responses to the handler and the reference to an Spmlv2SoapSender. This operation allows the handler to send its own SPMLv2 requests before or after those sent by the connector.

The product DVD folder **Additions/SpmlV1toV2Connector** contains the Java documentation of the interfaces and also the sources of some default handlers. The following handler classes are delivered with the product:

DefaultPasswordHandler.java - a password handler.

SimpleReferenceHandler.java - a handler for simple object-to-object DN references.

RoleParamHandler.java - a handler for processing role parameters of user-to-role assignments.

TargetSystemCapabilityHandler.java - a handler that implements all capabilities for target system management.

You can find more details about these handlers in the section "Sample Handlers".

2.22.5.1. Interface Spmlv2HandlerOptions

The interface “com.siemens.dxm.connector.spmlv1tov2.api.Spmlv2HandlerOptions” is the basic interface for all capability handlers. It allows the connector to set the configuration options with its method **setOptions(ConfigurationOptions options)**.

The **ConfigurationOptions** parameter contains the options in a map and other additional convenience methods to get the capability, reference and password attributes and handlers.

The method is called after the handler is instantiated and before the other methods are invoked. The options contain the values taken from the SPMLv1 operational attributes, if there are any. Therefore, the method is called in each request.

2.22.5.2. Interface Spmlv2ReferenceHandler

The interface “com.siemens.dxm.connector.spmlv1tov2.api.Spmlv2ReferenceHandler” extends the Spmlv2HandlerOptions and is to be implemented by a reference handler. The handler is responsible for processing all attributes that are configured as reference attributes.

It provides the following methods:

includeReferenceIntoRequest(...):

The connector passes the SPMLv1 request and the SPMLv2 request generated so far and expects the updated SPMLv2 request to be returned.

The method is called after the connector has included the normal attributes into the request.

includeReferenceIntoResponse(...):

The connector passes the SPMLv2 response received and the SPMLv1 response generated so far and expects the updated SPMLv1 response to be returned.

The method is called after the connector has included the normal attributes into the response.

includeReferenceIntoResultEntry(...):

The method is called while the connector transforms a SPMLv2 lookup- or searchResponse to a SPMLv1 searchResponse for each PSO, which is part of the SPMLv2 response. The connector passes one SPMLv2 PSO received and the SPMLv1 search result entry generated so far and expects the handler to update the search result entry. Therefore the handler may ignore these responses when performing the “includeReferenceIntoResponse” method.

The search result entry contains already the non-capability attributes.

The default implementation “com.siemens.dxm.connector.spmlv1tov2.handler.SimpleReferenceHandler” transforms each reference attribute into a SPMLv2 <reference> capability. The following snippet shows such a sample for the attribute “dxrPermissionLink” as part of an addRequest:

```
<capabilityData
capabilityURI="urn:oasis:names:tc:SPML:2.0:reference">
    <spmlref:reference typeOfReference="dxrPermissionLink">
```

```
        <spmlref:toPsoID ID="cn=Project Manager,cn=Project
Specific,cn=Corporate Permissions,cn=Permissions,cn=My-Company"/>
    </spmlref:reference>
</capabilityData>
```

2.22.5.3. Interface Spmlv2CapabilityHandler

The interface “com.siemens.dxm.connector.spmlv1tov2.api.Spmlv2CapabilityHandler” extends the Spmlv2HandlerOptions and is to be implemented by a capability handler. The handler is responsible for processing all attributes that are configured as capability attributes.

It provides the following methods:

includeCapabilitiesIntoRequest(...):

The connector passes the SPMLv1 request, the SPMLv2 request generated so far and the SPMLv2 SOAP sender and expects the updated SPMLv2 request to be returned.

The method is called after the connector has included the normal attributes into the request. The handler may send additional SPMLv2 requests using the SOAP sender before the connector sends the passed request.

includeCapabilitiesIntoResponse(...):

The connector passes the SPMLv2 response received, the SPMLv1 response generated so far and the SPMLv2 soap sender and expects the updated SPMLv1 response to be returned.

The method is called after the connector has included the normal attributes into the response. The handler may send additional SPMLv2 requests using the SOAP sender after the connector has sent the “normal” request.

includeCapabilitiesIntoResultEntry(...):

The method is called while the connector transforms a SPMLv2 lookup- or searchResponse to a SPMLv1 searchResponse for each PSO, which is part of the SPMLv2 response. The connector passes one SPMLv2 PSO received and the SPMLv1 search result entry generated so far and expects the handler to update the search result entry. Therefore the handler may ignore these responses when performing the “includeCapabilitiesIntoResponse” method.

The search result entry already contains the non-capability attributes.

2.22.5.4. Interface Spmlv2PasswordHandler

The interface “com.siemens.dxm.connector.spmlv1tov2.api.Spmlv2PasswordHandler” extends the Spmlv2HandlerOptions and is to be implemented by a password handler. The handler is responsible for processing the configured password attribute.

It provides only one method:

setPasswordRequest(...):

The connector passes the SPMLv1 request and the SPMLv2 response received for the previously sent SPMLv2 request. It expects a SPMLv2 request to be returned, usually a setPasswordRequest.

The method is called after an add- or ModifyRequest.

The default implementation "com.siemens.dxm.connector.spmlv1tov2.handler.DefaultPasswordHandler" produces a SPMLv2 setPasswordRequest from the password attribute of a SPMLv1 request. It also adds the current password, if it is available as an operational attribute.

2.22.5.5. Sample Handlers

This section provides some details on the sample handlers delivered with the product. Find their sources and configuration files or sample requests in the folder **Additions/SpmlV1toV2Connector** of the product DVD.

2.22.5.5.1. DefaultPasswordHandler.java

The password handler expects the name of the SPMLv1 attribute with the password in the configuration property "passwordAttribute". The default is "userPassword".

If an add or modify request contains the password attribute, it creates an SPMLv2 setPassword request. It uses the PSO ID of the modify request or of the add response or request as the PSO identifier. If the SPMLv1 request contains an operational attribute "currentPassword", the handler adds it to the setPassword request.

This is the relevant snippet of the configuration:

```
    <connection type="spmlv2" url="..." ...
            <property name="passwordAttribute" value="userPassword"/>
            <property name="passwordHandler"
value="com.siemens.dxm.connector.spmlv1tov2.handler.DefaultPasswordHa
ndler"/>
    </connection>
```

2.22.5.5.2. SimpleReferenceHandler.java

The simple reference handler transforms SPMLv1 attributes into simple SPMLv2 object-to-object references and vice versa. It is applicable only for simple (for example, DN) links between objects.

For each attribute configured as a reference attribute, it creates a SPMLv2 reference capability with the attribute name as the reference type. From SPMLv2 result entries in lookup or search responses, it takes references matching the attribute names and puts them as attributes into the SPMLv1 result entry.

If a SPMLv1 search request contains a reference attribute in its filter, it generates a SPMLv2 hasReference clause.

This is a sample snippet with the relevant configuration properties:

```
<property name="referenceAttributes" value="dxrPermissionLink,
dxrRoleLink"/>
<property name="referenceHandler" value="com.siemens
```

2.22.5.5.3. RoleParamHandler.java

This handler is a sample for a proprietary, complex capability used in DirX Identity user Web services: role parameters of user-to-role assignments.

It expects role parameters in the SPMLv1 attribute "dxrRoleParams" and transforms them to a SPMLv2 capability with URI "urn:siemens:dxm:provisioning:role:matchrule:1:0" and vice versa.

In the SPMLv1 attribute, it expects the role parameters as an XML document according to the following sample:

```
<matchrule>
<uid>uid-7f001-cee271-fed935aa6d--7eb4</uid>
<roleparamDN>cn=Project,cn=My-Company,cn=RoleParams,cn=Customer
Extensions,cn=Configuration,cn=My-Company</roleparamDN>
<type>Group</type>
<attribute>dxrproject</attribute>
</matchrule>
```

The handler uses the Open Source tool Castor and its generated classes for marshaling and unmarshalling,

For a sample request, see the file "spmlv1RequestWithRoleParam.xml" in the "Additions" folder of the DVD.

If the dxrRoleParams attribute is requested in a search request, the handler generates an appropriate include capability for the SPMLv2 lookup or search request.

2.22.5.5.4. TargetSystemCapabilityHandler.java

This handler implements all capabilities needed for the DirX Identity Web service for target system management: options, connection-, environment- and create- options.

The handler expects all the options to be in tag/value format in the respective SPMLv1 attributes "dxroptions", "dxrconnectionoptions", "dxrenvironmentproperties" and "tscreateoptions". They are transformed to and from the SPMLv2 capabilities as defined by

the DirX Identity Target System Web Service.

If one of the options (except for the create option) is a requested attribute in an SPMLv1 search request, the handler creates the appropriate includeCapability clause for the SPMLv2 lookup or search request.

For sample requests, see the file "spmlv1RequestWithTSCapabilities.xml" in the "Additions" folder of the DVD.

Here is the relevant configuration snippet. Note that the capability attributes in the configuration are ignored because they are hard-coded:

```
<property name="capabilityAttributes"
value="dxrenvironmentproperties, dxrconnectionoptions, dxroptions,
tscreateoptions"/>
<property name="capabilityHandler"
value="com.siemens.dxm.connector.spmlv1tov2.handler.TargetSystemCapab
ilityHandler"/>
```

2.23. Unify Office Connector

The Java-based Unify Office connector runs inside the Identity Java Connector Integration Framework. It communicates using the RingCentral System for Cross-domain Identity Management (SCIM) API on the common URL **https://platform.ringcentral.com/scim/v2** via common HTTP protocol. The operations are authorized by a dedicated OAuth server available on the common URL **https://platform.ringcentral.com/restapi/oauth/token**.

The connector is implemented in the class UnifyOfficeConnector in the package net.atos.dirx.dxi.connector.ringcentral.

The connector implements the common methods for the DirX Identity Connector API: add, modify, delete and search.

The operations are simply converted to RingCentral API requests. The corresponding responses are again translated to SPMLv1 responses.

The RingCentral API is a Representational State Transfer (REST)-ful service comprised of endpoints that are accessed using standard HTTP requests. The connector uses JavaScript Object Notation (JSON) content types for requests and responses. The current workflow only uses the SCIM endpoint of the RingCentral API. The documentation of the functions can be found at https://developers.ringcentral.com/api-reference/SCIM.

The connector communicates using SSL/TLS only.

2.23.1. Prerequisites

The connector is based on the RingCentral API. The connector functionality is limited by the functionality of the RingCentral API, with only the SCIM API being stable and therefore

fully supported. The functionality with other RingCentral API endpoints cannot be guaranteed.

The connector appends a JSON Web Token (JWT) in the Authorization header of the request. This token is acquired by making a request to the OAuth endpoint and providing valid credentials. The connector supports the use of the OAuth 2.0 service using “Resource Owner Password Credentials Flow”, “Client Credentials Flow” or “Refresh Token Flow”.

The connector supports common RingCentral user objects as specified in the SCIM specification.

It also supports extension, device, call queue and call queue member, answering rule, phone number, user-role and user-template objects of the (non-SCIM) RingCentral API endpoints, but there are no channels provided for these by default.

2.23.2. Configuration

The connector receives its configuration from the Connector Framework in a format that is specified there and reflects an XML document. Note that DirX Identity Manager presents configuration options in a more convenient way. For example, bind credentials and service addresses are typically collected from appropriate LDAP entries found by selecting the appropriate connected directory and bind profile.

This section discusses the configuration options based on the XML format. These options are either specified attributes in the XML schema of the element <connection> (referred to as standard properties) or specified as <property> subelements of the <connection> element (referred to as non-standard properties).

The connector evaluates the following standard properties:

server

(required) This property provides information about the host name or IP address of the RingCentral API endpoint. An example is platform.ringcentral.com.

ssl

(required) This value enables SSL/TLS authentication of a Graph API server and secures the communication line.

user

This property is the User ID of a RingCentral user. It is used for the "Resource Owner Password Credentials Flow" at the OAuth Service. Providing the password will automatically select the right flow and implicitly set the Account ID (= tenant in RingCentral) to the one the user is managed in.

password

The password of the User used for the "Resource Owner Password Credentials Flow" at the OAuth Service.

type

(required) This is the Directory Type, here Unify Office.

The Unify Office connector evaluates the following non-standard properties beneath the <connection> element:

proxyHost

The IP or server name of a proxy server, if any.

proxyPort

The port of a proxy server, if any.

proxyUser

The user for authorization at the proxy server, if any.

proxyPassword

The password for authorization at the proxy server, if any.

clientId

required. The OAuth service requires a client ID, which is provided by RingCentral when creating the "App" for API access in their administrative console. The client ID is usually a generated UID.

clientSecret

required. A client secret is generated together with the client ID. This client secret should be kept secret and works like a password for client authentication.

accountId

When no user and password is provided, the connector runs in "Client Credentials flow" mode. In this case, the Account ID is needed to identify the account (= tenant in RingCentral) that is being managed.

path

required. This property provides the path to the RingCentral API endpoint. By default, the SCIM V2 endpoint "scim/v2" is used.

authPath

required. This property provides the path to the RingCentral OAuth service. This is always "restapi/oauth/token".

Here is a sample configuration using some of the properties described here:

```
    <connector
className="net.atos.dirx.dxi.connector.ringcentral.UnifyOfficeConnect
or" name="TS" role="connector">
        <connection password="{SCRAMBLED}aG5WPw==" port="443"
server="platform.devtest.ringcentral.com" ssl="TRUE" type="Unify
Office" user="&lt;&lt;E.164 Phonenumber&gt;&gt;">
            <property name="proxyHost" value="ProxyServer"/>
            <property name="proxyPort" value="3128"/>
```

```
            <property name="proxyUser" value="user"/>
            <property name="proxyPassword"
value="{SCRAMBLED}aG5WPw=="/>
            <property name="clientId" value="&lt;&lt;Application
Client Id&gt;&gt;"/>
            <property name="clientSecret" value=
"{SCRAMBLED}aG5WPw=="/>
            <property name="accountId" value="&lt;&lt;Account Id for
Client Credentials Flow&gt;&gt;"/>
            <property name="path" value="scim/v2"/>
            <property name="authPath" value="restapi/oauth/token"/>
            <property name="debugMode" value="false"/>
        </connection>
    </connector>
```

2.23.3. SCIM

The Unify Office Connector is based on the System for Cross-domain Identity Management (SCIM) connector implementing the AbstractRestConnector. Many methods used are simply SCIM standard functions and compliant with the specification. For details, please refer to https://tools.ietf.org/wg/scim/.

3. Identity Agents

The Identity agent component of DirX Identity is the interface to a specific connected directory via batch workflows. Its function is to import data into a connected directory and export data from a connected directory. The following figure illustrates the Identity agent component and its relationship to the meta controller and meta directory components.

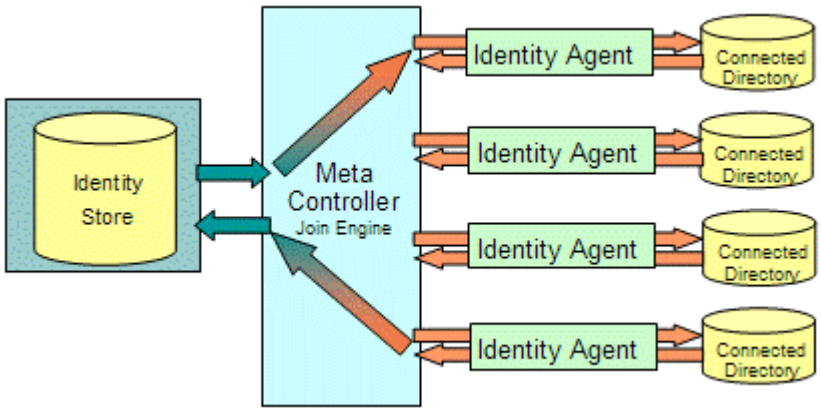


Figure 7. Meta Controller and Agent Control Flow

Identity agents can be designed to handle a particular connected directory, such as NT or Lotus Notes, or they can be designed to handle a specific set of connected directories; for example, ADSI directories or ODBC databases.

The meta controller can also perform the Identity agent function: it can export Identity store data into LDIF structured files, and it can import LDIF structured files into the Identity store. In this way, the meta controller can act as a generic LDAP agent for Identity stores.

The next sections provide a description of Identity agent architecture and the files used by Identity agent components. The remainder of this document provides reference information about each Identity agent, including:

- Command line format
- Import and export configuration file format, and how the data in the configuration file affect the Identity agent's import and export operation
- Import and export data file format
- Import error file format

3.1. Identity Agent Architecture

DirX Identity distinguishes between two types of agents:

- **Framework-based Agents** - implementation is based on the Identity Connector Integration Framework. A set of such agents is delivered with DirX Identity (for example, the JDBC agent). You can use the Identity Connector Integration Framework to build your own custom agents.
- **Non-framework-based Agents** - implementation of an executable in any programming

language. This type of agent comes with DirX Identity (for example the ADS or Notes agents).

In both cases, you can use the Identity Agent Integration Framework to run these agents within the C++-based Identity Servers.

3.1.1. Framework-based Agents

Framework-based agents are built with the Identity Connector Framework. It works internally with SPML, provides standard methods to integrate a target system API and has standard methods for configuration and reading and writing data.

3.1.2. Non Framework-based Agents

A non framework-based Identity agent is implemented in any programming language as an executable program that is invoked from a command line. It is either:

- The export of data from its associated connected directory into an export data file for subsequent import into the Identity store. Some Identity agents can export incremental data (deltas or changes). If the connected directory does not support deltas (or does, but only partially), the Identity agent keeps a copy of the connected directory data in a "delta base" file. The Identity agent uses this file to generate delta information or to complete it.
- The import of data into its associated connected directory that has been previously exported from the meta directory store. Some Identity agents can perform special administration tasks in the connected directory on import; for example, the creation of mailboxes or user accounts.

Some Identity agents can also handle entry and attribute filtering (this filtering is independent of the filtering performed by the meta controller).

An Identity agent requires the operation of the meta controller to complete a synchronization task.

3.2. Framework-based Agents

Framework-based agents use a standard configuration method. The next sections describe:

- The command line format to invoke an agent
- The exit codes provided by an agent
- General information about configuration file formats
- General information about the search request file format

3.2.1. Command Line Format

The command line format to invoke a framework-based agent is as follows:

***agent*.bat** *configuration_file* (on Windows platforms)

***agent*.sh** *configuration_file* (on UNIX platforms)

configuration_file

Specifies the name of the file that contains the specifications for the import procedure. All other parameters for correct agent operation are defined in the agent's configuration file in XML format.

3.2.2. Exit Codes

The following table describes the standard exit codes provided when an agent finishes running.

Exit Code	Description
0	Agent completed successfully.
1	Agent completed with errors. Details are described in the specified trace file unless this file cannot be created due to a file exception error.
60	Agent completed with warnings. For details see the specified trace file.

3.2.3. Configuration File Formats

Framework-based agents use configuration files that control import and export of data into a target system.

This section describes the general structure of a configuration file.

3.2.3.1. General Structure of a Configuration File

The configuration file's format is XML. It is composed of multiple sub-units (connectors). Normally you should not change the general structure of a configuration file. Instead, you configure some well-defined attribute values to the specific environment in which the agent runs.

Tags

The configuration files contain the tags job, connector, logging and connection.

- **job** - Defines the file's document tag, with connector sub-tags
- **connector** - Configures the properties of one connector, has connection and/or logging sub tags
- **connection** - Configures connection parameters, for example filename for a reader/writer or host/port/credentials for a network connector

Attributes

A connector tag can have the following attributes:

- **name** - The connector's name
- **role** - One of reader, controller, connector, responseWriter or requestCryptTransformer
- **className** - The name of the Java class that implements the connector
- **logging** - Configures the logging properties of a connector

The **connection parameters** of the specific connectors are described in their connection sub-tags.

Each connection tag has the attribute

- **type** - The type of connection (file format, protocol)

Readers and response writers are configured by the attribute

- **filename** - The pathname of the input or output file.

Dependent on the type of connectors additional properties may be defined in this section.

Encryption transformers are configured by these attributes

- ***firstAttribute*** ... ***lastAttribute*** - This list of parameters specifies the names of the attributes which can be encrypted in the input of the agent. All encrypted user attributes must be listed here to allow the agent to decrypt them. There's no limit of the number of encrypted attributes in the configuration file.

A **connection to the target system** is configured by some attributes that might differ dependent on the target system. Typical attributes are:

- **host** - the host to access
- **port** - the port to use
- **user** - the user for simple bind operation
- **password** - the user password for simple bind operation

The agent's **logging** is configured in the controller's logging tag by the attributes:

- **level** - The integers 0-9, where 0 indicates no logging and 9 indicates full logging
  0 - none
  1 - FatalError and Error
  2 - FatalError, Error and Warning
  3 - FatalError, Error and Warning
  4 - FatalError, Error and Warning
  5 - FatalError, Error, Warning and Trace
  6 - FatalError, Error, Warning and Trace
  7 - FatalError, Error, Warning and Trace
  8 - FatalError, Error, Warning and Trace
  9 - FatalError, Error, Warning and Trace (and additional HTML files)
- **filename** - The path and name of the trace file
- **debugmode** - a boolean switch to specify a special debug mode (possible values: true

and false)

3.2.3.1.1. Example of an Import Configuration File

The import configuration file has the format defined above. The following generic example describes shows the general layout. The attribute values that can be configured are shown in bold italic, e.g. ***level***:

```
<?xml version="1.0" encoding="UTF-8" ?>
<job>

<connector name="Default Controller" version="0.1" role="controller"
className="siemens.dxm.connector.framework.DefaultControllerStandalon
e">
<logging level="level" filename="traceFileName" />
</connector>

<connector role="reader" name="LDIF change file reader"
className="siemens.dxm.connector.framework.LdifChangeReader">
<connection type="LDIF change" filename="inputFileName" />
<property name="ExtractRDN" value="false"/>
<property name="IncludingNamingAttribute" value="false"/>
</connector>

<connector role="RequestCryptTransformer" name="Crypto Transformer"
className="siemens.dxm.connector.framework.CryptTransformer">
<mvproperty name="encryptedAttributes">
<value> firstAttribute </value>
...
<value> lastAttribute </value>
</mvproperty>
</connector>
<connector role="connector" name="agent_name"
className="siemens.dxm.connector...">
<connection type="connection_type"
user="account"
password="password"
... >
<property name="property_name" value="property_value"/>
</connection>
</connector>

<connector role="responseWriter" name="LDIF file writer"
```

```
className="siemens.dxm.connector.framework.LdifFileWriter">
<connection type="LDIF" filename="responseFilename" />
<property name="contenttype" value="LDIF-CONTENT"/>
</connector>

</job>
```

3.2.4. Search Request File Format

The objects to be exported are defined in a Service Provisioning Markup Language (SPML) search request. SPML is an XML format. The search request contains an LDAP-like filter and searchBase. Its configuration is described by the following template. The attribute values that can be configured are shown in bold (blue) italic; for example, ***subtree***:

```
<?xml version="1.0" ?>
<spml:searchRequest xmlns="urn:oasis:names:tc:SPML:1:0"
xmlns:spml="urn:oasis:names:tc:SPML:1:0" requestID="search_01">
  <spml:operationalAttributes>
    <attr name="scope"> <value>scope</value> </attr>
  </spml:operationalAttributes>
  <spml:searchBase type="urn:oasis:names:tc:SPML:1:0#DN">
    <spml:id>searchBase</spml:id>
  </spml:searchBase>
  <filter>
    <substrings name="attribute">
      <initial>value</initial>
    </substrings>
  </filter>
  <spml:attributes>
    <attribute name="attribute1" />
    <attribute name="attribute2" />
  </spml:attributes>
</spml:searchRequest>
```

searchBase

searchBase specifies the base object for the search .

This definition is dependent on the target system.

scope

scope specifies the depth of the search. It must have one of the following values:

base

The base object defined in **searchBase**

onelevel

The base object and all objects located one level below it.

subtree:::The base object and the entire subtree under this object.

filter

filter is a tag that specifies the search criteria. It supports an LDAP-like structure, with the operator tags <and>, <or>, <not>, corresponding to the LDAP operators (& ...), (|...), (! ...) and the tags <substrings> and <equalityMatch>.

The previous example uses a **substrings** tag, but you can use all the tags and combinations of the tags defined here.

equalityMatch

equalityMatch is a filter sub-tag that specifies that an attribute value must be exactly equal to the value defined in the <value> tag.

Example:

```
<equalityMatch name="objectclass">
         <value>user</value>
</ equalityMatch >
```

substrings

substrings is a filter sub-tag that specifies that an attribute value must start with, end with, or contain a value defined in its <initial>, <contains>, or <final> sub-tag.

Example:

```
<substrings name="objectclass">
        <initial>ro</initial>
</substrings>
```

and

and specifies the tags it contains to be "and" related.

Example:

```
<and>
    <substrings name="name">
         <initial>develop</initial>
```

```
    </substrings>
    <equalityMatch name="objectclass">
         <value>folder</value>
    </ equalityMatch >
   </and>
```

or

or specifies the tags it contains to be "or" related.

Example:

```
<or>
    <substrings name="name">
          <initial>develop</initial>
    </substrings>
    <substrings name="name">
          <initial>market</initial>
    </substrings>
</or>
```

not

not specifies that the tag it contains must evaluate to false.

Example:

```
<not>
    <equalityMatch name="objectclass">
         <value>folder</value>
    </ equalityMatch >
</not>
```

spml:attributes

spml:attributes specifies the attributes ***attribute1, attribute2,...*** to be returned by the search.

3.3. Non Framework-based Agents

Non framework-based agents use the following files:

- Import and export configuration files
- Import and export data files

Note: Non framework-based agents support only the ISO 8859-1 (Latin1) character set.

3.3.1. Agent Configuration Files

The non framework-based agents export and import configuration files "configure" a DirX Identity agent's import and/or export process.

The directory synchronization parameters for command line-based DirX Identity agent configuration are contained in "***.ini**" configuration files, which are generally specified on the command line that invokes the DirX Identity agent import or export operation. The configuration files consist of fields that control directory synchronization parameters such as:

- The category of entry to be exported from the connected directory; for example, NT accounts, NT local groups, or NT global groups
- The set of attributes to be exported from the connected directory; typically a subset of the total number of available attributes can be selected through the export configuration file
- Whether a full or delta import or export of entries is performed
- The import integration process, such as whether or not existing entries are updated with imported information, whether or not existing entries are deleted if they match entries in the import file, whether imported attribute values replace existing attributes, and so on.
- The server and/or target directory for the import or export

The synchronization profile-based DirX Identity agents' directory synchronization parameters are generally specified as profile switches in the variable section of the DirX Identity agent's synchronization profile. The synchronization profile-based DirX Identity agents also support import and export mapping rule files that configure the attribute mapping between the directories to be synchronized.

The directory synchronization parameters available in the export and import configuration files or the synchronization profile variable section depend on the individual DirX Identity agent. See the description of each DirX Identity agent for an explanation of the format and content of its export and import configuration files and how they affect the DirX Identity agent's operation.

3.3.2. Import and Export Data Files

Non framework-based agents use import and export data files to communicate with the meta controller, and both DirX Identity agents and the meta controller use these files as intermediate storage in the synchronization process. (Note that protocol access is possible for LDAP-enabled connected directories.)

Each DirX Identity agent supports a specific import data file format in which data to be imported into its connected directory must be formatted and supports a specific export data file format that it generates when exporting data from its connected directory. See the description of each DirX Identity agent for a description of its import and export data file formats.

3.4. JDBC Agent

The JDBC agent is the DirX Identity agent that handles the import and export of information into and out of relational databases. It is based on the Identity Integration Framework.

The JDBC agent can:

- Carry out search (SELECT) operations on configured tables
- Carry out add (INSERT), modify (UPDATE), and delete (DELETE) operations, and
- Execute stored functions and procedures.
- There are a variety of trace-file options.

The following figure illustrates the components of the JDBC agent.

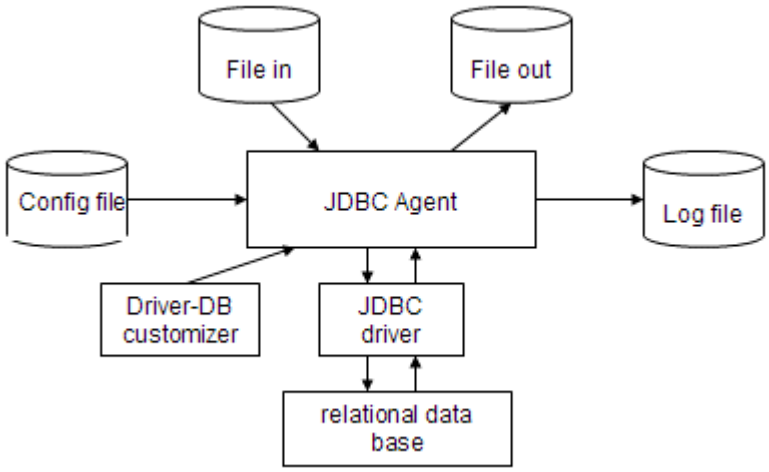


Figure 8. JDBC Agent Components

The JDBC agent carries out the actions specified by the file marked as "File In" in the diagram, and makes a return (depending on the action) to the file marked "File out".

The JDBC agent is able to accept as data input either an SPML request or (in the case of modify operations) LDIF-change format. Similarly, as data output the JDBC agent produces either SPML response or LDIF-content.

The actions of the JDBC agent are normally carried out on information within the relational database shown at the bottom of the diagram. The JDBC driver at lower centre provides a URL-based method for connecting to such databases, as well as giving the means of accessing it for searches and updates. JDBC can also access databases other than relational ones.

JDBC drivers are available for many standard databases. In addition, a generic JDBC/ODBC driver is available on certain target environments, and thus gives connectivity to many database systems for which ODBC access is available. This generic driver is not the preference for high-performance access, but has the necessary facilities to support the JDBC agent.

Databases vary in the list of data types that they can handle or not (e.g. text or decimal strings, integers, date/time, binary data such as photos, etc.). The Driver-DB Customizer shown at the lower left of the diagram provides an optional facility for identifying unsupported data types and for handling any special behavior for particular data types. There is a default customizer that should handle most normal cases.

Events (information-generating events, error events, warning events) are reported in the log-file.

This section describes:

- Agent-specific configuration files for export and import operations
- Data formats

The current agent supports name/password authentication only.

Command-line

The command-line to start the JDBC agent in stand-alone mode is:

java siemens.dxm.connector.framework.AgtSessionExe
-c *configfile*
-m *mappingfile*

The -m flag is mandatory, and must specify the location of the mapping file for the JDBC agent. The default file is jdbcMapping.xml. It is provided as part of the JDBC agent, and must not be modified. The batch file to start the agent inserts this parameter automatically and does not require it as input.

Parameters

configfile

All parameters of JDBC operation are defined in the agent's XML-formatted config file.

mappingfile

Mandatory, and must specify the location of the default file jdbcMapping.xml.

3.4.1. Configuration File

For details, see the section "Configuration" of the JDBC connector.

3.4.2. Input and Output Data File Formats

For details, see the section "Input and Output Data File Formats" of the JDBC connector.

3.4.3. CLASSPATH Environment Variable

You can use the CLASSPATH environment variable to define additional jar files for specific JDBC drivers to be used by the JDBC agent. To define or extend this environment variable:

Windows

If the JDBC agent should be executed under the **system account**, perform these steps:

- Extend the system environment variable CLASSPATH (or add the variable if it does not exist) so that the jar file of the requested JDBC driver is on the classpath and so that this extension is syntactically correct and does not interfere with other applications on your computer.
- Reboot the system

If the JDBC agent should be executed under a **Windows user account**, perform these steps:

- Extend the user environment variable CLASSPATH (or add this variable if it does not exist) so that the jar file of the requested JDBC driver is on the classpath and so that this extension does not interfere with other applications on your computer.
- Using the Expert View of the DirX Identity Manager, configure your JDBC agent job so that it runs under the specified user account (tab Authentication).

UNIX

Define a file **install_path/customer_rc.sh** to extend the environment variable CLASSPATH for the driver. Setting and exporting this variable must be done in separate commands, must be syntactically correct and must not interfere with other applications. The environment setting will become effective for the user after subsequent logins only and for the C++-based Server after the next restart only. Here is a sample content for this file:

```
CLASSPATH=$CLASSPATH:/opt/myjdbc/myjdbc.jar
export CLASSPATH
```

3.4.4. Error Handling

For details, see the section "Error Handling" of the JDBC connector.

3.5. IBM Notes Agent

NotesAgent is the DirX Identity agent that handles the import and export of entries to and from a public IBM Notes address book maintained on an IBM Domino server. NotesAgent can handle entries of any IBM Notes document type.

NotesAgent supports only IBM Notes server and client versions 7.03 or higher. Earlier versions are no longer supported. Use of additional functionality of the Notes APIs enforces this restriction. The agent runs on Windows and requires a co-located Notes Client. NotesAgent uses the Notes API to bind to a Notes server.

NotesAgent can:

- Perform a full or delta export of Person or Group entries from a Notes address book, including multiple attribute values

- Perform a full or delta import of Person or Group entries into a Notes address book, including multiple attribute values
- Create a separate "modify/delete" file of modified and deleted entries as part of the export process
- Create mailboxes and registered users in the Notes address book as part of the import process (includes support of mail replica servers)
- Rename, re-certify and delete registered users in the Notes address book as part of the import process
- Generate an import error file that records all entries that it fails to import
- Generate a log file (for tracing)

The following figures illustrate the components of NotesAgent export and import operations.

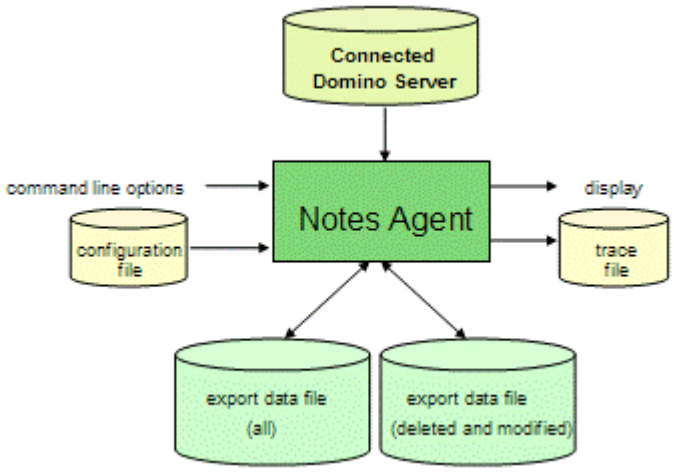


Figure 9. NotesAgent Export Components

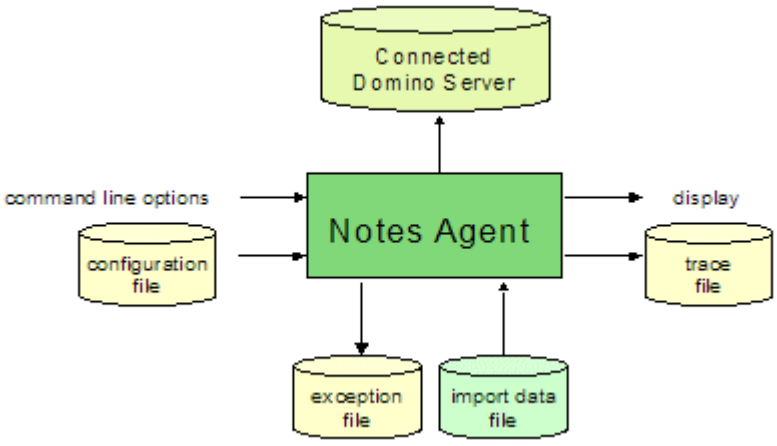


Figure 10. NotesAgent Import Components

The rest of this chapter describes:

- NotesAgent command line format for export and import operations
- NotesAgent configuration files for export and import operations

- The export data file format that NotesAgent generates
- The import data file format that NotesAgent recognizes
- NotesAgent import error file format

These functions use AdminP functionality:

RecertifyUser - uses the AdminP function **ADMINReqRecertify**

RenameUser - uses the AdminP function **ADMINReqRename**

MoveUserInHierarchy - uses the AdminP functions **ADMINReqMoveUserInHier** and **ADMINReqMoveComplete**

DeleteUser - uses the AdminP function **ADMINReqDeleteInNAB**

RegisterNewUser - calls internally **REGNewUser** and sets the flag **fREGCreateMailFileUsingAdminp** if the parameter **CreateMailDBNow** is set in the ini-file

RegisterNewPerson - calls internally **REGNewPerson** and sets the flag **fREGCreateMailFileUsingAdminp** if the parameter **CreateMailDBNow** is set in the ini-file

Sample configuration files and scripts are provided in the **\Samples\Notes** directory of the DirX Identity installation. See the file **NotesReadme.txt** for a description of these files and scripts.

3.5.1. Password Handling

NotesAgent uses the password that grants the credentials to log into a Notes server from an installed Notes client. NotesAgent must use password authentication to a Notes server in order to export data from an address book or import data into it.

The password can be supplied at login:

- Manually, at the user prompt
- Automatically, through the use of password information in the Export and Import ini files.

Information for users of older version of NotesAgent:

In older releases, an Extension Manager for Notes was defined in **notes.ini** of the IBM Notes installation.

The section

```
[Notes]
ExtMGR_ADDINS=nextpwd.dll
```

in **notes.ini** is no longer needed because the NotesAgent directly connects to the IBM Domino server with the “Admin” ID file defined in the Password section.

For details, see “Password Section” in "Password Configuration File Formats".

3.5.2. Command Line Format

The command line format to invoke NotesAgent is as follows:

NotesAgent.exe *sync_switch data_file configuration_file error_file*>_initial_error_file_

3.5.2.1. Parameters

sync_switch

Specifies the type of directory synchronization that NotesAgent is to perform. Possible values are:

/e - Invokes the NotesAgent export function
/i - Invokes the NotesAgent import function

data_file

For export: specifies the pathname of the target export data file that is to contain the entries that NotesAgent extracts from a Notes address book. For delta exports, this file must already exist and is used as the delta base to generate delta information.*
For import:* specifies the pathname of the source file that contains the data to be imported into the Notes address book.

configuration_file

Specifies the name of the file that contains the specifications for the import or export procedure.

error_file

Specifies the name of the file to which NotesAgent is to write error messages about errors that occur during the import or export process, in the format:

error_code
error_message
[*error_specific_information*]

where *error_code* is the code for the error that occurred, *error_message* is a description of the error, and *error_specific_information* is additional information that can appear depending on the type of error. For example:

```
#ProcessAddress error:
#Find more as one document with the following ItemIdentityName(s):
LastName: Test00000
FirstName: Hugo
ShortName: hTest00000
```

In this example, the last three lines are specific for this error.

For an import operation, NotesAgent writes additional information about the entries that it cannot import into the Notes address book into the file specified in *error_file*. See "Import Error File Format" for more details about the contents of the error file on an import operation.

initial_error_file

Specifies the name of the file to which NotesAgent is to write error messages about errors that occur before it creates *error_file*. The error format is the same as that of *error_file*.

3.5.3. Configuration File Formats

NotesAgent uses the following configuration files:

- Notes export configuration file - controls the export of data from a Notes address book
- Notes import configuration file - controls the import of data into a Notes address book
- Password configuration files - automates password authentication during NotesAgent login to a Notes server and enables the assignment of a default password for registered users

See "General Structure of a Configuration File" for a description of the basic organization.

Templates of these configuration files are provided with the NotesAgent installation. The filenames are:

- NotesExport.ini
- NotesImport.ini

In general, you must customize these files to support the requirements of your Notes import and export operations.

3.5.3.1. General Structure of a Configuration File

A NotesAgent configuration file consists of sections and fields defined within those sections. A configuration file has the following structure:

[SectionName***]*
<**;***comment*>
*sectionField**=***fieldValue***
.
.
. *
[SectionName***]*
<**;***comment*>
*sectionField**=**fieldValue*
.
.
.

SectionName is a keyword enclosed in square brackets ([ ]) that identifies the purpose of the section. *sectionField* is a keyword that identifies the field and *fieldValue* is the value assigned to the section field, preceded by the equal sign (=). For example:

```
UpdateExportFile=1
```

Comments can be inserted anywhere in the configuration file and are identified by a semicolon (;) at the beginning of a line.

3.5.3.2. Export Configuration File Format

The NotesAgent export configuration file consists of these sections:

- The Version section (required)
- The Export section (required)
- The Password section (optional)
- The Export Items section (optional)

These sections are described below.

3.5.3.2.1. The Version Section

The Version section consists of a single field that specifies the export configuration file version. The syntax is:

*Version=**version_number*

where *version_number* is the version number assigned to the configuration file, in the format *n*.*nn*. The current version is:

```
Version=1.03
```

This is a mandatory field. This document describes the latest version of the NotesAgent export configuration file. The NotesAgent is able to process configuration files with version number **1.00**, **1.01**, **1.02, 1.03** or "old" files that do not contain a Version section. The following table provides information about the differences between export configuration file versions and about the support of older export configuration file versions for compatibility reasons:

	"Old"	**1.00**	**1.01**	**1.02, 1.03**
FileForDeleteAddr	Supported	Not Supported	Not Supported	Not Supported
FileForModifiedAddr	Supported	Not Supported	Not Supported	Supported

If the version is greater than 1.02, the new field "TypeName" must be in the export configuration file.

3.5.3.2.2. The Export Section

The Export section consists of fields that define the parameters of an export operation for NotesAgent. The next sections describe these fields.

Server

The **Server** field specifies the name of the Notes server that contains the Notes address book from which entries are to be exported. The syntax is:

Server=[*server_name*]

where *server_name* is the name of a Notes server, in the format:

"CN=*server_name*/O=***organization_name***[/...]**"*

For example:

```
Server="CN=westford/O=IRIS/O=NOTES"
```

If *server_name* is not specified, the NotesAgent uses the local Notes address book (the address book that is present on the machine on which NotesAgent is running) as the export target.

This is a mandatory field.

AdrBook

The **AdrBook** field specifies the name of the Notes address book from which entries are to be exported. The syntax is:

AdrBook=*filename*[.nsf*]

where *filename* is the name of a Notes address book managed by the Notes server specified in the **Server** field and **.nsf** is the file extension, which is automatically supplied by NotesAgent if you do not specify it explicitly. (You cannot use a different extension). A single Notes server can support multiple Notes address books. NotesAgent can export from only one Notes address book at a time.

This is a mandatory field.

FormName

The **FormName** field specifies the Notes form of a document. Forms allow users to create documents that store data.

The syntax is:

FormName=*form_name*

where *form_name* is a Notes form name. For example:

FormName=Person

This is a mandatory field.

TypeName

The **TypeName** field specifies the Notes document type to be extracted from the Notes address book. The syntax is:

TypeName=*document_type*

where *document_type* is a Notes document type. For example:

TypeName=Person

This is a mandatory field.

FirstDeltaIsFull

The **FirstDeltaIsFull** field controls weather NotesAgent writes all entries also in the "modify" file. The syntax is:

FirstDeltaIsFull=[*switch*]

where *switch* is one of the following values:

- 0 - Perform the first delta export only in the data file (default)
- 1 - Perform the first delta export in both files.

This is an optional field. If it is not specified (or the field is not present in the configuration file), the NotesAgent exports all entries only in the data file.

SMTPHostDomain

The **SMTPHostDomain** field controls the generation of Internet addresses for Person entries exported from a Notes address book. The syntax is:

SMTPHostDomain=[*domain_name* | **None**]

Notes address books do not store Internet addresses. You can use the **SMTPHostDomain** field to control:

- whether or not Internet addresses are generated for Person entries that are exported from the Notes address book
- the domain supplied in the generated Internet address for each Person entry

Specify the name of a host in *domain_name* to generate an Internet address for each exported entry that uses this domain. For each exported entry, the NotesAgent generates an Internet address in the SMTP format:

name@*domain_name*

where *name* is the value of the **ShortName** attribute of the Notes entry and *domain_name* is the value supplied in *domain_name*. For example, if **SMTPHostDomain** is:

SMTPHostDomain=wstfd.ibm.us

and the value of **ShortName** for the entry is:

ShortName: Ray.Ozzie

the generated Internet address for the entry is:

Ray.Ozzie@wstfd.ibm.us

The Internet address is written to the export data file in this form:

InternetAddress: Ray.Ozzie@wstfd.ibm.us

Specify the keyword **None** to suppress Internet address generation for exported entries.

If no value is specified in this field, NotesAgent generates SMTP-format Internet addresses for the exported entries using the **ShortName** attribute for the *name* part of the address and the value of the **SMTPFullHostDomain** attribute of the Notes server entry ("document", in Notes terminology) in the *hostname* part of the address. It also updates the **SMTPHostDomain** field in the export configuration file with the retrieved Notes **SMTPFullHostDomain** attribute value.

This is a mandatory field.

ModifiedAddresses

The **ModifiedAddresses** field controls whether NotesAgent performs a full or delta export of the document type specified in the **FormName** field from the Notes address book. The syntax is:

ModifiedAddresses=[*switch*]

where *switch* is one of the following values:

- **0 -** Export all entries of the selected document type (default)
- **1 -** Export only those entries that have been added, deleted, or modified after the date specified in the **ModifiedDate** field

If 0 is specified, NotesAgent creates one file that contains all of the entries of the selected document type that are present in the address book. This file is called the "export data file" (or "full export data file") and is the file specified as an import data file to **metacp**. If 1 is specified, NotesAgent creates two files:

- A file that contains all of the entries of the selected document type that are present in the address book (the full export data file); this is the file specified in *data_file* on the command line
- A "modify and delete" file, which contains the entries that have been added, modified or deleted since the date specified in **ModifiedDate** (delta export data file). New and modified entries are identified by a "modify" changetype attribute. Deleted entries are identified by a "delete" changetype attribute. See the section "Delta Export data file Format" for further details about "modify/delete" file format.

NotesAgent creates the "modify/delete" file using the pathname specified in the **FileForModifiedAddr** field.

This is an optional field. If it is not specified (or the field is not present in the

configuration file), NotesAgent exports all entries of the selected document type that are present in the address book.

ModifiedDate

The **ModifiedDate** field specifies the date to be used to select entries for export. The syntax is:

ModifiedDate=*date_and_time*

where *date* is one of the date and time formats supported by Windows. For example:

```
ModifiedDate=22.06.98 14:20:25
```

specifies the date in European date and time format. In this example, all entries added, deleted, or modified after the specified date are to be exported. After NotesAgent performs a delta export, it updates this field in the export configuration file with the current date and time to enable subsequent delta exports. The date and time format specified in this field must match the date and time format selected in the Windows Regional Settings Properties Date and Time tabs.

This is an optional field unless **ModifiedAddresses** is set to 1.

UpdateExportFile

The **UpdateExportFile** field controls (for delta exports only) whether or not the full export data file created by the NotesAgent can be updated. The syntax is:

UpdateExportFile=[*switch*]

where *switch* is one of the following values:

- **0** - Do not update the full export data file
- **1** - Update the full export data file (default)

If **0** is specified, NotesAgent creates a new full export data file on subsequent export operations and preserves the original full export data file it creates on the initial export. If **1** is specified, NotesAgent overwrites the original full export data file on subsequent export operations.

This is an optional field. If it is not specified (or is not present in the configuration file), NotesAgent preserves the original full export data file on subsequent export operations. NotesAgent evaluates this field only when performing a delta export (**ModifiedAddresses** set to 1).

CopyDeletedAdrInModFile

The **CopyDeletedAdrInModFile** field controls whether or not NotesAgent retrieves the contents of entries of the document type selected with the **FormName** field that have been deleted since a full export data file was last generated. The syntax is:

CopyDeletedAdrInModFile=[*switch*]

where *switch* is one of the following values:

- **0** - Do not retrieve the contents of deleted entries
- **1** - Retrieve the contents of deleted entries (default)

The Notes address book retains the identifiers of deleted entries, although their contents are removed. Specifying 0 in this field directs NotesAgent to write the identifiers of the deleted entries to the "modified/deleted" file that it creates if the **ModifiedAddresses** field is set to 1. Specifying 1 in this field directs NotesAgent to retrieve the entry contents associated with the deleted entry identifiers from the most recently generated full export data file, and write the contents and the identifiers into the "modify/delete" file. To use this functionality the field **UpdateExportFile** must be set to 1.

This is an optional field. If it is not specified (or is not present in the configuration file), NotesAgent retrieves the contents of deleted entries. NotesAgent evaluates this field only when performing a delta export (**ModifiedAddresses** set to 1).

FileForModifiedAddr

The **FileForModifiedAddr** field specifies the pathname of the file to which NotesAgent is to write modified (and deleted) entries during a delta export operation. The syntax is:

FileForModifiedAddr=*pathname*

where *pathname* is the name for the "modify/delete" file. For example:

```
FileForModifiedAddr=c:\ibmnotes\ModDelFile
```

This field is optional unless **ModifiedAddresses** is set to 1. NTAgent does not evaluate this field if **ModifiedAddresses** is set to 0.

ExportAllItems

The **ExportAllItems** field controls whether all of the attributes ("items", in Notes terminology) of entries of the document type selected with the **FormName** field are exported, or whether a specified subset of attributes is exported. The syntax is:

ExportAllItems=*switch*

where *switch* is one of the following values:

- **0** - Export only the entry attributes specified in the **ExportItems** section of the configuration file
- **1** - Export all of the entry attributes (default)

This is an optional field. If it is not specified in the configuration file, NotesAgent exports all entry attributes for entries of the selected document type.

SearchDocuments

The **SearchDocuments** field controls whether or not NotesAgent searches for and exports specific entries ("documents", in Notes terminology) of the document type specified in the **FormName** field. The syntax is:

SearchDocuments=*switch*

where *switch* is one of the following values:

- **0** - All entries are exported (no attribute selection criteria are established for entry export) (default)
- **1** - Export only the entries described by the **SearchItemName** and **SearchItemValue** fields

If **SearchDocuments** is set to 1:

- The **ModifiedAddresses** field is ignored
- Values must be supplied for the **SearchItemName** and **SearchItemValue** fields

This is an optional field. If it is not present in the configuration file, NotesAgent exports all entries of the selected document type.

SearchItemName

The **SearchItemName** field specifies an attribute within an entry to search for. The syntax is:

SearchItemName=*attribute_name*

where *attribute_name* is a Notes attribute name for an attribute ("item") that can be present in an entry of the document type specified in the **FormName** field. For example:

```
SearchItemName=Department
```

directs NotesAgent to search all entries of the selected document type for the Department attribute. The entry is exported if it has the value specified in the **SearchItemValue** field.

This field is optional unless **SearchDocuments** is set to 1.

SearchItemValue

The **SearchItemValue** field specifies a value to search for (case exact match), given an attribute name to search for that is specified in the **SearchItemName** field. The syntax is:

SearchItemValue=*attribute_value*

For example:

```
SearchItemName=Department
SearchItemValue=Iris
```

directs NotesAgent to search all entries in the Notes address book for the Department attribute, and export entries whose value for the Department attribute is Iris.

This field is optional unless **SearchDocuments** is set to 1.

Separator

The **Separator** field specifies a value to be used to separate the individual attribute values of a multivalued attribute. The syntax is:

Separator=[*character*]

where *character* is a character or a string used as a multi-valued attribute separator.

This field is optional. If it is not specified (or not present in the configuration file), NotesAgent uses the comma (,) as the multi-valued attribute separator.

Trace

The **Trace** field controls whether NotesAgent performs program flow tracing on an export operation. The syntax is:

Trace=[*switch*]

where *switch* is one of the following values:

- **0** - Do not perform program flow tracing on the export operation (default)
- **1** - Perform program flow tracing on the export operation

If **1** is specified, NotesAgent writes information about the export operation to the pathname specified in the **TraceFileName** field. The type of information stored in the trace file depends upon the settings of the **TraceLevel_1**, **TraceLevel_2**, and **TraceLevel_3** fields. If **Trace** is set to 1, one of the trace level fields must also be set to 1.

TraceLevel_1

The **TraceLevel_1** field controls whether NotesAgent writes level 1 tracing information about the export operation. Level 1 tracing information includes a dump of the configuration file, number of documents, and other program flow variables.

The syntax is:

TraceLevel_1=[*switch*]

where *switch* is one of the following values:

- **0** - Do not write level 1 trace information (default)
- **1** - Write level 1 trace information

If **1** is specified, NotesAgent writes level 1 trace information about the export operation to the pathname specified in the **TraceFileName** field.

TraceLevel_2

The **TraceLevel_2** field controls whether NotesAgent writes level 2 tracing information about the export operation. Level 2 tracing provides more detailed information about program flow than is provided in level 1 tracing.

The syntax is:

TraceLevel_2=[*switch*]

where *switch* is one of the following values:

- **0** - Do not write level 2 trace information (default)
- **1** - Write level 2 trace information

If **1** is specified, NotesAgent writes level 2 trace information about the export operation to the pathname specified in the **TraceFileName** field.

TraceLevel_3

The **TraceLevel_3** field controls whether NotesAgent writes level 3 tracing information about the export operation. Level 3 tracing provides more detailed information about program flow than is provided in level 2 tracing The syntax is:

TraceLevel_3=[*switch*]

where *switch* is one of the following values:

- **0** - Do not write level 3 trace information (default)
- **1** - Write level 3 trace information

If **1** is specified, NotesAgent writes level 3 trace information about the export operation to the pathname specified in the **TraceFileName** field.

TraceFileName

The **TraceFileName** field specifies the pathname of the trace file to which NotesAgent is to write information about the export operation. The syntax is:

TraceFileName=*pathname*

where *pathname* is the name for the trace file. For example:

```
TraceFileName=c:\ibmnotes\ExportTraceFile
```

This field is optional unless **Trace** is set to 1. NotesAgent does not evaluate this field if **Trace** is set to 0.

3.5.3.2.3. The Password (Password) Section

The Password section consists of fields that define the parameters for NotesAgent automated password authentication. The next sections describe these fields.

PathFilePassword

The **PathFilePassword** field is only used for the old password handling mechanism.

For a description of the old password handling mechanism, see the section "Password Configuration File Formats".

The syntax for this field is:

PathFilePassword=*pathname*

where *pathname* is the path to the password configuration file NotesPassword.ini. For example:

PathFilePassword=a:\NotesSync\NotesPassword.ini

For security reasons, it is recommended that the password configuration file not be stored on disk, since it contains human-readable representations of Notes address book administrator and registered user passwords.

AutomaticPW

The **AutomaticPW** field specifies the password that NotesAgent (in conjunction with a NotesAgent DLL) is to use to decode the admin.id certificate; this is the certificate that grants it the credentials to log in to the Notes server

The "PathAndFileOfNotesIDFile" field can alternatively be indicated.

If the field "PathAndFileOfNotesIDFile" is available for a ID file, it has higher priority.

The syntax is:

AutomaticPW*=password*

For example:

AutomaticPW=notes

PathAndFileOfNotesIDFile

The **PathAndFileOfNotesIDFile** field specifies the password that NotesAgent (in conjunction with a NotesAgent DLL) is to use to decode the admin.id certificate; this is the certificate that grants it the credentials to log in to the Notes server

For this Notes ID file the Notes Agent needs a pair of "Path and file name of ID file" and the "corresponding password".

Path and file Name of notes.id=password

For example:

C:\IBM\Notes\admin.id=notes

The "AutomaticPW" field can alternatively be indicated.

If the field "PathAndFileOfNotesIDFile" is available for a ID file, it has higher priority

3.5.3.2.4. The Export Items Section

The **Export Items** section is an optional section of the export configuration file that specifies a set of entry attributes to be exported from a Notes address book. The section is only present if the **ExportAllItems** field in the **Export** section is set to 1. The syntax is:

attribute_name=switch

where *attribute_name* is the name of an entry attribute and *switch* is one of the following values:

- **0** - Do not export the attribute value for *attribute_name*
- **1** - Export the attribute value for *attribute_name*

For example:

```
[ExportItems]

LastName=1
FirstName=1
Location=0
```

Use the *switch* parameter to select or exclude attributes in the list for export. See the installed **NotesExport.ini** file for a list of attribute names known to the NotesAgent.

3.5.3.3. Import Configuration File Format

The NotesAgent import configuration file consists of the following sections:

- The **Version** section (required)
- The **Import** section (required)
- The Registered User (**RegUser**) section (optional)
- The Password (**Password**) section
- The EncryptedAttributes (**EncryptedAttributes**) section

The next sections describe these sections.

3.5.3.3.1. The Version Section

The Version section consists of a single field that specifies the import configuration file version. The syntax is:

Version=*version_number*

where *version_number* is the version number assigned to the configuration file, in the format *n*.*nn*. The current version is:

`Version=1.02`

This is a mandatory field. This document describes the latest version of the NotesAgent import configuration file. There are no differences between import configuration file versions.

3.5.3.3.2. The Import Section

The Import section consists of fields that define the parameters of the import operation for NotesAgent. The next sections describe these fields.

Server

The **Server** field specifies the name of the Notes server that contains the Notes address book to which entries are to be imported. It has the same syntax and default as the **Server** field in the export configuration file and is a mandatory field.

AdrBook

The **AdrBook** field specifies the name of the Notes address book to which entries are to be imported. It has the same syntax and default as the **AdrBook** field in the export configuration file and is a mandatory field.

FormName

The **FormName** field specifies the Notes document type to be imported into the target Notes address book. It has the same syntax as the **FormName** field in the export configuration file and is a mandatory field.

ItemIdentityName[1,2,3]

The **ItemIdentityName** fields control how NotesAgent matches entries in the target Notes address book with entries to be imported into the address book. The syntax is:

ItemIdentityName1=*attribute_name* | **ViewSearchName**
ItemIdentityName2=*attribute_name*
ItemIdentityName3=*attribute_name*

where *attribute_name* is the name of a Notes attribute in an import entry whose value NotesAgent is to use match against entries in the Notes address book. NotesAgent uses case-exact match unless the **CaseSensitive** field is set to 0. Specifying wildcards is not supported.

If the **Update** field is set to 1, at least one **ItemIdentityName** field must be specified. When multiple **ItemIdentityName** fields are specified, NotesAgent "ANDs" the fields; there is no "OR" function.

When the **ItemIdentityName1** field contains the **ViewSearchName** value and **ViewFolder** specifies a Notes view, it indicates that the view specified in **ViewFolder** has been sorted by a composite attribute and that the entries in the import data file have a **ViewSearchName** attribute that contains the composite attribute value. For example, suppose the view specified in **ViewFolder** has been sorted by the composite attribute:

LastName **,** *FirstName*

The **ItemIdentityName1** field must specify:

ItemIdentityName1=ViewSearchName

and each entry in the import data file contains a ViewSearchName attribute type. For example:

```
FirstName: Thomas
LastName: Diaz
```

```
ViewSearchName: Diaz , Thomas
```

The composite attribute format used in the import data file must match exactly with the format used in the sorted view. NotesAgent uses the sorted view and the ViewSearchName attribute values to match entries in the import data file against entries in the Notes address book. The **ViewSearchName** value is only relevant when the **ViewFolder** field is used.

This field can be in each entry in the import data file. If it is specified in the configuration file it acts as a default value. This means that the value in the import data file can override this default setting.

Update

The **Update** field controls whether or not existing Notes entries are modified with imported information or whether a new entry with the imported information is created, even if a matching entry already exists in the address book. The syntax is:

Update=[*switch*]

where *switch* is one of the following values:

- **0** - Always create a new Notes entry for an imported entry (create a new Notes ID), even if a Notes entry that matches it already exists in the address book
- **1** - Modify matching Notes entries in the address book and create new Notes entries if there are no matches for them in the address book (default)

NotesAgent uses the values supplied in the **ItemIdentityName** fields to determine whether matching entries exist.

This is an optional field. If it is not specified (or not present in the configuration file) NotesAgent updates existing Notes entries and creates non-existent Notes entries.

This field can be in each entry in the import data file. If it is specified in the configuration file it acts as a default value. This means that the value in the import data file can override this default setting.

ExactAction

The **ExactAction** field controls whether or not the ChangeType and the ItemType fields are used exactly as defined in the import data file.

Exact action for ChangeType means that the agent does not create an entry if ChangeType is set to modify but the entry does not exist. Otherwise a new entry is created.

Exact action for ItemType means also that the agent does not import an entry if the item in the import data file does not exist in the Notes database or has another ItemType as Text, Number or DateTime. Otherwise the item is created with item type "Text".

The syntax is:

ExactAction =[*switch*]

where *switch* is one of the following values:

- 0 - No exact action for ChangeType, no exact action for ItemType (default).
- 1 - Exact action for ChangeType, no exact action for ItemType.
- 2 - No exact action for ChangeType, exact action for ItemType.
- 3 - Exact action for ChangeType, exact action for ItemType.

This is an optional field.

ReplaceItem

The **ReplaceItem** field controls whether or not existing attribute values of Notes entries in the address book are overwritten with imported values. The syntax is:

ReplaceItem=[*switch*]

where *switch* is one of the following values:

- **0** - Add imported attribute values as multiple attribute values for the attribute (create a multi-valued attribute) (default)
- **1** - Replace existing attribute values with imported attribute values

This is an optional field. If it is not specified (or is not present in the configuration file), NotesAgent adds the imported attribute values to the existing attribute or performs operations on attributes as specified in a "modify" changetype entry.

If the **ReplaceItem** field is set to 1, NotesAgent reads the full entry and sorts it according to the attribute modification operations present in the entry. If an attribute has more than one attribute modification specified for it, NotesAgent performs a "replace" attribute modification operation, regardless of the attribute modification operation specified in the import data file.

This field can be in each entry in the import data file. If it is specified in the configuration file it acts as a default value. This means that the value in the import data file can override this default setting.

SaveOrgDB

The **SaveOrgDB** field controls whether or not NotesAgent backs up the target Notes address book before performing the import operation. The syntax is:

SaveOrgDB=*switch*

where *switch* is one of the following values:

- **0** - Do not back up the target Notes address book before import
- **1** - Back up the target Notes address book before import

If **SaveOrgDB** is set to 1, NotesAgent writes the contents of the target Notes address

book to the file specified by the **SaveDBName** field on the Notes server specified in the **SaveServerName** field.

This is an optional field. If it is not specified (or is not present in the configuration file), NotesAgent does not perform a backup.

SaveServerName

The **SaveServerName** field specifies the name of a Notes server that NotesAgent is to use as a storage target when backing up a Notes address book before an import operation. The syntax is:

SaveServerName=[*server_name*]

where *server_name* is the name of the Notes server to which the Notes address book is to be written, in the syntax:

"CN=*server_name***/O=***organization_name*[/...]"

For example:

SaveServerName="CN=Cambridge1/O=Notes/O=IBM"

If no value is specified for this field, NotesAgent writes the contents of the Notes address book to the Windows system on which it is running.

This is a mandatory field.

SaveDBName

The **SaveDBName** field specifies the name of the file to which NotesAgent is to write the contents of a target Notes address book before an import operation. The syntax is:

SaveDBName=[*filename*[**.nsf**]]

When specifying a filename, you can omit the **.nsf** file extension; it is automatically supplied by NotesAgent if you do not specify it explicitly. (You cannot supply a different extension, or Notes will be unable to open the saved file). For example:

```
SaveDBName=namessave.nsf
```

This is a required field if **SaveOrgDB** is set to 1.

DeleteEntries

The **DeleteEntries** field controls whether or not entries that exist in the Notes address book are to be deleted if matching entries exist in the import data file. The syntax is:

DeleteEntries=*switch*

where *switch* is one of the following values:

- **0** - Do not delete entries in the Notes address book that match entries to be imported (default)

- **1** - Delete entries in the Notes address book that match entries to be imported

NotesAgent uses the **ItemIdentityName** field(s) to determine whether entries in the address book match entries to be imported. If you plan to set **DeleteEntries** to 1, it is strongly recommended that you use all three **ItemIdentityName** fields and that you back up the Notes address book before performing the import.

The **DeleteEntries** field takes precedence over the **Update** field. That is, if **Update** is set to 0, and **DeleteEntries** is set to 1, NotesAgent will delete entries from the Notes address book that match entries to be imported. This field has a higher precedence than any per-entry "changetype" operations specified in the import data file.

This is an optional field. If it is not specified (or is not present in the configuration file), NotesAgent does not delete any entries in the Notes address book when it performs the import.

CreateTestAddresses

The **CreateTestAddresses** field is used to implement a test function on the import process. The syntax is:

CreateTestAddress=*number* | **0**

You can append a 5-digit "test" address to one or more attributes in a Notes import data file. Initially, the number is set to 0. For example:

```
LastName: Kawell00000
```

Specifying a number in *number* directs NotesAgent to process the import data file that number of times. On each processing cycle, NotesAgent increments the 5-digit test addresses you have inserted. You can use the test address function to distinguish the imported entries from the entries that exist in the Notes address book.

This is an optional field. If it is not specified (or is not present in the configuration file), NotesAgent does not increment test addresses.

Separator

The **Separator** field specifies a value to be used to separate the individual attribute values of a multi-valued attribute. It has the same syntax as the **Separator** field in the export configuration file.

RegisterUser

The **RegisterUser** field controls whether or not NotesAgent registers imported entries as Notes users. The syntax is:

RegisterUser=*switch*

where *switch* is one of the following values:

- **0** - Do not register imported entries as Notes users (default).
- **1** - Register imported entries as Notes users and create corresponding mail files immediately. (Internally the C++-API is called: LNCertifier.RegisterUser.)

- **2** - Register imported entries as Notes users and create requests for the Administration Process to create corresponding mail files. (Internally the API is called: REGNewWorkstationExtended.)
- **3** - Register imported entries as Notes users and select whether mail files shall be created immediately or only requests to create corresponding mail files for the Administration Process shall be created. In this mode you can set the mail template and quota for mail files. (Internally the API is called: REGNewUser.)
- **4** - Register imported entries as Notes users and select whether mail files shall be created immediately or only requests to create corresponding mail files for the Administration Process shall be created. In this mode you can set the mail template and quota for mail files. Furthermore mail replica servers can be defined. (Internally the API is called: REGNewPerson.)

If the **RegisterUser** field is set to 1, 2, 3 or 4 and the **Update** field is set to 0, NotesAgent always registers all imported entries as Notes users. If the **RegisterUser** field is set to 1, 2, 3 or 4 and the **Update** field is set to 1, NotesAgent only registers an entry as a Notes user if the entry has not already been registered.

Notes uses a distributed architecture and so several Notes servers participate in the user registration operation. If **RegisterUser** is set to 1, the operation is synchronous and completes successfully only if all of the required Notes servers are available. NotesAgent receives a response about successful or unsuccessful completion.

If **RegisterUser** is set to 2, the user registration operation is asynchronous: NotesAgent registers the users and then submits request documents (to create the mail files) to the Notes Administration Process (**adminp**) request database. When the Administration Process starts (the administrator specifies a time interval for startup), it consults its database and attempts to perform the request. If one of the necessary Notes servers is not available, the Administration Process tries the operation again the next time it starts up. NotesAgent does not receive a response about successful or unsuccessful completion.

If **RegisterUser** is set to 3, you can select wether mail files shall be created immediately or only requests to create corresponding mail files (CreateMailDBNow) for the Notes Administration Process shall be created. In this mode you can set the mail template (MailTemplate), quota for mail files (DbQuotaSizeLimit, DbQuotaWarningThreshold), the SMTP Host Domain for the internet address of the user (SMTPHostDomain), and the mail parameters (MailOwnerAccess, MailSystem, MailACLManager, MailForwardAddress).

If **RegisterUser** is set to 4, the same options as for option 3 apply. Furthermore you can define the following additional attributes: MailReplicaServer, PreferredLanguage, AltLanguage, OnDuplicate, .PasswordKeyWIdth, KeyWidth, InetKeyWIdth, PasswordQuality. Note that this option should be used if you want to define mail replica servers. Calling the API **REGNewPerson** sets the mail replica servers. The API **REGNewPerson** internally requires the additional attributes listed above.

You can use the Notes Client user interface to configure the Notes Administration Process. You will find the configuration parameters in the section "Administration Process" in the Server/Servers document of the address book.

This field can be in each entry in the import data file. If it is specified it acts as a default value. This means that the value in the import data file can override this default setting.

PathFileTargetCertId

The **PathFileTargetCertId** field specifies the pathname to the certificate ID file of a target organizational unit. The file contains the certificate that grants NotesAgent the right to create registered users for the organizational unit. The syntax is:

PathFileTargetCertId=*pathname*

where *pathname* is the pathname to the certificate ID file. For example:

```
PathFileTargetCertId=a:\German.id
```

This is a required field if the import operation is to process the "MoveUserInHier" changetype operation. See the section "Import Data File Format" for further details about these operations.

This field can be in each entry in the import data file. If it is specified it acts as a default value. This means that the value in the import data file can override this default setting.

Trace

The **Trace** field controls whether NotesAgent performs program flow tracing on an import operation. The syntax is:

Trace=[*switch*]

where *switch* is one of the following values:

- **0** - Do not perform program flow tracing on the import operation (default)
- **1** - Perform program flow tracing on the import operation

If **1** is specified, NotesAgent writes information about the import operation to the pathname specified in the **TraceFileName** field. The type of information stored in the trace file depends upon the settings of the **TraceLevel_1**, **TraceLevel_2**, and **TraceLevel_3** fields.

TraceLevel_1

The **TraceLevel_1** field controls whether NotesAgent writes level 1 tracing information about the import operation. It has the same syntax as the **TraceLevel_1** field in the export configuration file and is an optional field.

TraceLevel_2

The **TraceLevel_2** field controls whether NotesAgent writes level 2 tracing information about the import operation. It has the same syntax as the **TraceLevel_2** field in the export configuration file and is an optional field.

TraceLevel_3

The **TraceLevel_3** field controls whether NotesAgent writes level 3 tracing information about the import operation. It has the same syntax as the **TraceLevel_3** field in the

export configuration file and is an optional field.

TraceFileName

The **TraceFileName** field specifies the pathname of the trace file to which NotesAgent is to write information about the import operation. It has the same syntax as the **TraceFileName** field in the export configuration file and is an optional field unless the **Trace** field is specified.

TraceItemTypes

The **TraceItemTypes** field controls whether NotesAgent writes all item types of all items of the data base (AddrBook) in trace file.

The syntax is:

TraceItemTypes=[*switch*]

where switch is one of the following values:

- **0** - Do not write item types in trace file (default)
- **1** - Write item types in trace file

For example:

```
Item data types:
=========================
PhoneNumber = TEXT
Form = TEXT
PasswordChangeInterval = NUMBER
PasswordChangeDate = TIME
```

AdminReqDB

The **AdminReqDB** field specifies the name of the Notes Administration Process (**adminp**) request database to which NotesAgent is to send request documents during "DeleteUser" changetype processing. The syntax is:

AdminReqDB=[*filename*[**.nsf**]]

When specifying a filename, you can omit the **.nsf** file extension; it is automatically supplied by NotesAgent if you do not specify it explicitly. (You cannot supply a different extension, or Notes will be unable to open the saved file). For example:

```
AdminReqDB=admin4.nsf
```

This is a required field if the import data file to be processed contains "DeleteUser" changetype entries.

AdminReqAuthor

The **AdminReqAuthor** field specifies the author name of the Notes Administration

Process (**adminp**) request database to which NotesAgent is to send request documents during "DeleteUser" changetype processing. The syntax is:

AdminReqAuthor=*name*

where *name* is the author name in canonical format. For example:

```
AdminReqAuthor=CN=Thomas Diaz/OU=USA/O=Iris
```

This is a required field if the import data file to be processed contains "DeleteUser" changetype entries.

SearchUniversalID

The **SearchUniversalID** field controls whether NotesAgent uses the Universal identifier to match entries to be updated in the target Notes address book with entries to be imported into the address book. The syntax is:

SearchUniversalID=switch

where switch is one of the following values:

- **0** - Do not use the Universal identifier to search for a matching entry (default)
- **1** - Use the Universal identifier to search for a matching entry

If **SearchUniversalID** is set to 1, NotesAgent uses the Universal identifier (UniversalIDPart1: to UniversalIDPart4:) of an entry in the import data file as a search key for finding a matching entry in the Notes address book. Using the Universal identifiers to match import data file entries with their counterparts in the Notes address book results in faster import operations.

This is an optional field. If it is not specified (or is not present in the configuration file), NotesAgent uses the **SearchNoteID** functionality to match entries. If SearchNoteID is not specified, NotesAgent uses the **ViewFolder** field to match entries. If ViewFolder is not specified, NotesAgent uses the **ItemIdentityName** fields to match entries.

SearchNoteID

The **SearchNoteID** field controls whether or not NotesAgent uses the Notes identifier to match entries to be updated in the target Notes address book with entries to be imported into the address book. The syntax is:

SearchNoteID=*switch*

where *switch* is one of the following values:

- **0** - Do not use the Notes identifier to search for a matching entry (default)
- **1** - Use the Notes identifier to search for a matching entry

If **SearchNoteID** is set to 1, NotesAgent uses the Notes identifier of an entry in the import data file as a search key for finding a matching entry in the Notes address book. Using the Notes identifiers to match import data file entries with their counterparts in the Notes address book results in faster import operations.

This is an optional field. If it is not specified (or is not present in the configuration file), NotesAgent uses the **ViewFolder** field to match entries. If **ViewFolder** is not specified, NotesAgent uses the **ItemIdentityName** fields to match entries.

ViewFolder

The **ViewFolder** field controls whether NotesAgent uses a Notes view sorted by the Notes attribute specified in the **ItemIdentityName1** field to match entries to be updated in the target Notes address book with the entries to be imported into the address book. The syntax is:

ViewFolder=[*view_name*]

where *view_name* is a Notes view. For example:

ViewFolder=People

The specified Notes view must contain the sorted column that has been sorted by the attribute specified in the **ItemIdentityName1** field. For example, if **ViewFolder** specifies "People", and **ItemIdentityName1** specifies "LastName", the "People" view must contain a column that has been sorted by the "LastName" attribute. The **ItemIdentityName1** field can also specify the value **ViewSearchName** to indicate that the view has been sorted by a composite attribute, for example, FirstName , LastName. See the **ItemIdentityName** field description for further details.

NotesAgent uses the view specified in **ViewFolder** sorted by the Notes attribute specified in **ItemIdentityName1** to match the entry when **SearchNoteID** is set to 0, or when **SearchNoteID** is set to 1 but a Notes identifier does not exist for the entry.

This is an optional field. If it is not specified (or is not present in the configuration file), and the **SearchNoteID** field is set to 0, NotesAgent uses the values specified in **ItemIdentityName1**, **ItemIdentityName2**, or **ItemIdentityName3** fields to match entries.

CaseSensitive

The **CaseSensitive** field controls whether or not NotesAgent uses case-exact match when using a sorted Notes view to match entries in the target Notes address book with entries to be imported. The syntax is:

CaseSensitive=*switch*

where *switch* is one of the following values:

- **0** - Do not use case-exact match
- **1** - Use case-exact match (default)

When **CaseSensitive** is set to 0, NotesAgent uses case-insensitive matching when matching the values in the sorted view against Notes address book entries. When **CaseSensitive** is set to 1, NotesAgent uses case-sensitive matching.

This is an optional field and is only relevant if the **ViewFolder** field is used. If it is not specified (or is not present in the configuration file), NotesAgent uses case-sensitive matching.

ComputeWithFormIgnoreErrors

The **ComputeWithFormIgnoreErrors** field specifies the way the Notes-API "ComputeWithForm" is called before the Notes document is saved. ("ComputeWithForm" calculates computed fields and evaluates validation formulas defined in the form used by the Notes document.)

The syntax is:

ComputeWithFormIgnoreErrors=*switch*

where *switch* is one of the following values:

- **0** - if you want the function to stop at the first error
- **1** - if you do not want the function to stop executing if a validation error occurs

This field can be in each entry in the import data file. If it is specified in the configuration file it acts as a default value. This means that the value in the import data file can override this default setting. ++ If **ComputeWithFormIgnoreErrors** is not defined then the Notes-API "ComputeWithForm" is not called.

3.5.3.3.3. The Registered User (RegUser) Section

NotesAgent can register entries that it imports into a Notes address book as Notes users with their own mailboxes during the import process. The Registered User section provides the information that NotesAgent needs in order to perform this task and is only required if the **RegisterUser** field in the Import section is set to 1, 2, 3 or 4. The next sections describe the fields of the Registered User section.

MailboxName

The **MailboxName** field specifies the mailbox name. The syntax is:

MailboxName=*mailbox_name*

where *mailbox_name* is the name of the mailbox. For example:

```
MailboxName=mail/thcook,nsf
```

If the **MailboxName** field is specified then it is used to setup the mname of the mailbox; in this case the **ItemMailboxName** field is ignored.

ItemMailboxName

The **ItemMailboxName** field specifies the attribute to use as the mailbox name. The syntax is:

ItemMailboxName=*attribute_name*

where *attribute_name* is the name of a Notes attribute whose value NotesAgent should use as a mailbox name when registering the entry as a Notes user and creating a mailbox for it. For example:

```
ItemMailboxName=ShortName
```

If the **ItemMailboxName** field is specified, the entries in the import data file must contain values in the attribute specified in *attribute_name*.

This field can be in each entry in the import data file. If it is specified in the configuration file it acts as a default value. This means that the value in the import data file can override this default setting.

Internally the following value for the mailbox name is generated:

Mail/*value_of_attribute_name***.nsf**

The **ItemMailboxName** field is ignored if the **MailboxName** field is specified.

ItemUserId

The **ItemUserId** field specifies the attribute to use as the User ID. The syntax is:

ItemUserId=*attribute_name*

where *attribute_name* is the name of a Notes attribute whose value NotesAgent should use as a user ID when registering the entry as a Notes user and creating a mailbox for it. For example:

```
ItemUserId=ShortName
```

If the **ItemUserId** field is specified, the entries in the import data file must contain values in the attribute specified in *attribute_name*.

This field can be in each entry in the import data file. If it is specified in the configuration file it acts as a default value. This means that the value in the import data file can override this default setting.

PathFileCertId

The **PathFileCertId** field specifies the pathname to the certificate ID file **cert.id**, which is a binary file that is supplied with the Notes Server installation software. This file contains the certificate that grants NotesAgent the right to create registered users. The syntax is:

PathFileCertId=*pathname*

where *pathname* is the pathname to the certificate ID file. For example:

```
PathFileCertId=a:\cert.id
```

This is a required field if the import operation is to process the "RenameUser" and "RecertifyUser" changetype operations or if the **RegisterUser** field is set to 1, 2, 3 or 4. This is a required field that must specify the pathname to the certificate ID file of the source organizational unit if the import operation is to process the "MoveUserInHier" changetype operation.

See the section "Import Data File Format" for further details about these operations.

This field can be in each entry in the import data file. If it is specified in the configuration file it acts as a default value. This means that the value in the import data file can

override this default setting.

PathFileCertLog

The **PathFileCertLog** field specifies the pathname to the certifier logging file certlog.nsf on the server. This file contains the certifier logging entries of the registered users. The syntax is:

PathFileCertLog=*pathname*

where pathname is the pathname to the certifier logging file. For example:

```
PathFileCertLog=d:\ibm\domino\data\certlog.nsf
```

This is a required field if the RegisterUser field is set to 1, 2, 3 or 4.

This field can be in each entry in the import data file. If it is specified in the configuration file it acts as a default value. This means that the value in the import data file can override this default setting.

PathUserId

The **PathUserId** field specifies the directory in which NotesAgent is to store Notes user IDs created during the user registration process. The syntax is:

PathUserId=*directory*

where *directory* is a directory pathname. For example:

```
PathUserId=e:\notes\data
```

Notes User IDs are binary user certificate files that NotesAgent creates during the registration process if **CreateIdFile** is set to 1. NotesAgent writes these user ID files to the directory specified in the **PathUserId** field if **SaveIdInFile** field is set to 1.

This field can be in each entry in the import data file. If it is specified in the configuration file it acts as a default value. This means that the value in the import data file can override this default setting.

RegistrationServer

The **RegistrationServer** field specifies the name of the Notes registration server that is to register the users in the Notes server address book. The syntax is:

RegistrationServer=*server_name*

where *server_name* is a the name of a Notes server in the format:

"CN=*server_name***/O=***organization_name*[/...]"

For example:

```
RegistrationServer="CN=Cambridge3/O=Notes/O=IBM"
```

This field can be in each entry in the import data file. If it is specified in the configuration

file it acts as a default value. This means that the value in the import data file can override this default setting.

MailServer

The **MailServer** field specifies the name of a Notes server on which NotesAgent is to create user mailboxes during the user registration process. The syntax is:

***MailServer=**/*server_name*

where *server_name* is a the name of a Notes server in the format:

"CN=*server_name***/O=***organization_name*[/...]"

For example:

```
MailServer="CN=Cambridge4/O=Notes/O=IBM"
```

Entries in the import data file can also specify (as an attribute of the entry) the name of the Notes server on which to create user mailboxes. Mailboxes specified in import entries override the specification in the **MailServer** field.

This field can be in each entry in the import data file. If it is specified in the configuration file it acts as a default value. This means that the value in the import data file can override this default setting.

MinPasswordLength

The **MinPasswordLength** field specifies the minimum number of characters that a user password must have. The syntax is:

MinPasswordLength=*number*

For example:

```
MinPasswordLength=5
```

NotesAgent sets the specified value as an attribute of the registered user entry.

If the value is set to 0 the **SaveIdInAddressBook** field also must be set to 0.

This field can be in each entry in the import data file. If it is specified in the configuration file it acts as a default value. This means that the value in the import data file can override this default setting.

CreateAddressBookEntry

The **CreateAddressBookEntry** field controls whether NotesAgent creates Notes entries in the target Notes address book for Notes users that it registers during the import process. The syntax is:

CreateAddressBookEntry=*switch*

where *switch* is one of the following values:

- **0** - Register Notes users, but do not create Notes entries for them
- **1** - Register Notes users and create Notes entries for them

This field can be in each entry in the import data file. If it is specified in the configuration file it acts as a default value. This means that the value in the import data file can override this default setting.

CreateMailDatabase

The **CreateMailDatabase** field controls whether NotesAgent creates user mailboxes for Notes users that it registers during the import process. The syntax is:

CreateMailDatabase=*switch*

where *switch* is one of the following values:

- **0** - Register Notes users, but do not create mailboxes for them
- **1** - Register Notes users and create mailboxes for them

This field can be in each entry in the import data file. If it is specified in the configuration file it acts as a default value. This means that the value in the import data file can override this default setting.

CreateIdFile

The **CreateIdFile** field controls whether NotesAgent creates a user ID file for Notes users that it registers during the import process. The syntax is:

CreateIdFile=+*switch*

where *switch* is one of the following values:

- **0** - Register Notes users, but do not create a user ID file for them
- **1** - Register Notes users and create a user ID file for them

If **CreateIDFile** is set to 1, either the **SaveIdInAddressBook** field or the **SaveIdInFile** field (or both) must be set to 1 to specify where NotesAgent is to store the user ID files it creates.

This field can be in each entry in the import data file. If it is specified in the configuration file it acts as a default value. This means that the value in the import data file can override this default setting.

SaveIdInAddressBook

The **SaveIdInAddressBook** field controls whether or not NotesAgent saves the user ID files it creates as attachments of the Notes entries for the registered users. The syntax is:

SaveIdInAddressBook=*switch*

where *switch* is one of the following values:

- **0** - Do not save user ID files as attachments of the Notes entries for the registered

users

- **1** - Save user ID files as attachments of the Notes entries for the registered users in the Notes address book

If **SaveIdInAddressBook** is set to 1, NotesAgent creates the user ID file and stores it as an attachment of the corresponding Person entry for the registered user. If **SaveIdInAddressBook** is set to 1, the registered user must have got a password.

This field can be in each entry in the import data file. If it is specified in the configuration file it acts as a default value. This means that the value in the import data file can override this default setting.

SaveIdInFile

The **SaveIdInFile** field controls whether or not NotesAgent saves the user ID files it creates in individual files. The syntax is:

SaveIdInFile=*switch*

where *switch* is one of the following values:

- **0** - Do not save user ID files in individual files
- **1** - Save user ID files in individual files

If **SaveIdInFile** is set to 1, NotesAgent creates the user ID files and stores them in the directory specified in the **PathUserId** field.

This field can be in each entry in the import data file. If it is specified in the configuration file it acts as a default value. This means that the value in the import data file can override this default setting.

SaveInternetPassword

The **SaveInternetPassword** field controls whether or not NotesAgent saves the user ID password also for use as an Internet password. The syntax is:

SaveInternetPassword=*switch*

where switch is one of the following values:

- **0** - Do not save user ID password also as Internet password
- **1** - Save user ID password also as Internet password

If **SaveInternetPassword** is set to 1, NotesAgent saves the user ID password also in the field for the Internet password.

This field can be contained in each entry in the import data file. If it is specified in the configuration file it acts as a default value. This means that the value in the import data file can override this default setting.

CreateNorthAmericanId

The **CreateNorthAmericanId** field controls whether or not NotesAgent creates United

States security encrypted User ID files. The syntax is:

CreateNorthAmericanId=*switch*

where *switch* is one of the following values:

- **0** - Do not create U.S.-encrypted user ID files
- **1** - Create U.S.-encrypted user ID files

If **CreateNorthAmericanId** is set to 1, the Notes registered user can only be used within the United States. This field is disabled for NotesAgent installations outside the United States.

This field can be in each entry in the import data file. If it is specified in the configuration file it acts as a default value. This means that the value in the import data file can override this default setting.

ClientType

The **ClientType** field specifies the type of Notes client that NotesAgent is to associate with the registered users it creates during the import process. The syntax is:

ClientType=*number*

where *number* is one of the following values:

- **1** - Create registered users of client type "desktop"
- **2** - Create registered users of client type "complete"
- **3** - Create registered users of client type "mail"

The client types correspond to the different kinds of licenses available for Notes clients.

This field can be in each entry in the import data file. If it is specified in the configuration file it acts as a default value. This means that the value in the import data file can override this default setting.

SMTPHostDomain

The **SMTPHostDomain** field specifies the domain name of the internet addresses of the user. The syntax is:

SMTPHostDomain=*domain name*

This field can be in each entry in the import data file. If it is specified in the configuration file it acts as a default value. This means that the value in the import data file can override this default setting.

MailTemplate

The **MailTemplate** field specifies the name of the mail template database. The syntax is:

MailTemplate=*name of the template data base*

This field can be in each entry in the import data file. If it is specified in the configuration file it acts as a default value. This means that the value in the import data file can override this default setting.

DbQuotaSizeLimit

The **DbQuotaSizeLimit** field specifies the size limit of the mail file. The syntax is:

DbQuotaSizeLimit=*number*

where *number* is the size in MB.

This field can be in each entry in the import data file. If it is specified in the configuration file it acts as a default value. This means that the value in the import data file can override this default setting.

DbQuotaWarningThreshold

The **DbQuotaWarningThreshold** field specifies the size of the mail file at which a warning is displayed. The syntax is:

DbQuotaWarningThreshold=*number*

where *number* is the size in MB.

This field can be in each entry in the import data file. If it is specified in the configuration file it acts as a default value. This means that the value in the import data file can override this default setting.

CreateMailDBNow

The **CreateMailDBNow** field specifies that the mail file is created during the registration. The syntax is:

CreateMailDBNow=*number*

where *number* is one of the following values:

- **0** - Create mail file later with the administration process
- **1** - Create mail file during the registration

This field can be in each entry in the import data file. If it is specified in the configuration file it acts as a default value. This means that the value in the import data file can override this default setting.

MailOwnerAccess

The **MailOwnerAccess** field specifies the mail owner's ACL privileges. The syntax is:

MailOwnerAccess=*number*

where *number* is one of the following values:

- **0** - Manager (default)
- **1** - Designer

- **2** - Editor

This field can be in each entry in the import data file. If it is specified in the configuration file it acts as a default value. This means that the value in the import data file can override this default setting.

MailSystem

The **MailSystem** field specifies the type of the mail system. The syntax is:

MailSystem=*number*

where *number* is one of the following values:

- **0** - NOTES (default)
- **1** - CCMAIL
- **2** - VINMAIL
- **99** - NONE

This field can be in each entry in the import data file. If it is specified in the configuration file it acts as a default value. This means that the value in the import data file can override this default setting.

MailACLManager

The **MailACLManager** field specifies the manager name of the access control list of the mail file. The syntax is:

MailACLManager=*name*

where *name* is the manager name in canonical format. For example:

```
MailACLManager=CN=Administrator/O=MyCompany
```

This field can be in each entry in the import data file. If it is specified in the configuration file it acts as a default value. This means that the value in the import data file can override this default setting.

MailForwardAddress

The **MailForwardAddress** field specifies the forwarding address of a Domino domain or foreign mail gateway. The syntax is:

MailForwardAddress=name of the forwarding address

This field can be in each entry in the import data file. If it is specified in the configuration file it acts as a default value. This means that the value in the import data file can override this default setting.

CreateMailFullTextIndex

The **CreateMailFullTextIndex** field specifies that a full text index is created when creating the mailbox. The syntax is:

CreateMailFullTextIndex=*number*

where *number* is one of the following values:

- **0** - Do not create mail full text index
- **1** - Create mail full text index

This field can be in each entry in the import data file. If it is specified in the configuration file it acts as a default value. This means that the value in the import data file can override this default setting.

If absent, then the mail full text index is created. (That default behavior is compatible with older releases of the NotesAgent where that parameter was not configurable.)

CreateMailReplicas

The **CreateMailReplicas** field specifies that the mail replicas should be created with the administration process. The syntax is:

CreateMailReplicas=*number*

where number is one of the following values:

- **0** - Do not create mail replicas
- **1** - Create mail replicas with the administration process

This field can be in each entry in the import data file. If it is specified in the configuration file it acts as a default value. This means that the value in the import data file can override this default setting.

If absent, then no mail replicas are created. (That default behavior is compatible with older releases of the NotesAgent where that parameter was not configurable.)

MailReplicaServer

The **MailReplicaServer** field specifies the Notes servers that holds a mail replica. The syntax is:

MailReplicaServer=*server_name 1* | *server_name 2* | ... | *server_name n*

where *server_name* is a the name of a Notes mail server in the format:

"CN=*server_name***/O=***organization_name*[/...]"

For example:

```
MailReplicaServer="CN=Cambridge4/O=Notes/O=IBM | "CN=New York/O=IBM"
```

In the INI file, the mail replica servers are defined on a single line and are separated by "|". A "|" at the end of the definition will be accepted, but ignored.

This field is an optional field and will only be evaluated if the RegisterUser field is set to 4.

This field can be in each entry in the import data file. If it is specified in the configuration file it acts as a default value. This means that the value in the import data file can override this default setting.

In the import data file, the definition looks a little bit different. Each mail replica server is defined in a seperate line. The syntax is as follows:

MailReplicaServer:*server_name 1*

MailReplicaServer:*server_name 2*

...

MailReplicaServer:*server_name n*

PreferredLanuage

The **PreferredLanguage** field specifies the user's language. The syntax is:

PreferredLanguage=*language*

where *language* is the user's preferred language. For example:

PreferredLanguage=de

This field can be in each entry in the import data file. If it is specified in the configuration file it acts as a default value. This means that the value in the import data file can override this default setting.

AltLanguage

The **AltLanguage** field specifies a user's alternate language. The syntax is:

AltLanguage=*language*

where *language* is the user's alternate language. For example:

```
AltLanguage=de
```

This field can be in each entry in the import data file. If it is specified in the configuration file it acts as a default value. This means that the value in the import data file can override this default setting.

OnDuplicate

The **OnDuplicate** field specifies the action to execute in case the new user is already available. The syntax is:

OnDuplicate=*option*

where *option* is one of the following values:

- **0** - terminate without creating the user (REG_FILE_DUP_STOP); default
- **1** - create a unique user (?) (REG_FILE_DUP_UNIQUE)

- **2** - overwrite the existing user (REG_FILE_DUP_OVERWRITE)

For example:

OnDuplicate=0

This field can be in each entry in the import data file. If it is specified in the configuration file it acts as a default value. This means that the value in the import data file can override this default setting. If this attribute is neither specified in the import data file nor in the configuration file, then the default value **0** applies.

PasswordKeyWidth

The **PasswordKeyWidth** specifies the encryption strength of the user's password in bits. The syntax is:

PasswordKeyWidth=*encryption_strength*

where *encryption_strength* is one of the following values:

- **0** - default; (means 64 bits for PasswordKeyWidth < 1024 else 128 bits)
- **64**
- **128**

For example:

```
PasswordKeyWidth=0
```

This field can be in each entry in the import data file. If it is specified in the configuration file it acts as a default value. This means that the value in the import data file can override this default setting. If this attribute is neither specified in the import data file nor in the configuration file, then the default value **0** applies.

KeyWidth

The **KeyWidth** field specifies the key width in bits. The syntax is:

KeyWidth=*width*

where *width* is one of the following values:

- **0**
- **630** - Compatible with all releases
- **1024** - Compatible with R6 and later
- **2048** - Compatible with R7 and later

For example:

```
KeyWidth=0
```

This field can be in each entry in the import data file. If it is specified in the configuration file it acts as a default value. This means that the value in the import data file can

override this default setting. If this attribute is neither specified in the import data file nor in the configuration file, then the default value **0** applies.

InetKeyWidth

The **InetKeyWidth** field specifies the width of the internet key in bits. The syntax is:

InetKeyWidth=*width*

where *width* is one of the following values:

- **0** - default width
- **1024**

For example:

```
InetKeyWidth=0
```

This field can be in each entry in the import data file. If it is specified in the configuration file it acts as a default value. This means that the value in the import data file can override this default setting. If this attribute is neither specified in the import data file nor in the configuration file, then the default value **0** applies.

PasswordQuality

The **PasswordQuality** field specifies the quality of the user's password required for this server. The syntax is:

PasswordQuality=*quality*

where *quality* is a value between **0** and **16**.

For example:

```
PasswordQuality=0
```

This field can be in each entry in the import data file. If it is specified in the configuration file it acts as a default value. This means that the value in the import data file can override this default setting. If this attribute is neither specified in the import data file nor in the configuration file, then the default value **0** applies.

3.5.3.3.4. The Password (Password) Section

The Password section consists of fields that define the parameters for NotesAgent automated password authentication. The next sections describe these fields.

PathFilePassword

The PathFilePassword field is only used for the old password handling mechanism.

For a description of the old password handling mechanism, see the section "Password Configuration File Formats".

The syntax for this field is:

PathFilePassword=*pathname*

where *pathname* is the pathname to the password configuration file NotesPassword.ini. For example:

PathFilePassword=a:\NotesSync\NotesPassword.ini

For security reasons, it is recommended that the password configuration file not be stored on disk, since it contains human-readable representations of Notes address book administrator and registered user passwords.

AutomaticPW

The AutomaticPW field specifies the password that NotesAgent (in conjunction with a NotesAgent DLL) is to use to decode the admin.id certificate; this is the certificate that grants it the credentials to log in to the Notes server.

The "PathAndFileOfNotesIDFile" field can alternatively be indicated.

If the field "PathAndFileOfNotesIDFile" is available for a ID file, it has higher priority.

The syntax is:

AutomaticPW=*password*

For example:

```
AutomaticPW=notes
```

PathAndFileOfNotesIDFile

A certifier Notes ID can be protected with up to three passwords. The standard is the protection with just one password. For each certifier Notes ID file the Notes Agent needs a pair of "Path and file name of ID file" and the "corresponding password". The syntax is:

Path and file Name of notes.id{[**_1**|**_2**|**_3**]}=*password*{[**1**|**2**|**3**]}

For example:

C:\IBM\Notes\certs\cert.id=notes
to protect the certifier Notes ID with just one password

or

C:\IBM\Notes\certs\cert.id_1=notes1
C:\IBM\Notes\certs\cert.id_2=notes2
to protect the certifier Notes ID with two passwords.

UserPassword

The UserPassword field specifies the default password that NotesAgent is to assign to any registered user it creates during the import process. The syntax is:

UserPassword=*password*

For example:

```
UserPassword=notes
```

NotesAgent uses the default password supplied in this field for entries in an import data file that do not contain a Password attribute value. If an entry in an import data file contains a Password attribute value, NotesAgent assigns this value as the user password when it creates the registered user.

This is an optional field unless the NotesAgent is to create registered users (the RegisterUser field is set to 1 or 2).

3.5.3.3.5. The EncryptedAttributes (EncryptedAttributes) Section

The EncryptedAttributes section is an optional section that lists attributes which are encrypted in the import data file and have to be decrypted by the agent before they are passed to the Notes Interface. 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). The attributes are listed in the format:

attribute=1

where attribute specifies the attribute names to be imported.

For example:

```
[EncryptedAttributes]
Password=1
```

3.5.3.4. Password Configuration File Formats

This chapter is provided only for compatibility reasons. It describes the old password handling mechanisms of the NotesAgent.

NotesAgent uses password configuration files to:

- Supply the password that grants it the credentials to log in to a Notes server from an installed Notes client. NotesAgent must use password authentication to a Notes server in order to export data from an address book or import data into it. The password can be supplied at login:
  - Manually, at the user prompt
  - Automatically, through the use of password configuration files
- Supply the password that grants it the credentials to register users during an import operation
- Provide a default password for registered users created during an import.

Templates of the password configuration files are provided with the NotesAgent installation. The filenames are:

- **NotesPathPWIni.ini**
- **NotesPassword.ini**

The next sections describe the password configuration file formats.

3.5.3.4.1. Notes Password Pathname Configuration File

The Notes password pathname file specifies the pathname to the password configuration file that contains:

- The password that NotesAgent is to use to automate the granting of credentials to log in into a Notes server
- The password(s) that NotesAgent is to use to obtain the credentials required to register users
- The default password that NotesAgent is to assign to any registered users that it creates during an import. NotesAgent must be able to assign a password to a registered user in order to create a user ID file for it in the Notes address book.

The Notes Password Pathname configuration file consists of one section - **Password** - which contains one field-**PathFilePassword**. The syntax for this field is:

PathFilePassword=*pathname*

where *pathname* is the pathname to the password configuration file **NotesPassword.ini**. For example:

```
PathFilePassword=a:\NotesSync\NotesPassword.ini
```

For security reasons, it is recommended that the password configuration file not be stored on disk, since it contains human-readable representations of Notes address book administrator and registered user passwords.

3.5.3.4.2. Password Configuration File

The Password configuration file stores passwords used by NotesAgent. It contains two sections - **Password** and **certifierPW** (only up to version 1.03). The next sections describe the fields within these sections.

The Password Section (up to Version 1.03)

AutomaticPW

The **AutomaticPW** field specifies:

- The password that NotesAgent (in conjunction with a NotesAgent DLL) is to use to decode the **admin.id** certificate; this is the certificate that grants it the credentials to log in to the Notes server
- The password that NotesAgent is to use to decode the **cert.id** certificate; this is the certificate that grants it the credentials to register users

The password to decode the **cert.id** certificate must be the same as the

> password to decode the **admin.id** certificate in order to be able to use the **AutomaticPW** field for automating authenticated login. There are also steps that must be followed during the NotesAgent installation to enable automatic password authentication at login; see the *DirX Identity Release Notes* for details.

The syntax is:

AutomaticPW=*password*

For example:

```
AutomaticPW=notes
```

CertifierPW

The **CertifierPW** field specifies the password that NotesAgent is to use to decode the **cert.id** certificate (the certificate that grants it the credentials to register users) during RenameUser, RecertifyUser, DeleteUser and MoveUserInHier operations and during user registration where the **RegisterUser** field is set to 2, 3 or 4. The syntax is:

CertifierPW=*password*

For example:

```
CertifierPW=notes
```

This is an optional field. If it is not specified (or is not present in the configuration file), the agent uses the password from the CertifierPW section (see below). If there the password for the cert.id is also not specified, the user is prompted for the password.

TargetCertifierPW

The **TargetCertifierPW** field specifies the password that NotesAgent is to use to decode the **cert.id** certificate (the certificate that grants it the credentials to register users) of the target organizational unit during "MoveUserInHier" operations. The syntax is:

TargetCertifierPW=*password*

For example:

```
TargetCertifierPW=notes
```

This is an optional field. If it is not specified (or is not present in the configuration file), the agent uses the password from the CertifierPW section (see below). If there the password for the cert.id is also not specified, the user is prompted for the password.

UserPassword

The **UserPassword** field specifies the default password that NotesAgent is to assign to any registered user it creates during the import process. The syntax is:

UserPassword=*password*

For example:

`UserPassword=notes`

NotesAgent uses the default password supplied in this field for entries in an import data file that do not contain a Password attribute value. If an entry in an import data file contains a Password attribute value, NotesAgent assigns this value as the user password when it creates the registered user.

This is an optional field unless the NotesAgent is to create registered users (the **RegisterUser** field is set to **1** or **2**).

The CertifierPW Section (only up to version 1.03)

If the agent registers users in several organizational units and each unit uses an own cert.id (with password), the agent needs for each cert.id a password.

In this section each line is a pair of "Path and file name of cert.Id" and the "corresponding password".

`Path and file Name of cert.id=password`

For example:

[CertifierPW]
C:\IBM\Notes\certs\cert.id=notes

This is an optional field. If it is not specified (or is not present in the configuration file), the user is prompted for the password.

The Password Section (version 1.04 and newer)

The Notes Agent needs to provide ID files and the corresponding passwords so that it enables the Notes client to decode the certificates that are stored in the relevant ID files.

The following ID files and passwords are needed:

- Full path name of administrator ID file for connecting to the IBM Domino Server and its relevant password
- Full path name of **cert.id** and its relevant password
- Full path name of other certifier ID files and their relevant passwords

These certifier ID files are needed if you plan to move users to different organizational units.

That information is stored in the bind profiles of a Notes target system.

The Notes Agent needs to know which one of the ID files is the one it could use for connecting to the Domino server. Therefore the display name of bind profile is part of the entries in the password section. The Notes Agent uses the entry with display name “Admin” for connecting to the Domino Server.

The format of the Password section is as follows:

[Password]

display_name|full_path_name_of_ID_files=password

Example:

```
[Password]
Admin|c:\Program Files\ ibm\notes\data\ids\admin.id=pwd1
Certifier| c:\Program Files\ ibm\notes\data\ids\cert.id=pwd2
Sales-OU| c:\Program Files\ ibm\notes\data\ids\sales.id=pwd3
```

3.5.4. Export and Import Data File Format

The NotesAgent import and export data files use a tagged file format. The next sections describe the:

- General characteristics of export and import data file formats
- Delta export data file format
- Import data file format

3.5.4.1. General Data File Format

The NotesAgent export and import data files have the following characteristics:

- Each entry attribute is contained on one line; line continuation is not permitted.
- The representation of each attribute is:

  attribute_name:attribute_value(s)

- Leading and trailing whitespace between attribute_name and attribute_value is ignored. For example, in the attribute:

  FullName: Timothy Michael Halvorsen

  The white space between the colon (:) and the start of the attribute value is ignored, but the white space within the attribute value is returned

- The form-feed character (0x0c) is used as a record (entry) separator
- The form-feed character can optionally appear as the first line in the file
- There is no special character processing (there is no "escaping" mechanism)

Here is an example of a person entry:

NoteID: 8453
Form: Person

Type: Person
Department: ENG,eng3
FullName: Alan Eldredge
City: Westford
ShortName: aeldredge
FirstName: Alan
LastName: Eldredge
Password: secret
State: MA
CompanyName: Iris Associates
InternetAddress: aeldredge@eng.iris.com
(0x0c is here as the record (entry) separator)
NoteID: 8454
Form: Person
Type: Person
...

Here is an example for a group entry:

NoteID: 8498
Form: Group
Type: Group
GroupType: 0
Form: Group
ListName: IrisAdminGroup
LocalAdmin: CN=Alan Eldredge/O=ENG3
$UpdatedBy: CN=Alan Eldredge/O=ENG3
GroupTitle: 0
Members: Ian Gillan,Roger Waters
ListOwner: CN=Alan Eldredge/O=ENG3
DocumentAccess: [GroupModifier]
AvailableForDirSync: 1

The following group attributes have numeric values:

- GroupType - specifies the use of the group and can have the values:
  0 (multi purpose)
  1 (mail only)
  2 (access control list only)
  3 (deny list only)
- GroupTitle - specifies the title of the group and can have the values:
  0 (group)
  1 (mailing list)
  2 (access list)
  3 (deny access list)
- AvailableForDirSync - specifies whether the group is available for synchronization (so that NotesAgent can export it) and can have the values:
  0 (not available for synchronization)
  1 (available for synchronization)

3.5.4.2. Delta Export Data File Format

The delta export data file ("modify/delete") generated when **ModifiedAddresses** is set to 1 uses LDIF per-entry "changetype" attributes to indicate the type of modification made to the entry since the last full export. The "modify" changetype attribute is applied to new or modified entries, and the "delete" changetype attribute is applied to entries that have been deleted. For example:

```
Changetype: delete
NoteID: 5430
Form: Person
Type: Person
Department: ENG,eng3
FullName: Jack Ozzie
City: Westford
ShortName: jozzie
FirstName: Jack
LastName: Ozzie
Password: secret
State: MA
CompanyName: Iris Associates
InternetAddress: jozzie@eng.iris.com
(0x0c is here as the record (entry) separator)
Changetype: modify
NoteID: 5478
Form: Person
Type: Person
Department: ENG,eng2
FullName: Len Kawell
City: Westford
ShortName: lkawell
FirstName: Len
LastName: Kawell
Password: secret
State: MA
CompanyName: Iris Associates
InternetAddress: lkawell@eng.iris.com
...
```

3.5.4.3. Import Data File Format

An entry in an import data file can contain the following optional attributes:

- An optional (text format) attribute **UniqueOrgUnit**, whose value is used as an additional value for OrganizationUnit to distinguish between entries with identical names; that is, identical values for the **FirstName**, **MiddleInitial**, **LastName** attributes.
- An optional (integer) attribute **Validity**, whose value specifies the lifetime, in days, for which the user certificate is valid (the default is 730 (2 years)).

The import data file format supports the LDIF per-entry "changetype" attribute that indicates the type of modification to be made to the entry in the Notes address book. The value for "changetype" is one of "add", "modify", "delete", "RenameUser", "RecertifyUser", "DeleteUser" or "MoveUserInHier". The changetype attribute name and its values are case-insensitive.

The attributes for a multivalued attribute specified in a "modify" changetype operation appear on separate lines. For example:

```
add: OfficeFaxPhoneNumber
OfficeFaxPhoneNumber: 123458
OfficeFaxPhoneNumber: 345892
-
```

Entries with a "modify" changetype contain attributes that indicate one or more "add", "delete", or "replace" attribute value modifications. The "replace" modification has a higher precedence than the "add and "delete" modifications; if it is present for an attribute, it is the only modification evaluated. For the "modify" changetype, NotesAgent adds a new entry in the Notes address book if it does not find a matching entry.

The "RenameUser" changetype renames a registered user. The user may need to confirm renaming when he logs on to Notes the next time. The entry must contain the OldUserName (in canonical format) and **LastName** attributes. The **FirstName**, **MiddleInitial**, **UniqueOrgUnit**. and **Validity** attributes are optional. For example:

```
OldUserName: CN=Armen Varteressian/OU=USA/O=MyCompany
LastName: Varteressian
Validity: 365
```

To perform this operation, the **PathFileCertId** field in the RegUser section of the import configuration file must be specified.

The "RecertifyUser" changetype re-certifies a registered user. The re-certification is completed when the user logs on the next time using the new certificate. The entry must contain the **UserName** (in canonical format) attribute and the **PathFileCertId** field in the RegUser section of the import configuration file must be specified.

The "DeleteUser" changetype deletes the user in "Person Documents", "Access Control List" and in "Reader/Author" fields and deletes his mail file subject to confirmation in the request database (approve file deletion) by the Notes Administrator. The following attributes must

be present in the import entry:

- **UserName** (in canonical format)
- **MailServer** (in canonical format)
- **MailFile** (mail file name including path relative to the Notes data directory)
- **DeleteMailFile** (0=don't delete mail file;1=delete just mail file specified in person record;2=delete mail file specified in person record and all replicas)

For example:

```
UserName: CN=Armen Varteressian/OU=USA/O=MyCompany
MailServer: CN=Neptune/O=MyCompany
MailFile: mail\Varteres
DeleteMailFile: 2
```

In order to perform this operation, the **AdminReqDB** and **AdminReqAuthor** fields in the Import section of the import configuration file must be specified, and the **DeleteEntries** field must be set to 0. If **DeleteEntries** is set to 1 or a "delete" changetype entry is processed, the user is deleted in the Notes address book only and his mail file is retained.

The "MoveUserInHier" changetype moves a user to a different organizational unit and renames the full username. In order to perform this operation:

- The Notes Client V5.0 must be installed
- The source and target organizational units must have different certificate ID (cert.id) files
- The **PathFileTargetCertID** field specifies the pathname to the certificate ID file of the target organizational unit, for example:

  PathFileTargetCertID: a:\German.id

- The **PathFileCertID** field specifies the pathname to the certificate ID file of the source organizational unit, for example:

  PathFileCertID: a:\cert.id

- The password configuration file must contain the passwords for both source and target organizational unit certificate IDs in the Password section, for example:

  CertifierPW=*password_for_source_organization*
  TargetCertifierPW=*password_for_target_organization*

  or in the CertifierPW section, for example:

  C:\IBM\certs\source_cert.id=*password_for_source_organization*
  C:\IBM\certs\target_cert.id=*password_for_target_organization*

- The **FullName** and **TargetCertifier** attributes must be present in the import entry in canonical format. For example:

  ```
  FullName: CN=Armen Varteressian/O=MyCompany
  TargetCertifier: OU=Germany/O=MyCompany
  ```

The values in the relevant fields in the import configuration file have a higher precedence than the changetype operations specified in the import data file. For example, if **DeleteEntries** is set to 1, NotesAgent deletes entries that match entries in the import data file from the Notes address book regardless of the change types specified for the entries in the import data file.

The import data file can contain comments, which are identified by a # character at the beginning of a line.

For Person entries, the **LastName** attribute must be the first attribute for the entry, the **FirstName** attribute must be the second attribute, and the **MiddleInitial** attribute must be the third attribute. For Group entries, the **ListName** attribute must be the first attribute for the entry. The ordering for all other attributes for Person and Group entries is arbitrary.

3.5.5. Import Error File Format

During the import process, NotesAgent writes the original attributes and values of entries that it is unable to import into the error file specified on the command line along with an error message that describes the error. Each line in the import error file generated by NotesAgent on an import operation has the following format:

source_entry
#error_code
#error_message

where *source_entry* is the original entry that NotesAgent was unable to import, *error_code* is the code for the error that occurred, and *error_message* is a description of the error. For example:

```
FirstName: Armen
LastName: Varteressian
CompanyName: Digital
Type: Person
FullName: Armen Varteressian
ShortName: avart
City: Nashua
Department: PUBS,VMSpubs
State: New Hampshire
#ProcessAddress error:
#Find more as one document with the following ItemIdentityName(s):
```

```
FirstName, LastName
```

Any entry that cannot be imported into the Notes address book is written into the import error file. Consequently, you can use the file as an input file and re-run the import operation, after first fixing the errors reported in the file.

3.5.6. Notes Agent Import Procedure

If NotesAgent encounters a single-valued attribute in an import data file that has more than one value defined, it takes the first value.

The order of operation on attributes is arbitrary. An import entry should not contain inconsistent attribute operations for the entry.

NotesAgent creates groups with GroupType 0-2 in the Groups folder of the Notes address book. It creates groups with GroupType 3 in the Server/Deny Access Groups folder.

The import configuration fields RegisterUser, ClientType, PathFileCertId and PathFileTargetCertId can be present as attributes of an entry to be imported. The syntax is:

field_name:* *field_value*

The colon character (**:**) is the name and value separator. For example:

```
RegisterUser: 1
ClientType: 1
PathFileCertId: d:\notes\data\certs\cert.id
PathFileTargetCertId: d:\notes\data\certs\sales.id
```

When present in the entry, the values in these attributes override the values specified in the fields of the import configuration file. When absent from the entry, NotesAgent uses the fields' default values from the configuration file.

3.6. Microsoft ADS Agent

ADSAgent is the DirX Identity agent that handles the import and export of Active Directory user and group objects to and from a Microsoft Windows Active Directory. ADSAgent uses the ADSI LDAP provider to bind to the Active Directory and runs on Windows.

ADSAgent can:

- Perform a full or a delta export of object classes from an Active Directory, including multiple attribute values and using LDAP search filters
- Perform a full or a delta import of object classes into an Active Directory, including multiple attribute values
- Generate an import error file that records all user and group entries that it fails to

import

- Generate a log file (for tracing)

The following figures illustrate the components of the ADSAgent export and import operations.

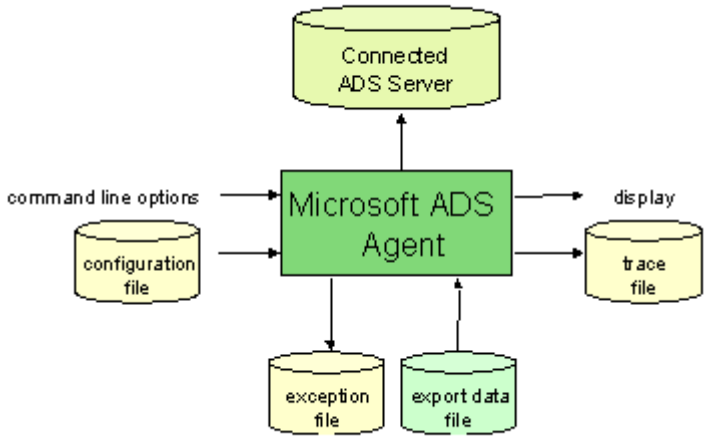


Figure 11. ADSAgent Export Components

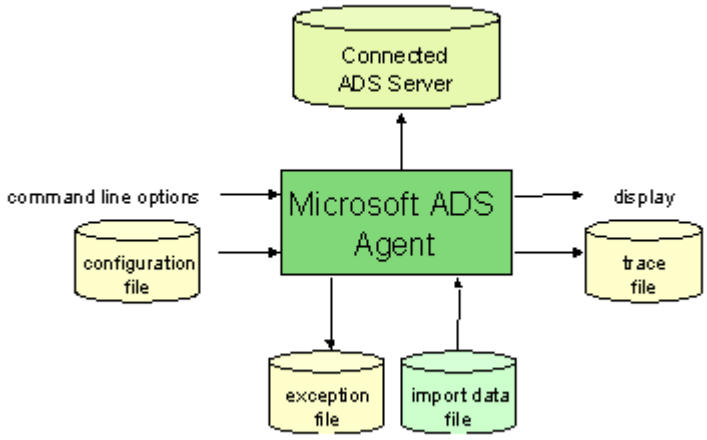


Figure 12. ADSAgent Import Components

This section describes:

- ADSAgent command line format for export and import operations
- ADSAgent configuration files for export and import operations
- The export data file format that ADSAgent generates
- The import data file format that ADSAgent recognizes
- ADSAgent import error file format
- How to create Exchange mail-enabled and mailbox-enabled users in Active Directory

Sample ADSAgent configuration files and scripts are provided in the **\Samples\ADS** directory of the DirX Identity installation. See the file **ADSReadme.txt** for a description of these files and scripts.

3.6.1. Command Line Format

The command line format to invoke ADSAgent is as follows:

AdsAgent.exe *sync_switch data_file configuration_file error_file* **[/a]>***initial_error_file* **[-Enc** *encryption_mode* **-Timeout** *timeout_value* **-AuditLevel** *audit_level* **-CryptLogLevel** *crypt_level*]

3.6.1.1. Parameters

sync_switch

Specifies the type of directory synchronization that ADSAgent is to perform. Possible values are:

/e Invokes the ADSAgent export function
/i Invokes the ADSAgent import function

data_file

For export: specifies the pathname of the target export data file that is to contain the entries that ADSAgent extracts from an Active Directory.*
For import:* specifies the pathname of the source file that contains the data to be imported into an Active Directory.

configuration_file

Specifies the name of the file that contains the specifications for the export and import procedure.

If the file is located in the working directory, you must explicitly indicate this fact by using the **.** notation before the file name, as shown in the example. It is not sufficient to specify only the file name, as it is for the *data_file* and *error_file* parameters.

error_file

Specifies the name of the file to which ADSAgent is to write error messages about errors that occur during the export or import process. For export errors, the format is:

###*date_and_time command_line*
Error! *error_message*
###*date_and_time command_line*

where *error_message* contains the function name that failed and an error code and error text.

For example:

```
### 04/07/2001 08:16:57 AM AdsAgent /e Data\ExportDirx.adr
.\ExportDirx.ini ExportDirx.log
Error! ADsOpenObject failed. Error Code: 80005000 Error Text:
An invalid Active Directory Pathname was passed.
### 04/07/2001 08:16:58 AM End
```

See "Import Error File Format" for a description of import error format.

/a (On export only)

Specifies that ADSAgent is to append the results of the export operation to *data_file* and *error_file* and write a timestamp at the start and end of the results. Use this switch on an export operation to append extracted entries to an existing export data file and to append error information to an existing error log file.

If the switch is not specified, ADSAgent overwrites the contents of the specified *data_file* and *error_file*, if they already exist.

initial_error_file

Specifies the name of the file to which ADSAgent is to write error messages for errors that occur before it creates *error_file*.

-ENC *encryption_mode*

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

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 must 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).

-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).

The following table describes the codes provided when ADSAgent.exe finishes running:

Exit Code	Description
0	ADSAgent completed successfully.
1	ADSAgent completed with errors, which are described in the specified *error_file* unless this file cannot be created due to a file exception error.
60	ADSAgent completed with warnings, which are described in the specified *error_file*.

3.6.2. Configuration File Formats

ADSAgent uses the following configuration files:

- ADS export configuration file - controls the export of data from an Active Directory
- ADS import configuration file - controls the import of data into an Active Directory

Templates of these configuration files are provided with the ADSAgent installation. The filenames are:

- **ExportAds.ini** (to export all object classes (Users, Groups, Sites, Services, Computers, Schema Objects))
- **ImportAds.ini** (to import all User and Group object classes to Active Directory)

In general, you must customize these files to support the requirements of your Active Directory import and export operations.

This section also describes the general structure of a configuration file.

3.6.2.1. General Structure of a Configuration File

An ADSAgent configuration file consists of sections and fields defined within those sections. A configuration file has the following structure:

```
[SectionName]

<comment> _
sectionField_*=*fieldValue
.
.
.
[SectionName]

<comment>
```

sectionField=fieldValue
.
.
.

SectionName is a keyword enclosed in square brackets ([ ]) that identifies the purpose of the section. *sectionField* is a keyword that identifies the field and *fieldValue* is the value assigned to the section field, preceded by the equal sign (=). For example:

SearchScope=2

Comments can be inserted anywhere in a configuration file and are identified by any character-for example, a # character or a semicolon (;)-that appears at the beginning of a line.

3.6.2.2. Export Configuration File Format

The ADSAgent export configuration file consists of the following sections:

- The Version section (required)
- The Connection section (required)
- The SearchPreferences section (optional)
- The SearchFilter section (optional)
- The SelAttributes section (optional)
- The Attributes section (optional)
- The Configuration section (optional)
- The DeltaExport section (optional)

3.6.2.2.1. The Version Section

The Version section consists of a single field that specifies the export configuration file version. The syntax is:

Version=*version_number*

where *version_number* is the version number assigned to the configuration file, in the format *n*.*nn*. The current version is:

```
Version=1.05
```

This is a mandatory field. This document describes the latest version of the ADSAgent export configuration file. The ADSAgent is able to process configuration files with version number **1.05** or lower as well as "old" files that do not contain a Version section. The following table provides information about the differences between export configuration file versions and about the support of older export configuration file versions for compatibility reasons:

	"Old"	1.00	1.01 or higher
TraceLevel	Supported	Not supported	Not supported
Trace	Not supported	Supported (1)	Supported

(1) **TraceLevel** has been replaced by **Trace**.

The following new sections or section fields have been added to the specified version and do not conflict with older versions. These sections and fields are optional: if present, they are performed, if not, the default behavior is performed.

Version 1.03:

[Connection]
UseSealing

Version 1.04:

[SearchPreferences]
ChaseReferrals

Version 1.05:

[Connection]
UseSigning
UseDelegation

3.6.2.2.2. The Connection Section

The Connection section is a mandatory section that consists of fields that define the parameters of an export operation for ADSAgent. The next sections describe these fields.

SearchBase

The **SearchBase** field specifies the base within the Active Directory from which to export entries. The syntax is:

SearchBase=LDAP://*host_name*[:*port_number*][/*distinguished_name*]

where:

- *host_name* specifies a computer name, an IP address, or a domain name. This is an optional component when ADSAgent is running on a Windows system. If it is not specified, the ADSI protocol locates the best domain controller in the system's site (the local area network to which the machine belongs) and connects to that controller.
- *port_number* specifies the port on *host_name* on which the Microsoft Active Directory LDAP server listens for requests. If *port_number* is not specified, ADSAgent uses the default LDAP port number 389.
- *distinguished_name* specifies the name of the target Active Directory root, in top-

down (DAP-style) or bottom-up (LDAP-style) naming format.

For example:

```
SearchBase=LDAP://Saturn/DC=MyCompany/DC=DirXIdentity/OU=Development
```

or:

```
SearchBase=LDAP://DC=MyCompany/DC=DirXIdentity/OU=Development
```

on Windows systems. Any comma (,) and forward slash (/) characters that are present in naming attribute values of *distinguished_name* must be "escaped" with the backslash character. For example:

```
SearchBase=LDAP://Venus/DC=OpTech\, Inc./DC=Talk2/OU=Sales
```

Active Directory supports the concept of "server-less" binding, which means that you can bind to Active Directory on the default domain without having to specify the name of a domain controller. When processing a server-less binding call, ADSI finds the "best" Windows domain controller in the default domain, which is the domain associated with the current security context of the thread that is performing the bind (the logged-on user on the machine on which the ADSAgent runs). ADSI uses DNS to find the domain controller and first looks in the client's computer's site, which is usually defined as an IP subnet.

The **SearchBase** field is a mandatory field.

UserName

The **UserName** field specifies the Windows account that ADSAgent is to use when binding to the Active Directory server during the export procedure. The syntax is:

UserName=*Windows_account_name*

For example:

```
UserName=Smith@dirxidentity.mycompany
```

This is an optional field; if it is not specified or is not present in the configuration file, ADSAgent uses the Windows account that invoked it when binding to the Active Directory server. If you specify a **UserName** field value, you must also specify a **Password** field value.

You can specify **UserName** in user principle name (UPN) format (which is the recommended form) as in the example just shown, or in a DN format, like cn=Smith,ou=sales,dc=dirxidentity,dc=mycompany or in the format of previous Windows versions like dirxidentity\Smith, where dirxidentity is the domain_name and Smith the account_name. If you use this format, you must set the **UseSecureAuthentication** field to **1**.

If ADSAgent runs on a Windows NT system, you must specify the **UserName** in the form for previous Windows versions or specify no

UserName (and consequently set **UseSecureAuthentication** field to **1**) if you want to set passwords for the users to be imported to ADS.

On Windows XP you must also set the **UseSecureAuthentication** field to **1** if you want to set passwords for the users to be imported to ADS. For further hints concerning password setting for users see the section Import Data File Format → Setting a Password for a User.

If you want to get the deleted objects in a delta export the account specified under **UserName** must be a member of the DomainAdmin group of the domain from which the entries are exported.

Password

The **Password** field specifies the password for the Windows account name specified in the **UserName** field. The syntax is:

Password=*password*

For example:

Password=fidlajsks

This is an optional field; if no value is specified in this field or the field is not present in the configuration file, ADSAgent uses the password used with the Windows account that invoked it when binding to an Active Directory server during an export procedure. If you specify a **Password** field value, you must also specify a **UserName** field value.

UseSecureAuthentication

The **UseSecureAuthentication** field controls the level of authentication that ADSAgent uses when binding to an Active Directory server during the export procedure. The syntax is:

UseSecureAuthentication=*switch*

where *switch* is one of the following values:

- **0** - Use simple authentication (default)
- **1** - Use secure authentication

The **UseSecureAuthentication** field is used in conjunction with the **UseEncryption** field to set the level of security services used during the export procedure. If the **UseSecureAuthentication** field is set to **1**, a secure authentication is requested using the Security Support Provider Interface (SSPI). In Windows 20xx, an SSP for Kerberos and an SSP for NT LAN Manager (NTLM) is included. Either of these protocols can be used for authentication. The SSP used depends on the capabilities of the computer on the other side of the connection, but Kerberos is always the first choice. When the UserName and Password are NULL, ADSI binds to the object using the security context of the calling thread, however in simple binds when using NULL credentials, ADSI does an anonymous bind.

The **UseSecureAuthentication** field must be set to **1** on Windows NT and Windows XP systems in order to set user passwords. However, on Windows NT (due to a bug in the Adsi NT version), a bind to the deleted objects container fails if this field is set. As a result, you must use different bind settings for ADSAgent import (set **UseSecureAuthentication** to 1) and ADSAgent export (set **UseSecureAuthentication** to 0).

UseEncryption

The **UseEncryption** field controls whether or not the Secure Socket Layer (SSL) port is used to provide a secure channel during the export procedure. The syntax is:

UseEncryption=*switch*

where *switch* is one of the following values:

- **0** - Do not use SSL encryption (default)
- **1** - Use SSL encryption

The **UseEncryption** field is used in conjunction with the **UseSecureAuthentication** field to set the level of security services used during the export procedure. If the **UseEncryption** field is set to 1 data will be encrypted using SSL. Active Directory requires that the Certificate Server is installed to support SSL encryption.

ADSI is designed to use simple binds when using SSL. Simple binds send UserName and Password in clear text across the network. Without using SSL this is not acceptable method under security aspects, but using SSL the network traffic is encrypted and the UserName and Password are protected. Because ADSI does an anonymous bind when using NULL credentials in **simple** binds, which would result in not having sufficient permissions to view and modify objects in the Active Directory, we recommend the following combination of the flags if a secure connection is wanted:

Set the **UseSecureAuthentication** field to 0 and the **UseEncryption** field to 1 to establish an SSL connection and pass a **UserName** and a **Password**. You can pass the **UserName** either in DN form, such as cn=Smith,ou=Development, dc=dirxidentity,dc=mycompany or in the UPN form, such as Smith@dirxidentity.mycompany. We recommend the UPN form. To use the UPN form, you must have assigned the appropriate UPN value for the **userPrincipalName** attribute of the targeted user object.

Another possible method is to specify the SSL channel in the **SearchBase** field, such as

LDAP://Saturn.dom.comp:636/dc=mycompany/dc=dirxidentity/ou=development

and also passing the **UserName** with a **Password**. This has the same effect as setting the UseEncryption field.

Prerequisites for running ADSAgent with SSL:

If you want to run ADSAgent with SSL you must perform the following steps:

- Install a CA (Certificate Authority) on the Active Directory Server. This installation generates the root certificate which must be named with the full qualified server name.
- Import the certificate generated in the step above to the Windows client certificate store on the machine where the agent runs. To import the certificate you can use the Internet Explorer. (Menu **Tools** → **Internet Options** → **Content** → **Certificates** → **Trusted Root Certification Authorities** → **Import**.)
- **Note**: When importing the certificate into the **Trusted Root Certification Authorities** as described above you must be logged in into Windows as the same user as the ADSAgent runs with. The ADSAgent runs under the user the C-Server Service was started with or if you explicitly configured a user in the ADS Job Configuration Authentication Tab it runs with that user.

See section "LDAP SSL Setup" in chapter "Identity Connectors" how to setup an SSL connection to an Active Directory Server.

UseSealing

The **UseSealing** field controls whether or not the data is encrypted using Kerberos during the export procedure. The syntax is:

UseSealing=*switch*

where *switch* is one of the following values:

- **0** - Do not use Kerberos encryption (default)
- 1 - Use Kerberos encryption

If the **UseEncryption** field is set to 1 data will be encrypted using Kerberos. The **UseSecureAuthentication** field must also be set to 1 in order to use the sealing. Kerberos encryption and authentication work under the following conditions:

- The client computer must be a member of a Windows mixed mode or native mode domain.
- The client must be logged on to the Windows domain, or to a domain trusted by a Windows domain.

UseServerBind

The **UseServerBind** field can be used for Windows versions greater than Windows 2000 SP1 if a server name is specified in the SearchBase (instead of a serverless bind) to reduce network traffic . The syntax is:

UseServerBind=*switch*

where *switch* is one of the following values:

- **0 - for serverless binds and as default for binds to a specific server (default)**

- 1 - If a servername is specified and network traffic is very high.

UseSigning

The **UseSigning** field can be used to verify data integrity. The **UseSecureAuthentication** flag must also be set to use signing. The syntax is:

UseSigning=*switch*

where *switch* is one of the following values:

- **0** - no signing (default)
- **1** - verifies data integrity.

UseDelegation

The **UseDelegation** field can be used to delegate the bind user security context to another domain. The syntax is:

UseDelegation=*switch*

where *switch* is one of the following values:

- **0** - no delegation (default)
- **1** - delegates the bind user security context to another domain.

3.6.2.2.3. The SearchPreferences Section

The SearchPreferences section is an optional section that consists of fields that specify parameters for search operations in Active Directory. The next sections describe these fields.

SearchScope

The **SearchScope** field specifies the search scope for search filters specified in the SearchFilter section. The syntax is:

SearchScope=*number*

where *number* is one of the following values:

- **0** - Limits the search scope to the entry specified in the **SearchBase** field
- **1** - Limits the search scope to the children of the entry specified in the **SearchBase** field
- **2** - Limits the search scope to the subtree below the entry specified in the **SearchBase** field (default)

PageSize

The **PageSize** field controls how Microsoft Active Directory server is to return search results to ADSAgent for a single search operation. The syntax is:

The syntax is:

PageSize=*number*

where *number* is one of the following values:

- **0** - Processes the entire search result set before returning it to ADSAgent (default)
- *n* - Processes and returns the search result in pages, where each page has a maximum of *n* entries. When a search results page contains *n* entries, Active Directory server returns the page to ADSAgent.

You can use the **PageSize** field to maintain client-server performance in cases where large result sets are being processed and returned. Specifying a number in **PageSize** directs the Active Directory server to process only that number of entries before returning the data to ADSAgent; this prevents large amounts of Active Directory server memory from being tied up while the server acquires the search results data and allows for scalable search operations. If you have more than 1000 entries to export, you must set a **PageSize** field value. We recommend that you set this field to a value between 100 and 500 for best performance.

PagedTimeLimit

The **PagedTimeLimit** field controls the length of time that Microsoft Active Directory server is to search for a single page. The syntax is:

The syntax is:

PagedTimeLimit=*number*

where *number* is one of the following values:

- **0** - No time limit on the search of one page (default)
- *n* - The maximum number of seconds to search for one page; specify a non-negative integer

AsynchronousSearch

The **AsynchronousSearch** field controls whether Microsoft Active Directory server performs synchronous or asynchronous search operations. The syntax is:

AsynchronousSearch=*switch*

where *switch* is one of the following values:

- **0** - A single search operation must complete before a new search operation can begin (default)
- **1** - A new search operation can start while a current search operation is being processed

When **AsynchronousSearch** is set to **1**, a new search can be started when the Active Directory server returns the first entry. When a number in **PageSize** is specified and **AsynchronousSearch** is set to **1**, a new search can be started when the Active Directory server returns the first page of search results. If **AsynchronousSearch** is set to **0**, a new search operation cannot be started until Active Directory server returns the entire results

set.

CacheResults

The **CacheResults** field controls whether ADSAgent caches search results in its local memory. The syntax is:

CacheResults=*switch*

where *switch* is one of the following values:

- **0** - Do not cache results
- **1** - Cache results (default)

TimeLimit

The **TimeLimit** field controls whether ADSAgent imposes a time limit for search results to be returned from Microsoft Active Directory server. The syntax is:

The syntax is:

TimeLimit=*number*

where *number* is one of the following values:

- **0** - No time limit is imposed on search operations (default)
- *n* - A time limit *n* is imposed on search operations, where *n* is the time in seconds after which ADSAgent is to abandon the search operation

ChaseReferrals

The **ChaseReferrals** field controls whether and how ADSAgent chases referrals. When an Active Directory server determines that another server holds relevant information (for example, when you have child domains and search in the parent domain), it may refer the ADsAgent to another server to obtain the result. Referral chasing is the action taken by a client to contact the referred-to server to continue the directory search. When the ADSAgent receives a referral message, it can decide whether to ignore or chase (follow) this referral. The syntax is:

ChaseReferrals=*switch*

where switch is one of the following values:

- **0** - Referrals are not chased (default).
- **1** - Referrals are chased in subordinate namespaces (turned off for paged searches)
- **2** - Referrals are chased in external namespaces.
- **3** - Referrals are always chased.

3.6.2.2.4. The SearchFilter Section

The SearchFilter section is an optional section that specifies the Active Directory entries that are to be exported from the Active Directory and can control whether or not ADSAgent

performs a delta export. The section consists of one or more **LdapFilter** fields. Each **LdapFilter** field specifies a collection of entries to locate and export. The syntax is as follows:

LdapFilter=*filter*

where *filter* is a search string specified in LDAP filter syntax; see RFC 2254 for an explanation of this syntax. For example:

LdapFilter=(objectClass=*)

The following table shows some sample LDAP filters and their results.

Filter Value	Action Taken
(objectClass=*)	Export all entries
(objectClass=user)	Export all user entries
(objectClass=group)	Export all group entries
(&(objectClass=*) (sn=a*))	Export all entries whose surname begins with "a"
(&(objectCategory=attributeSchema) (isMemberOfPartialAttributeSet=TRUE))	Export all attributeSchema entries which are in the Global Catalog

When specifying attributes in *filter*, you must use the LDAP names for the attributes.

ADSAgent creates one export data file. If the **DeltaExport** field in the DeltaExport section is set to 0, this file contains all of the entries extracted from the Active Directory that match the search criteria specified in the SearchPreferences and SearchFilter sections. This file is called the "export data file" (or "full export data file").

Use the Delta Export fields **DeltaExport** (set to 1) and the **HighestCommittedUSN** field to create a search filter that performs a "delta" export of modified entries into the generated export data file. See the description of the DeltaExport section for further details.

Active Directory moves deleted entries ("objects", in Active Directory terminology) to the "Deleted Objects" container in the naming context in which the entries originally existed. For example, the user Smith in //Saturn/DC=mycompany/DC=DirXIdentity/OU=Development is moved to //Saturn/DC=mycompany/DC=DirXIdentity/CN=Deleted Objects. Since the Deleted Objects container is itself marked as deleted, it is not seen in a normal search. When Active Directory deletes an entry, it sets the entry's "isDeleted" attribute to TRUE; the entry is then known as a tombstone. Active Directory retains a tombstone for a configurable period of time (60 days by default), after which it completely removes it. The RDN and objectGUID attribute values of deleted entries are always saved; the schema determines the other attributes that are to be saved. The RDN is changed to ensure uniqueness within the Deleted Objects container.

This is an optional field. If no value is specified or the field is not present in the configuration file, ADSAgent exports all entries.

3.6.2.2.5. The SelAttributes Section

The SelAttributes section is an optional section of the export configuration file that controls whether or not ADSAgent retrieves all entry attributes with a value, or only those attributes set to 1 in the Attributes section. The section consists of one field, which is **SelectAttributes**. The syntax is:

SelectAttributes=*switch*

where *switch* is one of:

- **0** - Export all attributes with a value (default)
- **1** - Export only those attributes set to 1 in the Attributes section

This field must be set to 1 to perform a delta export operation. See the DeltaExport section for more information about the delta export operation.

3.6.2.2.6. The Attributes Section

The **Attributes** section is an optional section of the export configuration file that specifies a set of Active Directory attributes to be exported from a Active Directory. The syntax is:

attribute_name=switch

where *attribute_name* is the name of an Active Directory attribute and *switch* is one of the following values:

- **0** - Do not export the attribute value for *attribute_name*
- **1** - Export the attribute value for *attribute_name*

For example:

```
[Attributes]
#Attributes of the Object Class Person
#Subclass of Top
#
seeAlso=0
sn=1
telephoneNumber=1
#
#Attributes of the Object Class organizationalPerson
#Subclass of Person
#
co=1
company=0
countryCode=1
department=1
```

```
facsimileTelephoneNumber=1
...
#
```

Use the *switch* parameter to select or exclude attributes in the list for export.

If the Attributes section is not specified in the configuration file and **SelectAttributes** is set to 0 in the SelAttributes section, ADSAgent exports all of the attributes of Active Directory entries that match the search criteria specified in the SearchPreferences and SearchFilter sections. If **SelectAttributes** is set to 1 and the Attributes section does not contain any attributes that are set to 1, nothing is exported.

3.6.2.2.7. The Configuration Section

The Configuration section is an optional section that contains information that ADSAgent is to use when evaluating entry attributes during the export procedure. The next sections describe the fields in the Configuration section.

MultiValueSeparator

The **MultiValueSeparator** field specifies a value to be used to separate the individual attribute values of a multi-valued attribute. The syntax is:

MultiValueSeparator=[*character*]

where *character* is a character or a string used as a multi-valued attribute separator. For example:

```
MultiValueSeparator=#
```

This field is optional. If it is not specified (or not present in the configuration file), ADSAgent uses the pound sign (#) as the multi-valued attribute separator.

Trace

The **Trace** field controls whether ADSAgent performs program flow tracing on an export operation. The syntax is:

Trace=[*switch*]

where *switch* is one of the following values:

- **0** - Do not perform program flow tracing on the export operation (default)
- **1** - Perform program flow tracing on the export operation

If 1 is specified, ADSAgent writes information about the export operation to the pathname specified in the **TraceFileName** field.

TraceFileName

The **TraceFileName** field specifies the pathname of the trace file to which ADSAgent is to write information about the export operation. The syntax is:

TraceFileName=*pathname*

where *pathname* is the name for the trace file. For example:

```
TraceFileName=c:\Adssync\ExportTraceFile
```

This field is optional unless **Trace** is set to 1. ADSAgent does not evaluate this field if **Trace** is set to 0.

3.6.2.2.8. The DeltaExport Section

The DeltaExport section is an optional section that provides information that can be used to direct ADSAgent to perform a delta export of entries from the Active Directory.

The next sections describe the fields in the DeltaExport section.

DeltaExport

The **DeltaExport** field controls whether ADSAgent performs a full or delta export of the Active Directory. The syntax is:

DeltaExport=[*switch*]

where *switch* is one of the following values:

- **0** - Export all entries (or those entries specified with the **LDAPFilter** field) from the directory (default)
- **1** - Export only those entries whose uSNChanged attribute value is greater than or equal to the value of the **HighestCommittedUSN** field.

If the **DeltaExport** field is set to 1:

- The Delta Export section must contain the **HighestCommittedUSN** field and value
- The **SelectedAttributes** field in the SelAttributes section must be set to 1.

This is an optional field. If it is not specified (or the field is not present in the configuration file), ADSAgent exports all entries in the directory (or all the entries selected using the specifications in the **LDAPFilter** field, if it is present in the configuration file) from the Active Directory.

HighestCommittedUSN

The **HighestCommittedUSN** field specifies the highest-numbered uSNChanged attribute value in the Active Directory. The syntax is:

HighestCommittedUSN=*USN*

where *USN* is an integer that represents the highest-number USN assigned to an Active Directory container entry in the directory. For example:

```
HighestCommittedUSN=6426
```

The uSNChanged attribute is a Microsoft Active Directory attribute that is assigned to

every entry in the Active Directory. When the Active Directory server carries out a modification to an entry, it assigns the highest USN to the entry's uSNChanged attribute as its value.

For the initial delta export, you must:

- Set the value of **HighestCommittedUSN** to 0:
- Set the **DeltaExport** field to 1

ADSAgent appends the search filter in **LdapFilter** with the part (uSNChanged greater than or equal to 1) to select only those entries modified after the uSNChanged value provided in the filter (all entries, in this export run). When it completes the export, ADSAgent updates the **HighestCommittedUSN** field with the current highest uSNChanged attribute value.

On subsequent exports, each time ADSAgent performs an export and the **DeltaExport** field is set to 1, it writes the value in the **HighestCommittedUSN** field to the **LdapFilter** field, performs the export, and updates the **HighestCommittedUSN** field with the current highest uSNChanged attribute value.

that performing incremental delta exports works only if you do not change the search filters in the export configuration file. If you make changes to search filters in the SearchFilter section, you will need to perform a full export and re-set the **HighestCommittedUSN** attribute.

3.6.2.3. Import Configuration File Format

The ADSAgent import configuration file consists of three sections:

- The Version section (required)
- The Connection section (required)
- The Configuration section (optional)
- The Ignore Empty Attributes section (optional)
- The Encrypted Attributes section (optional)
- The AttributeTypes section (required)

3.6.2.3.1. The Version Section

The Version section consists of a single field that specifies the import configuration file version. The syntax is:

Version=*version_number*

where *version_number* is the version number assigned to the configuration file, in the format *n*.*nn*. The current version is:

```
Version=1.02
```

This is a mandatory field. This document describes the latest version of the ADSAgent import configuration file. This document describes the latest version of the ADSAgent import configuration file. The ADSAgent is able to process configuration files with version number **1.03** or lower as well as "old" files that do not contain a Version section. The following table provides information about the differences between import configuration file versions and about the support of older import configuration file versions for compatibility reasons:

	"Old"	1.00	1.01 and higher
TraceLevel	Supported	Not supported	Not supported
Trace	Not supported	Supported (1)	Supported
SearchBase	Not supported	Not supported	Supported

(1) **TraceLevel** has been replaced by **Trace**.

The following new sections or section fields have been added in the specified version and do not conflict with older versions. These sections and fields are optional: if present, they are performed, if not, the default behavior is performed.

Version 1.02:

[IgnoreEmptyAttrValues]
[EncryptedAttributes]
Version 1.03:
[Connection]
UseSealing

3.6.2.3.2. The Connection Section

The Connection section is a mandatory section that consists of fields that define the parameters of an import operation for ADSAgent. The next sections describe these fields.

UserName

The **UserName** field specifies the Windows account that ADSAgent is to use when binding to the Active Directory server during the import procedure. It has the same syntax as the **UserName** field in the export configuration file.

Password

The **Password** field specifies the password for the Windows account name specified with the **UserName** field. It has the same syntax as the **Password** field in the export configuration file.

UseSecureAuthentication

The **UseSecureAuthentication** field controls the level of authentication that ADSAgent uses when binding to an Active Directory server during the import procedure. It has the same syntax as the **UseSecureAuthentication** field in the export configuration file.

UseEncryption

The **UseEncryption** field controls whether or not the Secure Socket Layer (SSL) port is used to provide a secure channel during the import procedure. It has the same syntax as the **UseEncryption** field in the export configuration file.

UseSealing

The **UseSealing** field controls whether or not data is Kerberos-encrypted during the import procedure. It has the same syntax as the **UseSealing** field in the export configuration file.

UseServerBind

The **UseServerBind** field can be used in Windows versions greater than Windows 2000 SP1 if a server name is specified in the Ads Bind Path (instead of a serverless bind) to reduce network traffic. It has the same syntax as the **UseServerBind** field in the export configuration file.

UseSigning

The **UseSigning** field controls whether or not data integrity is verified during the import procedure. It has the same syntax as the **UseSigning** field in the export configuration file.

UseDelegation

The **UseDelegation** field controls whether or not the bind user security context is delegated to another domain during the import procedure. It has the same syntax as the **UseDelegation** field in the export configuration file.

SearchBase

The **SearchBase** field specifies the base within the Active Directory from which to search for matching entries using the search criteria specified in the "ldapFilter" attribute of each entry in the import data file. The syntax is:

SearchBase=LDAP://*host_name*[:_port_number_][/*distinguished_name*]

where:

- *host_name* specifies a computer name, an IP address, or a domain name. This is an optional component when ADSAgent is running on a Windows system. If it is not specified, the ADSI protocol locates the best domain controller in the system's site (the local area network to which the machine belongs) and connects to that controller.
- *port_number* specifies the port on *host_name* on which the Microsoft Active Directory LDAP server listens for requests. If *port_number* is not specified, ADSAgent uses the default LDAP port number 389.
- *distinguished_name* specifies the name of the target Active Directory root, in top-down (DAP-style) or bottom-up (LDAP-style) naming format.

For example:

```
SearchBase=LDAP://Saturn/DC=MyCompany/DC=DirXIdentity/OU=Development
```

or

```
SearchBase=LDAP://DC=MyCompany/DC=DirXIdentity/OU=Development
```

on Windows systems. Any comma (,) and forward slash (/) characters that are present in naming attribute values of *distinguished_name* must be "escaped" with the backslash character. For example:

```
SearchBase=LDAP://Venus/DC=OpTech\, Inc./DC=Talk2/OU=Sales
```

The **SearchBase** field is a mandatory field when the import data file uses the "ldapFilter" attribute. See the section "Import Data File Format" for further details.

3.6.2.3.3. The Configuration Section

The Configuration section is an optional section that consists of fields that contain information that ADSAgent is to use when evaluating entry attributes in an import data file. The next sections describe these fields.

MultiValueSeparator

The **MultiValueSeparator** field specifies a value to be used to separate the individual attribute values of a multi-valued attribute. The syntax is the same as the **MultiValueSeparator** field in the export configuration file.

IgnoreObjectClass

The **IgnoreObjectClass** field controls whether ADSAgent evaluates or ignores the objectClass attribute of entries for which the "modify" LDIF changetype operation has been specified. The syntax is:

IgnoreObjectClass=*switch*

where *switch* is one of the following values:

- **0** - Evaluate the objectClass attribute when the changetype operation is "modify"
- **1** - Ignore the objectClass attribute when the changetype operation is "modify" (default)

Active Directory server does not currently permit the modification of the objectClass attribute through an LDIF "changetype" operation. Consequently, you can set the IgnoreObjectClass field to 1 to direct ADSAgent not to pass the ObjectClass attribute to the Active Directory server. However, other LDAP servers do permit the modification of the objectClass attribute. Setting IgnoreObjectClass to **0** in this case permits ADSAgent to pass the objectClass attribute to the LDAP server for evaluation.

Trace

The **Trace** field controls whether ADSAgent performs program flow tracing on an import operation. It has the same syntax as the **Trace** field in the export configuration file and is an optional field.

TraceFileName

The **TraceFileName** field specifies the pathname of the trace file to which ADSAgent is to write information about the import operation. It has the same syntax as the **TraceFileName** field in the export configuration file and is an optional field unless the **Trace** field is specified.

RejectSpecialCharacters

The **RejectSpecialCharacters** field controls whether ADSAgent evaluates the ADsPath attribute of entries in the import data file for special characters. The syntax is:

RejectSpecialCharacters=*switch*

where *switch* is one of the following values:

- **0** - Do not evaluate the ADsPath attribute for special characters (default)
- **1** - Evaluate the ADsPath attribute for special characters

If 1 is specified, ADSAgent scans the Common-Name (cn) RDN of each import entry's ADsPath attribute for the characters specified in the **RejectedCharacters** field and rejects the entry for import if it contains one of these characters.

RejectedCharacters

The **RejectedCharacters** field specifies the characters in the import entries' AdsPath attribute that ADSAgent is to scan for; ADSAgent is to reject the entry for import if it contains one of these characters. The syntax is:

RejectedCharacters=*characters*

Where *characters* specifies the characters in the Common-Name RDN of the AdsPath attribute for which ADSAgent is to search.

This field is optional unless **RejectSpecialCharacters** is set to 1. ADSAgent does not evaluate this field if **RejectSpecialCharacters** is set to 0.

3.6.2.3.4. The Ignore Empty Attributes Section

The Ignore Empty Attributes section is an optional section that lists attributes which are ignored if they exist in the import data file and if they are empty. Normally an attribute with an empty value results in clearing that attribute in the Active Directory. The attributes are listed in the format:

name_of_attribute=1

where *name_of_attribute* is the name for the attribute to be imported.

For example:

```
[IgnoreEmptyAttrValues]
description=1
Password=1
```

3.6.2.3.5. The Encrypted Attributes Section

The Encrypted Attributes section is an optional section that lists attributes which are encrypted in the import data file and have to be decrypted by the agent before they are passed to the Adsi Interface. 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). The attributes are listed in the format:

name_of_attribute=1

where name_of_attribute is the name for the attribute to be imported.

For example:

[EncryptedAttributes]
Password=1

3.6.2.3.6. The Attribute Types Section

The Attribute Types section is a mandatory section that specifies the attribute syntax for each Active Directory attribute to be imported into the Active Directory. The section consists of one or more attribute syntax specifications in the format:

LDAP_name_of_attribute=attribute_syntax

where *LDAP_name_of_attribute* is the LDAP name for the Active Directory attribute to be imported and *attribute_syntax* is one of the following keywords:

Boolean
CaseExactString
CaseIgnoreString
DNstring
Integer
LargeInteger
NumericString
ObjectClass
OctetString
PrintableString
ProviderSpecific
UTCTime

Each of these keywords corresponds to a data type that can be passed over the Active Directory Services Interface (ADSI). Because ADSAgent uses this interface, it must specify the data type of each attribute it passes over the interface. The Attribute Types section provides ADSAgent with the information it needs about each attribute's ADSI data type.

For example:

```
[Attribute Types]
changetype=CaseIgnoreString
```

```
objectClass=CaseIgnoreString
company=CaseIgnoreString
cn=CaseignoreString
department=CaseIgnoreString
member=DNString
userAccountControl=Integer
```

3.6.3. Export and Import Data File Format

The ADSAgent import and export data files use a tagged file format. This section describes:

- The general characteristics of export and import data file formats
- The specific features of the import data file format

3.6.3.1. General Data File Format

The ADSAgent import and export data files have the following characteristics:

- Each entry attribute is contained on one line; line continuation is not permitted.
- The representation of each attribute is *attribute_name:attribute_value(s)*
- Leading and trailing whitespace between *attribute_name* and *attribute_value* is ignored. For example, in the attribute:

  ```
  cn:   SallyAnn K. Quebec
  ```

  The whitespace between the colon (:) and the start of the attribute value is ignored, but the whitespace within the attribute value is returned

- The form-feed character (0x0c) is used as a record (entry) separator
- The form-feed character can optionally appear as the first line in the file
- There is no special character processing (there is no "escaping" mechanism)

Here is an example:

```
(0x0c is here as a record (entry) separator)
changetype: delete
objectClass: user
cn: Robert Amber
telephoneNumber: 603 555 8845
givenName: Robert
l:Nashua, New Hampshire
postalAddress: 110 Spitbrook Road
```

```
postalCode: 03060
sn: Amber
(0x0c is here as the record (entry) separator)
...
```

3.6.3.2. Import Data File Format

ADsPath

Individual entries in a single import data file can be targeted for import to different containers or Active Directory servers. Consequently, each entry in an ADSAgent import data file must contain an ADsPath attribute that identifies the fully-qualified pathname of the user or group entry to be added, modified, or deleted. The attribute syntax is:

ADsPath:LDAP://*host_name*[:_port_number_][/*distinguished_name*]

where:

- *host_name* specifies a computer name, an IP address, or a domain name. This is an optional component when ADSAgent is running on a Windows system. If it is not specified, the ADSI protocol locates the best domain controller in the system's site (the local area network to which the machine belongs) and connects to that controller.
- *port_number* specifies the port on *host_name* on which the Microsoft Active Directory LDAP server listens for requests. If *port_number* is not specified, ADSAgent uses the default LDAP port number 389.
- *distinguished_name* specifies the name of the target Active Directory root, in top-down (DAP-style) or bottom-up (LDAP-style) naming format.

For example:

`ADsPath: LDAP://Mars/DC=MyCompany/DC=DirXIdentity/OU=TestUsers/CN=Hans Hase`

or

`ADsPath: LDAP://DC=MyCompany/DC=DirXIdentity/OU=TestUsers/CN=Hans Hase`

on Windows systems.

ldapFilter

The ADSAgent import data file format also supports a per-entry ldapFilter attribute. The value of this attribute is a search filter that specifies an attribute that acts as a unique key for matching the entry in the import data file with an entry in an Active Directory. The attribute syntax is:

ldapFilter: *filter*

where *filter* is a search string specified in LDAP filter syntax (see RFC 2254 for an explanation of this syntax) that uses an attribute as a unique identifier. For example:

```
ldapFilter: (&(objectClass=user) (sAMAccountName=Hase4))
```

We recommended using the **sAMAccountName** attribute as the unique key in the ldapFilter attribute; however, other attributes can be defined and used as keys. If a new attribute is defined for use as a unique key, the meta directory schema must be extended to include this attribute definition.

When a "modify" or "delete" entry (see the per-entry changetype attribute) in the import data file contains an ldapFilter attribute, and an ADsPath attribute is not present, ADSAgent uses the ldapFilter attribute to search the Active Directory specified in the **SearchBase** field using the scope specified in the **SearchScope** field (or the default). If ADSAgent finds one entry ("object" in Active Directory terminology) that matches the filter criteria, ADSAgent modifies or deletes the entry, according to its changetype attribute. If ADSAgent finds more than one matching entry, or does not find a matching entry at all, it writes an error to the import error file.

Per-Entry Changetype

The ADSAgent import data file format supports the LDIF per-entry "changetype" attribute that indicates the type of modification to be made to the entry in the Active Directory and the attribute operation codes "add" or "delete". The value for "changetype" is one of "add", "modify", "delete", or "move". The changetype attribute name and its values are case-insensitive and can appear anywhere in the entry. If a changetype attribute is not present (or does not contain a value), ADSAgent attempts an "add" operation for the entry. If the "add" operation fails with the error code "entry already exists", it attempts a "modify" operation. If the "modify" operation fails with the error code "no such object", ADSAgent attempts a "move" operation if either the attribute AdsPathOld or ldapFilter exists in the entry. After every move operation, the ADSAgent performs a modify on the attributes contained in the entry.

Entries that contain the "add" "modify" or "delete" changetype attributes must contain the ADsPath attribute or the ldapFilter attribute (or both). Entries with a "modify" changetype attribute value must also contain at least one attribute to be modified. If the modify operation fails with the error code "no such object", it attempts to find the object using the ldapFilter attribute, if present, and then performs the modify operation. Entries with an "add" changetype must contain an object class attribute. Entries that contain the "move" changetype attribute must contain the ADsPath attribute, and either the ADsPathOld attribute or the ldapFilter attribute (in the "move" case, the ADsPath attribute specifies the destination for the entry and ADsPathOld or ldapFilter are used to identify the entry to be moved.)

In a "modify" operation, attributes can be deleted either by setting their values to an empty string or by setting them to the string value <clear>.

In a "modify" or a "delete" operation, the AdsPath can contain the GUID of an object, which is kept in the attribute "objectGuid", instead of its distinguished name. For example:

changetype: delete

ADsPath: LDAP://Saturn/<GUID=2e7330f0e8d24f49bc98de7045bf54b5>

Mandatory Attributes

Depending on the object class attribute value, the following attributes must also be present in the entry:

- For users (objectClass=user), the attribute **sAMAccountName** must be present and should be fewer than 20 characters in length. The second mandatory attribute **cn** (=RDN) is taken from ADsPath and for the third mandatory attribute **userAccountControl** a default is taken. Mailbox-enabled user entries must have the mandatory attributes **mail**, **legacyExchangeDN**, **proxyAddresses**, **showInAddressBook**, **textEncodedORAddress**, **msExchHideFromAddressLists**, **homeMTA**, **homeMDB**, **msExchHomeServerName**, **mailNickName**, and **mDBUseDefaults**. When it creates a mailbox-enabled user in the Active Directory, ADSAgent uses the value in **msExchHomeServerName** to create the mandatory attribute **msExchMailboxSecurityDescriptor**. Mail-enabled user entries must have the mandatory attributes **mail**, **legacyExchangeDN**, **proxyAddresses**, **showInAddressBook**, **textEncodedORAddress**, **msExchHideFromAddressLists**, **mailNickName**, and **mDBUseDefaults**.
- For groups (objectClass=group) the attribute **sAMAccountName** must be present. Also here the mandatory attribute **cn** is taken from ADsPath and for the mandatory attribute **groupType** a default is taken.
- All other objects with different object classes can also be imported, if the mandatory attributes for this object class are passed in the correct syntax and if the administrative rights of the user specified in the import configuration file are sufficient for this operation.

userAccountControl

In the following Microsoft Knowledge Base article describes how the values for the attribute userAccountControl can be set to manipulate user account properties:

http://support.microsoft.com/default.aspx?scid=kb;en-us;305144

Multi-Valued Attributes and Operation Codes for Attributes

The attributes for a multi-valued attribute appear on one line and are separated by the multi-valued attribute separator specified in the **MultiValueSeparator** field in the import configuration file. For example:

```
...

member:
cn=Test1,ou=TestUsers,DC=DirXIdentity,DC=MyCompany#cn=Test2,ou=Test
Users,DC=DirXIdentity,DC=MyCompany#Test3,ou=TestUsers,DC=DirXIdenti
ty,DC=MyCompany
```

For entries with a "modify" changetype, ADSAgent overwrites the specified attributes with the new values and the other attributes retain their old value. If an operation code for an attribute is specified, values for this attribute can be added or deleted. A sample for the syntax is given for the following entry:

```
changetype: modify
ADsPath: LDAP://Server1/CN=Hans
Hase6,OU=TestUsers,DC=DirXIdentity,DC=mch,DC=sni,DC=de
sn: Hase6
add: otherTelephone
otherTelephone: 113#114#115
delete: otherTelephone
otherTelephone: 114#115
```

When a "move" entry in the import data file contains the ldapFilter attribute, ADSAgent uses the ldapFilter attribute to search the Active Directory specified in the **SearchBase** field. If ADS finds one matching entry, it moves the entry to the destination specified in the ADsPath attribute for the entry in the import data file. When a "move" entry contains the OldADsPath attribute, ADSAgent uses this ADsPath to locate the entry, then moves it to the destination specified in the ADsPath attribute. An ADS entry can be moved within the same domain or from different domains in the same directory tree. The following restrictions apply for cross-domain moves:

- The destination domain must be in native mode.
- The object to be moved must be a leaf object or an empty container
- The operation requires Kerberos authentication (NTLM will not work). Set the **UseSecureAuthentication** field to 1 to enable Kerberos authentication.
- When ADSAgent moves a security principal (user, group, computer and so on), a new SID for the object is created at the destination. However, the old SID from the source (stored in the **sIDHistory** attribute) and the object's password are preserved.
- Security principals that belong to a global group cannot be moved.

Comments

The import data file can contain comments, which are identified by a # character at the beginning of a line.

Setting a Password for a User

When a user entry in the import data file contains the Password attribute, ADSAgent passes the specified value to the Adsi function IADsUser::SetPassword, which sets the password for that user. The password is stored in the Active Directory user object attribute unicodePwd, which can be written under restricted conditions but cannot be read. In order for the Adsi function to work correctly, the following rules apply:

- The user account specified in the import configuration file must have administrative rights in the Active Directory domain to which the user entry is imported.
- If ADSAgent runs on a Windows NT system, the **UserName** field in the import configuration file must be specified in the form *domain_name*_username_ or not specified at all and the **UseSecureAuthentication** field must be set to 1.
- If ADSAgent runs on a Windows XP system, the **UseSecureAuthentication** field must be set to 1.

- The user account under which ADSAgent runs must have administrative rights in the destination domain, because the Adsi function setting the password calls Windows Security functions that use the credentials of the calling thread. On Windows XP, this function has changed: the password function uses the credentials specified in the configuration file.
- For both Windows NT and Windows XP, if the machine on which ADSAgent runs is a member of a domain that is different from the domain into which the user entry is to be imported, a trust relationship must be established between both domains.
- In a DirX Identity environment, a DirX Identity agent can be run with a special account (see the authentication tab in the job object), if the account of the DirX Identity server has advanced user rights in the Windows operating system. See the *DirX Identity Connectivity Administration Guide* for information about how to set these rights.
- The AdsPath attribute of the entry in the import data file must either contain the server name of the destination Active Directory domain controller or an IP address without a port number. The reason for this restriction is that the Adsi function IADsUser::SetPassword does not work properly if the user object has bound with an AdsPath that contains an IP address and a port number.
- If Windows password policies are set for the domain make sure that the user password and account flags contained in the attribute userAccountControl for each user are consistent with those domain policy settings. If for example the domain policies require a password for a user you must set the password_not_required flag (which is the default) in the userAccountControl attribute to be able to create the user by temporarily overwriting the domain settings. After creation of the user the password is set. For a detailed description of the attribute userAccountControl see the link mentioned above under the userAccountControl section.

Attributes of type OctetString

If an attribute in the **import.ini** file is specified with the OctetString data type, ADSAgent expects it to be base64-encoded in the import data file. In the export direction, ADSAgent writes attributes with type OctetString base64-encoded into the export data file.

3.6.4. Import Error File Format

During the import process, ADSAgent writes the original attributes and values of user or group entries that it is unable to import into the error file specified on the command line along with an error message that describes the error that caused the import to fail on the entry. Each error record in the import error file has the following format:

#*warning_message*
source_entry
#*error_message*

Where *warning_message* contains a warning text, *source_entry* is the original entry that ADSAgent was unable to import and *error_message* contains the function name that failed and an error code and error text. Entries can have either *warning_message* or *error_message* or both of them. Here is an example of an import error record:

```
#Warning! Cannot find Attribute Type of xxx. Attribute Ignored.
changetype: add
objectClass: organizationalPerson
ADsPath:
LDAP://Mars:390/DC=MyCompany/DC=DirXIdentity/OU=TestUsers/CN=Hans
Hase
sAMAccountName: Hase
userPrincipalName: Hase@DirXIdentity.mchp.mycompany.de
displayName: Hans Hase3
givenName: Hans
sn: Hase
userAccountControl: 544
streetAddress: Otto-Hahn-Ring 36
info: Notes1
company: MyCompany
department: MDS
description: Description1
xxx: test
mail: hans.hase@icn.mycompany.de
#Error! CreateDSObject of
LDAP://Mars:390/DC=MyCompany/DC=DirXIdentity/OU=TestUsers/CN=Hans
Hase failed. Error Code: 80071392 Error Text: The object already
exists.
```

Any entry that cannot be imported into the Active Directory is written into the import error file. Consequently, you can use the error file as an input file and re-run the import operation, after first fixing the errors reported in the file. A timestamp is written at start and end of the import error file.

3.6.5. Creating Mail- and Mailbox-Enabled Users in Active Directory

When installing Exchange the schema of Active Directory is extended by Exchange related object classes and attributes. This allows the ADS Agent to create mail- and mailbox-enabled objects in Active Directory. A mail-enabled object can receive messages at an external address. A mailbox-enabled object has an Exchange mailbox associated with it, and can thus send and receive messages.

During the DirX Identity setup some sample import data files are installed to the samples subfolder, which show you how to create mail-enabled and mailbox-enabled users and mail-enabled groups and contacts.

3.6.5.1. Provisioning Exchange 2007 and Newer

The ADS Agent from Version 1.1.9.2 on supports mailbox guid generation, which is necessary for a full-functioning Exchange 2007 SP1 and newer mailbox.

If a data record in the import data file for the ADS Agent contains the attribute **msExchRecipientTypeDetails** it is assumed that an Exchange mailbox is supposed to be created or modified. The ADS Agent then generates a random globally unique mailbox identifier and modifies the user in Active Directory with the **msExchMailboxGuid** attribute set to this generated identifier in case the user's msExchMailboxGuid attribute is not already set. The msExchMailboxGuid attribute of the Active Directory user is the link to the mailbox object in the Exchange Server Mailbox database and should not be overwritten.

3.6.6. Deleting Non-Leaf Objects

The ADS Agent supports the deletion of non-leaf objects. Non-leaf objects in Active Directory are no container objects, like OUs, but objects that are usually expected to be leaf objects, like users. Nevertheless, sometimes these objects are non-leaf objects because they have subentries in certain cases. For example, Active Directory creates subentries for mailbox-enabled users in special situations. Those subentries are only shown by the "Active Directory Users and Computers" tool if the "Users, Contacts, Groups and Computers as containers" setting is checked in the View menu entry.

If such a non-leaf object is to be deleted the ADS Agent automatically deletes this object with all its subentries.

3.7. Microsoft Exchange Agent

ExchangeAgent is the DirX Identity agent that handles the import and export of Exchange Mailboxes, Remote Addresses, and Distribution Lists to and from a Microsoft Exchange directory maintained on an Exchange server. ExchangeAgent supports Exchange V5.5; the Microsoft Exchange Server Administrator (**admin.exe**) must be used to import and export from older versions. ExchangeAgent uses the ADSI LDAP provider to bind to the Exchange server and runs on Windows.

ExchangeAgent can:

- Perform a full or a delta export of Mailboxes, Remote Addresses and Distribution Lists from an Exchange directory, including multiple attribute values and using LDAP search filters
- Perform a full or a delta import of Mailboxes, Remote Addresses and Distribution Lists into an Exchange directory, including multiple attribute values
- Generate an import error file that records all Mailbox, Remote Address or Distribution List entries that it fails to import
- Generate a log file (for tracing)

The following figures show the components of the ExchangeAgent export and import operations.

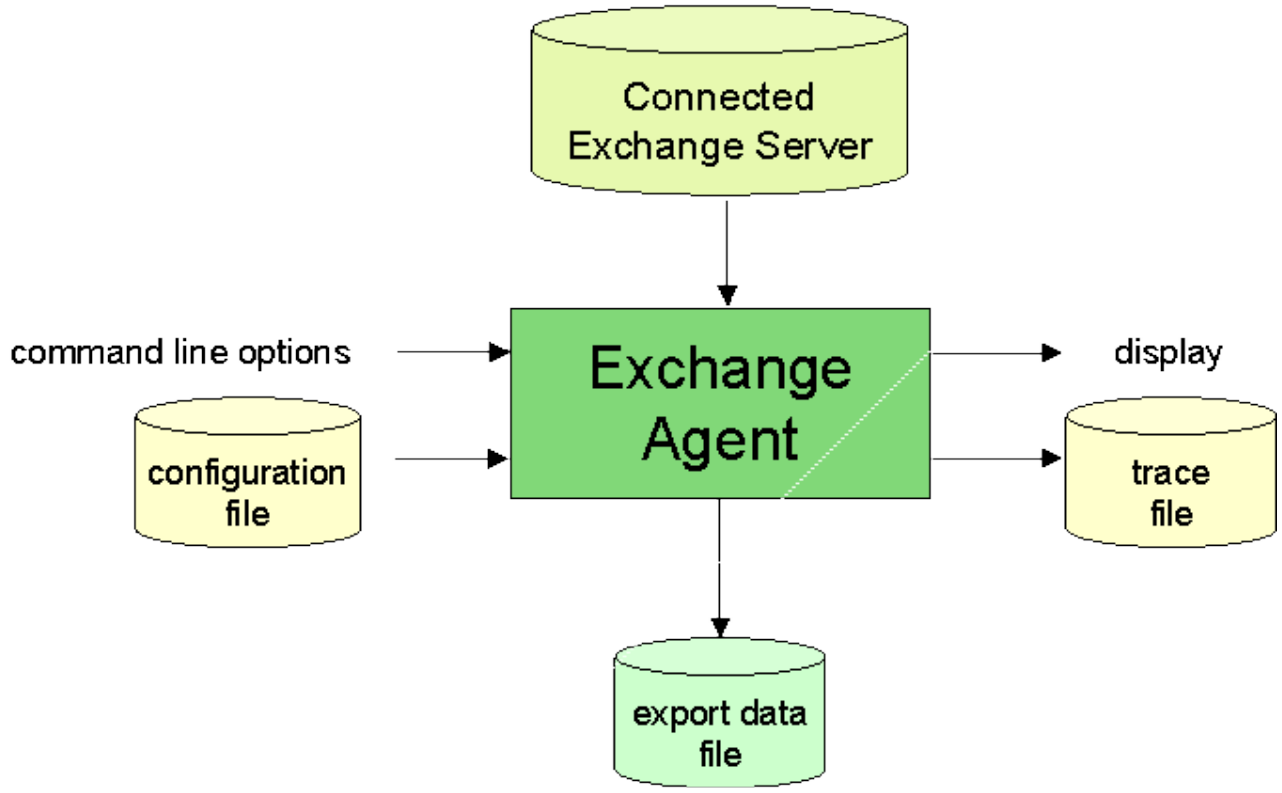


Figure 13. ExchangeAgent Export Components

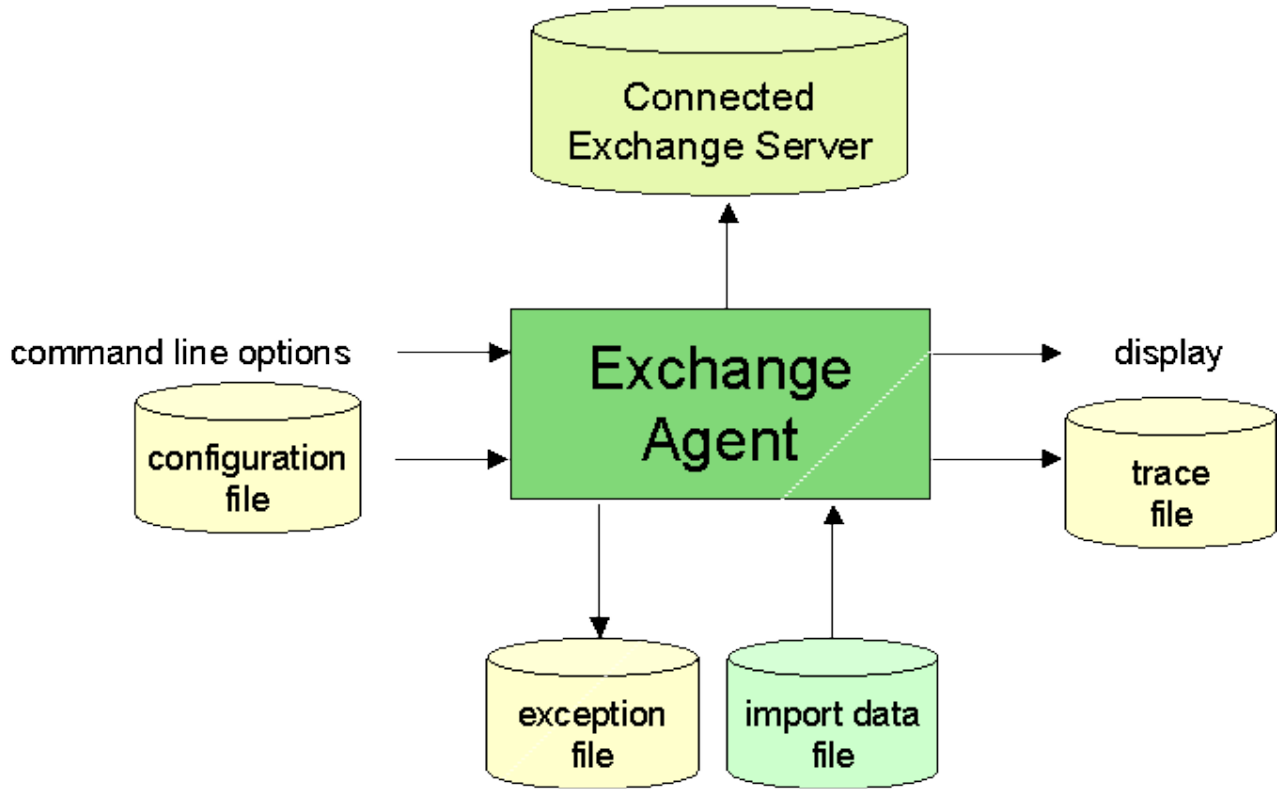


Figure 14. ExchangeAgent Import Components

This section describes:

- ExchangeAgent command line format for export and import operations
- ExchangeAgent configuration files for export and import operations

- The export data file format that ExchangeAgent generates
- The import data file format that ExchangeAgent recognizes
- ExchangeAgent import error file format
- Exchange Server administration for import and export operations

Sample ExchangeAgent configuration files and scripts are provided in the **\Samples\Exchange** directory of the DirX Identity installation. See the file **ExchangeReadme.txt** for a description of these files and scripts.

You can also use Microsoft Exchange Server Administrator (**admin.exe**) to import Mailboxes, Remote Addresses, and Distribution Lists into an Exchange directory. See "Using Exchange Server Administrator" for information on how to use Administrator.

3.7.1. Command Line Format

The command line format to invoke ExchangeAgent is as follows:

ExchangeAgent.exe *sync_switch data_file configuration_file error_file* ***[/a]>****initial_error_file* *[****-Enc*** *_encryption_mode* **-Timeout** *timeout_value* **-AuditLevel** *audit_level* **-CryptLogLevel** *crypt_level*]

3.7.1.1. Parameters

sync_switch

Specifies the type of directory synchronization that ExchangeAgent is to perform. Possible values are:

/e Invokes the ExchangeAgent export function
/i Invokes the ExchangeAgent import function

data_file

For export: specifies the pathname of the target export data file that is to contain the entries that ExchangeAgent extracts from a Exchange directory.*
For import:* specifies the pathname of the source file that contains the data to be imported into an Exchange directory.

configuration_file

Specifies the name of the file that contains the specifications for the export and import procedure.

If the file is located in the working directory, you must explicitly indicate this fact by using the **.** notation before the file name, as shown in the example. It is not sufficient to specify only the file name, as it is for the *data_file* and *error_file* parameters.

error_file

Specifies the name of the file to which ExchangeAgent is to write error messages about

errors that occur during the export or import process. For export errors, the format is:

###*date_and_time command_line*
Error! *error_message*
###*date_and_time command_line*

where *error_message* contains the function name that failed and an error code and error text.

For example:

```
### 06/07/2000 08:16:57 AM ExchangeAgent /e Data\ExportDirx.adr
.\ExportDirx.ini ExportDirx.log
Error! ADsOpenObject failed. Error Code: 80005000 Error Text: An
invalid Active Directory Pathname was passed.
### 06/07/2000 08:16:58 AM End
```

See "Import Error File Format" for a description of import error format.

/a (On export only)

Specifies that ExchangeAgent is to append the results of the export operation to *data_file* and *error_file* and write a timestamp at the start and end of the results. Use this switch on an export operation to append extracted entries to an existing export data file and to append error information to an existing error log file.

If the switch is not specified, ExchangeAgent overwrites the contents of the specified *data_file* and *error_file*, if they already exist.

initial_error_file

Specifies the name of the file to which ExchangeAgent is to write error messages for errors that occur before it creates *error_file*.

-ENC *encryption_mode*

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

This function only works correctly in an appropriate security environment, such as the DirX Identity environment configured in security mode. (see the *DirX Identity Connectivity Administration Guide* for details).

-Timeout *timeout_value*

Specifies the timeout value for the security mode, in microseconds.

This function only works correctly in an appropriate security environment, such as the DirX Identity environment configured in security mode. (see the *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 0

through 4.

This function only works correctly in an appropriate security environment, such as the DirX Identity environment configured in security mode. (see the *DirX Identity Connectivity Administration Guide* for details).

-CryptLogLevel *crypt_level*

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

This function only works correctly in an appropriate security environment, such as the DirX Identity environment configured in security mode. (see the *DirX Identity Connectivity Administration Guide* for details).

The following table describes the codes provided when ExchangeAgent.exe finishes running:

Exit Code	Description
0	ExchangeAgent completed successfully.
1	ExchangeAgent completed with errors, which are described in the specified *error_file* unless this file cannot be created due to a file exception error.
60	ExchangeAgent completed with warnings, which are described in the specified *error_file*.

3.7.2. Configuration File Formats

ExchangeAgent uses the following configuration files:

- Exchange export configuration file - controls the export of data from a Exchange directory
- Exchange import configuration file - controls the import of data into a Exchange directory

See "General Structure of a Configuration File" for a description of the basic organization.

Templates of these configuration files are provided with the ExchangeAgent installation. The filenames are:

- **ExchExport.ini** (to export all object classes (Mailboxes, Remote Addresses and Distribution Lists))
- **ExchImport.ini** (to import all object classes (Mailboxes, Remote Addresses and Distribution Lists) from an import data file)

In general, you must customize these files to support the requirements of your Exchange import and export operations.

3.7.2.1. General Structure of a Configuration File

An ExchangeAgent configuration file consists of sections and fields defined within those sections. A configuration file has the following structure:

```
[SectionName]

+ <comment> _
sectionField_*=*fieldValue
.
.
.
[SectionName]

+ <comment>
sectionField=fieldValue
.
.
.
```

SectionName is a keyword enclosed in square brackets ([ ]) that identifies the purpose of the section. *sectionField* is a keyword that identifies the field and *fieldValue* is the value assigned to the section field, preceded by the equal sign (=). For example:

SearchScope=2

Comments can be inserted anywhere in an configuration file and are identified by any character - for example, a # character or a semicolon (;) - that appears at the beginning of a line.

3.7.2.2. Export Configuration File Format

The ExchangeAgent export configuration file consists of the following sections:

- The Version section (required)
- The Connection section (required)
- The SearchPreferences section (optional)
- The SearchFilter section (optional)
- The SelAttributes section (optional)
- The Attributes section (optional)
- The Configuration section (optional)
- The DeltaExport section (optional)

The next sections describe these sections.

3.7.2.2.1. The Version Section

The Version section consists of a single field that specifies the export configuration file version. The syntax is:

Version=*version_number*

where *version_number* is the version number assigned to the configuration file, in the format *n.nn*. The current version is:

Version=1.03

This is a mandatory field. This document describes the latest version of the ExchangeAgent export configuration file. The ExchangeAgent is able to process configuration files with version number **1.03** or lower as well as "old" files that do not contain a Version section. The following table provides information about the differences between export configuration file versions and about the support of older export configuration file versions for compatibility reasons:

	"Old"	1.00	1.01 and higher
TraceLevel	Supported	Not supported	Not supported
Trace	Not supported	Supported (1)	Supported
LdapFilter	(2)		

(1) TraceLevel has been replaced by Trace.
(2) If the field **DeltaExport** is set to **1**, **(USN-Changed>=*n*)*** must be specified in the **LdapFilter** field. (See **The SearchFilter Section** for details.)

The following new sections or section fields have been added to the specified version and do not conflict with older versions. These sections and fields are optional: if present, they are performed; if not, the default behavior is performed.

Version 1.02:
NTAccountFormat

3.7.2.2.2. The Connection Section

The Connection section is a mandatory section that consists of fields that define the parameters of an export operation for ExchangeAgent. The next sections describe these fields.

SearchBase

The **SearchBase** field specifies the base within the Exchange directory from which to export entries. The syntax is:

SearchBase=LDAP://*host_name*[:_port_number_][/*distinguished_name*]

where:

- *host_name* specifies a computer name, an IP address, or a domain name.
- *port_number* specifies the port on *host_name* on which the Microsoft Exchange LDAP server listens for requests. If *port_number* is not specified, ExchangeAgent uses the default LDAP port number 389.
- *distinguished_name* specifies the name of the target Exchange directory root, in top-down (DAP-style) or bottom-up (LDAP-style) naming format.

For example:

SearchBase=LDAP://Saturn/O=MyCompany/OU=Talk1/CN=Recipients

Any comma (,) and forward slash (/) characters that are present in naming attribute values of *distinguished_name* must be "escaped" with the backslash character. For example:

```
SearchBase=LDAP://Venus/O=OpTech\, Inc./OU=Talk2/CN=Recipients
```

The **SearchBase** field is a mandatory field.

UserName

The **UserName** field specifies the NT account that ExchangeAgent is to use when binding to the Exchange server during the export procedure. The syntax is:

UserName=CN=NT_account_name***,CN=**NT_domain_name*

For example:

```
UserName=cn=Smith,cn=TestDomain
```

The Exchange server retains deleted entries for a specific period of time (the default is 30 days, and can be changed by the Exchange administrator; see "Exchange Server Administration for Import and Export Operations" for further details). If your export procedure is to read extracted deleted entries from the Exchange directory, you must append the NT Administrator account name cn=admin to the values specified in the **UserName** field. For example:

```
UserName=CN=Smith,CN=TestDomain,CN=admin
```

The Exchange server must also be set up to enable ExchangeAgent to extract deleted entries from the Exchange directory; see "Exchange Server Administration for Import and Export Operations" for further details.

A recipient (Mailbox, Remote-Address and/or Distribution-List) can be hidden in an Exchange directory if its Hide-From-Address attribute is assigned the value True. To export hidden recipients, you must append the NT Administrator account name cn=admin to the values specified in the **UserName** field. For example:

```
UserName=CN=Beninga,CN=Saturn,CN=admin
```

This is an optional field; if it is not specified or is not present in the configuration file, ExchangeAgent uses the NT account that invoked it when binding to the Exchange

server. If you specify a **UserName** field value, you must also specify a **Password** field value.

Password

The **Password** field specifies the password for the NT account name specified with the **UserName** field. The syntax is:

Password=*password*

For example:

```
Password=fidlajsks
```

This is an optional field; if no value is specified in this field or the field is not present in the configuration file, ExchangeAgent uses the password used with the NT account that invoked it when binding to an Exchange server during an export procedure. If you specify a **Password** field value, you must also specify a **UserName** field value.

UseSecureAuthentication

The **UseSecureAuthentication** field controls the level of authentication that ExchangeAgent uses when binding to an Exchange server during the export procedure. The syntax is:

UseSecureAuthentication=*switch*

where *switch* is one of the following values:

- **0** - Use simple authentication (default)
- **1** - Use secure authentication

The **UseSecureAuthentication** field is used in conjunction with the **UseEncryption** field to set the level of security services used during the export procedure. If **UseSecureAuthentication** is set to **1**, the Exchange server managing the target Exchange directory must also have secure authentication enabled. See "Exchange Server Administration for Import and Export Operations" for further details.

UseEncryption

The **UseEncryption** field controls whether or not the Secure Socket Layer (SSL) port is used to provide a secure channel during the export procedure. The syntax is:

UseEncryption=*switch*

where *switch* is one of the following values:

- **0** - Do not use SSL encryption (default)
- **1** - Use SSL encryption

The **UseEncryption** field is used in conjunction with the **UseSecureAuthentication** field to set the level of security services used during the export procedure. Note that the SSL encryption setting on the Exchange server managing the target Exchange directory

must match the **UseEncryption** setting specified in the configuration file. See "Exchange Server Administration for Import and Export Operations" for further details.

ADSI is designed to use simple binds when using SSL. Simple binds send UserName and Password in clear text across the network. Without using SSL this is not an acceptable method under security aspects, but using SSL the network traffic is encrypted and the UserName and Password are protected. Because ADSI does an anonymous bind when using NULL credentials in simple binds, which would result in not having sufficient permissions to view and modify objects in the Active Directory, we recommend the following combination of the flags if a secure connection is wanted:

Set the UseSecureAuthentication flag to 0 and the UseEncryption flag to 1 to establish a SSL connection and pass a UserName and a Password. Another possible method is to specify the SSL channel in the SearchBase, such as

LDAP://Saturn:636/O=MyCompany/OU=Identity/CN=Recipients

and also passing the UserName with a Password. This has the same effect as setting the UseEncryption flag.

3.7.2.2.3. The SearchPreferences Section

The SearchPreferences section is an optional section that consists of fields that specify parameters for search operations. The next sections describe these fields.

SearchScope

The **SearchScope** field specifies the search scope for search filters specified in the SearchFilter section. The syntax is:

SearchScope=*number*

where *number* is one of the following values:

- **0** - Limits the search scope to the entry specified in the **SearchBase** field
- **1** - Limits the search scope to the children of the entry specified in the **SearchBase** field
- **2** - Limits the search scope to the subtree below the entry specified in the **SearchBase** field (default)

PageSize

The **PageSize** field controls how Microsoft Exchange server is to return search results to ExchangeAgent for a single search operation. The syntax is:

The syntax is:

PageSize=*number*

where *number* is one of the following values:

- **0** - Processes the entire search result set before returning it to ExchangeAgent (default)
- *n* - Processes and returns the search result in pages, where each page has a maximum of *n* entries. When a search results page contains *n* entries, Exchange server returns the page to ExchangeAgent.

You can use the **PageSize** field to maintain client-server performance in cases where large result sets are being processed and returned. Specifying a number in **PageSize** directs the Exchange server to process only that number of entries before returning the data to ExchangeAgent; this prevents large amounts of Exchange server memory from being tied up while the server acquires the search results data and allows for scalable search operations.

PagedTimeLimit

The **PagedTimeLimit** field controls the length of time that Microsoft Exchange server is to search for a single page. The syntax is:

The syntax is:

PagedTimeLimit=*number*

where *number* is one of the following values:

- **0** - No time limit on the search of one page (default)
- *n* - The maximum number of seconds to search for one page; specify a non-negative integer

AsynchronousSearch

The **AsynchronousSearch** field controls whether Microsoft Exchange server performs synchronous or asynchronous search operations. The syntax is:

AsynchronousSearch=*switch*

where *switch* is one of the following values:

- **0** - A single search operation must complete before a new search operation can begin (default)
- **1** - A new search operation can start while a current search operation is being processed

When **AsynchronousSearch** is set to **1**, a new search can be started when the Exchange server returns the first entry. When a number in **PageSize** is specified and **AsynchronousSearch** is set to **1**, a new search can be started when the Exchange server returns the first page of search results. If **AsynchronousSearch** is set to **0**, a new search operation cannot be started until Exchange server returns the entire results set.

CacheResults

The **CacheResults** field controls whether ExchangeAgent caches search results in its local memory. The syntax is:

CacheResults=*switch*

where *switch* is one of the following values:

- **0** - Do not cache results
- **1** - Cache results (default)

TimeLimit

The **TimeLimit** field controls whether ExchangeAgent imposes a time limit for search results to be returned from Microsoft Exchange server. The syntax is:

TimeLimit=*number*

where *number* is one of the following values:

- **0** - No time limit is imposed on search operations (default)
- *n* - A time limit *n* is imposed on search operations, where *n* is the time in seconds after which ExchangeAgent is to abandon the search operation

3.7.2.2.4. The SearchFilter Section

The SearchFilter section is an optional section that specifies the Exchange entries that are to be exported from the Exchange directory and can control whether ExchangeAgent performs a delta export. The section consists of one or more **LdapFilter** fields. Each **LdapFilter** field specifies a collection of entries to locate and export. The syntax is as follows:

LdapFilter=*filter*

where *filter* is a search string specified in LDAP filter syntax; see RFC 2254 for an explanation of this syntax. For example:

```
LdapFilter=(objectClass=*)
```

The following table shows some sample LDAP filters and their results.

Filter Value	Action Taken
(objectClass=*)	Export all entries
(objectClass=organizationalPerson)	Export all Mailbox entries
(objectClass=Remote-Address)	Export all Remote Address entries
(&(objectClass=*) (sn=a*))	Export all entries whose surname begins with "a"
(&(objectClass=organizationalPerson) (postalAddress=*))	Export all Mailbox entries that have the attribute postalAddress set
(&(objectClass=*) (Is-Deleted=True))	Export all deleted entries
(&(objectClass=*) (USN-Changed>=6200))	Export all entries whose USN-changed value is higher than the latest USN-changed value (delta export)

When specifying attributes in *filter*, you must use the LDAP names for the attributes. See "Microsoft Exchange Directory Schema" for tables of LDAP name-to-Exchange name mappings.

ExchangeAgent creates one export data file. If the **DeltaExport** field in the DeltaExport section is set to 0, this file contains all of the entries extracted from the Exchange directory that match the search criteria specified in the SearchPreferences and SearchFilter sections. This file is called the "export data file" (or "full export data file").

ExchangeAgent uses the **LdapFilter** field in conjunction with the Delta Export fields **DeltaExport** (set to 1) and the **USNChangedMax** field to create a search filter that performs a "delta" export of modified entries into the generated export data file. See the DeltaExport section for further details.

The Exchange server retains deleted entries for a specific period of time (the default is 30 days, and can be changed by the Exchange administrator; see "Exchange Server Administration for Import and Export Operations" for further details). For deleted entries, it updates the "Is-Deleted" Exchange attribute to TRUE. Use the "Is-Deleted" Exchange attribute as a search filter component to direct ExchangeAgent to extract deleted entries into the generated export data file. Note that ExchangeAgent can only extract and read deleted entries if the Exchange server that is managing the target directory has been set up with the appropriate access rights. See "Exchange Server Administration for Import and Export Operations" for further details.

Because ExchangeAgent always generates only one export data file, it is recommended that you perform separate export operations to extract modified entries and deleted entries. (If the search filter supplied in **LdapFilter** selects both modified and deleted entries, they are written to the same export data file.)

Only one LDAP search filter can be activated per export operation.

This is an optional field. If no value is specified or the field is not present in the configuration file, ExchangeAgent exports all entries.

3.7.2.2.5. The SelAttributes Section

The SelAttributes section is an optional section of the export configuration file that controls whether or not ExchangeAgent retrieves all entry attributes with a value, or only those attributes set to 1 in the Attributes section. The section consists of one field, which is **SelectAttributes**. The syntax is:

SelectAttributes=*switch*

where *switch* is one of:

- **0** - Export all attributes with a value (default)
- **1** - Export only those attributes set to 1 in the Attributes section

This field must be set to 1 to perform a delta export operation. See the DeltaExport section for more information about the delta export operation.

3.7.2.2.6. The Attributes Section

The **Attributes** section is an optional section of the export configuration file that specifies a set of Exchange attributes to be exported from an Exchange directory. The syntax is:

attribute_name=switch

where *attribute_name* is the name of an Exchange attribute and *switch* is one of the following values:

- **0** - Do not export the attribute value for *attribute_name*
- **1** - Export the attribute value for *attribute_name*

For example:

```
[Attributes]
#Attributes specific to Remote Addresses
#
Target-Address=1
userPassword=0
#
#Attributes specific to Distribution Lists
#
DL-Member-Rule=0
Hide-DL-Membership=0
member=1
OOF-Replay-To-Originator=0
owner=0
Report-To-Owner=0
#
```

Use the *switch* parameter to select or exclude attributes in the list for export.

If the Attributes section is not specified in the configuration file and **SelectAttributes** is set to 0 in the SelAttributes section, ExchangeAgent exports all of the attributes of Exchange entries that match the search criteria specified in the SearchPreferences and SearchFilter sections. If **SelectAttributes** is set to 1 and the Attributes section does not contain any attributes that are set to 1, nothing is exported.

Note that the USN-Changed attribute must be set to 1 in the Attributes section in order to perform a delta export operation. See the DeltaExport section for more information about delta export operations.

3.7.2.2.7. The Configuration Section

The Configuration section is an optional section that contains information that

ExchangeAgent is to use when evaluating entry attributes during the export procedure. The next sections describe the fields in the Configuration section.

MultiValueSeparator

The **MultiValueSeparator** field specifies a value to be used to separate the individual attribute values of a multi-valued attribute. The syntax is:

MultiValueSeparator=[*character*]

where *character* is a character or a string used as a multi-valued attribute separator. For example:

```
MultiValueSeparator=#
```

This field is optional. If it is not specified (or not present in the configuration file), ExchangeAgent uses the pound sign (#) as the multi-valued attribute separator.

Trace

The **Trace** field controls whether ExchangeAgent performs program flow tracing on an export operation. The syntax is:

Trace=[*switch*]

where *switch* is one of the following values:

- **0** - Do not perform program flow tracing on the export operation (default)
- **1** - Perform program flow tracing on the export operation

If 1 is specified, ExchangeAgent writes information about the export operation to the pathname specified in the **TraceFileName** field.

TraceFileName

The **TraceFileName** field specifies the pathname of the trace file to which ExchangeAgent is to write information about the export operation. The syntax is:

TraceFileName=*pathname*

where *pathname* is the name for the trace file. For example:

```
TraceFileName=c:\Exchsync\ExportTraceFile
```

This field is optional unless **Trace** is set to 1. ExchangeAgent does not evaluate this field if **Trace** is set to 0.

NTAccountFormat

The **NTAccountFormat** field controls the format in which the attribute **Assoc-NT-Account** is exported. The syntax is:

NTAccountFormat=*format*

where *format* is one of the following values:

- **Text** - The **Assoc-NT-Account** is exported in the format ***domain_name*_user_name_** (default)
- **Hex** - The **Assoc-NT-Account** is exported as a hex string.

3.7.2.2.8. The DeltaExport Section

The DeltaExport section is an optional section that provides information that can be used to direct ExchangeAgent to perform a delta export of entries from the Exchange directory.

The next sections describe the fields in the DeltaExport section.

DeltaExport

The **DeltaExport** field controls whether ExchangeAgent performs a full or delta export of the Exchange directory. The syntax is:

DeltaExport=[*switch*]

where *switch* is one of the following values:

- **0** - Export all entries (or those entries specified with the **LDAPFilter** field) from the directory (default)
- **1** - Export only those entries whose USN-Changed attribute value is greater than or equal to the value of the **USNChangedMax** field.

If the **DeltaExport** field is set to 1:

- The Delta Export section must contain the **USNChangedMax** field and value
- The Attributes section must specify the USN-Changed attribute set to 1
- The **SelectAttributes** field in the SelAttributes section must be set to 1

This is an optional field. If it is not specified (or the field is not present in the configuration file), ExchangeAgent exports all entries in the directory (or all the entries selected using the specifications in the **LDAPFilter** field, if it is present in the configuration file) from the Exchange directory.

USNChangedMax

The **USNChangedMax** field specifies the highest-numbered USN-changed attribute value in the Exchange directory. The syntax is:

USNChangedMax=*USN*

where *USN* is an integer that represents the highest-number USN assigned to an Exchange container entry in the directory. For example:

```
USNChangedMax=6426
```

The USN-Changed attribute is a Microsoft Exchange attribute that is assigned to every entry in the Exchange directory. When the Exchange server carries out a modification to an entry, it assigns the highest USN to the entry's USN-Changed attribute as its value.

For the initial delta export, you must:

- Set the value of **USNChangedMax** to 0:
- Set the **DeltaExport** field to 1

ExchangeAgent selects only those entries modified after the value provided in the **USNChangedMax** field (all entries, in this export run). When it completes the export, ExchangeAgent updates the **USNChangedMax** field with the current highest USN-Changed attribute value.

On subsequent exports, each time ExchangeAgent performs an export and the **DeltaExport** field is set to 1, it automatically appends the string USN-Changed[3]*USN* to the filter specified in the **LdapFilter** field, where *USN* is the current value in the **USNChangedMax** field. It then performs the export and updates the **USNChangedMax** field with the current highest USN-Changed attribute value.

Performing incremental delta exports works only if you do not change the search filters or the attributes in the export configuration file. If you make changes to search filters in the SearchFilter section or to the attribute list in the Attributes section, you will need to perform a full export and re-set the **USNChangedMax** attribute.

3.7.2.3. Import Configuration File Format

The ExchangeAgent import configuration file consists of three sections:

- The Version section (required)
- The Connection section (required)
- The Configuration section (optional)
- The Ignore Empty Attributes section (optional)
- The Encrypted Attributes section (optional)
- The AttributeTypes section (required)

3.7.2.3.1. The Version Section

The Version section consists of a single field that specifies the import configuration file version. The syntax is:

Version=*version_number*

where *version_number* is the version number assigned to the configuration file, in the format *n.nn*. The current version is:

Version=1.03

This is a mandatory field. This document describes the latest version of the ExchangeAgent import configuration file. The ExachangeAgent is able to process configuration files with version number **1.03** or lower as well as "old" files that do not contain a Version section. The

following table provides information about the differences between import configuration file versions and about the support of older import configuration file versions for compatibility reasons:

	"Old"	1.00	1.01 and higher
TraceLevel	Supported	Not supported	Not supported
Trace	Not supported	Supported (1)	Supported
SearchBase	Not supported	Not supported	Supported

(1) **TraceLevel** has been replaced by **Trace**.

The following new sections or section fields have been added to the specified version and do not stay in conflict to older versions. These sections and fields are optional: if present, they are performed; if not, the default behavior is performed.

Version 1.02:
CreateNTAccounts
DeleteNTAccounts
[IgnoreEmptyAttrValues]

3.7.2.3.2. The Connection Section

The Connection section is a mandatory section that consists of fields that define the parameters of an import operation for ExchangeAgent. The next sections describe these fields.

UserName

The **UserName** field specifies the NT account that ExchangeAgent is to use when binding to the Exchange server during the import procedure. It has the same syntax as the **UserName** field in the export configuration file.

A recipient (Mailbox, Remote-Address and/or Distribution-List) can be hidden in an Exchange directory if its Hide-From-Address attribute is assigned the value True. To modify hidden recipients on import, you must append the NT Administrator account name cn=admin to the values specified in the **UserName** field. For example:

```
UserName=CN=Beninga,CN=Saturn,CN=admin
```

Deleting hidden recipients does not require the cn=admin privilege.

Password

The **Password** field specifies the password for the NT account name specified with the **UserName** field. It has the same syntax as the **Password** field in the export configuration file.

UseSecureAuthentication

The **UseSecureAuthentication** field controls the level of authentication that

ExchangeAgent uses when binding to an Exchange server during the import procedure. It has the same syntax as the **UseSecureAuthentication** field in the export configuration file.

UseEncryption

The **UseEncryption** field controls whether or not the Secure Socket Layer (SSL) port is used to provide a secure channel during the import procedure. It has the same syntax as the **UseEncryption** field in the export configuration file.

SearchBase

The SearchBase field specifies the base within the Exchange directory from which to export entries. The syntax is:

SearchBase=LDAP://*host_name*[:_port_number_][/*distinguished_name*]

where:

- *host_name* specifies a computer name, an IP address, or a domain name.
- *port_number* specifies the port on *host_name* on which the Microsoft Exchange LDAP server listens for requests. If *port_number* is not specified, ExchangeAgent uses the default LDAP port number 389.
- *distinguished_name* specifies the name of the target Exchange directory root, in top-down (DAP-style) or bottom-up (LDAP-style) naming format.

  For example:

  ```
  SearchBase=LDAP://Saturn/O=MyCompany/OU=Talk1/CN=Recipients
  ```

  Any comma (,) and forward slash (/) characters that are present in naming attribute values of *distinguished_name* must be "escaped" with the backslash character. For example:

  ```
  SearchBase=LDAP://Venus/O=OpTech\, Inc./OU=Talk2/CN=Recipients
  ```

  The SearchBase field is a mandatory field when the import data file uses the "ldapFilter" attribute.

3.7.2.3.3. The Configuration Section

The Configuration section is an optional section that consists of fields that contain information that ExchangeAgent is to use when evaluating entry attributes in an import data file. The next sections describe these fields.

MultiValueSeparator

The **MultiValueSeparator** field specifies a value to be used to separate the individual attribute values of a multi-valued attribute. The syntax is the same as the **MultiValueSeparator** field in the export configuration file.

IgnoreObjectClass

The **IgnoreObjectClass** field controls whether ExchangeAgent evaluates or ignores the

ObjectClass attribute of entries for which the "modify" LDIF changetype operation has been specified. The syntax is:

IgnoreObjectClass=*switch*

where *switch* is one of the following values:

- **0** - Evaluate the ObjectClass attribute when the changetype operation is "modify"
- **1** - Ignore the ObjectClass attribute when the changetype operation is "modify" (default)

Exchange server does not currently permit the modification of the ObjectClass attribute through an LDIF "changetype" operation. Consequently, you can set the IgnoreObjectClass field to 1 to direct ExchangeAgent not to pass the ObjectClass attribute to the Exchange server. However, other LDAP servers that manage Exchange directories do permit the modification of the ObjectClass attribute. Setting IgnoreObjectClass to **0** in these cases permits ExchangeAgent to pass the ObjectClass attribute to the LDAP server for evaluation.

Trace

The **Trace** field controls whether ExchangeAgent performs program flow tracing on an import operation. It has the same syntax as the **Trace** field in the export configuration file and is an optional field.

TraceFileName

The **TraceFileName** field specifies the pathname of the trace file to which ExchangeAgent is to write information about the import operation. It has the same syntax as the **TraceFileName** field in the export configuration file and is an optional field unless the **Trace** field is specified.

RejectSpecialCharacters

The **RejectSpecialCharacters** field controls whether ExchangeAgent evaluates the ADsPath attribute of entries in the import data file for special characters. The syntax is:

RejectSpecialCharacters=*switch*

where *switch* is one of the following values:

- **0** - Do not evaluate the ADsPath attribute for special characters (default)
- **1** - Evaluate the ADsPath attribute for special characters

If 1 is specified, ExchangeAgent scans the Common-Name (cn) RDN of each import entry's ADsPath attribute for the characters specified in the **RejectedCharacters** field and rejects the entry for import if it contains one of these characters.

Use this field in conjunction with the **RejectedCharacters** field if you are working with Microsoft Exchange server 5.5; these fields provide a "work around" to a bug that exists in this version of Exchange server. For more information about the bug and its recommended resolution, see the URL:

http://support.microsoft.com/support/kb/articles/q222/6/47.asp

RejectedCharacters

The **RejectedCharacters** field specifies the characters in the import entries' AdsPath attribute that ExchangeAgent is to scan for; ExchangeAgent is to reject the entry for import if it contains one of these characters. The syntax is:

RejectedCharacters=*characters*

Where characters specifies the characters in the Common-Name RDN of the AdsPath attribute for which ExchangeAgent is to search.

This field is optional unless **RejectSpecialCharacters** is set to 1. ExchangeAgent does not evaluate this field if **RejectSpecialCharacters** is set to 0.

CreateNTAccounts

The **CreateNTAccounts** field controls whether ExchangeAgent creates an NT-Account if the attribute NT-Account is specified in the data record or if it only tries to assign this NT-Account to the mailbox. The syntax is:

CreateNTAccounts=*switch*

where *switch* is one of the following values:

- 0 - Don't create the NT-Account if it does not exist
- 1 - Create the NT-Account if it does not exist (default)

If 1 is specified, ExchangeAgent creates the NT-Account if the attribute NT-Account is specified in the data record and if the account does not exist yet. If 0 is specified, ExchangeAgent tries to assign the specified account to the mailbox, but does not create it if the assignment fails.

DeleteNTAccounts

The **DeleteNTAccounts** field controls whether ExchangeAgent deletes an NT-Account when deleting the mailbox if the attribute NT-Account is specified in a data record with changetype delete. The syntax is:

DeleteNTAccounts=*switch*

where *switch* is one of the following values:

- 0 - Don't delete the NT-Account when deleting the mailbox
- 1 - Delete the NT-Account when deleting the mailbox (default)

If 1 is specified, ExchangeAgent deletes the NT-Account when deleting the mailbox if the attribute NT-Account is specified in a data record with changetype delete. If 0 is specified, ExchangeAgent does not delete the specified NT-Account.

3.7.2.3.4. The Ignore Empty Attributes Section

The Ignore Empty Attributes section is an optional section that lists attributes which are ignored if they exist in the import data file and if they are empty. Normally an attribute with an empty value results in clearing that attribute in the Active Directory. The attributes are listed in the format:

name_of_attribute=1

where *name_of_attribute* is the name for the attribute to be imported.

For example:

```
[IgnoreEmptyAttrValues]
NT-Account=1
Submission-Cont-Length=1
```

3.7.2.3.5. The Encrypted Attributes Section

The Encrypted Attributes section is an optional section that lists attributes that are encrypted in the import data file and which must be decrypted by ExchangeAgent before they are passed to the Adsi interface. This function only works correctly in an appropriate security environment, such as the DirX Identity environment configured in security mode (see the *DirX Identity Connectivity Administration Guide* for more information). The attributes are listed in the format:

name_of_attribute=1

where *name_of_attribute* is the name of the attribute to be imported.

For example:

```
[EncryptedAttributes]
description=1
```

3.7.2.3.6. The Attribute Types Section

The Attribute Types section is a mandatory section that specifies the attribute syntax for each Exchange attribute to be imported into the Exchange directory. The section consists of one or more attribute syntax specifications in the format:

LDAP_name_of_attribute=attribute_syntax

where *LDAP_name_of_attribute* is the LDAP name for the Exchange attribute to be imported and *attribute_syntax* is one of the following keywords:

Boolean
CaseExactString
CaseIgnoreString
DNstring
Integer
LargeInteger

NumericString
ObjectClass
OctetString
PrintableString
ProviderSpecific
UTCTime

Each of these keywords corresponds to a data type that can be passed over the Active Directory Services Interface (ADSI). Because ExchangeAgent uses this interface, it must specify the data type of each attribute it passes over the interface. The Attribute Types section provides ExchangeAgent with the information it needs about each attribute's ADSI data type.

For example:

```
[Attribute Types]
changetype=CaseIgnoreString
objectClass=CaseIgnoreString
Company=CaseIgnoreString
cn=CaseignoreString
department=CaseIgnoreString
member=DNString
Replication-Sensitivity=Integer
Report-To-Originator=Boolean
```

3.7.3. Export and Import Data File Format

The ExchangeAgent import and export data files use a tagged file format. The next sections describe the:

- General characteristics of export and import data file formats
- Specific features of the import data file format

3.7.3.1. General Data File Format

The ExchangeAgent import and export data files have the following characteristics:

- Each entry attribute is contained on one line; line continuation is not permitted.
- The representation of each attribute is: *attribute_name*:_attribute_value(s)_
- Leading and trailing whitespace between *attribute_name* and *attribute_value* is ignored. For example, in the attribute:

```
      cn: SallyAnn K. Quebec
```

The whitespace between the colon (:) and the start of the attribute value is ignored, but the whitespace within the attribute value is returned.

- The form-feed character (0x0c) is used as a record (entry) separator
- The form-feed character can optionally appear as the first line in the file
- There is no special character processing (there is no "escaping" mechanism)

Here is an example:

```
(0x0c is here as a record (entry) separator)
changetype: delete
objectClass: Remote-Address
cn: Robert Amber
rdn: Robert Amber
Replication-Sensitivity: 20
telephoneNumber: 603 555 8845
uid: AliasAmber
givenName: Robert
l:Nashua, New Hampshire
postalAddress: 110 Spitbrook Road
postalCode: 03060
sn: Amber
Target-Address: SMTP:robert_amber@spitbrook.digital.com
(0x0c is here as the record (entry) separator)
...
```

"Microsoft Exchange Directory Schema" describes the Microsoft Exchange directory schema.

3.7.3.2. Import Data File Format

ADsPath

Individual entries in a single import data file can be targeted for import to different containers or Exchange servers. Consequently, each entry in an ExchangeAgent import data file must contain an ADsPath attribute that identifies the fully-qualified pathname of the Remote Address, Distribution List, or Mailbox entry to be added, modified, or deleted. The attribute syntax is:

ADsPath:LDAP://*host_name*[:_port_number_][/*distinguished_name*]

where:

- *host_name* specifies a computer name, an IP address, or a domain name.
- *port_number* specifies the port on *host_name* on which the Microsoft Exchange

LDAP server listens for requests. If *port_number* is not specified, ExchangeAgent uses the default LDAP port number 389.

- *distinguished_name* specifies the name of the target Exchange directory root, in top-down (DAP-style) or bottom-up (LDAP-style) naming format.

For example:

```
ADsPath=LDAP://Mars/O=MyCompany/OU=Talk1/CN=Recipients/CN=Arno Held
```

ldapFilter

The ExchangeAgent import data file format also supports a per-entry ldapFilter attribute. The value of this attribute is a search filter that specifies an attribute that acts as a unique key for matching the entry in the import data file with an entry in an Exchange Directory. The attribute syntax is:

ldapFilter: *filter*

where *filter* is a search string specified in LDAP filter syntax (see RFC 2254 for an explanation of this syntax) that uses an attribute as a unique identifier. For example:

```
ldapFilter: (&(objectClass= Remote-Address)(uid=Held))
```

It is recommended to use the **uid** attribute as the unique key in the ldapFilter attribute; however, other attributes can be defined and used as keys. When a "modify" or "delete" entry (see the per-entry changetype description) in the import data file contains an ldapFilter attribute, and an ADsPath attribute is not present, ExchangeAgent uses the ldapFilter attribute to search the Exchange Directory specified in the **SearchBase** field using the scope specified in the **SearchScope** field (or the default). If ExchangeAgent finds one entry that matches the filter criteria, ExchangeAgent modifies or deletes the entry, according to its changetype attribute. If ExchangeAgent finds more than one matching entry, or does not find a matching entry at all, it writes an error to the import error file.

Per-Entry Changetype

The ExchangeAgent import data file format also supports the LDIF per-entry "changetype" attribute that indicates the type of modification to be made to the entry in the Exchange directory and the attribute operation codes "add" or "delete". The value for "changetype" is one of "add", "modify", or "delete". The changetype attribute name and its values are case-insensitive and can appear anywhere in the entry. If a changetype attribute is not present (or does not contain a value), ExchangeAgent attempts an "add" operation for the entry. If the "add" operation fails with the error code "entry already exists", it attempts a "modify" operation.

As with all other entries in an import data file, entries that contain changetype attributes must contain the ADsPath attribute. Entries with a "modify" changetype attribute value must also contain at least one attribute to be modified. If the modify operation fails with the error code "no such object", ExchangeAgent attempts to find the object using the ldapFilter attribute (if present) and then performs the modify operation. Entries with an "add" changetype must contain an object class attribute. For a "modify" operation, attributes can be deleted either by setting their values to an empty string or by setting

them to the string <clear>.

Mandatory Attributes

Depending on the object class attribute value, the following attributes must also be present in the entry:

- For Remote Addresses (objectClass=Remote-Address), the attributes cn, uid, sn, Target-Address, textEncodedORaddress, Hide-From-Address-Book, MAPI-Recipient, Replication-Sensitivity, and rfc822Mailbox must be present
- For Distribution Lists (objectClass=groupOfNames) the attributes cn, member, Hide-DL-Membership, Hide-From-Address-Book, Replication-Sensitivity, rfc822Mailbox, and textEncodedORaddress must be present
- For Mailboxes (objectClass=organizationalPerson), the attributes cn, uid, sn, Assoc-NT-Account, Home-MDB, Home-MTA, mailPreferenceOption, MDB-Use-Defaults, MAPI-Recipient, Replication-Sensitivity, rfc822Mailbox, and textEncodedORaddress must be present

Multi-valued Attributes and Operation Codes for Attributes

The attributes for a multi-valued attribute appear on one line and are separated by the multi-valued attribute separator specified in the **MultiValueSeparator** field in the import configuration file. For example:

```
...
member:
cn=Test1,cn=Recipients,ou=identity1,o=MyCompany#cn=Test2,cn=Recipie
nts,ou=identity1,o=MyCompany#cn=Test3,cn=Recipients,ou=identity1,o=
MyCompany
```

For entries with a "modify" changetype, the specified attributes are overwritten with the new value and the other attributes retain their old value. If an operation code for an attribute is specified, values for this attribute can be added or deleted. A sample for the syntax is given for the following entry:

```
changetype: modify
ADsPath: LDAP://IdentityServ2000/CN=Arno Held,CN=Recipients,
     OU=identity1,O=MyCompany
sn: Held
add: member
member: cn=Test4,cn=Recipients,ou=identity1,o=MyCompany
delete: member
member: cn=Test1,cn=Recipients,ou=identity1,o=MyCompany#cn=Test2,
    cn=Recipients,ou=identity1,o=MyCompany
```

Comments

The import data file can contain comments, which are identified by a # character at the beginning of a line.

Assoc-NT-Account Attribute

Mailbox entries to be imported can contain the Assoc-NT-Account pseudo attribute, which ExchangeAgent uses to assign an NT account to a mailbox entry to be added. The attribute has the syntax:

Assoc-NT-Account: *domain\user*

where *domain* is the domain name and *user* is the user account name. For example:

Assoc-NT-Account: ASW\Testuser

ExchangeAgent uses the domain name and user account name to create the LDAP attributes Assoc-NT-Account and NT-Security-Descriptor and then passes these attributes to the Exchange server. The attribute syntax for these LDAP attributes is OctetString, for example, **00515000000EE490F6B657A1E603031**. The attribute Assoc-NT-Account is represented in the export data file in the form: domain\user or in the OctetString form if specified so (hex string) in the export.ini file.

Attributes of Type OctetString

If an attribute in the **import.ini** file is specified with the OctetString, data type, ExchangeAgent expecteds it to be base64 encoded in the import data file. In export direction, ExchangeAgent writes attributes with type OctetString in Base64 encoding into the export data file.

3.7.4. Import Error File Format

During the import process, ExchangeAgent writes the original attributes and values of Remote Address, Distribution List, or Mailbox entries that it is unable to import into the error file specified on the command line along with an error message that describes the error that caused the import to fail on the entry. Each error record in the import error file has the following format:

#*warning_message*
source_entry
#*error_message*

Where *warning_message* contains a warning text, *source_entry* is the original entry that ExchangeAgent was unable to import and *error_message* contains the function name that failed and an error code and error text. Entries can have either *warning_message* or *error_message* or both of them. For example:

```
#Warning! Cannot find Attribute Type of xxx. Attribute Ignored.
changetype: add
objectClass: organizationalPerson
ADsPath:
```

```
LDAP://Premium:390/o=MyCompany/ou=identity1/cn=Recipients/cn=Marc
Held0
cn: DisplayMBHeld0
uid: Marc Held0
sn: Held0
xxx: test
NT-Account: ASW\Testuser
Home-MDB: cn=Microsoft Private
MDB,cn=ExchServer1,cn=Servers,cn=Configuration,ou=identity1,o=MyCompa
ny
Home-MTA: cn=Microsoft
MTA,cn=ExchServer1,cn=Servers,cn=Configuration,ou=identity1,o=MyCompa
ny
mailPreferenceOption: 0
MDB-Use-Defaults: True
MAPI-Recipient: True
Replication-Sensitivity: 20
rfc822Mailbox: marc.held2@icn.mycompany.de
textEncodedORaddress: c=de;a=abc;p=MyCompany;o=identity1;s=Held2;
#Error! CreateDSObject of
LDAP://Premium:390/o=MyCompany/ou=identity1/cn=Recipients/cn=Marc
Held0 failed. Error Code: 80071392 Error Text: The object already
exists.
```

Any entry that cannot be imported into the Exchange directory is written into the import error file. Consequently, you can use the error file as an input file and re-run the import operation, after first fixing the errors reported in the file. A timestamp is written at start and end of the error file.

3.7.5. ExchangeAgent Import Notes

Each mailbox in Exchange is associated with an NT account. As part of the import procedure, ExchangeAgent manages the NT accounts associated with mailboxes as follows:

- When ExchangeAgent creates a mailbox and the NT-Account attribute is present in the import entry and did not previously exist and the CreateNTAccounts field is not set to 0 in the import.ini file, it creates an NT account and associates it with the mailbox. This operation is performed for import entries with no changetype attribute and for entries whose changetype attributes are "add" or "modify". For the operations creating and associating NT accounts to succeed:
  - the machine running ExchangeAgent must be a member of a domain that trusts all domains from which NT accounts are to be assigned

    - the Windows/NT Logon procedure must be carried out using an account that has the rights to create NT accounts in the specified domain.
    - the Windows/NT Logon procedure must be carried out using an account that has the rights to carry out security functions in the domain the machine running the ExchangeAgent is a member of.
- When ExchangeAgent deletes a mailbox (because the import entry has the changetype attribute "delete") and the NT-Account attribute is present in the import entry and the DeleteNTAccounts field is not set to 0 in the import.ini file, it deletes the associated NT account. To prevent the associated NT account from being deleted either set the DeleteNTAccounts field to 0 resulting in preventing the deletion for all associated NT accounts or make sure you filter out the attribute in your mapping procedure. If the NT-Account attribute is not present in the import entry, ExchangeAgent deletes only the mailbox.
- When ExchangeAgent modifies a mailbox and the NT-Account attribute is present in the import entry, it associates the mailbox with the NT account specified in the attribute. This operation is performed for import entries with no changetype attribute and for entries with the changetype attribute "modify".

In order for ExchangeAgent to perform these operations on NT accounts, the Exchange server Service account must be set up with "Service Account Admin" rights. Refer to the section "Enabling NT Account Management during Import Operations" in the sections that follow. ExchangeAgent can only automatically create and associate NT accounts with Exchange mailboxes; it cannot automatically create Windows 20xx or Active Directory accounts and associate them with Exchange mailboxes. It also cannot create a mailbox that is associated with a Windows 20xx or Active Directory account.

When importing mailboxes that are to be accessed from POP3 clients:

- The NT account associated with the mailbox must be identical to the mailbox aliasname attribute value (which is a UID)
- The POP3 client must use the mailbox aliasname as the logon name

3.7.6. Exchange Server Administration

The administrator of an Exchange server that is to be the target of an ExchangeAgent import or export operation may need to perform some administrative tasks on the server to ensure proper operation of the ExchangeAgent import and/or export procedure. The next sections describe how to:

- Manage the Exchange Server’s LDAP Interface
- Export Deleted Entries
- Set the Tombstone Lifetime for Deleted Entries
- Monitor LDAP Operations on the Exchange Server
- Enable NT Account Management during Import Operations

3.7.6.1. Managing the Exchange Server's LDAP Interface

ExchangeAgent accesses the Exchange server through its LDAP interface. Consequently, the administrator of an Exchange server that is to be the target of an ExchangeAgent export or import operation should ensure that the server is accessible through its LDAP interface.

Next, the administrator should ensure that the settings for the LDAP interface correspond to the requirements of the ExchangeAgent import and export configurations in the following areas:

- The port number set for the Exchange server must match the port number specified in the **SearchBase** field of the export configuration file
- The authentication method selected in the **UseSecureAuthentication** field of the export or import configuration files must be enabled in the Exchange server
- The maximum number of entries returned for a single search must be large enough to accommodate the number of entries to be exported in an ExchangeAgent export operation

Use the Exchange Server Administrator (**admin.exe**) to manage the Exchange server's LDAP settings:

- Select container **Organization** → **Site** → **Configuration** → **Protocols**, and then select **Properties**.
- Select **General** to check and change the Exchange server port number.
- Select **Authentication** to check and change the authentication methods (the default is no authentication; this selection does not need to be changed if the **UseSecureAuthentication** field is set to 0 (no authentication)
- Select **Search** to check and change the maximum number of entries returned in a search result (the default is 100 entries)

If you do not want to inherit Site Protocol settings, you can use the **Ldap-Protocol** menu of the **Server** container to check and change the LDAP settings.

The Exchange server must be restarted after changing any LDAP settings.

3.7.6.2. Exporting Deleted Entries

To enable ExchangeAgent to extract and read deleted entries from an Exchange directory, the Exchange server that manages the target Exchange directory must be administered as follows:

- The Exchange server Service on the NT system that is running the target Exchange server must be started with an NT account that is a member of the Administrators local group.
- The Exchange server Service account must be set up with "Service Account Admin" rights. To establish these rights, use Exchange Server Administrator (**admin.exe**) as follows:

- Open **Properties** of the **Site** container
- Select **Permissions**
- Add the Exchange server Service account
- Select "Service Account Admin" rights for the account
- The NT account that is running the ExchangeAgent export procedure (the account that is specified in the **UserName** and **Password** fields of the export configuration file) must be granted "Admin" rights on the target Exchange server. To grant these rights, use Exchange Server Administrator (**admin.exe**) as follows:
- Open **Properties** of the **Site** container
- Select **Permissions**
- Add the ExchangeAgent NT account
- Select "Admin" rights for the account

In addition to the Exchange server setup just described, the NT account specified in the **UserName** field must be appended with cn=admin. For example:

```
UserName=CN=Smith,CN=TestDomain,CN=admin
```

3.7.6.3. Setting the Tombstone Lifetime for Deleted Entries

The Exchange server retains deleted entries and attributes for the period of time specified in its Tombstone Lifetime property. The default time period is 30 days and can be changed using Exchange Server Administrator (**admin.exe**) as follows:

- Select the container **Organization** → **Site** → **Configuration**
- Select **Properties** of **DS Site Configuration**
- Select **General**, and then set **Tombstone Lifetime**

3.7.6.4. Monitoring LDAP Operations on the Exchange Server

The administrator of an Exchange server that is the target of ExchangeAgent import and export operations can enable LDAP logging in the Exchange server to monitor incoming LDAP operations, in cases where ExchangeAgent import or export operations are not returning the expected results. To enable LDAP logging, use Exchange Server Administrator (**admin.exe**) as follows:

- In the **Servers** container, open **Properties** of the Exchange server
- Select **Diagnostics Logging**
- Select **MSExchangeDS**, **LDAP Interface**, and the maximum logging level in **Logging Level**

Use the Event Viewer to review the LDAP logging information returned by the Exchange server.

3.7.6.5. Enabling NT Account Management during Import Operations

To enable ExchangeAgent to manage the NT accounts associated with mailboxes during its import procedure, the Exchange server Service account must be set up with "Service Account Admin" rights. To establish these rights use Exchange Server Administrator (**admin.exe**) as follows:

- Open **Properties** of the **Site** container
- Select **Permissions**
- Add the Exchange server Service account
- Select "Service Account Admin" rights for the account

3.8. ODBC Agent

DirX Identity provides two agents to handle the import and export of data from ODBC-based databases:

- ODBCAgentImp-the DirX Identity agent that handles the import of data into an ODBC database
- ODBCAgentExp-the DirX Identity agent that handles the export of data from an ODBC database

The ODBC agents run on Windows and Linux systems and can be used to access any database that is accessible through an ODBC driver.

ODBC agents can:

- Perform a full or a delta export of selected rows from a table or a join of tables
- Perform a full or a delta import to a single table, with insert (add), update (modify), and delete
- Perform a full or delta import by calling a Stored Procedure for each record
- Perform single-step operations, in which one entry is processed at a time, after which user input is required to continue
- Generate an import error file that records all rows that it fails to import
- Generate a log file (for tracing)

The following figures show the components of the ODBC agents.

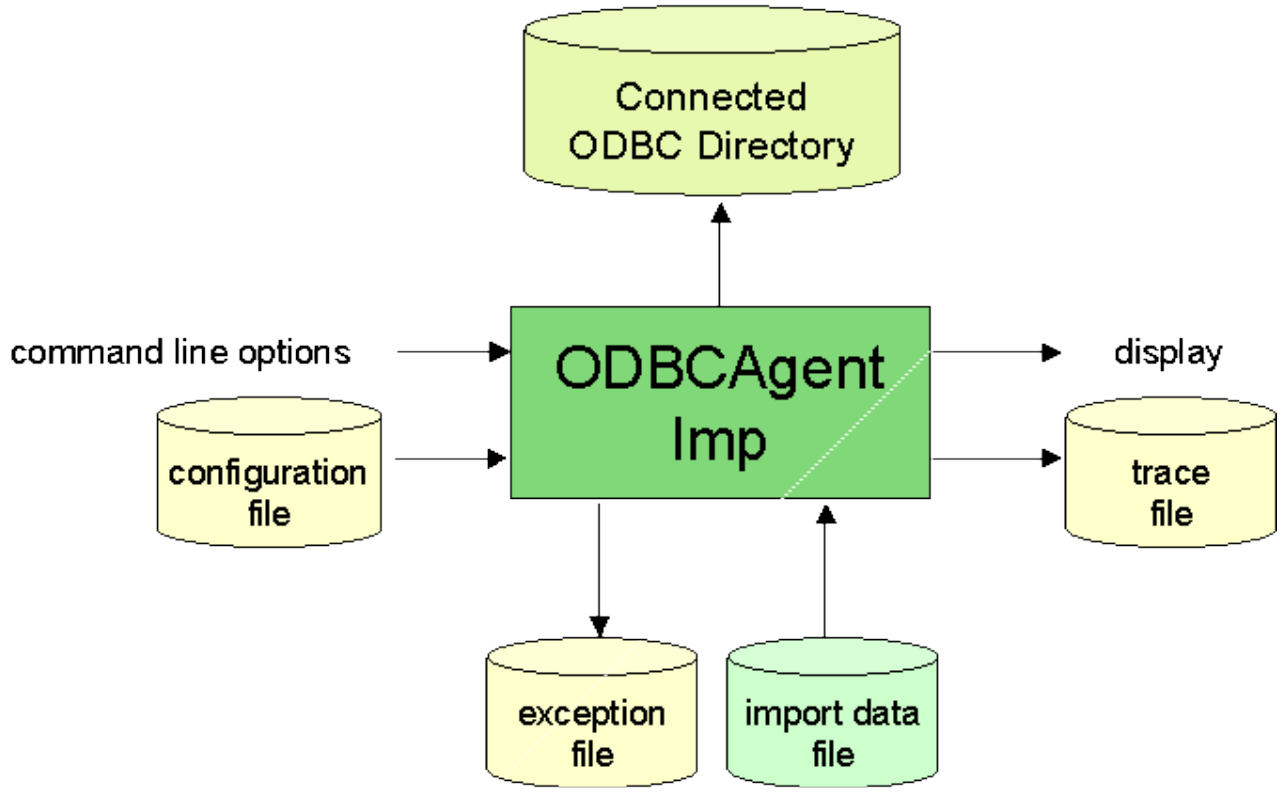


Figure 15. ODBCAgentImp Components

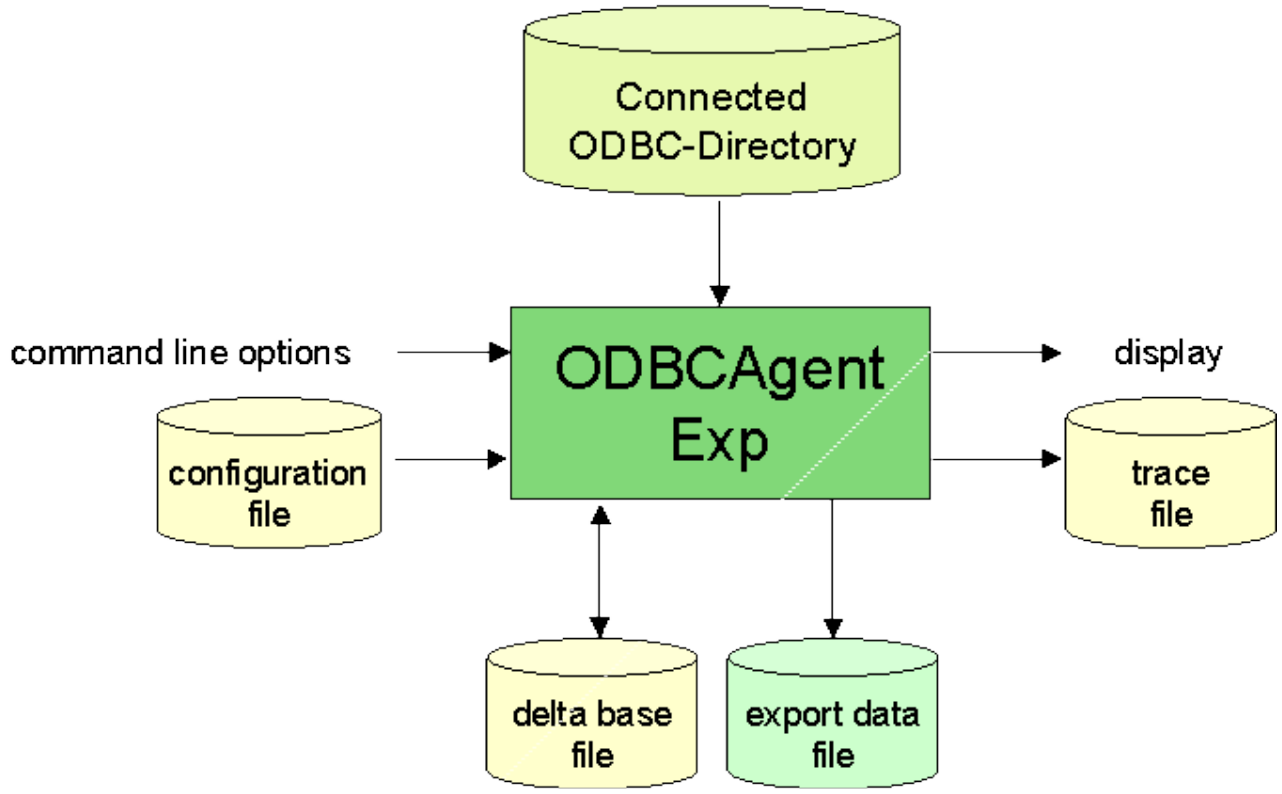


Figure 16. ODBCAgentExp Components

This section describes:

- ODBCAgentImp and ODBCAgentExp command line format
- ODBCAgentImp and ODBCAgentExp configuration files

- The export data file format that ODBCAgentExp generates
- The import data file format that ODBCAgentImp recognizes
- ODBCAgentImp import error file format
- The ODBC agent import procedure
- The ODBC agent full export procedure
- The ODBC agent delta export procedure

Sample configuration files and scripts are provided in the **ODBC Agent\Samples** directory of the DirX Identity installation. See the file **OdbcReadme.txt** for a description of these files and scripts.

Transactions in ODBC can be in one of two modes: auto-commit mode or manual-commit mode. By default, ODBC transactions are in auto-commit mode. In auto-commit mode, every database operation is a transaction that is committed when performed, but the degree of effective support for transactions is driver-defined. The ODBC agents use the default.

3.8.1. ODBCAgentImp Command Line Format

The command line format to invoke ODBCAgentImp is as follows:

ODBCAgentImp -f *configuration_file* [**-i** *data_file*] [**-n** *name*] [**-p** *password*] [**-s**] [**-v**] [-**Enc** *encryption_mode* -**Timeout** *timeout_value* -**AuditLevel** *audit_level* -**CryptLogLevel** *crypt_level*]

3.8.1.1. Parameters

-f *configuration_file*

Specifies the name of the file that contains the specifications for the import procedure. The agent assumes that the file exists in the current working directory unless a pathname is specified. This is a mandatory command line parameter.

-i *data_file*

Specifies the filename or the full pathname of a source file that contains the data to be imported into the ODBC database. If a file name is specified, the agent assumes that it is relative to the current working directory. If this option is not specified, the agent uses the filename **odbc_in.txt** relative to the current working directory.

-n *name*

Specifies a user name, where *name* is case-sensitive. This parameter may be required in order to access the database.

-p *password*

Specifies a user password, where *password* is case-sensitive. This parameter may be required in order to access the database.

-s

Runs the import operation in single-step mode.

-v

Directs ODBCAgentImp to use verbose reporting.

-ENC *encryption_mode*

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

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 must 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*).

-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*).

The following table describes the codes provided when ODBCAgentImp finishes running:

Exit Code	Description
0x00	Import successfully done
0x01	Import not done due to errors
0x3C	Import done with warnings

3.8.1.2. Command Line Description

The ODBCAgentImp command line parameters can be specified in any order. Each parameter can only be specified once.

It is not necessary to provide whitespace between a command line parameter and its argument. For example:

```
-imy_import_file.txt
```

is equivalent to

```
-i my_import_file.txt
```

Placing a # character where a command line parameter is expected causes ODBCAgentImp to ignore the remainder of the command line.

To display help information about ODBCAgentImp parameters, enter **ODBCAgentImp** on the command line or follow **ODBCAgentImp** with the # character to ignore the remaining parameters.

When it is invoked, ODBAgentImp reports any errors found in the command line. For example:

```
missing configuration file name
cannot open import data file for reading
odbc_in.txt
```

The **-n** and **-p** parameters may be required in order to access the ODBC database. These parameters provide credential attributes to the underlying ODBC access, and must comply with any requirements made by ODBC and the underlying database. Access to Microsoft Access can, but need not be, user-sensitive; it may only require a password, depending on the security arrangements made for the database. You cannot specify username and password in the configuration file; the command line is the only method permitted for specifying them.

The **-s** parameter invokes single-step mode. In single-step mode, ODBCAgentImp imports one row at a time, and then waits for the input from the keyboard. Possible inputs are:

q<CR> or **Q<CR>** or **n<CR>** or **N<CR>** - to terminate the import procedure (case-insensitive)

g<CR> or **G<CR>** - to terminate single-step mode and continue the import procedure (case-insensitive)

<CR> - to continue with the next row

The **-v** parameter directs ODBCAgentImp to write trace information to the display on standard out.

3.8.2. ODBCAgentExp Command Line Format

The command line format to invoke ODBCAgentExp is as follows:

ODBCAgentExp -f *configuration_file* [**-o** *data_file* | **+**] [**-n** *name*] [**-p** *password*] [**-r** [*ref_file*]] [

-s] [**-v**]
[**-Enc** *encryption_mode* **-Timeout** *timeout_value* **-AuditLevel** *audit_level* **-CryptLogLevel** *crypt_level*]

3.8.2.1. Parameters

-f *configuration_file*

Specifies the name of the file that contains the specifications for the export procedure. The file is taken to be in the current working directory unless a pathname is specified.

-o *data_file* | +

Specifies the filename or the pathname of the target export data file that is to contain the entries that ODBCAgentExp extracts from the ODBC database, or directs the agent to write the extracted entries to standard output, if the plus sign (+) is specified. If a filename is specified, the agent assumes it is relative to the current working directory. If this parameter is not specified, the agent writes to the filename **odbc_out.txt** relative to the current working directory.

-n *name*

Specifies a user name, where *name* is case-sensitive. This parameter may be required in order to access the database.

-p *password*

Specifies a user password, where *password* is case-sensitive. This parameter may be required in order to access the database.

-r [*ref_file*]

Specifies the name of a delta export reference file that ODBCAgentExp is to use as the base for a delta export operation (specified by a **Mode** field of **delta** or **delta-or-full** in the export configuration file). The file is taken to be in the current working directory unless a pathname is specified; all name forms acceptable to the operating environment are accepted (for example, fred.ref or reference\fred.ref or ..\reference\fred.ref or \users\myusers\reference\fred.ref..). If the **-r** flag is present, but *ref_file* is not specified, ODBCAgentExp performs a full export regardless of the setting in the **Mode** field and creates a new reference file that represents the full export.

-s

Runs the export operation in single-step mode.

-v

Directs ODBCAgentExp to use verbose reporting.

-ENC *encryption_mode*

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

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 must 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).

-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).

Exit Code

The following table describes the codes provided when ODBCAgentExp finishes running:

Exit Code	Description
0x00	Export successfully done
0x01	Export not done due to errors
0x3C	Export done with warnings.

3.8.2.2. Command Line Description

The ODBCAgentExp command line parameters can be specified in any order. Each parameter can only be specified once.

It is not necessary to provide whitespace between a command line parameter and its argument. For example:

```
-omy_export_file.txt
```

is equivalent to

```
-o my_export_file.txt
```

Placing a # character where a command line parameter is expected causes ODBCAgentExp to ignore the remainder of the command line.

To display help information about ODBCAgentExp parameters, enter just **ODBCAgentExp** on the command line.

When it is invoked, ODBCAgentExp reports any errors found in the command line. For example:

```
missing configuration file name
```

The **-n** and **-p** parameters may be required in order to access the ODBC database. These parameters provide credential attributes to the underlying ODBC access, and must comply with any requirements made by ODBC and the underlying database. Access to Microsoft Access can, but need not be, user-sensitive; it may only require a password, depending on the security arrangements made for the database. You cannot specify username and password in the configuration file; the command line is the only method permitted for specifying them.

The **-r** parameter specified with the *ref_file* option directs ODBCAgentExp to override its process for selecting a base delta export reference file and to use the file specified in *ref_file*. The **-r** parameter specified without the *ref_file* option overrides the **Mode** field specified in the export configuration file and can be used to perform a full export that also creates a new base reference file (ODBCAgentExp does not create a new reference file when the **Mode** field is set to **full**.)

The **-s** parameter invokes single-step mode. In single-step mode, ODBCAgentExp outputs one row at a time, and then waits for the input from the keyboard. Possible inputs are:

q<CR> or **Q<CR>** or **n<CR>** or **N<CR>** - to terminate the export procedure (case-insensitive)

g<CR> or **G<CR>** - to terminate single-step mode and continue the export procedure (case-insensitive)

<CR> - to continue with the next row

The **-v** parameter directs ODBCAgentExp to write trace information to the display on standard out.

3.8.3. Configuration File Format

Both ODBC agents read control information about the export or import procedure from a common configuration file. You do not need to provide import configuration information when exporting, or export configuration information when importing. If you do provide import configuration information when exporting (or vice-versa), the values that you supply will be checked for syntax, and, if incorrect, will prevent the operation from being executed. If you are uncertain about import (or export) details when doing the other operation, you may find it helpful to "comment out" each import line (including the heading) by inserting a # character at the beginning of the line.

Templates of import and export configuration files are provided with the ODBC agent installation. The filenames are:

- **NWAcc70.ini** (Northwind ODBC and Microsoft Access Version 7.0)

- **NWAcc97.ini** (Northwind ODBC and Microsoft Access 97)
- **NWSQLServer70.ini** (Northwind ODBC and SQL Server Version 7.0)
- **hrora.cfg** (Unix: Oracle RDBMS)
- **hrtext.cfg** (Unix: Text database)

In general, you must customize these files to support the requirements of your ODBC import and export operations.

This section describes:

- The general structure of a configuration file
- The configuration file sections
- The configuration file error reporting

3.8.3.1. General Structure of a Configuration File

A ODBC agent configuration file consists of sections and fields defined within those sections. An ODBC agent configuration file has the following general structure:

```
[SectionName]

#comment ...

sectionField=fieldValue
.
.
.
[SectionName]

#comment ...

sectionField=fieldValue
.
.
.
```

SectionName is a keyword enclosed in square brackets ([ ]) that identifies the purpose of the section. *sectionField* is a keyword that identifies the field and *fieldValue* is the value assigned to the section field, preceded by the equal sign (=). Whitespace is allowed on either side of the equal sign. For example:

MaxTraceFileSize=1024, or
MaxTraceFileSize = 1024

SectionName and *sectionField* are case-insensitive. *fieldValue* is usually case-insensitive, although text used directly by ODBC (for example, in the Database section) may be case-sensitive. Whitespace is permitted before and after each of these tokens. Comment lines

can be inserted anywhere in the configuration file and are identified by a # character at the beginning of the line. Note, however, that the ODBC agents recognize the # character within a non-comment line as real data, and will not, for example, ignore the remainder of the line.

Long *fieldValue* information can be placed on multiple lines by placing a backslash character (\) at the very end of a line that is to be continued. Line length is unlimited.

3.8.3.2. Configuration File Sections

The ODBC agent configuration file consists of the following sections:

- The version section (required)
- The attributes section (required)
- The database section (required)
- The import section (required for import and optional for export; if present, ODBCAgentExp only checks it. Optional for import using Stored Procedures; if present ODBCAgentImp only checks it)
- The procedures section (required for Stored Procedures; if both import and procedures sections are present, ODBCAgentImp uses Stored Procedures for the import)
- The export section (required for export and optional on import; if present, ODBCAgentImp only checks it)
- The control section (optional; all fields have default values)
- The encrypted attributes section (optional)

3.8.3.2.1. The Version Section

The **version** section consists of a single field that specifies the export configuration file version. The syntax is:

Version=*version_number*

where *version_number* is the version number assigned to the configuration file, in the format *n*.*nn*. The latest version is:

Version=1.01

This is a mandatory field. This document describes the latest version of the ODBC agent configuration file. The ODBC Agent is able to process configuration files with version number **1.00**, **1.01** or "old" files that do not contain a Version section. The following table provides information about the differences between configuration file versions and about the support of older configuration file versions for compatibility reasons:

	“Old”	**1.00**	**1.01**	**1.02**	**1.03**	**1.04**	**1.05**	**1.06**
Trace	n.s.	s.	s.	s.	s.	s.	s.	s.
NewReferenceFile	n.s.	n.s.	s.	s.	s.	s.	s.	s.

	"Old"	1.00	1.01	1.02	1.03	1.04	1.05	1.06
Autos (on Unix)	n.s.	n.s.	n.s.	n.s.	s.	s.	s.	s.
Autos (additionally on Windows)	n.s.	n.s.	n.s.	n.s.	s.	s.	s.	s.
EncryptedAttributes Encryption command-line switches Stored Procedures User and password in the configuration file	n.s.	n.s.	n.s.	n.s.	n.s.	s.	s.	s.
SQLExecDirect	n.s.	n.s.	n.s.	n.s.	n.s.	n.s.	s.	s.
SetOption	n.s.	n.s.	n.s.	n.s.	n.s.	n.s.	n.s.	s.

n.s. = not supported
s. = supported

3.8.3.2.2. The Attributes Section

The **attributes** section is a required section of the configuration file that defines the attribute abbreviations for ODBC database attributes ("columns", in ODBC syntax) to be imported or exported and maps the abbreviations to the corresponding ODBC table and column. The **attributes** section must appear before the **import** or **export** section.

Each field in the **attributes** section specifies an attribute definition. The field syntax is:

attribute_abbreviation[*qualifier*]*=**column_identifier*

where:

- *attribute_abbreviation* consists of one or more alphanumeric characters, including the underscore (*) and hyphen (-) that represent the short form or convenient notation for the column of information. The number of characters that can be specified for _attribute_abbreviation* is unlimited. Attribute abbreviations are case-insensitive in terms of matching.
- *qualifier* is an optional syntax that specifies any special data type and control information for the attribute, in the format:
  (*data_type*[[[*minimum*]-*maximum*]])[#]
  where *data_type* is the name of the attribute's data type (only **text** is currently permitted), *minimum* and *maximum* are non-negative integers that specify a lower (optional) and upper limit of attribute length, and the # character is a flag that, if present, causes the import of an entry to fail when the row to be imported contains an oversize attribute. The definition of "oversize" can be specified explicitly in *maximum*, but the ODBC database itself will often specify a maximum length value for the attribute in the relevant table, and the user-supplied value will be reduced to this value (if necessary).
- *column_identifier* identifies the column and generally has the form:
  table_name.column_name

For example:

```
        Employees.LastName
```

or

```
        HR.Employees.LastName
```

but can be in any form permitted by a SELECT statement. (Please read the notes concerning Oracle below first.) Specifically, the *table_name* value and the following period (.) can be omitted when exporting from a single table. (Import is normally performed on a single table, although update-modification, row insertion and row deletion-of joined tables is possible with some restrictions.) *column_identifier* can for export only take the value of an expression or a subquery (that is, joining up the strings formed by two or more columns, or adding up arithmetical values). In this case put parenthesis around the expression like in the following example. (Note that the MaxPrice line is splitted here into 3 lines with continuation symbol (\) just for better reading):

```
…
[attributes]
orderID=Orders.OrderID
MaxPrice=(SELECT MAX(OrdDet.UnitPrice) FROM \
     Northwind.dbo.[Order Details] AS OrdDet WHERE \
     Orders.OrderID = OrdDet.OrderID)
orderDate=Orders.OrderDate
[database]
DSN=…
[export]
SELECT=orderID, MaxPrice, orderDate
…
```

Microsoft Access or SQL Server: Table names and column names that contain the hyphen character (-) or the space character ( ) must be enclosed in square brackets [ ], for example, [My Table].[First-Name].

Oracle: Table names and column names that contain the hyphen or space character or other special characters must be enclosed in double quotation marks.

The following example illustrates an attribute definition that uses the optional *qualifier*:

```
SN(text[1-64])#=Employees.LastName
```

- For Oracle, import can be performed on a single table. It is possible to perform import on a „view“ which has been created to represents a join of two or more tables, but there are significant restrictions. (See the Oracle documentation for details.)
- For Oracle, only referring objects in the own schema is supported. In the syntax you omit the schema name.

An attribute definition field must exist for each ODBC attribute to be imported or exported. Each *attribute_identifier* must be unique, but a specific *column_identifier* can be mapped to more than one *attribute_identifier*. To obtain the column identifiers for ODBC attributes, it is recommended that you use the ODBC tools to access the ODBC database, and then copy the column identifiers as they appear in the resulting data file, for example, using a simple select statement like:

```
SELECT * FROM Employees
```

The ODBC agents convert attributes being supplied to the ODBC database to the column identifier using the first matching abbreviation.

You can use the abbreviations in the From and Where fields. Be careful in the naming of abbreviation because the substitution is a simple text substitution. You can see the substitution result in the SQL statements if you set the tracelevel SQL.

3.8.3.2.3. The Database Section

The **database** section is a required section that provides information that the ODBC agents need to access the ODBC database. The fields in the **database** section represent the information that is required by the ODBC driver for access to the ODBC databases it manages and differ depending on the ODBC driver in use. Specifying the DSN (*Data Source Name*) alone or with credentials should be adequate; otherwise, it will be necessary to consult the ODBC driver documentation to determine which fields are required and what values should be used.

The fields described in the next section vary depending which RDBMS you are using. At least DSN is necessary and in many cases sufficient. See the manual of the ODBC distribution you use for further information.

DSN

The **DSN** field specifies the name of the ODBC database as set up by the ODBC Data Source Administrator, which is available in the Windows 2003 Control Panel. On Unix this is configured in the .odbc.ini file in the HOME directory of the account in which the agent runs. The syntax is:

DSN=*name*

where *name* is the name of an ODBC database. For example:

```
DSN=my_database
```

This is the only mandatory field, and using it by itself will usually be sufficient.

If the DirX Identity Server is to start the agent and the Server is running as a service under a local system account, you must set up a System DSN (this DSN is shared by all users and services on the machine) or a User DSN.

For simplest operation, use the ODBC Data Source Administrator to bind a specific name, such as **my_database**, to a specific database, such as **C:\MSOffice\Access\Samples\Northwind.mdb**. You can also use the Administrator to select a class of databases (not selecting a specific database); in this case, running the agent causes a selection window to pop up to do the selection at run time.

User and Password Fields

The Access ODBC driver specifies username and password fields as credentials for authentication to the ODBC driver during ODBC database access. In some cases, it may be necessary to supply user and password credentials in order for the ODBC agents to gain access to the ODBC database.

Starting from version 1.04 user and password fields are allowed in the configuration file because passwords can now be written in scrambled or encrypted format into the file.

It is still possible to use the **-n** and **-p** parameters on the ODBCAgentImp or ODBCAgentExp command line to specify the credentials.

Other Driver-Specific Fields

Using the ODBC Data Source Administrator to establish an ODBC Data Source Name (DSN) also establishes a number of other driver-specific fields, which are used for internal purposes. These fields include the **Driver** field, which specifies the type of ODBC driver, and some other fields. You can obtain the values of these fields using the TraceLevel facility quoting ODBC, but their use in the context of Windows is not material.

The configuration file can be used to contain default values for these fields. Including values for these fields in the configuration file other than those supplied by the TraceLevel facility will cause the system default values to be replaced, but this should only be done with detailed knowledge of the effect on the database.

For example, if you use Microsoft SQL Server the fields DATABASE and SERVER are needed additionally.

3.8.3.2.4. The Export Section

The Export section consists of fields that define the parameters of an export operation for ODBCAgentExp. The next sections describe these fields. If present when importing, the values supplied here should be syntactically and semantically correct. If in doubt, deactivate the entire section by prefixing each line with the # character.

Mode

The **Mode** field specifies the type of export operation that ODBCAgentExp is to perform. The syntax is:

Mode=*mode*

where *mode* is one of the following keywords:

- **delta** - Perform a delta export and fail if delta processing cannot be performed
- **delta-or-full** - Perform a delta export, or perform a full export if delta processing cannot be performed
- **full** - Perform a full export

ODBCAgentExp cannot perform delta processing when:

- The **-r** *ref_file* parameter has been specified on the command line, and it cannot find or cannot open the specified reference file
- The **-r** parameter without the *ref_file* option has been specified on the command line
- It detects a change in the export specification; almost any change in configuration will cause the reference files no longer to match in their basic characteristics

If the **Mode** field is set to **delta-or-full** and the second and third cases occur, ODBCAgentExp performs a full export and creates a new reference file that represents it.

This is a mandatory field if delta exporting is to be performed and is optional for full exporting.

Select

The **Select** field specifies the set of entry attributes that ODBCAgentExp is to export from the ODBC database. The syntax is:

Select=*attribute_list*

where *attribute_list* specifies the attributes to export, in the format:

abbreviation[,*abbreviation* ...]

For example:

```
Select=GN, SN, T, TOC, BD, HD, A, CITY, REG, C
```

Keys and save-attributes (see SaveAttr below) are included in the exported attributes.

This is a mandatory field.

Keys

The **Keys** field specifies the set of attributes that ODBCAgentExp is to use to uniquely identify each entry to be exported from the ODBC database. The syntax is:

Keys=*attribute_list*

where *attribute_list* specifies the attributes to be used as unique identifiers, in the format:

abbreviation[*,**abbreviation* ...]

For example:

```
Keys=EID
```

The set of attributes specified in the **Keys** field should correspond to the set of primary keys defined for the ODBC database since the export procedure assumes that the combination of key values is unique in the selected export table. Any combination for which this is true will work properly. The attribute that represents the most significant primary key for ordering should appear as the first attribute in the list; the remainder should follow in order of precedence. The attributes specified in the **Keys** field can overlap the attributes specified in **Select**, but they cannot overlap the attributes specified in the **SaveAttr** field. This is a mandatory field if delta exporting is to be performed. It is mandatory for full export with the option to generate a delta export reference file.

SaveAttr

The **Keys** field is used to relate delta reference information to the ODBC database. The **SaveAttr** information enables the delta reference information to be related to information in the target database. For example, if an entry is removed from the ODBC database, it may be required to remove the corresponding entry in the target database; the **SaveAttr** field is used to specify any additional attributes that may be used to identify the entry in the target database that is to be removed. Otherwise, removal is impossible.

For example, an ODBC table may have a simple integer as a primary key. When synchronizing to a database which does not store the key, information such as surname, given-name, and initials may be required to identify the correct entry in the target database.

The **SaveAttr** field therefore specifies the set of entry attributes that ODBCAgentExp is to store in the delta export reference file that it creates as part of the delta export process. The syntax is:

SaveAttr=*attribute_list*

where *attribute_list* specifies the attributes to be saved in the reference file, in the format:

abbreviation[,*abbreviation* ...]

For example:

```
SaveAttr=GN, SN, T, TEL
```

By default, ODBCAgentExp does not store complete entries in the delta reference files it creates. Instead, it stores the attributes defined as keys (with the **Key** field) to uniquely identify each entry, and stores the complete contents of the entry as a hash value which cannot be used to reconstruct the complete entry. Use the **SaveAttr** field to store attributes that are important to the directory synchronization process in the reference

file. The attributes specified in the **SaveAttr** field cannot overlap the attributes specified in the **Key** field but can overlap the attributes specified in the **Selection** field.

This is an optional field.

From

The **From** field specifies the table or tables in the ODBC database from which ODBCAgentExp is to extract entries. The syntax is:

From=*tables*

where *tables* is a valid SQL-like expression that can be used in a FROM expression in a SQL SELECT statement. The value in *tables* can specify a simple table, for example:

```
/From=Employees
```

It can also specify a single or multiple union of tables, for example:

```
From=HR, PABX
```

or

```
From=Employees LEFT OUTER JOIN "org-units" on UnitId = EmpUnit
```

or

```
From=HR INNER JOIN PABX ON PNR = PID) AND (HR.EmployeeId = PABX.EmployeeId
```

You can also use a self-join to obtain further information from the same table. For example:

```
From=Employees INNER JOIN Employees AS Employees_1 ON Employees.ReportsTo
= Employees_1.EmployeeID;
```

In the previous example, Employees 1 is a correlation name used to distinguish multiple uses of an object. If correlation names are used, they must be specified in the **attributes** section of the configuration file; for example:

```
[attributes]
GN=Employees.FirstName
SN=Employees.LastName
BName=Employees_1.Title
BTitle=Employees_1.LastName
```

When using a join of tables in the **From** field, the appropriate table name should be used to prefix a column name; for example, **Employees.LastName**. You can use column name abbreviations in JOIN predicates (as PNR and PID were used in an earlier example).

This is a mandatory field.

If you are not completely familiar with SQL SELECT statements, when developing the SQL statement that defines tables to be exported, it is recommended that you use Access or another tool to design the query, then use the tool to view the FROM component of the resulting SQL statement. The agent should work with any FROM component that works in an SQL statement for the target database.

Where

The **Where** field controls whether or not ODBCAgentExp searches for and exports specific entries ("rows" in ODBC terminology). The syntax is:

Where=*predicate*

where *predicate* is a valid SQL expression that can be used in a WHERE expression in a SQL SELECT statement. For example:

```
Where=Employees.LastName LIKE 'D%'
```

The LIKE element in this case selects last-names that start with D, and is part of standard SQL. Examples of SQL predicates are:

```
ProductID>2
ProductName='Chai'
ProductName LIKE 'C%'
```

The repertoire of supported predicates varies with ODBC database, and some databases provide extensions to the SQL standard. Refer to the database documentation for details.

As for the **From** field above, you may find it convenient to develop a working SELECT statement using Access or another tool, and then "lift" the predicate from it.

This is an optional field. If it is not specified, ODBCAgentExp exports all of the rows in the selected table or join of tables.

MaxRows

The **MaxRows** field controls the number of entries that ODBCAgentExp writes to the export data file. The syntax is:

MaxRows=*number*

where *number* specifies the maximum number of entries ("rows" in ODBC terminology) to be output.

This is an optional field. If it is not specified, ODBCAgentExp exports a theoretical maximum of 2^{31}-1 rows (a little more than 2 billion).

ReferencePath

The **ReferencePath** field specifies the pathname to the directory in which ODBCAgentExp is to store delta export reference files. The syntax is:

ReferencePath=*directory_pathname*

For example:

```
ReferencePath=D:\Program Files\DirX Identity\ODBC\Data\myrefdir
```

This is an optional field; if it is not specified (or is not present in the configuration file), ODBCAgentExp stores the delta reference files it creates in the current working directory.

NewReferenceFile

The **NewReferenceFile** field stores the name of a reference file that ODBCAgentExp is to use as the base for a delta export operation. The syntax is:

NewReferenceFile=*ref_file*

For example:

```
NewReferenceFile=fred.ref
```

ODBCAgent writes a new reference file name into this field each time it performs an export operation using the naming convention described in "ODBCAgentImp Delta Export Process". See that section for a complete description of the reference file generation process and format.

SortControl

When the ODBCAgentExp creates a reference file for delta export, it sorts the records in the file by ordering the key fields. This ordering then permits fast analysis of changes between the previous state of the database and the present one, and allows the modified information to be selected. The process of sorting and extraction is much faster if the sorting of the reference file information corresponds to the order in which the ODBC database is sorted. The **SortControl** facility enables the sorting to be optimized where necessary. In many cases, the sorting will be correct anyway. (You can usually determine how sorting is done by the database by inspection of a full export data file.)

Use this field if you are exporting a large database and seek to optimize export times. A problem that may be resolved by using the field may be indicated by a delta export that takes a much longer time than a full export.

Thus, the **SortControl** field controls how ODBCAgentExp sorts attributes in the reference file when exporting from an ODBC database and can be used to override the agent's default sorting algorithm. The syntax is:

SortControl=[*form_list*]

where *form_list* specifies the matching rule form to apply to each key specified in the **Keys** field, in the format:

form[,*form* ...]

And *form* is one of the following (case-insensitive) keywords:

- **Integer** (or **Int**) - the key is to be interpreted as an integer and the first four characters are to be taken as a binary 32-bit number. For example, '0A2B' in memory is 0x42324130 in hex or is 1,110,589,744 in decimal format (Windows NT is a little-endian system). The sorting is endian sensitive.
- **Numeric** (or **Num**) - the key is to be right-justified before sorting and taken as a number. Numeric is handled as a lexical comparison based on the encoding value. It is not a "real" numeric sorting, which ignores leading zeros. For example, the agent sorts "39" < "0040"; as "0040" < "39".
- **CaseIgnore** (or **CI**) - the key is to be left-justified before sorting, ignoring case
- **CaseSensitive** (or **CS**) - the key is to be left-justified before sorting and case is significant
- **Any or empty** - the key is to be matched using the default sorting algorithm (see below)

For example:

```
SortControl=CI,,Num
```

specifies that the sorting for first three keys are to be specified as case-ignore, default, numeric.

The ordering of the keywords in *form_list* must correspond to the ordering of the **Keys** attributes; these must be ordered in accordance with the desired sort key precedence. If the **SortControl** field specifies **Any** or is empty, ODBCAgentExp uses the default sorting algorithm, where ODBCAgentExp sorts the ODBC database according to the data type of the column, ignoring case where relevant. The following table shows the ODBCAgentExp default sorting algorithm.

ODBC Code	Matching Rule Form
SQL_BIGINT	integer form
SQL_BINARY	numeric form
SQL_BIT	numeric form
SQL_CHAR	case-sensitive form
SQL_WCHAR	case-ignore form
SQL_DATE	case-ignore form
SQL_DECIMAL	numeric form
SQL_DOUBLE	fail
SQL_FLOAT	fail
SQL_INTEGER	integer form
SQL_LONGVARBINARY	fail
SQL_LONGVARCHAR	fail
SQL_WLONGVARCHAR	fail

ODBC Code	Matching Rule Form
SQL_NUMERIC	numeric form
SQL_REAL	fail
SQL_SMALLINT	integer form
SQL_TIME	case-ignore form
SQL_TIMESTAMP	case-ignore form
SQL_TINYINT	integer form
SQL_VARBINARY	integer form
SQL_VARCHAR	case-sensitive form
SQL_WVARCHAR	case-ignore form

This is an optional field.

3.8.3.2.5. The Import Section

The Import section consists of fields that define the parameters of the import operation for ODBCAgentImp. The next sections describe these fields.

If present when exporting, the values supplied here should be syntactically and semantically correct. If in doubt, deactivate the entire section by prefixing each line with the # character.

The procedures section has precedence over the import section.

Table

The **Table** field specifies the ODBC name of the table (or joined set of tables) into which entries are to be imported. The syntax is:

Table=*table_name*

or

Table=*joined-table*

An example of **Table=***table_name* syntax is:

```
Table=Employees
```

With the **Table=***joined-table* syntax, it is only possible to use the method of join that uses the JOIN keyword. For example, the following SQL statement block is permitted:

```
Table=Categories INNER JOIN (Suppliers INNER JOIN Products ON
Suppliers.SupplierID = Products.SupplierID)
ON Categories.CategoryID = Products.CategoryID
```

However, the following legal SQL statement block cannot be used as a basis for import because the predicate ("WHERE") as part of the SQL SELECT construct is already controlled by the **SelectBy** field:

```
SELECT ... FROM Categories, Suppliers, Products
WHERE (Suppliers.SupplierID = Products.SupplierID) AND
(Categories.CategoryID = Products.CategoryID);
```

The use of joined tables is permitted with certain limitations. The first table that is specified is called the "primary table" and must contain all of the columns specified by the **SelectBy** field. The primary table will be the *only* table whose rows are removed by a "delete" changetype entry, or inserted by an "add" changetype entry.

Both inner and outer joins are possible. If inner join is specified, ODBCAgentImp only evaluates the rows in the primary table that satisfy a join to the other table (or tables). If outer join is specified, all rows in the primary table are evaluated.

In the case of modification of a row in a joined table, only the columns specified by the **Modify** field will potentially be changed. You should note that two "rows" from a join of tables are not necessarily independent. For example, changing a Products.SupplierName from Organic Growers Ltd to OGL in one row has the effect of changing the name for all products that have the same supplier. If a subsequent row includes a setting of Products.SupplierName to the original value, the change will be undone for *all* rows. To control this effect, you must use the **Modify** field carefully to select the columns to be changed.

Note that the target ODBC database may refuse to carry out modifications if "referential integrity" will be violated; see "Import Procedure" for further details. Referential integrity problems can occur with delete commands, even when a single table is involved.

The **Table** field is a mandatory field.

SelectBy

The **SelectBy** field specifies one or more naming attributes that ODBCAgentImp is to use as selection criteria during the import procedure. The syntax is:

SelectBy=*predicate*

where *predicate* is built up in a natural way from abbreviations, ampersand (&) characters representing logical ANDs, vertical bar (|) characters representing logical ORs, exclamation point (!) characters representing logical NOT, and parentheses ( ) to coerce an order of evaluation.

For example:

A&!(B|C|!(D&E&(!F)))

When the predicate is evaluated, the values to be used for a particular abbreviation are taken from the row information being imported at any one time. The predicate is used

to select (if possible) a single row from the database.

The use of the **SelectBy** field is in strong contrast to the **Where** field for export, in which fixed values are used for every row to be selected for export. As an example (FirstName & LastName) would be used by a row to be imported that specified FirstName as Joe and LastName as Bloggs to select the single row that used this combination of FirstName and LastName. If there is more than one such row in the target database, the import of that particular row will fail because the import operation is ambiguous for the supplied data.

The precedence of AND, OR, and NOT items is as specified for SQL (NOT binds tightest, then AND, then OR). When in doubt, you should use parentheses ( ) to group these items.

This is a mandatory field.

Modify

The **Modify** field specifies the entry attributes in the ODBC database that ODBCAgentImp is to modify. The syntax is:

Modify=*attribute_list*

where *attribute_list* specifies the attributes to import, in the format:

abbreviation[,*abbreviation* ...]

For example:

```
Modify=T, TOC, BD, HD, A, CITY, REG, C
```

Naming attributes specified in the **SelectBy** field cannot be specified in the **Modify** field, nor can attributes that correspond to expressions in an export table (that is, which compute a value by combining more than one column value).

This is a mandatory field. ODBCAgentImp modifies only the columns that correspond to the attributes specified in *attribute_list*.

When using joined tables, remember that **Modify** field attributes can be selected to alter the column values in several tables at the same time.

CreateIfAbsent

The **CreateIfAbsent** field controls whether or not ODBCAgentImp creates a new ODBC entry ("row" in ODBC terminology) in the ODBC database if it does not find a matching entry using the naming attributes supplied in **SelectBy**. The syntax is:

CreateIfAbsent=*boolean*

where *boolean* is one of the following values:

- **TRUE** - Create a new entry using the attribute values supplied in the import data file (default)

- **FALSE** - Do not create a new entry

This is an optional field. If it is not present in the configuration file, ODBCAgentImp creates new entries if possible. If the **Table** field specifies a join of tables, ODBCAgentImp creates an entry (inserts a row) in the primary table only.

Exceptions

The **Exceptions** field specifies the import error file that ODBCAgentImp is to create and write error information about entries that cannot be imported into the ODBC database. The syntax is:

Exceptions=*filename*

For example:

`Exceptions=Exceptions.txt`

The filename is taken as within the current working directory unless it includes a relative or absolute pathname, such as:

```
Exceptions=..\except\Exceptions.txt
Exceptions=\users\fred\odbc\Exceptions.txt
```

ODBCAgentImp saves the complete row information, with diagnostic and other information, for each row that it is unable to import into this file. You can use this file as input to ODBCAgentImp and re-run the import operation, after first fixing the reported errors.

This is an optional field. If it is not specified in the configuration file, ODBCAgentImp creates the file **except.txt** in the current working directory.

InsertOnly

The **InsertOnly** field controls whether existing entries in the ODBC database are updated with attribute values from the import data file. The syntax is:

InsertOnly=*boolean*

where *boolean* is one of the following values:

- **TRUE** - Do not modify existing entries, and create new entries if there are no matches in the database using the naming attributes specified in the **SelectBy** field.
- **FALSE** - Modify existing entries with the attribute values supplied in the import data file (default) unless otherwise permitted.

An entry is created when no matching entry is found when any one of the following is true:

- The import entry (record) explicitly specifies that a new entry is to be added.

- **CreateIfAbsent** is set to **TRUE**
- **InsertOnly** is set to **TRUE**.

Thus, the **InsertOnly** field overrides a **CreateIfAbsent** field that is set to **FALSE** and vice versa.

This is an optional field. If it is not present in the configuration file, ODBCAgentImp only modifies existing entries unless otherwise directed.

ChangeType

The **ChangeType** field specifies the alphanumeric string used in the import data file to indicate the "changetype" for ODBC entries. The syntax is:

ChangeType=*string*

For example:

```
ChangeType=change_it
```

This is an optional field. If it is not present in the configuration file, ODBCAgentImp recognizes the string **ChangeType** as the "changetype" identifier. The value supplied to the "changetype" identifier must be one of **add**, **delete**, or **modify**.

If the **ChangeType** field is not defined, the control of the imported rows in the import data file is done by quasi-attributes using **ChangeType** such as:

```
ChangeType: delete
```

If it *is* defined (for example, to change_it), the control of imported rows is done by:

```
change_it: delete
```

Relationships

The **Relationships** field specifies references from one table to another for which referential integrity enforcement can be handled by nullifying the reference. Use the **Relationships** field to permit entries ("rows" in ODBC terminology) to be deleted when entries in other tables affected by referential integrity point to them, or when it is unacceptable for the reference to a deleted entry to continue to exist.

In order for ODBCAgentImp to implement this function:

- The reference that points to the entry to be removed must be nullifiable
- The access control that permits ODBCAgentImp to nullify the reference must be in force

The syntax is:

Relationships=*relationship*[,*relationship* ...]

where *relationship* is a string in the format:

abbreviation⇒abbreviation

The first abbreviation specifies the column in the table that contains a reference; this is the table that is affected by referential integrity. The second abbreviation specifies the column in the table that supplies the value of the reference. For example, Employees.[org-unit-id] could be a reference in the Employees table to an entry ("row", in ODBC terminology) in the OrgUnit table, using the value of OrgUnit.Id as the value used in the reference. In this case, the relationship would be specified as:

```
Employees.[org-unit-id]⇒OrgUnit.Id
```

Removing an OrgUnit entry (if successful) invalidates all the pointers Employees.org-unit-id that point to the OrgUnit entry. If referential integrity enforcement is switched on in the database for this relationship, removing the OrgUnit entry is impossible. However, if referential integrity enforcement is switched off, the OrgUnit entry can be deleted, leaving the Employees.[org-unit-id] references pointing into "thin air".

You must create abbreviations for all related table elements whose referential integrity you want to override. Thus, an entry, or "row", that is referenced by another row with referential integrity policing cannot be deleted. However, when the **Relationships** field has been used to specify a referential integrity override and ODBCAgentImp detects a failure to remove a row for this reason, it nullifies the column value for all rows that would otherwise refer to the row specified by the relationship. For example, consider two tables-"Clients" and "Websites"-with a set of row values as follows:

```
Clients
Id Surname Given Name Email Address
16 Smith Fenella fenella@mysp.net
Websites
Id Owner WWW Site
32 16 (to be nullified) www.fenella.mysp.net
45 16 (to be nullified) www.fenella-import.com
```

Referential integrity between these two tables is defined by **Websites.Owner ⇒ Clients.Id**. Row 16 in the Clients table cannot be removed while rows 32 and 45 in the Websites table point to it and the database applies referential integrity enforcement to the relationship. However, the referential integrity link can be broken by setting the Owner value to null. The **Relationships** field permits this action to occur automatically.

When the database does not apply referential integrity enforcement to the relationship, an attempt to remove Row 16 will succeed, and the references in the Websites table will stay set to the same (now-nonsensical) value. Use the **AlwaysFollowReferences** field to cause the relationships to be followed in the absence of referential integrity enforcement, or, better, establish referential integrity enforcement. This is an optional field.

AlwaysFollowReferences

The **AlwaysFollowReferences** field controls whether ODBCAgentImp follows the references defined in the Relationship field if referential integrity enforcement has not been configured in the database for the specific relationships specified. The syntax is:

AlwaysFollowReferences=*boolean*

where *boolean* is one of the following values:

- **TRUE** - always follow the references defined in the Relationships field even if referential integrity is not enforced for the references specified by **Relationships**
- **FALSE** - do not follow references if referential integrity is not configured in the database for the references specified by **Relationships** (default)

Using the database referential integrity enforcement mechanism is a more efficient solution than leaving referential integrity unenforced and using **AlwaysFollowReferences**. This is because **AlwaysFollowReferences** always checks for references, whether one exists or not for the particular row being removed. Thus, more operations are typically carried out when using **AlwaysFollowReferences** by comparison with using referential integrity enforcement.

This is an optional field.

ModifyAnyway

The **ModifyAnyway** field controls whether ODBCAgentImp performs a comparison operation before modifying an ODBC entry. The syntax is:

ModifyAnyway=*boolean*

where *boolean* is one of the following values:

- **TRUE** - compare the ODBC entry with the import data entry before modifying the entry
- **FALSE** - modify the ODBC entry without performing a comparison operation first (default)

When **ModifyAnyway** is set to TRUE, ODBCAgentImp compares each attribute value in a "modify" import data entry with the corresponding entry in the ODBC database. If all of the values match, ODBCAgentImp does not modify the ODBC entry. The following table shows the matching rules that ODBCAgentImp uses depending on the corresponding column data type. If <none> is specified then no comparison is performed.

SQL Data Type	Matching Rule Form
SQL_BIGINT	integer form
SQL_BINARY	numeric form
SQL_BIT	numeric form

SQL Data Type	Matching Rule Form
SQL_CHAR	case-sensitive form
SQL_DATE	case-ignore form
SQL_DECIMAL	numeric form
SQL_DOUBLE	none
SQL_FLOAT	none
SQL_INTEGER	integer form
SQL_LONGVARBINARY	none
SQL_LONGVARCHAR	none
SQL_NUMERIC	numeric form
SQL_REAL	none
SQL_SMALLINT	integer form
SQL_TIME	case-ignore form
SQL_TIMESTAMP	case-ignore form
SQL_TINYINT	integer form
SQL_VARBINARY	integer form
SQL_VARCHAR	case-sensitive form
SQL_WCHAR	case-sensitive form
SQL_WLONGVARCHAR	case-sensitive form
SQL_WVARCHAR	case-sensitive form

This is an optional field.

Autos

The **Autos** field specifies named sequences to be used with Oracle databases.

To generate an automatically incremented field for new rows a named sequence has to be used (comparing to autoNumber fields in Microsoft Access). This helps to generate unique primary keys. The syntax is:

Autos=*attribute_abbreviation*(*sequence_name*)[, *attribute_abbreviation* (*sequence_name*) ...]

Use comma-separated values in case there are multiple columns which require sequences are allowed.

An example for the attributes ID and SER, and the sequences employeeid and serial would be as follows:

Autos=ID(employeeid), SER(serial)

The presence of this field modifies the INSERT statements in the following way:

- If id is present, its value is automatically derived from the nextval value of the given sequence.

Internally the next value is derived from the Oracle-internal table “dual”.

- This feature is only supported with Oracle databases.
- You must provide values for the fields in the import data file although these values are irrelevant.

This is an optional field.

SQLExecDirect

The **SQLExecDirect** field controls whether ODBCAgentImp uses the ODBC function SQLExecDirect or SQLPrepare/SQLExec in stored procedure mode. The latter is the default. It has the advantage, that a repeatedly-used SQL statement is interpreted just once on the data base side. With SQLExecDirect in each cycle the statement has to be interpreted again. But SQLExecDirect avoids the issue with open cursors in some ODBC driver implementations, e.g. in the Microsoft SQL Server driver.

The syntax is:

SQLExecDirect=*boolean*

where *boolean* is one of the following values:

- **TRUE** – use SQLExecDirect in stored procedure mode
- **FALSE** – use a combination of SQLPrepare/SQLExec in stored procedure mode (default).

This is an optional field.

SETOPTIONS

The **SETOPTIONS** can be used to execute SQL SET statements one time before the import procedure starts. For example, if on the table a trigger is defined that starts a stored procedure it might be necessary to set some options. On Microsoft SQL Server data bases, certain options can not be set in the stored procedure but must be set in the client.

The statement is executed once after the structure of the targeted primary table has been evaluated.

The syntax is:

SETOPTIONS=*set_statement* [; ...]

where *set_statement* is a valid SQL SET statement for the target data base.

In the following example the options ARITHABORT and ANSINULLS are specified:

SETOPTIONS = SET ARITHABORT ON; SET ANSI_NULLS OFF

This is an optional field.

3.8.3.2.6. The Procedures Section

The Procedures section consists of fields that define the parameters of the import operation using Stored Procedures for ODBCAgentImp. The next sections describe these fields.

Procedure

The Procedure field specifies a list of configured procedures, of which the first is the active one. The syntax is:

Procedure=*proc-id* [, *proc-id2*]

Example:

```
Procedure=Full_SP
```

This is an mandatory field for import using stored procedures.

procedure declaration

The procedure declaration field specifies the stored procedure by name proc-id including response codes. This name must correspond precisely to the stored name of the Stored Procedure/Function in the RDBMS which is to be invoked.

proc-id = [*return-tag*] CALL *procedure-name* ([RETURN] *tag-1* [= *preset*] [,[RETURN] *tag-2* [= *preset*]] ...) RETURNING *range-item* [, *range-item*]

where

preset = “ literal-string “ | number

range-item = (range-list) [range-output]

range-list = (number | (number : number)) [, range-list]

range_output = [!] (“ literal-string “ | token)

This is a required field for import.

The identifiers (*proc-id, return-tag, tag-1* etc.) are arbitrary combinations of :

- Alphanumerics
- Underscore
- Hyphen

In the *range-output* production, the initial exclamation mark distinguishes between a warning and an error. The exclamation mark can be placed within a quoted string rather than preceding it, without change in effect; for example:

```
FullSP =res call SP_BENUTZERROLLE (op = 2, rname, mitarb) \
```

```
returning (0:9999),(-1)!"make it a good error!", \
(-2) ! "quite extraordinary", (-3) "!amazing"
DeltaSP =result call ADD_FUNC (base=2, addend = 5 )\
returning (0 , 1: 6 ),(7:10)"Too many for comfort", \
(11:1000 )"!far too many altogether"
```

Tag names *tag-1* etc. correspond directly to record attribute names in the input (and output) files. They match in sequence the return (if there is one) and the parameters of the stored procedure. Output tags may be involved in future developments to produce export files, but are not currently supported except as potential providers of a return value.

If there is a return tag (i.e. the procedure is in fact a function), the RETURN indicator cannot be used (since there already is a return). Otherwise, RETURN is only permitted with an argument that is defined as OUT or IN/OUT in the underlying function or procedure definition; only one RETURN is permitted per procedure definition, and the value returned must resolve to a numeric value.

The names of the parameters (*tag-1, tag-2* ...) must be the same (without case sensitivity) as the names of lines in the input file to which they are potentially to correspond (removing the need for an [attributes] section). Setting preset allows a tag to be defaulted to a particular value if a value is not set in an input record. Note, that the tag “ChangeType” the value of the ChangeType field of the import section is allowed.

Presets must be of a format that corresponds to the nature of the target parameter, e.g.:

- Numeric values (format depending on the precise data-type)
- String values
- Dates

Presets can only be checked against the ruling stored procedure characteristics *after* connection to the database, and not when the configuration file is analysed. Thus, a value that is accepted with a particular stored procedure definition could cause an error return after a modification to the stored procedure.

The tags need not be the same as those (formal parameter names) stored in the stored procedure definition. For example, Oracle9i defines procedure creation, with some simplifications, as:

```
CREATE [OR REPLACE] PROCEDURE procedure-name (arg1-name [ IN | OUT
| IN OUT ] data-type , … ) ( AS | IS ) pl-sql-or-other;
```

but *tag-1* need not be the same as *arg1-name*. The parameter characteristics (data-type, IN/OUT values, etc.) are not specified in the configuration file.

The stored procedure mechanism makes use of a returned value. This is the value provided by the return value of the procedure/function when return-tag is provided. This value is optional, although desirable. If absent, the key word RETURN must prefix the tag corresponding to the return value, which must be:

- Designated as an OUT parameter in the definition of the stored procedure
- Of value mapping to an integer.

The return tag is mapped to a string value as defined by RETURNING. Returned values must be numeric, but can be positive, negative, or zero. The result of the RETURNING process is a string (if specified) defined by range-output (if specified). Evaluation is left to right. The strings have the following semantics:

- No string: OK
- String not preceded by "!" - warning described by the string
- String preceded by "!" - failure described by the string (e.g. "!ERROR!! - review record!"
- Return is out of range - failure.

In both the latter two cases, an error notification is made to the exceptions file. This also applies if no return is made (null response) or if the return is not an integer. In the warning case, the trace file contains a warning message if the appropriate trace-level settings are made.

The returned value of data-type must be numeric, as given in the following table:

Allowed	Not supported
SQL_DECIMAL SQL_NUMERIC SQL_SMALLINT SQL_INTEGER SQL_DOUBLE SQL_TINYINT SQL_REAL SQL_FLOAT	SQL_CHAR SQL_VARCHAR SQL_LONGVARCHAR SQL_BITSQL_BINARY SQL_VARBINARY SQL_LONGVARBINARY SQL_TYPE_DATE SQL_TYPE_TIME SQL_TYPE_TIMESTAMP SQL_BIGINT Interval types

The data-types associated with a particular function or procedure can be determined by the Agent by using the "PC" facility in the configured TraceLevel value.

Dates

Date values (in the configuration file and in input files) are supported in a single configured form selected from one of the following general forms (see method of specification below):

...YYYY...MM...DD
...YYYY...MON...DD

...DD...MM...YYYY
...DD...MON...YYYY

Specific examples are:

"YYYY/MM/DD" which accepts dates like "2003/4/1"

"date-of-birth: DD-MON-YYYY" which accepts dates like "date-of-birth: 1-Apr-2003"

Here:

- YYYY means a sequence of four digits
- DD, MM means a sequence of one or two digits representing day and month;
- MON means one of jan feb etc (the first three letters of the month), case insensitively;
- ... means any string of letters not in %YMD.

The default format is:

YYYY-MM-DD

which permits dates of form "2003-04-02".

In all cases, only the date itself is passed to the database.

Changes to the default may be specified in the [control] section of the configuration file in the following form:

DateFormat=*format*

Where format is one of the form s given above. Values are accepted with the following relaxations:

- Single-digit days and months are accepted
- The value in an input file can take the form of an ODBC escape; for example: {d '1995-01-15'}

Dates support arithmetic in stored procedures, so that if date1 and date2 are dates, date1-date2 is the positive or negative time between them.

3.8.3.2.7. The EncryptedAttributes Section

The EncryptedAttributes section is an optional section that lists attributes which are encrypted in the import data file and have to be decrypted by the agent before they are passed to the ODBC Interface. 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). The attributes are listed in the format:

abbreviation=1

where abbreviation specifies the attributes to be imported.

For example:

[EncryptedAttributes]
Password=1

3.8.3.2.8. The Control Section

The **control** section is an optional section that consists of fields that provide control information that is common to both export and import procedures. The next sections describe these fields.

RecordSeparator

The **RecordSeparator** field specifies the record separator that distinguishes between successive import or export entries. The syntax is

RecordSeparator=*string*

where *string* is a value that can contain a form-feed, expressed as "\f". No other escapes are permitted; backslashes are not permitted other than as a prefix to "f" (not even to "F").

This is an optional field. If it is not specified in the configuration file, the ODBC agents use a default string **row:**.

Trace

The **Trace** field controls whether the ODBC agents perform program flow tracing on an export or import operation. The syntax is:

Trace=[*switch*]

where *switch* is one of the following values:

- **0** - Do not perform program flow tracing on the export or import operation (default)
- **1** - Perform program flow tracing on the export or import operation

If 1 is specified, the ODBC agents write information about the export or import operation to the pathname specified in the **TraceFile** field.

TraceLevel

The **TraceLevel** field controls the amount of program event information that is written to the trace file during import and export operations. The syntax is:

TraceLevel=*trace_string_list*

where *trace_string_list* is one or more of the following strings or abbreviations, separated by whitespace:

- **ConnectAttributes** (**CA**) - include connection "attributes" (fields)
- **FailSummary** (**FS**) - include a summary of failed entries sent to the error file

- **ODBC** - report ODBC versions
- **SQL** - include SQL statements that are to be executed
- **Summary** (**S**) - include a summary of all entries imported or exported (mark failed entries with a trailing # character
- **Warnings** (**W**) - include ODBC warnings (ODBC errors are always written to the trace file)
- **Columns** (**Cols**) - include information on columns stored as part of the database schema
- **RefData** (**Ref**) - include information on the reference data used for delta export
- **Statistics** (**Stats**) - include statistics about the import operation, such as the number of creates, updates, and deletes, the number of entries unprocessed because of errors, the total of all entries handled with or without error but not skipped, and skipped entries
- **AllProcedures (AP)** – provides summary details of all functions and procedures
- **ProcedureCharacteristics (PC)** - provides details of all functions and procedures configured into the configuration file
- **ProcedureReturns (PR)** - provides details of (numerical) procedure returns if available

ConnectAttributes

When *trace_string_list* includes **ConnectAttributes**, the ODBC agents write the connection "attributes" that fully define the connection to the ODBC database to the trace file (and to standard output) in the format they would have as fields in the configuration file (username and password are not included). You can copy the connection attributes from the trace file into the configuration file if you wish to change any of the parameters at agent startup. For example:

```
database connection attributes:
DSN=my_database
DBQ=C:\MSOffice\Access\Samples\Northwind.mdb
DriverId=25
FIL=MS Access
MaxBufferSize=512
PageTimeout=5
```

FailSummary

When *trace_string_list* includes **FailSummary**, the ODBCAgentImp provides a summary of each entry that it fails to process on an import operation, in the format:

entry_introducer attribute_1 / attribute_2 ... line-number-text disposition-text error-sign
diagnostic-event
diagnostic-event

...

Here is an example:

```
record Annette/Dodsworth at line 129 (absent) #

warning: empty import attribute
REG

insertion forbidden
```

ODBC

When *trace_string_list* includes **ODBC**, the ODBC agents write the version of the ODBC subsystem and the driver in use to the trace file. For example:

```
ODBC version info
03.00.0000

ODBC driver version
02.50
```

SQL

When *trace_string_list* includes **SQL**, the ODBC agents write to the trace file the SQL statements that they execute at various stages of the import or export procedure. Because the ODBC agents can only run successfully if the SQL is correct, you can use this information to determine the precise location at which an import or export operation has failed by running the SQL statement in a native SQL environment for the database being accessed.

ODBCAgentExp makes only one SQL call which obtains all matching records sequentially. This is the form of the output:

```
SELECT statement
SELECT Employees.FirstName, Employees.LastName, Employees.Title,
Employees.TitleOfCourtesy, Employees.BirthDate,
Employees.HireDate, Employees.Address, Employees.City,
Employees.Region, Employees.PostalCode, Employees.Country,
Employees.HomePhone, Employees.Extension, Employees.Notes FROM
Employees ;
```

All of the SELECT statement is on a single line.

ODBCAgentImp makes several SQL calls which prepare it for the various import

scenarios it may encounter. The agent performs this preparation phase before it carries out any importing, and must complete this preparation phase successfully. Here is an example:

```
dummy SELECT statement
SELECT Employees.FirstName, Employees.LastName,
Employees.PostalCode, Employees.Title, Employees.TitleOfCourtesy,
Employees.BirthDate, Employees.HireDate, Employees.Address,
Employees.City, Employees.Region, Employees.Country,
Employees.HomePhone, Employees.Extension, Employees.Notes FROM
Employees
SELECT statement
SELECT Employees.FirstName, Employees.LastName, Employees.Title,
Employees.TitleOfCourtesy, Employees.BirthDate,
Employees.HireDate, Employees.Address, Employees.City,
Employees.Region, Employees.PostalCode, Employees.Country,
Employees.HomePhone, Employees.Extension, Employees.Notes FROM
Employees WHERE (Employees.FirstName=?) AND
(Employees.LastName=?)
UPDATE statement
UPDATE Employees SET Employees.Title=?,
Employees.TitleOfCourtesy=?, Employees.BirthDate=?,
Employees.HireDate=?, Employees.Address=?, Employees.City=?,
Employees.Region=?, Employees.PostalCode=?, Employees.Country=?,
Employees.HomePhone=?, Employees.Extension=?, Employees.Notes=?
WHERE (Employees.FirstName=?) AND (Employees.LastName=?)
INSERT statement
INSERT INTO Employees (FirstName, LastName, PostalCode, Title,
TitleOfCourtesy, BirthDate, HireDate, Address, City, Region,
Country, HomePhone, Extension, Notes) VALUES (?, ?, ?, ?, ?, ?,
?, ?, ?, ?, ?, ?, ?, ?)
DELETE statement
DELETE FROM Employees WHERE (Employees.FirstName=?) AND
(Employees.LastName=?)
```

ODBCAgentImp uses the dummy SELECT statement to "prepare" an execute statement; the preparation enables the table characteristics to be extracted. The SQL is never executed.

ODBCAgentImp uses the SELECT statement to test each supplied entry to see how many of its kind there are in the database, using the predicate that follows WHERE, in this case:

```
(Employees.FirstName=?) AND (Employees.LastName=?)
```

This condition would have been the result of a **SelectBy** field of:

```
SelectBy = GN & SN
```

where GN maps to Employees.FirstName and SN maps to Employees.LastName.

The question mark (?) characters are used internally to indicate entry-specific arguments, as supplied. This construct is also used in each of the following SQL statements.

ODBCAgentImp uses the UPDATE statement to modify rows according to the specification of the incoming entry; that is, each of the '?' signs are substituted for with the incoming information.

ODBCAgentImp uses similar statements for INSERT (to add a new row) and DELETE (to remove a row).

Summary

When *trace_string_list* includes **Summary**, ODBCAgentImp provides a summary of each entry processed in the import operation, in the format:

entry_introducer attribute_1 / *attribute_2 ... line-number-text disposition-text* [*error-sign*]

each followed by diagnostic information if necessary. For example:

```
record Nancy/Davolio at line 1 (updated)

record Andrew/Fuller at line 17 (updated)

record Janet/Leverling at line 33 (updated)

record Margaret/Peacock at line 49 (updated)

record Steven/Buchanan at line 65 (updated)

record Michael/Suyama at line 81 (updated)
```

Warnings

When *trace_string_list* includes **Warnings**, the ODBC agents write warning messages into the trace file. The messages have the following format:

- The first line announces a warning.
- The second and third lines provide information generated by the underlying ODBC support. Consult the ODBC Help for explanations of the meaning.

- The fourth line is provides information generated by the ODBC driver manager. This is occasionally useful, particularly in conjunction with the SQL statements included by using the **SQL** trace string.
- The fifth line is occasionally generated, and is based on experience. (The generation is an open-ended capability.)

Here is a sample (benign) warning that occurs on agent startup:

```
connection with information
state=01000
native error=0
error message=[Microsoft][ODBC Driver Manager] The driver doesn't
support the version of ODBC behavior that the application
requested (see SQLSetEnvAttr).
this error is probably irrelevant.
```

Here is a warning generated as the result of misspelling the table-name in an export operation:

```
SQLExecDirect failure
state=42S02
native error=-1305
error message=[Microsoft][ODBC Microsoft Access 97 Driver] The
Microsoft Jet database engine cannot find the input table or
query 'xEmployees'. Make sure it exists and that its name is
spelled correctly.
```

Here is a warning generated as the result of misspelling the table-name in an import operation:

```
SQLDescribeCol failure
state=42S02
native error=-1305
error message=[Microsoft][ODBC Microsoft Access 97 Driver] The
Microsoft Jet database engine cannot find the input table or
query 'xEmployees'. Make sure it exists and that its name is
spelled correctly.
state=07009
native error=53
error message=[Microsoft][ODBC Microsoft Access 97 Driver]Invalid
```

```
column number
```

Columns

When *trace_string_list* contains **Columns**, the ODBC import agent writes information about the columns to which it is about to import. For example:

```
column info
attribute=GN
column-id=FirstName
column-expansion=Employees.FirstName
configured-data-type=SQL_CHAR
C-data-type=SQL_C_CHAR
configured-minimum=1
configured-maximum=10
fail-if-too-big=false
SQL-data-type=SQL_VARCHAR
precision=10
scale=0
nullable=SQL_NULLABLE
column info
attribute=SN
column-id=LastName
column-expansion=Employees.LastName
configured-data-type=SQL_CHAR
C-data-type=SQL_C_CHAR
configured-maximum=20
fail-if-too-big=false
SQL-data-type=SQL_VARCHAR
precision=20
scale=0
nullable=SQL_NULLABLE
etc.
```

In this information:

- attribute gives the abbreviation;
- column-id defines the name of the column within the table;
- column-expansion defines the full column definition as defined in the attributes section;
- configured-data-type and C-data-type give information about the representation of the ODBC data-type within the agent;

- configured-maximum defines the maximum size specified for the column, either by configuration or by taking information from the ODBC database;
- SQL-data-type give information about the native ODBC data-type for the column;
- precision gives (for text information) the field length within the ODBC database; scale is not relevant at present;
- nullable indicates whether the column information is permitted to be absent (TRUE), or whether it must always be present (FALSE).

RefData

When *trace_string_list* includes **RefData**, ODBCAgentExp writes the following information to the trace file:

- When the **Mode** field in the export configuration file is set to **full**, ODBCAgentExp writes a message indicating that full export is to take place
- When the **Mode** field in the export configuration file is set to **delta-or-full** or **delta**, ODBCAgentExp writes:
  - A list of reference files that are currently available (possibly none)
  - The name of the file identified as the reference file to be used in the delta export procedure (possibly none)
  - The information contained in the reference file header (if present)
  - A message indicating that delta export is to take place

If the **-v** parameter has been specified on the command line, ODBCAgentExp also displays this information on the user's console.

ODBCAgentImp ignores the **RefData** option.

Here is an example trace file for a full export when the **RefData** option is used:

```
reference data files:
none
no previous reference file found
full export with reference file:
R9122200
```

Here is an example trace file for a delta export when the **RefData** option is used:

```
reference data files:
R9122200
reference file found:
.\R9122200
reference info:
time-stamp=Wed Dec 22 16:47:42 1999
```

```
record count=17
key data width=92
support-8-bits=TRUE
data hash size=8
selection=Employees.FirstName, Employees.LastName,
Employees.Title, Employees.TitleOfCourtesy, Employees.BirthDate,
Employees.HireDate, Employees.Address, Employees.City,
Employees.Region, Employees.PostalCode, Employees.Country,
Employees.HomePhone, Employees.Extension, Employees.Notes
keys=Employees.FirstName, Employees.LastName,
Employees.HomePhone, Employees.Country, Employees.City
from=Employees
where=null

delta export with new reference file:

R9122201
```

RefData

When *trace_string_list* includes **Stats**, ODBCAgentImp writes statistical information into the trace file. ODBCAgentExp ignores the **Stats** option. Statistics are provided on entries created and updated, entries deleted, entries that could not be processed as the result of an error, the total number of entries processed, and the total number of entries skipped (these entries are not included in the total processed). The number of updated attributes and the number of attributes on which an update was unnecessary are also listed. Items of statistics for which the value is zero are not included. For example:

```
import statistics:
creates: 21
updates: 1105
updates (attr): 995
updates (not necess.)2320
deletes: 112
errors: 6
total imports: 1244
skips: 2
```

ProcedureCharacteristics

When *trace_string_list* contains **ProcedureCharacteristics**, the ODBC import agent writes information about the parameters of the stored procedure to which it is about to import. For example:

```
column info
procedure=test
procedure-cat = ""
procedure-schema = "SCOTT"
procedure-identifier = "SP_BENUTZERROLLE"
parameter-name = "RETURN_VALUE"
parameter-type = SQL_RETURN_VALUE
data-type = SQL_DECIMAL
type-name = "NUMBER"
column-size = 38
buffer-length = 39
decimal-digits = 0
num-prec-radix = 10
nullable = SQL_NULLABLE
remarks = "null"
column-def = "null"
sql-data-type = SQL_DECIMAL
sql-datetime-sub = SQL_LONGVARCHAR
char-octet-length = -1
ordinal-position = 0
is-nullable = "YES"
etc

parameter bind info
parameter-number = 1
parameter-type = SQL_PARAM_OUTPUT
c-data-type = SQL_C_CHAR
sql_type = SQL_DECIMAL
precision = 18
scale = 0
buffer-width = 20
pvd->slip = 0
etc
```

The output gives detailed information about data type mapping between the target database and the ODBC subsystem and the ODBC Agent:

1. The native data-type of procedure parameters (NUMERIC, VARCHAR2 etc).
2. The data-type supported by and visible within the ODBC subsystem (SQL_DECIMAL, SQL_CHAR, etc)
3. The C-data-type into which the ODBC data-type is mapped (SQL_C_FLOAT, etc.).

Column info covers mapping 1. to 2.,bind info 2. to 3.

ProcedureReturns

When *trace_string_list* contains **ProcedureReturns**, the ODBC import agent writes information about the procedure returns if available. For example:

```
procedure return value at line n
result=0.000000
```

AllProcedures

When *trace_string_list* contains **AllProcedures**, the ODBC import agent writes information about all the defined procedures in the target database. For example:

```
procedure-catalog = "mydatabase" +
procedure-schema-identifier = "dbo" +
procedure-identifier = "sp_benutzerrolle;1" +
remarks = "null" +
procedure-type = function
```

This trace level can be used to check the names of the defined procedures to which the agent has access.

This should be used only for test purposes because the access costs a lot of performance if many stored procedures are defined in the target database.

TraceFile

The **TraceFile** field specifies the name of the trace file to which the ODBC agents are to write information about their execution. The syntax is:

TraceFile=*filename*

where *filename* can be a pathname or a file name. When a pathname is specified, the entire path must pre-exist. If *filename* has a suffix, the ODBC agents use this file to write tracing information. If *filename* does not have a suffix (that is, it does not contain a period (.)), the ODBC agents use *filename* as a prefix for a multiple trace file naming scheme of the form:

*filenamennn***.txt**

where *filename* represents the supplied prefix and *nnn* is a three-digit decimal string that starts at 000 and has a maximum value of the **MaxTraceFiles** field minus 1. For example:

imp_trace038.txt

The ODBC agents write multiple trace files up to one less than the maximum specified in the **MaxTraceFiles** field and use the **MaxTraceSize** field to determine when to create a new trace file in the series. You can supply a pathname to specify where the trace files are to be located.

This is an optional field. If it is not specified in the configuration file, the ODBC agents direct program event output to the trace file **trace.txt** in the current working directory.

MaxTraceFiles

The **MaxTraceFiles** field determines the number of trace files that the ODBC agents are permitted to create, in rotation. The syntax is:

MaxTraceFiles=*number*

where *number* is a non-negative integer between 2 and 1000. Each trace file name has a suffix ranging from 000**.txt** (first trace file created) to *nnn**.txt*, where *nnn* is one less than the value of the **MaxTraceFiles** field.

This field is optional. If it is not specified, the ODBC agents use a maximum number of 1000.

MaxTraceFileSize

The **MaxTraceFileSize** field determines the maximum size of a trace file. The syntax is:

MaxTraceFileSize=*size*

where *size* is a value between 1024 and 2^{31}-1. Actual trace files will always be a little smaller than this size (by up to 256 bytes).

The ODBC agents use the value in the **MaxTraceSize** field to determine when to create a new trace file. The change of file to the next in the series occurs when the ODBCAgentImp or ODBCAgentExp determines that the report to be written could cause the size of the trace file to exceed the maximum size set by the **MaxTraceFileSize** field.

TraceFlow

The **TraceFlow** field specifies the level of tracing information written to the trace file. The syntax is:

TraceFlow=*level*

where *level* is an integer from 0 to 9. The higher the number, the more tracing information is written. Currently, only trace level 1 is implemented; at this level, the ODBC agents give an indication of entrance and exit for main functions. For example:

```
...
entering CONVERT()
exiting CONVERT()
entering SQLFetch()
```

```
exiting SQLFetch()
exiting odbc_export()
```

8bit

The **8bit** field controls whether or not ODBCAgentExp accepts characters larger than 7 bits without escaping them to hex notation (**\x***hh*). The syntax is:

8bit=*boolean*

where *boolean* is one of the following values:

- **TRUE** - Accept characters where bit 8 is non-zero
- **FALSE** - Escape characters where bit 8 is non-zero (default)

On export, ODBCAgentExp changes characters in the text that is output, if necessary, to the escaped hex code. This conversion always occurs for non-text characters like NEWLINE. If the **8bit** field is set to TRUE, ODBCAgentExp transmits characters with encodings in the range \xa0 to \xfe unchanged; if set to FALSE, it converts the characters to the hex notation. A backslash \ is encoded as two backslashes \\. On import, ODBCAgentImp reverses the encoding to reproduce the original text without encoding.

This is an optional field. If it is not specified in the configuration file, ODBCAgentExp escapes characters where bit 8 is non-zero. ODBCAgentImp ignores the **8bit** field on import.

DataHash

The **DataHash** field specifies the number of octets that ODBCAgentExp is to use when creating the hash value for an entry's attributes in the delta reference file. The syntax is:

DataHash=*number*

where *number* is a non-negative integer between 4 and 16. The hash values for entry attributes that ODBCAgentExp creates are not guaranteed to be unique for every entry. Therefore, the larger the number of octets used by ODBCAgentExp to create a hash value, the more unlikely it is that ODBCAgentExp will create duplicate hash values for different entries.

This is an optional field. If it is not specified in the configuration file, ODBCAgentExp uses 8 octets to create the hash value for the entry.

3.8.3.3. Configuration File Error Reporting

The ODBC agents attempt to pinpoint the cause of any errors in the configuration file. For example, the output:

```
E:\transfer\run>h:\development\exporter\exporter\ODBCAgentExp -v
-ftransfer.cfg
missing equals sign at line 34
```

```
From ??Employees
```

detects an error in the export section in the configuration file. Where errors occur (as in this case) in simple analysis of the characters, the "??" marker occurs at the point where the error is detected. Where the error requires some context, it will normally follow the offending token. For example, a second **SelectBy** creates this error:

```
duplicate SelectBy setting at line 45
SelectBy=??GN
```

3.8.4. Import and Export Data File Format

The ODBC agent import and export data files use a tagged file format with the following characteristics:

- The only supported encoding is ISO 8859-1.
- Each entry attribute is contained on one line; line continuation is permitted using the backslash (\) as the line continuation character.
- The representation of each attribute is: *attribute_name*: *attribute_value(s)*
- Leading and trailing whitespace between *attribute_name* and *attribute_value* is ignored. For example, in the attribute:

  ```
        SN:   Lowell Jr.
  ```

  the whitespace between the colon (:) and the start of the attribute value is ignored, but the whitespace within the attribute value is returned
- The record (entry) separator is either the default string **row:** or the string defined in the **RecordSeparator** field in the configuration file. For an import data file, the ODBC agent requires that a record separator be present at the end of each entry. However, it does not require the presence of a record separator at the start of the first entry in the file or at the end of the last entry in the file.
- An optional file termination indicator string **end:** can be placed anywhere in an import data file to direct ODBCAgentImp to ignore data that occurs after the **end:** terminator during import processing. The file terminator string is never present in an export data file.
- Comment lines can be placed anywhere in the file and are identified by a # character at the beginning of the line. A non-comment line that would otherwise start with a # character starts instead with \#.
- An optional skip record indicator string **skip:** can be placed in a record to direct ODBCAgentImp to ignore the record during import processing. The skip record string is never present in an export data file.
- The data file format supports a per-entry "changetype" attribute that specifies the type

of modification indicated for the entry in the ODBC database. The value for "changetype" is one of "add", "modify", or "delete". The changetype attribute name and its values are case-insensitive. The attribute name is either **ChangeType** or the name specified in the **ChangeType** field.

- The export data file format permits <CR> or <LF> to be used in attribute values.
- Boolean attribute values are represented as **0** (for FALSE) and **1** (for TRUE).
- Characters in import data file entries can be specified by their hex value in the format **\x***hh*.
- Backslash characters in import data file entries must be specified as \\.

Here is an example (CR LF is represented as \x0d\x0a in line eight):

```
row:
GN:  Nancy
SN:  Davolio
T:   Sales Representative
TOC:  Ms.
BD:  1948-12-08 00:00:00
HD:  1992-05-01 00:00:00
A:   507 - 20th Ave. E.\x0d\x0aApt. 2A
CITY:  Seattle
REG:  WA
PC:  98122
C:   USA
TEL:  (206) 555-9857
EXT:  4109
DESC:  Education includes a BA in psychology from Colorado State
University in 1970. She also completed "The Art of the Cold Call."
Nancy is a member of Toastmasters International.

row:
...
```

3.8.5. Import Error File Format

An import error file log entry generated by ODBCAgentImp on a failed import of an entry has the following format:

#*entry_identifier*
source_entry
#*error_messages*
#

where *entry_identifier* specifies the line number in the import data file at which the failed entry (record) exists, *source_entry* is the original entry that ODBCAgentImp was unable to import, and *error_messages* describe the error. Here is an example of an error log for an import operation in which the configuration file sets **InsertOnly** to TRUE:

```
# record at line 17
row:
GN: Nancy
SN: Davolio
T: Sales Representative
TOC: Ms.
BD: 1948-12-08 00:00:00
HD: 1992-05-01 00:00:00
A: 507 - 20th Ave. E.\x0d\x0aApt. 2A
CITY: Seattle
REG: WA
PC: 98122
C: USA
TEL: (206) 555-9857
EXT: 5467
DESC: Education includes a BA in Psychology from Colorado State
University in 1970. She also completed "The Art of the Cold Call."
Nancy is a member of Toastmasters International.

# error: row already exists
#
```

If the entry itself is invalid, ODBCAgentImp records it as a comment. For example, in the following entry, "Surname" was used in place of the abbreviation "SN":

```
# record at line 1
row:
GN: Nancy
Surname: Davolio
T: Sales Representative
TOC: Ms.
BD: 1948-12-08 00:00:00
HD: 1992-05-01 00:00:00
A: 507 - 20th Ave. E.\x0d\x0aApt.
CITY: Seattle
REG: WA
```

```
PC: 98122
C: USA
TEL: (206) 555-9857
EXT: 5467
DESC: Education includes a BA in Psychology

# warning: unrecognized attribute in import data file
# Surname
#
# unset naming attributes
# SN
```

The text for the failed entry is always as supplied; that is, no attempt is made to correct it. To use the entries, manual correction will probably be necessary to remove the problem.

ODBCAgentImp places a condensed version of this error message in the trace file (and also to the display if verbose mode was specified on the command line):

```
record Nancy at line 1 (error) #

warning: unrecognized attribute in import data file
Surname

unset naming attributes
SN
```

3.8.6. Import Procedure

Import takes place using data from an import data file. This file has the format described in "Import and Export Data File Format" and comprises a series of records, each stored on separate lines and consisting of an introducer string and a set of attribute lines. By default, the introducer string is a line containing **row:**. However, you can specify a different value for the introducer (for example, a form-feed) using the **RecordSeparator** field in the Control Section of the configuration file.

Each attribute is one of the following:

- A naming attribute (identified as on the **SelectBy** attribute list, and never modified);
- An attribute to be modified (identified by the **Modify** attribute list);
- An attribute that is ignored except when a new row (record) is to be created (other attributes listed in the **attributes** block);
- All other attributes - ignored (present in the import entry (record), but not specified in

the **attributes** block).

In addition, the quasi-attribute **ChangeType** or an attribute substituted for it using the **ChangeType** field in the Import section of the configuration file can be used to modify the behavior for the entry within which the attribute is placed.

ODBCAgentImp does not handle multi-valued attributes; attributes in the first three categories are not permitted to occur twice in a single import entry. If the agent encounters a multi-valued attribute, it discards the second and subsequent values creates a warning message in the error file.

The introducer **end:** may be used where **row:** would otherwise be used, and terminates the process.

The introducer **skip:** may be used anywhere following a **row:** introducer, and causes the complete record information to be ignored. It can be used to blank off a record which requires special attention. The record information does not appear in the error file, and is not reported upon in the trace file.

For each ODBC entry, ODBCAgentImp does the following:

- Checks the entry for compliance with the constraints specified for the attributes (or defined by the database). If constraints are broken for non-naming attributes and the attribute is not marked with the "fail-if-too-big" flag, the agent truncates the values (or pads them), and continues its processing. If constraints are broken for naming attributes or for any attribute when the "fail-if-too-big" flag is TRUE, the agent discards the entire entry and writes an explanatory message into the error file (and, if required, to the trace file).
- Using the naming attributes, SELECTs a row (record) that matches the resulting predicate in the specified table or join of tables. *All naming attributes used in the predicate must be present in the import entry.*
- If zero rows are found, and either **ChangeType** is set to **add**, or **CreateIfAbsent** is **TRUE** or **InsertOnly** is **TRUE**, creates a new entry that contains all the relevant attributes. If **ChangeType** is absent or is not set to **add**, or if both **CreateIfAbsent** and **InsertOnly** are **FALSE**, discards the entry and writes an explanatory message into the error file.
- If a join of tables is specified, and a new entry is to be created, this creates a new row in the primary table only (that is, the table first mentioned in the JOIN specification). Attributes matching rows in tables other than the primary table are ignored.
- If just one row is found, and **ChangeType** is set to **delete**, removes the row (just the row in the primary table, if a join of tables is selected). If **ChangeType** is set to **add** or **InsertOnly** is set, discards the entry and writes an explanatory message to the error file. Otherwise, modifies the attributes specified or removes them (using a blank line after the attribute); in both cases, attributes to be modified must be defined within the **Modify** list; others are ignored for modification.

The database can refuse to remove an entry if it detects that referential integrity would be broken. This means that a row which is pointed to by another row in the same or a different table cannot be removed by ODBC (or even by SQL tools in general). The referencing components

must be removed first, or the reference must be nullified. To make this happen automatically, you can use the **Relationships** control to specify where referential integrity enforcement is configured.

- If multiple rows are found, discards the entry and writes an explanatory message to the error file.
- If the **-s** parameter has been specified on the command line, ODBCAgentImp imports one row at a time, displaying an invitation to continue after each import. Possible inputs are:

q<CR> or **Q<CR>** or **n<CR>** or **N<CR>** - terminates the import procedure (case-insensitive)

g<CR> or **G<CR>** - terminates single-step mode and continues the import procedure (case-insensitive)

<CR> - continues with the next row

3.8.7. Export Procedure

ODBCAgentExp transcribes all or selected rows (records) from a table of an ODBC-accessible database to an export data file. The column information is either the stored value, or may be derived from expressions within the SQL language.

In the **export** section, the **Select** field defines attribute abbreviations that map exactly to those used in SELECT statements. The **From** field specifies the tables or combination of tables. These two fields normally need to be designed together.

The attribute abbreviations must be specified in the **attributes** section. Table names can be omitted in the specification, but it is recommended that table names are always included when the **From** table is a union of tables.

Here is an example of the relationship between the attribute definitions in the **attributes** section and the **From** and **Select** fields in the **export** section.

```
[attributes]
GN=Employees.FirstName
SN=Employees.LastName
PC=Employees.PostalCode
T=Employees.Title
TOC=Employees.TitleOfCourtesy
BD=Employees.BirthDate
HD=Employees.HireDate
.
.
.
[export]
Select=GN, SN, T, TOC, BD, HD, A, CITY, REG, PC, C, TEL, EXT, DESC
```

```
From=Employees

...
```

If the **-s** parameter has been specified on the command line, ODBCAgentExp exports one row at a time, displaying an invitation to continue after each row is exported. Possible inputs are:

q<CR> or **Q<CR>** or **n<CR>** or **N<CR>** - terminates the export procedure (case-insensitive)

g<CR> or **G<CR>** - terminates single-step mode and continues the export procedure (case-insensitive)

[*any_other input**]<CR>* - continues with the next row

If the **-v** parameter has been specified on the command line and **RefData** has been specified as an option to the **TraceLevel** field in the export configuration file, ODBCAgentExp displays a message indicating that the export procedure is to take place.

3.8.8. Delta Export Procedure

The ODBCAgentExp delta export procedure exports only those entries that have changed since the last export operation using a delta reference file mechanism to determine which entries have changed. This section describes:

- The delta export process that ODBCAgentExp follows
- The configuration file fields and command line parameters that you can use to configure and control the delta export process

3.8.8.1. ODBCAgentExp Delta Export Process

A delta reference file is a snapshot of the ODBC database contents. It is a binary file that contains a header and a sorted array of elements that represents the entries in the ODBC database. Each element in the array contains:

- Key values that uniquely identify the record (entry) within the database
- A hash value of the contents of the entry (its attribute values)
- Any key values that are necessary to uniquely identify the entry to the target database during the directory synchronization process

ODBCAgentExp creates an initial reference file on the first full export. On a subsequent delta export operation, ODBCAgentExp performs the following steps:

- Exports the information from the ODBC database into a temporary export data file
- Creates a temporary reference file that represents the temporary export data file contents in condensed form
- Sorts the temporary reference file (according to default ordering criteria or according to the criteria specified by the **SortControl** field in the export configuration file)

- Evaluates the temporary reference file against the reference file it previously created and builds the delta export data file as follows:
  - Each entry that is present in both reference files and for which the hash is the same is ignored
  - Each entry that was absent from the previous reference file but which is present in the temporary reference file is exported as an "add", sending the complete set of data
  - Each entry that was present in the previous reference file but which is absent in the temporary reference file is exported as a "remove", specifying only its identity using the key information in the reference file
  - Each entry that is present in both reference files, but for which the hash has changed in the temporary reference file, is exported as a "modify"
- Creates a new reference file from the temporary reference file, retains the "old" (previous) reference file, and removes all temporary files

For modified entries, ODBCAgentExp writes only the new values of attributes to the delta export data file. If the value of an attribute specified in the **Key** field has changed, ODBCAgentExp treats this change as the deletion of the entry and the creation of a new entry. Consequently, the delta export data file contains a "delete" changetype entry and an "add" changetype entry.

For deleted entries, ODBCAgentExp writes only the values of the attributes specified in the **Key** field and the **SaveAttr** field to the delta export data file. Values of attributes that are not specified in either the **Key** field or the **SaveAttr** field are not written to the delta export data file.

ODBCAgentExp names reference files in the format:

R*ymmddnn*

where:

- *y* represents the last digit of the year; for example, 9 for 1999
- *mm* represents the month (01 through 12)
- *dd* represents the day (01 through 31, calculated using GMT, not local time)
- *nn* is a sequence number that ODBCAgent calculates for the file

For example:

R9061015

When creating a new reference file based on a previous reference file that has a sequence number, ODBCAgentExp calculates the sequence number for the new file as follows:

- If the day (*dd*) of new and previous reference files are the same, it increments the sequence number by one (*nn*+1) for the new reference file, to a maximum of 99 for the previous reference file. If a new reference file is then needed, the sequence number re-starts at 00, and the day is incremented (taking into account all the usual month-length

and leap-year rules).

- If the day (*dd*) of the previous reference file is older, it assigns the sequence number **00** to the new reference file (any previous file of this name is overwritten)
- If the day (*dd*) of the previous reference file is newer (which could possibly occur around midnight GMT but otherwise indicates a serious problem), it uses the day of the previous reference file for the new reference file, and assigns it the sequence number of the previous file incremented by 1. This provision ensures that the agent always has a monotonic view of time, and that clock adjustments do not confuse the mechanism.

When creating a new reference file based on a previous reference file that does not have a sequence number (for example, because a reference filename that does not use the agent's naming format has been explicitly specified on the command line), ODBCAgent uses the next sequence number above the largest for the day, and uses **00** if no reference files are available (any previous file of this name is overwritten).

ODBCAgentExp interprets the year (*y*) in **R***y***0101***nn* as the last year or next year relative to **R***zmmddnn* depending on how close *y* and *z* are. For example, if *y* is 8, it is considered to be before *z*=9, 0, 1, 2 (the **1** cells with vertical stripes in the column 8 in the table below) but after *z*=3,4,5,6,7 (the -**1** cells with horizontal stripes in column 8):

Year 1 Year 2	0	1	2	3	4	5	6	7	8	9
0	0	-1	-1	-1	-1	-1	1	1	1	1
1	1	0	1	1	1	1	1	1	1	1
2	1	1	0	-1	-1	-1	-1	-1	1	1
3	1	1	1	0	1	1	1	1	1	1
4	1	1	1	1	0	-1	-1	-1	-1	-1
5	1	1	1	1	1	0	1	1	1	1
6	-1	1	1	1	1	1	0	-1	-1	-1
7	1	1	1	1	1	1	1	0	1	1
8	-1	-1	-1	1	1	1	1	1	0	-1
9	1	1	1	1	1	1	1	1	1	0

ODBCAgentExp creates delta reference files in the current working directory unless the configuration file specifies otherwise.

3.8.8.2. Configuration File Fields and Command Line Parameters for Delta Export

The following fields in the export configuration file are relevant to the delta export procedure:

- The **Mode** field in the Export section - use this field to select the delta export operation

- The **Keys** field in the Export section - use this field to establish the keys that ODBCAgentExp is to use for uniquely identifying each ODBC record (entry)
- The **SaveAttr** field in the Export section - use this field to establish the attributes required for directory synchronization that ODBCAgentExp should store in the reference file
- The **ReferencePath** field in the Export section - use this field to specify the directory in which ODBCAgentExp is to create delta reference files
- The **SortControl** field in the Export section - use this field to set up a specific ordering criteria that ODBCAgentExp is to use in place of its default ordering scheme when sorting the elements in the reference file
- The **DataHash** field in the Control section - use this field to specify the length of the hash value that ODBCAgentExp is to use when hashing a record (entry) attribute values for the reference file

See "Configuration File Format" for further details about these fields.

Use the **-r** parameter on the ODBCAgentExp command line to generate a full export data file and the initial reference file; the **-r** parameter overrides the delta operation specification in the **Mode** field of the export configuration file, but generates a reference file (if the **Mode** field is set to **full**, no reference file is generated). See the section "ODBCAgentExp Command Line Format" for more information about ODBCAgentExp command line parameters.

3.9. SAP ERP HR Agent

SAPAgent is the DirX Identity agent that handles the export of SAP ERP Human Resources (HR) Personnel Administration (PA) entries from a SAP ERP database for import into the meta directory store. Arbitrary objects of the Organizational Management (OM) component can also be exported.

SAPAgent is implemented as an ERP application (in ABAP) that can be executed in the SAP GUI.

SAPAgent Unicode supports ERP 6.0 HCM (Human Capital Management) and higher and runs on all SAP NetWeaver server (ABAP stack) platforms. It is not important whether the system is configured for Unicode or not.

SAPAgent can:

- Create one or more SAPAgent transfer configurations; a SAPAgent configuration describes a full or delta export of selected SAP ERP Personnel table entries or selected SAP ERP OM objects
- Allows detailed multiple selection of objects and employees by using the SAP standard ‘logical databases’ PCH’s and PNP’s selection screens
- Include records about new employees or modifications for specific time periods into the future in a full or delta export
- Be enhanced with customer exits for attribute computation or for the exclusion of

persons or OM objects

- Transport SAPAgent configurations to a production system for subsequent activation and scheduling
- Schedule SAPAgent configurations for immediate execution or for execution via the ERP job scheduler
- Generate log files of SAPAgent operations and any errors that occur

SAP ERP differentiates between customization, test and production phases. These phases are generally dedicated to separate ERP systems. Consequently, working with SAPAgent consists of the following phases:

1. Customizing, usually performed on an ERP development system using the ERP Implementation Guide (IMG)
2. Customizing transport to the production system
3. Activation on the production system
4. Background (or manual) execution through the ERP job scheduler

The following figure illustrates the SAPAgent components.

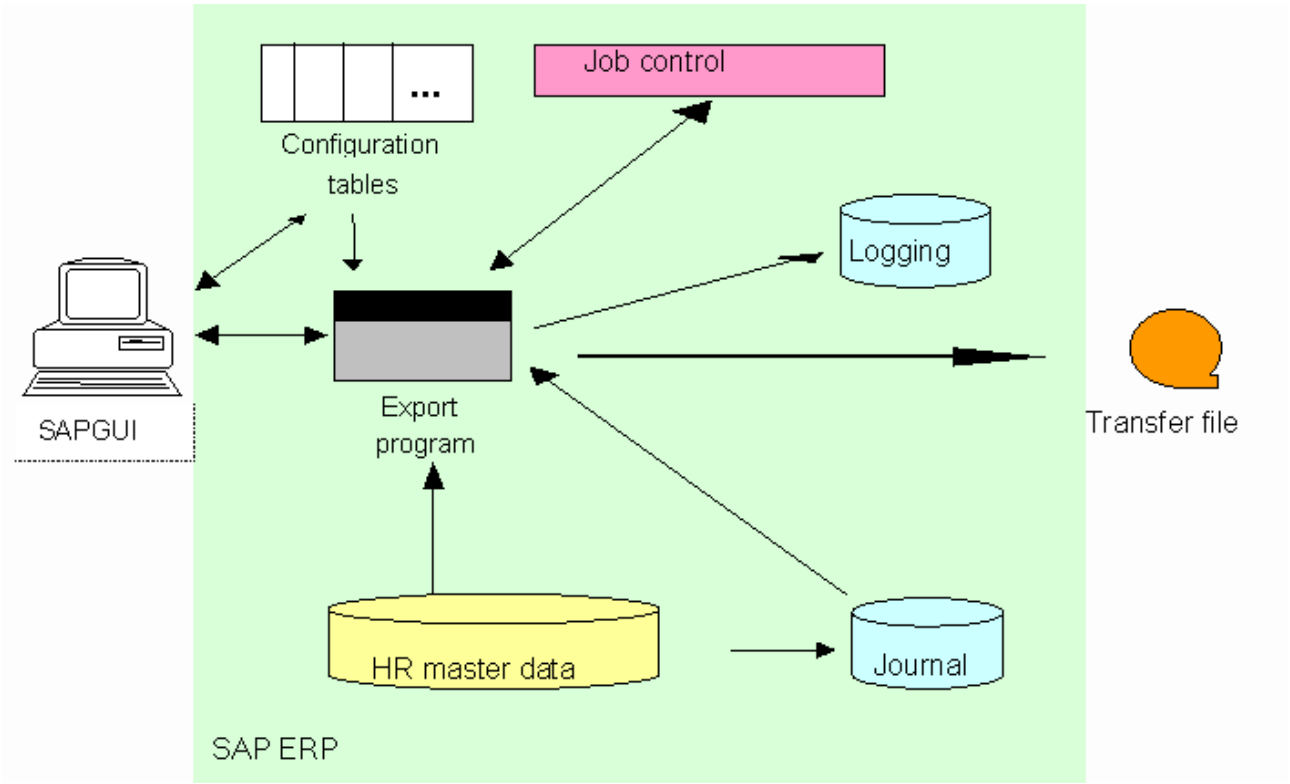


Figure 17. SAPAgent Components

This section describes:

- SAPAgent installation
- SAPAgent predefined roles
- SAPAgent command format

- SAPAgent transfer configuration
- Transport of SAPAgent configurations from customizing to production
- Configuration activation and immediate (ad-hoc) execution
- SAPAgent job scheduling
- The SAPAgent export procedures, security features, and customer exits
- The export data file formats that SAPAgent generates
- SAPAgent logging

SAPAgent provides a base configuration with a set of default values; see the sections "Configuration" and "Default Configuration" for an explanation of the base configuration and the default configuration values.

3.9.1. SAP ERP HR Agent Prerequisites

Before you can use the SAP ERP HR agent, you must extend the DirX Identity Store schema with SAP ERP HR target system-specific attributes and object classes so that the agent can store SAP ERP HR-specific information in the Identity Store. For instructions, see the section "Extending the Schema for the Target System Workflows" in the *DirX Identity Application Development Guide*.

3.9.2. Installing the SAP ERP HR Agent

This section describes how to install the SAP ERP HR agent (SAPAgent). It provides hints for the integration of test and production systems and a checklist for SAPAgent installation. The installation procedure uses the SAP ERP transport system, so you should be familiar with this component.

3.9.2.1. SAPAgent Installation Checklist

This checklist consists of three sections:

- Preparing the installation (before importing the application files) - describes how to prepare your system environment before running the actual installation.
- Importing the application files - describes how to install the application files into your ERP system.
- Completing the installation (after importing the application files) - describes how to complete the installation after the application files have been installed on the target ERP system. After this step, the application is ready for use on the target system. Once you have familiarized yourself with the application, you can use the **/sie/dirx_ag** transaction code to customize the application to your requirements.

3.9.2.2. Preparing the Installation (before Importing the Application Files)

Installing SAPAgent is a straightforward process and follows the rules set by SAP for the installation of ERP third party products.

3.9.2.2.1. Checking the ERP System

The ERP system in which the application is to be installed must have a release level of 6.0 or higher. The installation procedure assumes that the clients 000 and 001 are at least similar to the configuration that is delivered by SAP. The installation procedure also assumes that the Human Resources (HR) component in the client into which the application is to be installed is configured according to your needs.

3.9.2.2.2. Checking the Name Space

To prevent conflicts with other applications, you must check to make sure that the name space required by SAPAgent (**/sie/dirx**...) is not already in use. This should not normally be the case, because the name space is reserved for this application.

Use the repository browser to check the namespace in the ABAP workbench. If the name space contains objects, do not proceed with the installation until you have determined the source and purpose of these objects.

The installation creates an authorization class "YSDX". Check to make sure that it is not already in use. If it is, do not proceed with the installation, and contact your supplier.

3.9.3. Backing up the System

Before you import the application into the system, you should perform a backup of the full system so that you can restore the information in the event that problems occur during the installation.

3.9.3.1. Importing the Application Files

In order to import the SAPAgent, you must be familiar with the SAP transport management system (TMS) component.

Copy the files from the installation media to the subdirectories **trans\data** and **trans\cofiles** in the transport directory of the ERP system. Ensure that the copied subdirectories and files are read- and write-enabled for the SAP user (files copied from CD-ROM or DVD are usually read-only).

The transport consists of objects of the following categories:

- Workbench (consisting of a K... and an R... file)
- Customizing (consisting of a K... and an R... file)

3.9.3.1.1. Import Workbench

The Workbench import transports are indispensable for the installation and include the program logic and data structures.

3.9.3.1.2. Import Customizing

The Customizing import transports are only required for a new installation. If customizing

has already been performed, you should omit this step.

Importing the customizing transport deletes all previously created configurations.

3.9.3.1.3. Executing the Import

The imports must be performed in the following sequence:

- Workbench imports
- Customizing imports

Note: The workbench import must be performed once for a system. The customizing import must be performed for the actual client or clients that use the agent.

To import each of the transports:

- Log in to the ERP system as administrator.
- Start the transport management system (Transaction **STMS**).
- Change to the import queue overview (Menu **Overview** – **Imports** (or **F5**)).
- Select an import queue (double-click).
- Change to import insertion (Menu **Extras** – **Other requests** – **Add**)
- In the pop-up panel, enter the name of the transport to be added. This name is derived from the K... filename by the extension suffixed by the filename without the extension (e.g., the file **K900124.D13** indicates transport **D13K900124**).
- Select **Yes** in the pop-up **Add transport request**.
- The new request should now appear in the list of the requests of the import queue. Select it.
- Start the import (Menu **Request** – **Import** (or **Ctrl F11**)).
- In the pop-up **Import Transport Request**, specify the client into which the transport is to be imported; in the **Options** tab, select the two **Overwrite...** options. Verify that the import type is **synchronous**, then click **OK**.
- In the pop-up **Start Import**, click **Yes**.

After the transport has been imported, check the result code. Depending on the environment, a return code of **0** or **4** is okay. Other return codes (e.g., **8** or **12**) indicate severe problems – the installation is not successful.

3.9.3.2. Finishing the Installation (after Importing the Application Files)

While performing the following procedures, the system asks you for a transport request; you can use transaction SE10 to generate one.

3.9.3.2.1. Maintaining Users

It is recommended that you create new users for the maintenance and execution of

SAPAgent and that you maintain these users with the necessary authorizations. The user selected for SAPAgent execution also must be authorized to read the HR master data, to write into the file system of the application server and to execute external applications on the application server (if enabled during customization).

In order to simplify this process two pre-defined roles are included in the customizing data of the SAPAgent. See below for more information about roles.

Your authorization may not be sufficient to maintain the SAPAgent tables. In this case, consult the chapter 'management of authorizations' in the 'IMG' manual.

3.9.3.3. Checking the Installation

- From the menu, select Tools-→ ABAP Workbench -→ Repository Browser.
- Select **/sie/dirx_ag** in the "development class" field.

A list of several items should appear.

3.9.3.4. Testing the Installation

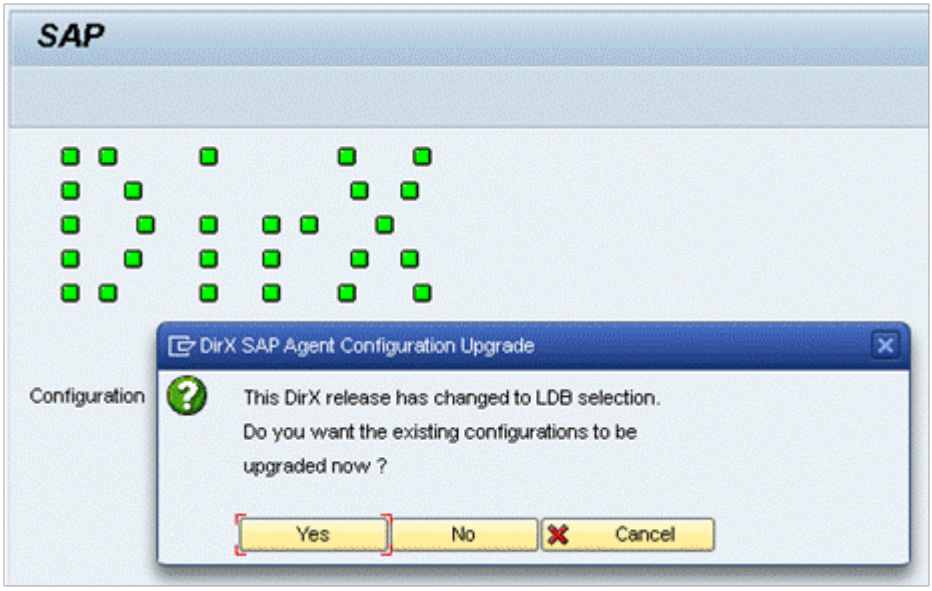
Select the transaction code **/sie/dirx_ag**. (You can include the transaction code **/sie/dirx_ag** into your start menu by selecting System → User profile → Own data → Profiles.

Test some menu entries.

The installation procedure installs the standard configurations "sample configuration 1 (PA)" and "sample configuration 2 (OM)", which do not require any customizing.

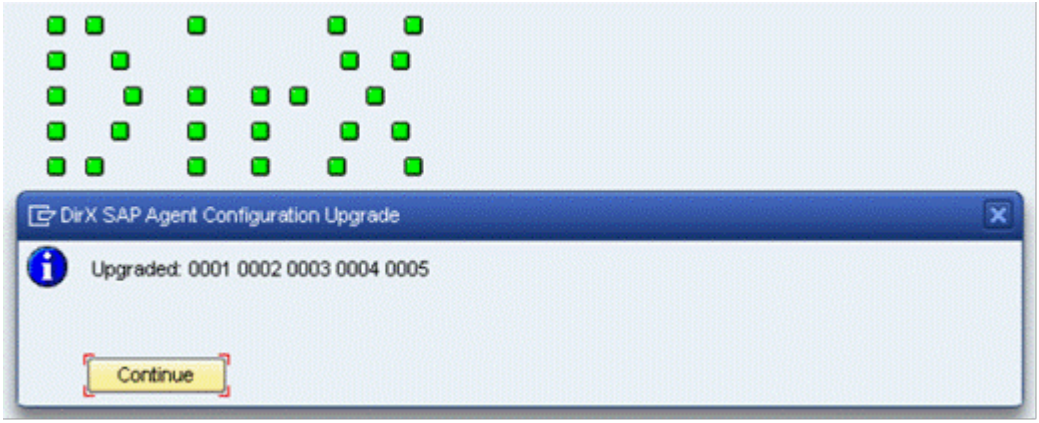
3.9.3.5. Upgrading Existing Configurations

At the initial start of an upgraded release of the agent, existing configurations must be updated to the new release 5 format which is based on the LDBs ("Logical DataBase") for PA and OM.



The upgrade can be made at any time, the release 5 agent terminates if the upgrade is not yet executed (Button **No** or **Cancel**).

After an upgrade has been performed (by choosing **Yes**) all existing configurations (including the sample configurations) are upgraded to LDB-based selection parameters. A list of upgraded configuration Ids is shown:



The SAPAgent is now ready to be used on the development system. The next section explains how to transport the customizing from the development system to a test system or the production system.

3.9.3.6. Initializing the Application

- Run **Scheduling** → **Synchronize DirX Conf's to ERP Jobs**. This action inserts or modifies a job into the ERP job management for every configuration.

3.9.3.7. Hints for Integrating Test and Production Systems

You need to create a "customizing order" to be able to bring ("transport") your customizing information from the development system to the test system and to the production system. From time to time during customizing, the system will prompt you to enter (or select) this order.

To install the customized application on your test system:

- Release and export the customizing order on the development system
- Import the application files from the installation media to the test system
- Import the customizing order from the development system to the test system

Follow the same procedure to install the customized application on the production system.

3.9.3.8. Transferring SAPAgent Configurations to another ERP System

The SAPAgent must be installed (use the installation procedure just described) on the target system

- On the source system, start the application and select **Extras** → **Transport Customizing**. If a transport request has not already been specified, the system prompts for one. Next,

execute the process to fill the transport.

- Release the transport (SE10); this action automatically exports the files to the operating system.
- Copy the files to the target system.
- Import the files (using **tp** or STMS)
- Open **Scheduling** → **Synchronize DirX Conf's to ERP Jobs** and run Execute synchronization. Check this item and uncheck others then click on execute (**F8**). This action inserts or modifies a job into the ERP job management for every configuration.

The target system is now ready to use. You can check the application's log file or the ERP Job management (SM37) to verify that the configuration is correct.

3.9.3.9. Upgrading the Installation

New SAPAgent versions can generally be installed over existing versions. See the ReadMe of the new version for further details.

In most cases, only the workbench transport must be installed (this transport includes the program and data dictionary definitions). The customizing transport contains the default configurations.

If you install the customizing transport again, any previous configurations are lost.

In version 5 the internal format of configurations has changed. At the first start of the new agent a dialog offers to migrate existing configurations.

3.9.3.10. Uninstalling SAPAgent

There is no common uninstallation procedure for ERP applications, nor is there one for SAPAgent. To delete SAPAgent from an installation, you must delete the development class **/sie/dirxag** from the system. (Note: developer rights and a developer key are required for this task.)

3.9.4. Predefined Roles

The customizing data of the agent provides the following two pre-defined roles that contain the necessary authorizations for a batch and an administrator role:

- **Batch role**
  The role /SIE/DIRX_HR_AGENT_BATCH contains authorizations for a system user (user type 'system') under which a background job can be scheduled.
- **Administrator role**
  An administrator who configures and executes the SAPAgent requires the role /SIE/DIRX_HR_AGENT_ADMIN.

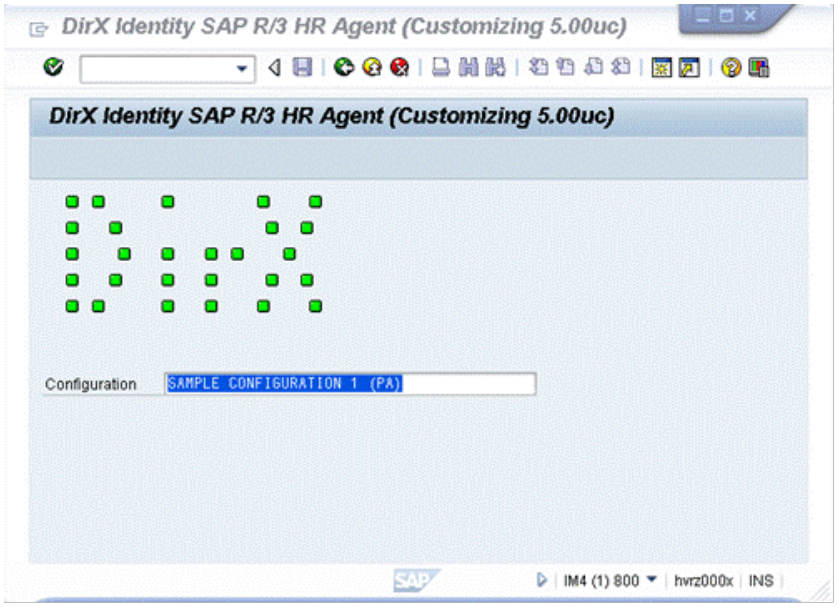
Roles are assigned to users in the SAP user administration (transaction SU01).

3.9.5. Command Format

SAPAgent is an ERP application that is integrated into the SAP GUI as a graphical user interface. To run SAPAgent:

1. Log on to the ERP server (It is recommended that you select English (EN) as the dialog language.)
2. In the ERP SAP GUI browser command line, type the transaction code **/sie/dirx_ag** (or **/n/sie/dirx_ag**), and then press **Enter**.

SAPAgent runs and displays its main window.



The title bar indicates the version of the agent and whether SAPAgent is running on a customizing or a production client. The name suffix "uc" indicates Unicode enabled releases. In general, you run SAPAgent on a customizing client to configure a transfer and transport it to the production client. You run SAPAgent on a production client to activate the transfer for ERP job scheduler execution (or for immediate execution without ERP scheduling).

To exit SAPAgent, click the back icon or press function key **F3**.

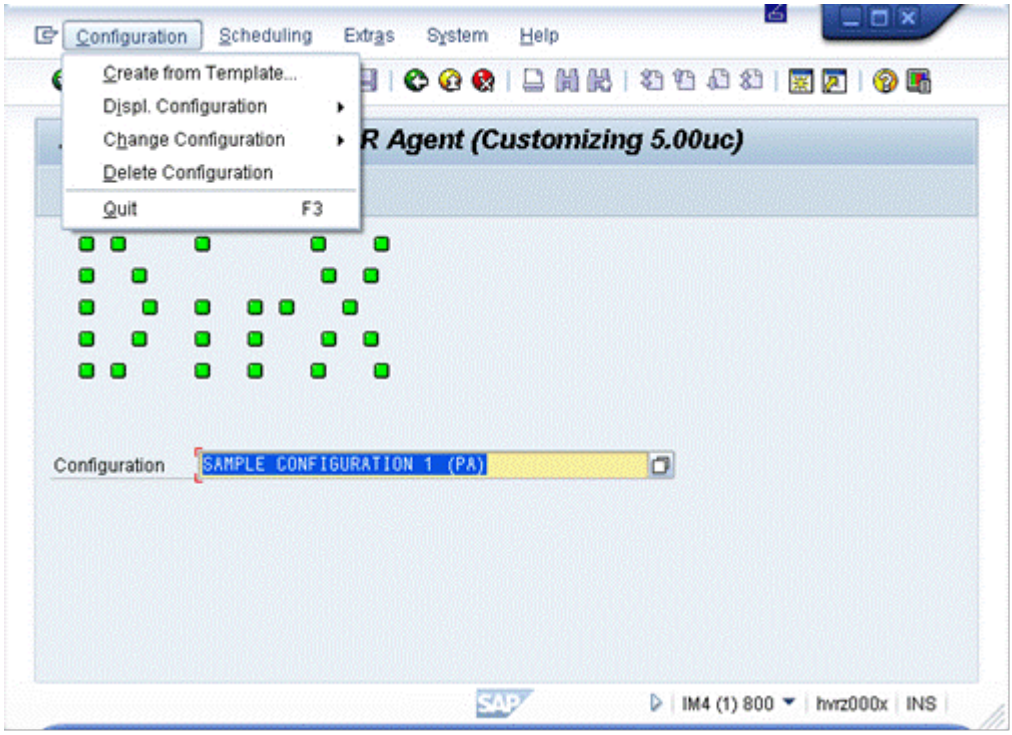
3.9.6. Configuration

Each SAPAgent transfer is described by a configuration. SAPAgent can manage multiple configurations. A configuration can be active (entered into the ERP job scheduler) or inactive.

The SAPAgent installation procedure creates two base (default) configurations (PA and OM). New configurations are inherited from one of these base configurations. The base configurations cannot be deleted. SAPAgent allows you to select a configuration out of a list of available configurations. All changes are then applied to this configuration. To select a configuration in the SAPAgent main window, use the selection button or press **F4**.

Use the **Configuration** menu selection in the main SAPAgent window to:

- Create a new configuration from a template configuration
- Display, edit and delete configurations
- Quit the agent



To create a new configuration from a template configuration:

1. Select an existing configuration or use the base configuration.
2. From the **Configuration** menu, select **Create from Template**. The New Configuration dialog box appears.

3. In **New Configuration**, enter the name of the new configuration, and then click **OK**. SAPAgent copies the existing configuration into the new configuration.

You use the **Change Configuration** menu selection to select the Human Resources data to be applied to the configuration. You select this data in two dimensions:

- Selection of persons or OM objects, called vertical selection
- Selection of person attributes or OM objects attributes, called horizontal selection

You also use the **Change Configuration** menu selection to define a job for a configuration.

This section also describes:

- Changing the Configuration
- The default configuration
- Transport from customizing to production
- Configuration activation and immediate (ad-hoc) execution

3.9.6.1. Vertical Selection (PA)

You use the SAPAgent vertical selection dialog to determine the persons that are to be exported from the SAP/ERP database. To display the vertical selection dialog:

1. Click **Configuration**.
2. Select **Change Configuration**, and then select **Attributes for PA**.

SAPAgent displays the vertical selection dialog. The dialog box contains four areas for vertical selection:

- **Multiple Selection including Period and Selection** - this section enables you to specify selection criteria which the agent then uses to determine which set of data is read. These selection "screens" are the standard SAP report selection screens used in HR management. It contains a series of selection fields and buttons which are described in the SAP manuals. For further information see http://help.sap.com/erp2005_ehp_06/helpdata/EN/e5/7c3438fd263402e10000009b38f8cf/frameset.htm
  The Selection part enables you to enter a single value or value range for each selection field, which you can then restrict still further using the selection options.

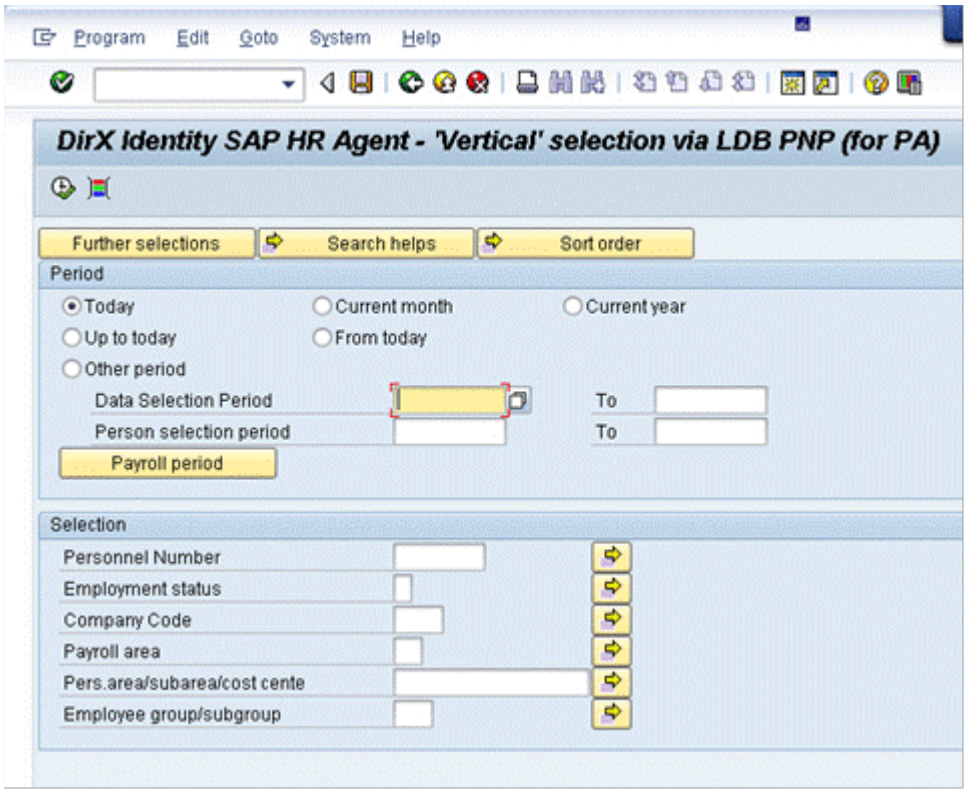
  Multiple selection also enables you to enter

  - Several single values or value ranges to be taken into account when the export is executed.
  - Several single values or value ranges to be excluded when the export is executed.

    You can also use a selection option for each value and range; for example, greater or less than a single value, and within or not within a range.
- **Other attributes** - defines the type of export to be performed (full or delta) and defines export data file format.
- **Job** - a job definition provides information about the configuration for the ERP job scheduler. You use the Job area of the vertical selection dialog box to define a job.

3.9.6.1.1. The Multiple Selection Area

The following figure shows the Selections area of the vertical selections dialog box.



Each selection criteria field is displayed on a separate line. Click in any selection criteria field (or press **F4**) to display a list of the entries that are allowed for the field.

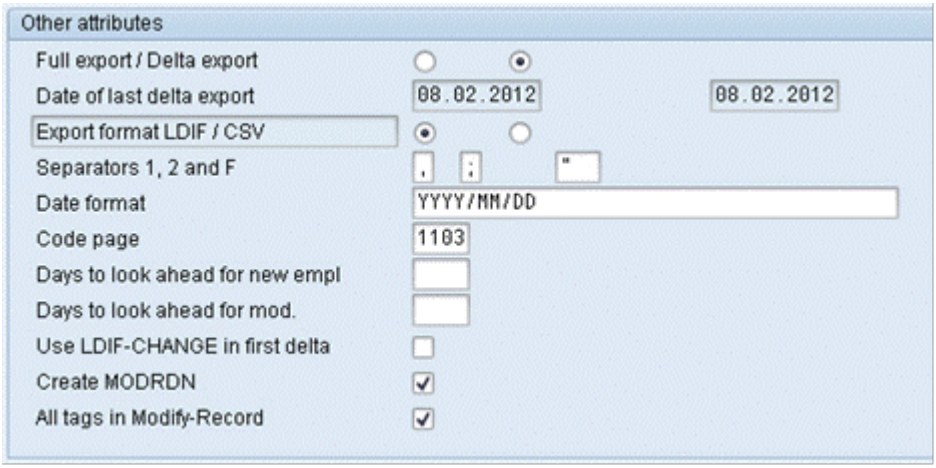
Click on the **Further selections** button to add more selection criteria. The Personnel number field is a mandatory selection field, and you must always supply a range. Selection criteria fields that have no values entered in the lower portion of a range are not used for limiting the selection.

In the **Period** section only **Today** is valid. Other date or range selections are **undefined**.

Fields with a zero value are left blank; this is an ERP issue.

3.9.6.1.2. The Other Attributes Area

The following figure shows the Other attributes area of the vertical selections dialog.



The next sections describe the fields of the Other attributes area.

Full Export / Delta Export

This field selects whether the configuration is a full export or a delta export of SAP/ERP person entries. Selecting this field marks the configuration for exporting all data within the vertical selection regardless of the type of export performed in the last export.

If you do not select this field, the configuration is a delta export. A delta export only selects the data that has changed since the last export. The first export after the creation or change of a configuration is always a full export.

This is a mandatory field for a full export.

Date of Last Delta Export

This field is an informational field that displays the date of the last and first delta export, or the last and first full export. SAPAgent automatically generates this field. Entering a different date into this field overwrites the internal date so that you can generate a delta export for a longer or shorter time period. Do not enter a date that is prior to the date of the first delta export for the configuration.

Export Format LDIF / CSV

This field selects the export data file format. The possible selections are:

- LDIF-Selects LDAP Data Interchange Format (LDIF) file format. Use this format for full exports or delta exports.
- CSV-Selects Character Separated Value (CSV) format. Use this format for full exports only.

This is a mandatory field.

Separators 1, 2 and F

These fields specify the separators and the field delimiter character where

- **Separator 1** - specifies the separator character between entries (fields) in the export data file.
- **Separator 2** - specifies the separator character between multiple values within one

field (entry).

- **Separator F** - specifies the field delimiter character. Usually the double-quote character (**"**) is used as field delimiter. Escape the field delimiter by a consecutive delimiter character if it is embedded in the field value.

These are mandatory fields if the **Export Format** selected is **CSV**.

Date Format

This field specifies the format in which date values are to be represented in the export data file. The standard format is:

DDD, *DD.MMM.YYYY*

for example:

```
MON, 26.JUN.2011
```

You can also use a two-digit representation for the month; for example:

```
MON, 26.06.2011
```

This is an optional field.

Code Page

This field selects a code page (the number assigned to a character code set) into which the characters of the export are to be converted. If this field is not specified, SAPAgent uses the standard code page (1103) for the configuration. Select code page 1133 to use the ISO-8859-1 character code set.

SAPAgent (Unicode) does not support code pages because SAP does not support code pages in Unicode systems. The created file is in UTF-8 format. The Meta Controller must be configured to import UTF-8 files.

Days to look ahead for new empl/obj

This field specifies the number of days that the SAPAgent looks into the future for new persons or new objects. You can use this field to extend the set of selected persons and the set of extracted person data that will be exported as new employees join the company. Since all HR data is marked with a begin date and an end date of validation, or additions can be made that will become valid in the future. You can use this feature to synchronize data of new employees into other directories, for example, to create an e-mail address, before the new employee begins work.

OM export: When set, the implementation aligns the key date of the export by the amount of days specified.

This field works with both full and delta configurations and is an optional field.

Days to look ahead for mod./mod.obj

This field specifies the number of days that the SAPAgent looks into the future for modifications of persons or objects. You can use this field to extend the set of extracted

person data that will be exported. Since all HR data is marked with a begin date and an end date of validation, updates can be made that will become valid in the future. You can use this feature to synchronize updated data in a future time period into other directories.

You might get several records in the export file for the same person in a strict order (actual additions, modifications followed by future additions or modifications). Therefore it is advisable to export the begin date and end date attributes of the relevant infotypes.

Unless you use “LDIF-CHANGE in first delta” you can get a mixture of LDIF-CONTENT and LDIF-CHANGE format in the first delta export.

This field works with both full and delta configurations and is an optional field.

Use LDIF-CHANGE in first delta

This field specifies whether LDIF-CHANGE or LDIF-CONTENT format is used for the first delta export that is always a full export. By default, LDIF-CONTENT format is written.

This is an optional field.

Create MODRDN

This field specifies whether changes of attributes in the pseudo-dn are exported using a modrdn changetype record. In earlier versions (before 2.0x) this was exported as a modify changetype record.

Example: Change of surname from Meyer to Mueller:

```
dn: GN=Anja, PNO=00001000, SN=Meyer
changetype: modrdn
newrdn: SN=Mueller
deleteoldrdn: 1
```

Because this feature changes the output format of the export files, you must adapt your meta controller import scripts if you are upgrading from a earlier version.

This field works with delta configurations and is an optional field.

All tags in Modify-Record

This field specifies whether in delta mode a new formatting style is used. The new formatting style lists all attributes as a tagged list. If this field is set any modify and any add record contains next to the pseudo-generated DN all non-empty attributes in a tagged list. A pseudo attribute "CHANGETYPE" is also part of the list. A delete record will only consist of the pseudo-generated DN and the CHANGETYPE: DELETE attribute. If the "Create MODRDN" field is set also this format will still include the CHANGETYPE: MORRDN record as it does with the default LDIF change format.

The difference to the default LDIF change format is that in a modify record all attributes are exported, not just the changed attributes. This format is helpful if the following step of processing the agent's export file fails to modify all changed attributes in the target directory, so that attributes must be changed in the ERP database again and a second export is necessary. With the default format this second export would just have the attributes that have changed in-between.

3.9.6.2. Vertical Selection (OM)

You use the SAPAgent vertical selection dialog to determine the OM objects that are to be exported from the SAP/ERP database. To display the vertical selection dialog:

1. Click **Configuration**.
2. Select **Change Configuration**, and then select **Attributes for OM**.

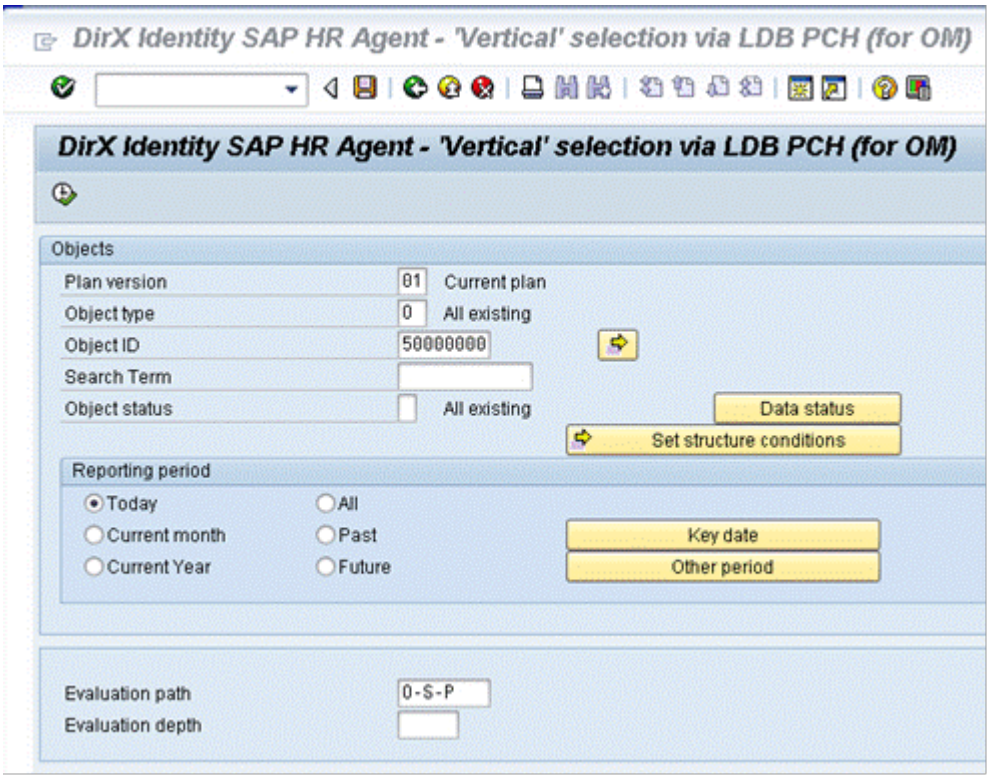
SAPAgent displays the vertical selection dialog. The dialog box contains five areas for vertical selection:

- **Multiple Selection including Objects and Reporting period** - this section enables you to specify selection criteria which the agent then uses to determine which set of data is read. These selection "screens" are the standard SAP report selection screens used in HR management. It contains a series of selection fields and buttons which are described in the SAP manuals. For further information see
  http://help.sap.com/erp2005_ehp_06/helpdata/EN/e5/7c3438fd263402e10000009b38f8cf/frameset.htm
  The Selection part enables you to enter a single value or value range for each selection field, which you can then restrict still further using the selection options.

  Multiple selection also enables you to enter

  - Several single values or value ranges to be taken into account when the export is executed.
  - Several single values or value ranges to be excluded when the export is executed.

    You can also use a selection option for each value and range; for example, greater or less than a single value, and within or not within a range.
- **Evaluation path** - This field and the evaluation depth field are optional. These fields can be used in combination with the multiple selection. If an evaluation path is given the path is additionally followed for each object from the multiple selection area. This was the only possible way in former versions of the SAPagent.
- **Configuration** - a read-only field that displays the configuration name.
- **Other Attributes** - defines the type of export to be performed (full or delta) and defines export data file format. (See also "Vertical Selection (PA)" for details.)
- **Job** - a job definition provides information about the configuration for the ERP job scheduler. You use the Job area of the vertical selection dialog box to define a job.



Each selection criteria field is displayed on a separate line. Click in any selection criteria field (or press **F4**) to display a list of the entries that are allowed for the field.

3.9.6.2.1. Selection via LDB

The Objects and Reporting period sections enables you to specify selection criteria which the agent then uses to determine which set of OM data is read. These selection “screens” are the standard SAP report selection screens used in HR management. It contains a series of selection fields and buttons which are described in the SAP manuals. For further information see

http://help.sap.com/erp2005_ehp_06/helpdata/EN/1b/16a7375de78668e10000009b38f889/frameset.htm

Plan version, Object type and at least one Object Id are mandatory fields. The object Id is used as a starting point. Object type defines what kind of OM objects are to be exported.

The Evaluation path field is an optional selection field and is used to navigate to the result set of the objects. You can only use evaluation paths that are defined in the ERP system. If you cannot find a suitable evaluation path, you must use the ERP system to define a new path. You can set an evaluation depth. The agent evaluates the objects to export along an evaluation path. With setting the depth you can reduce the number of evaluated objects. For example, the path **O-O** recursively describes the predecessor organizational unit. If you are interest just in the superior unit set the evaluation depth to 2.

Selection criteria fields that have no values entered in the lower portion of a range are not used for limiting the selection.

In the **Reporting period** section only **Today** is valid. Other date or range

selections are undefined.

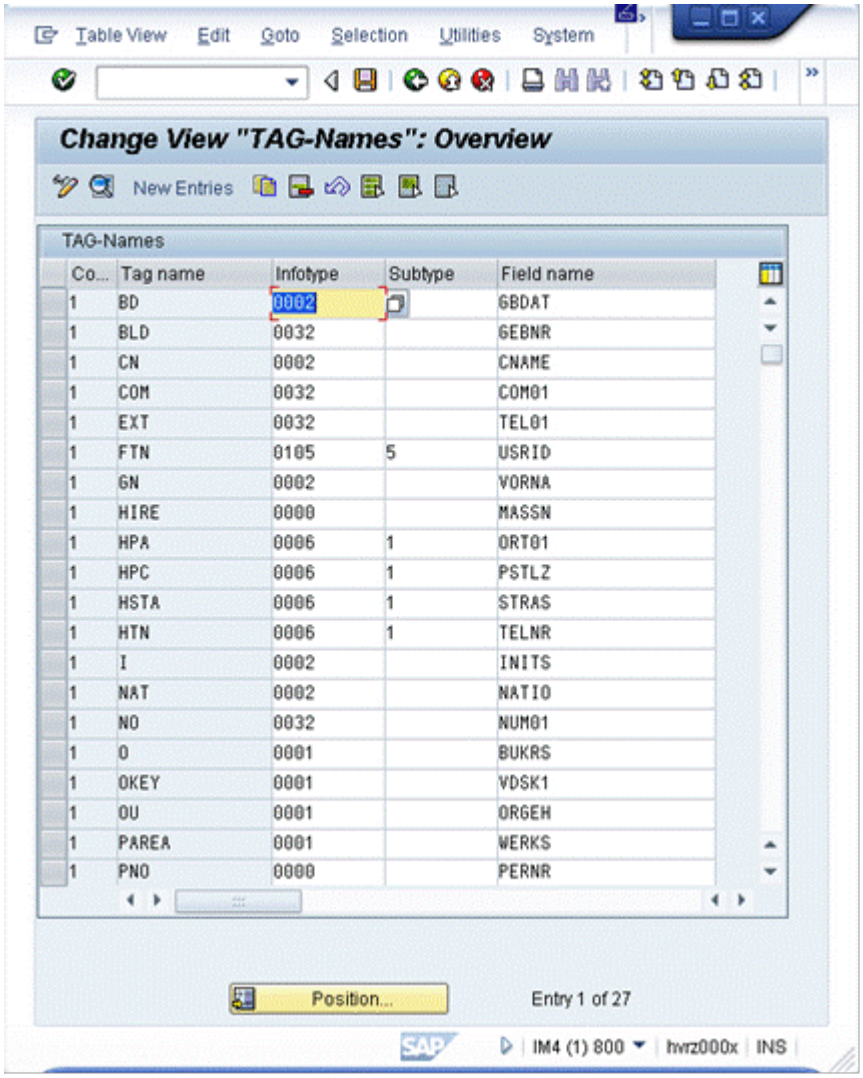
Fields with a zero value are left blank; this is an ERP issue.

3.9.6.3. Horizontal (Attribute) Selection

You use the SAPAgent horizontal selection function to determine the person attributes that are to be exported from the SAP/ERP database. To display the horizontal selection dialog:

1. Click **Configuration**.
2. Select **Change Configuration**, and then select **Tags (Fields)**.

SAPAgent displays the horizontal selection dialog.



The dialog consists of the following columns:

Configuration	Configuration ID. The unique identifier for the configuration. This is a mandatory field.
TAG name	The abbreviation for the attribute (tag); for example, GN. This is a mandatory field. Tag names cannot be longer than 30 characters
Infotype	The infotype for the attribute. This is a mandatory field.
Subtype	The subtype for the infotype. This is an optional field unless subtypes have been defined for the infotype.
Field name	The field per infotype (sub-table). This is a mandatory field.

Each entry (row) shown in the dialog represents an attribute name and the data associated with the attribute. An entry must have at least an infotype (**Infotype**) value and a field of that infotype (**Field name**).

Use the ERP data dictionary (transaction code SE11) to obtain the infotype, the subtype (if applicable), and the field name for an entry. The information about HR infotypes is located in tables PA*nnnn*, where *nnnn* represents the infotype. The information about OM infotypes is located in tables HRP*nnnn*.

Be careful when defining the tags. In delta mode, only entries are exported if any values of the selected tags have changed. For example if you want to export entries into a company you must at least export from infotype 0000 the fields pernr, subtyp, and begda.

If the subtype field is empty it is not evaluated by the SAPagent, all subtypes are exported. If you want to explicit filter the empty subtype (subtype 0) you must specify the characters as a mark for the empty one.

To insert a new entry, use the **New Entries** button or copy an entry and then edit it. When inserting new entries with the **New Entries** button (**F5**). Copying or the **New Entries** button opens the details view dialog screen.

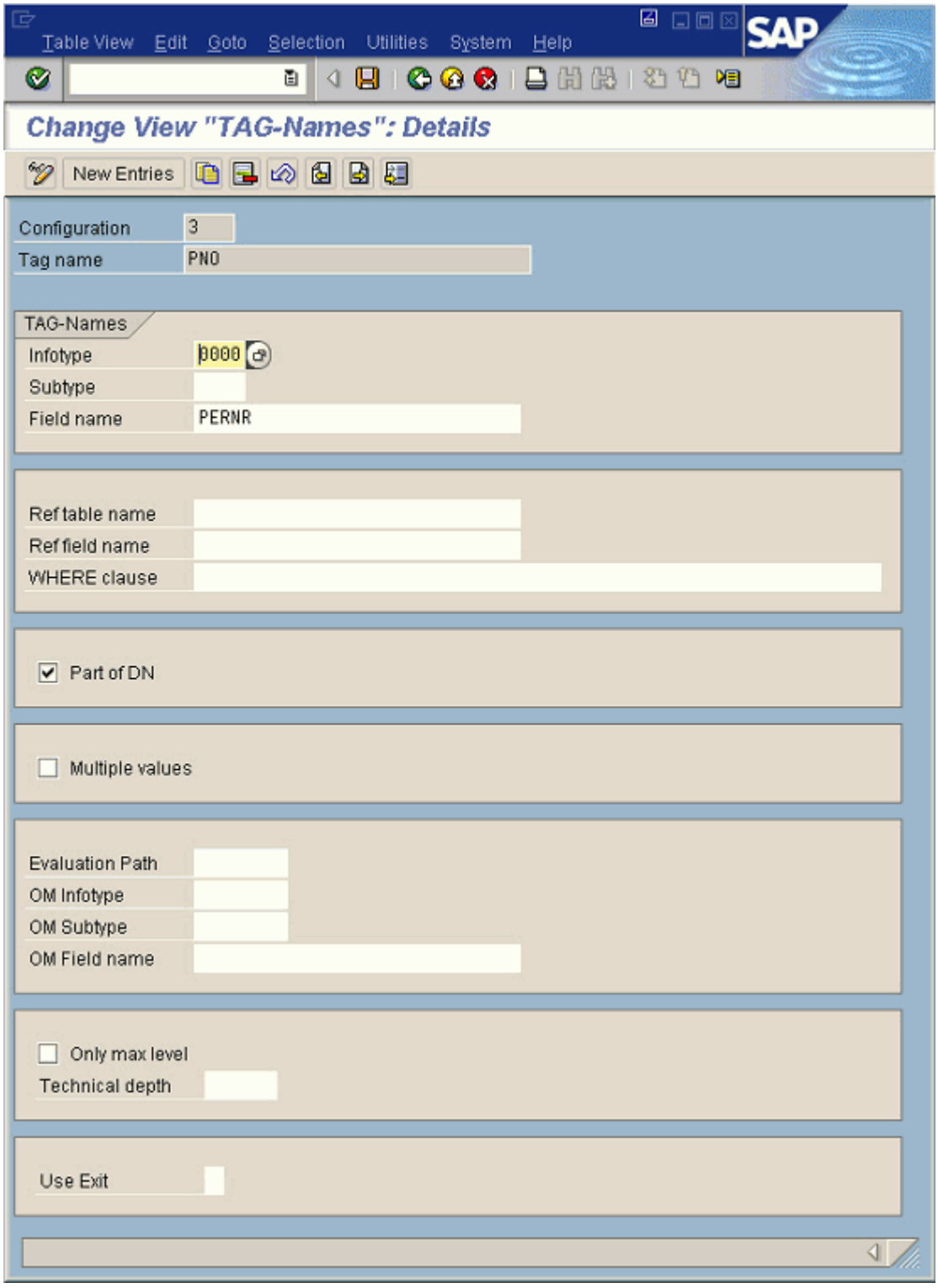
To delete an entry, select it and then click the **Delete** button (**Shift**+**F2**).

To edit an entry or view it in detail, double-click it.

The Change Tags Overview dialog only shows the most important configuration parameters. The detail view provides more detailed information.

To edit or view an entry in detail view:

1. Select the entry's TAG column, then click the magnifying glass icon in the application toolbar or double-click the selected entry. The Details dialog appears.



The dialog box provides the following fields:

Configuration	**Configuration ID. The unique identifier for the configuration. For new entries this is a mandatory field you must edit. For existing entries this is just an informational field.**
TAG name	The abbreviation for the attribute (tag); for example, GN. For new entries this is a mandatory field you have to edit. For existing entries this is just an informational field.

Configuration	**Configuration ID. The unique identifier for the configuration. For new entries this is a mandatory field you must edit. For existing entries this is just an informational field.**
Infotype	The infotype for the attribute. This is a mandatory field.
Subtype	The subtype for the infotype. This is an optional field unless subtypes have been defined for the infotype.
Field name	The field per infotype (sub-table). This is a mandatory field.
Ref table name	The name of a reference table. This is an optional field.
Ref Field name	The name of a reference field in the reference table. This is an optional field.
WHERE clause	A SQL expression that specifies the value of the entry, if the values in Ref table name and Ref field name are insufficient to describe the value. This is an optional field.
Part of DN	Whether or not the attribute is part of the generated distinguished name (DN) = key. This is an optional field.
Multiple values	Whether or not the attribute is a multi-valued field (relevant only for OM objects). This is an optional field.
Evaluation Path	The evaluation path to a related object, starting from the selected OM object or person. If this column is used, the Infotype, Subtype and Field name (in the box named *TAG-Names*) must identify the starting object (that is, the object id or the personnel number). This is an optional field.
OM Infotype	The infotype for the attribute of the related object. This is an optional field.
OM Subtype	The subtype for the infotype. This is an optional field unless subtypes have been defined for the OM Infotype field.
OM Field name	The field of the OM infotype. This is an optional field.
Only max level	Only the last object in the object chain along the path is evaluated, not all intermediate objects. This is an optional field.
Technical depth	Follows only the given number of objects in the object chain. This is an optional field.
Use Exit	Whether or not customer exits should be called for this configuration. This is an optional field.

Reference tables

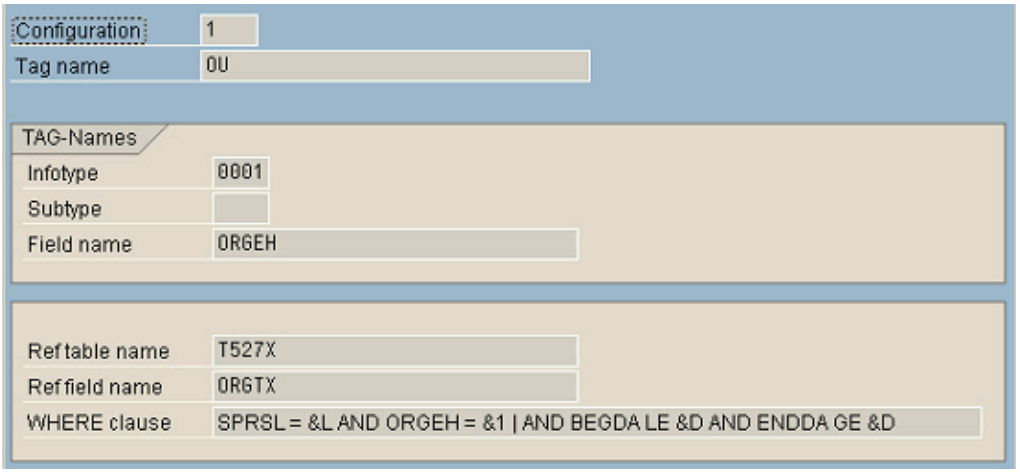
Reference tables - table, field and value - can be specified for any attribute that holds a key to the textual expression of its contents. Using this feature requires extensive knowledge of the HR component. Use the ERP data dictionary to determine reference tables and relationships for the selected attributes. For example, in the OU attribute of the default configuration, the statement:

```
SPRSL = &L AND ORGEH = &1 AND BEGDA LE &D AND ENDDA GE &D
```

selects the row from table T527X with the current language (SPRSL = &L), the reference key itself (ORGEH = &1), and provides for the correct time period (BEGDA LE &D AND ENDDA GE &D). From the row selected, the content of the field specified in the Field column (ORGTX) is then used as the attribute value.

Only variables &1, &D, and &L are allowed in the expression. Specifying a ABAP functional module call is prohibited.

- A WHERE clause can only be used for reference tables.
- Vertical bars ("|") are used to specify a line break in the generated ABAP program. This is strongly recommended to avoid syntax errors when executing the ABAP program.



OM data related a person or related to OM object

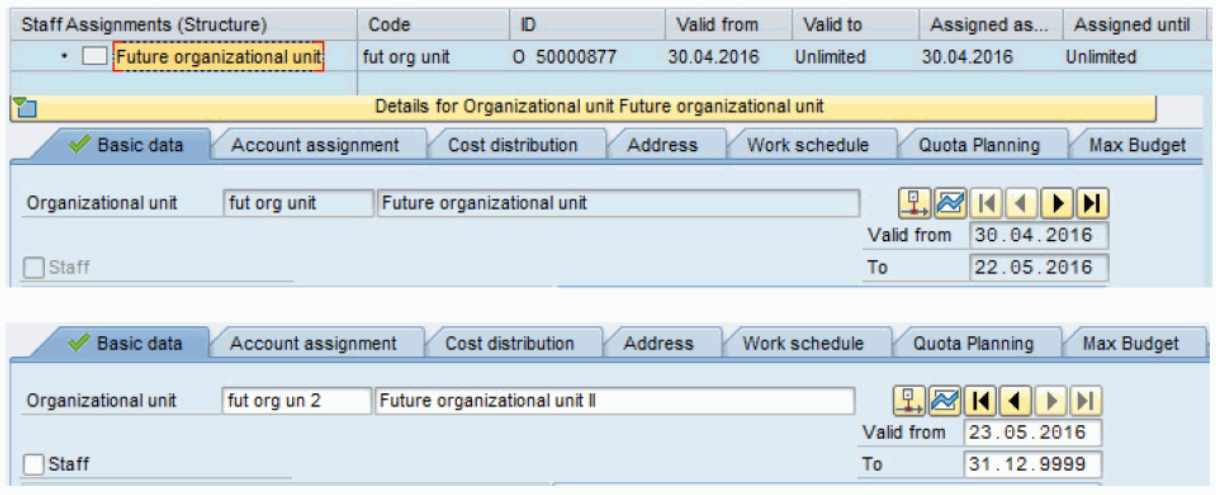
To export data from any OM object related to a person or related to one OM object, use the columns Evaluation Path, OM Infotype, OM Subtype, and OM Field name.

Generated DN - Key

SAPAgent exports the selected keys or just the Personnel number (PNO), if no keys are specified. The PNO is the primary key for ordering and the selected keys should follow in order of precedence.

Example 1. Future org. management data export

Consider a situation with an organizational unit which was initially created starting on April 30, 2016 and is split on May 22, 2016 because of a name change:



An export executed on May 1, 2016 without **Days to look ahead** yields this result:

```
# DEFAULT_CONF_OM20160501114917.TXT
version: 1
DN:ID_OF_NODE=50000877
ADDRESS_CITY_OF_AN_ORG_UNIT:
BEGDA:20160430
BELONGS_TO_ORG_UNIT_ID:50000000
BELONGS_TO_ORG_UNIT_NAME:target world organisation
ENDDA:20160522
FULL_NAME_OF_AN_EMPLOYEE:
ID_OF_NODE:50000877
LONG_TEXT_OF_OBJECT:Future organizational unit
RELATIONS:50000000
```

An export executed on May 1, 2016 with **Days to look ahead** set to **30** yields this result:

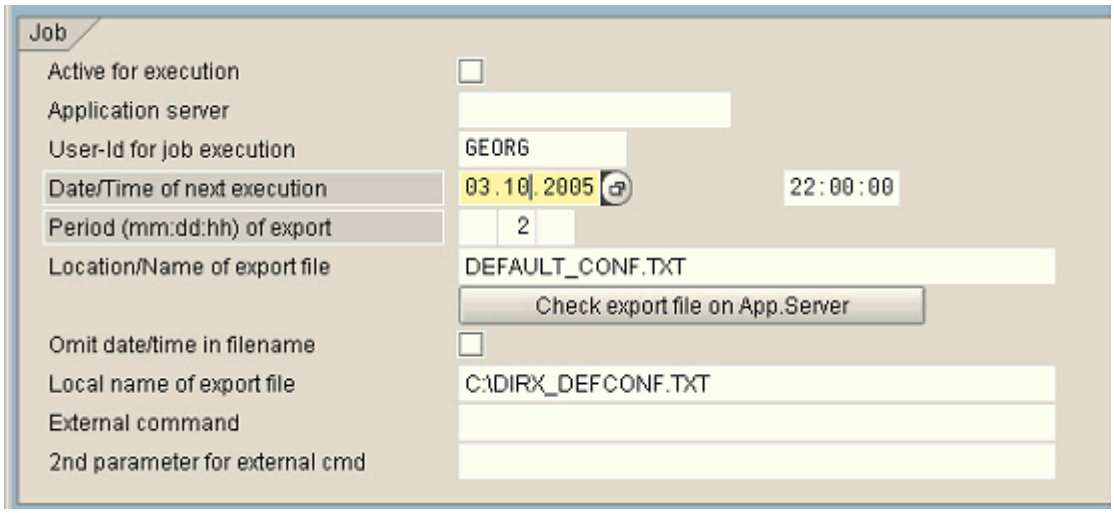
```
version: 1
DN:ID_OF_NODE=50000877
ADDRESS_CITY_OF_AN_ORG_UNIT:
BEGDA:20160430
BELONGS_TO_ORG_UNIT_ID:50000000
BELONGS_TO_ORG_UNIT_NAME:target world organisation
ENDDA:20160522
FULL_NAME_OF_AN_EMPLOYEE:
ID_OF_NODE:50000877
LONG_TEXT_OF_OBJECT:Future organizational unit
RELATIONS:50000000
DN:ID_OF_NODE=50000877
```

```
ADDRESS_CITY_OF_AN_ORG_UNIT:
BEGDA:20160523
BELONGS_TO_ORG_UNIT_ID:50000000
BELONGS_TO_ORG_UNIT_NAME:target world organisation
ENDDA:99991231
FULL_NAME_OF_AN_EMPLOYEE:
ID_OF_NODE:50000877
LONG_TEXT_OF_OBJECT:Future organizational unit II
RELATIONS:50000000
```

Look-ahead includes the complete history of the organizational unit. It would be an option to export ONLY the value at the aligned key date and then only the second data set would be exported.

3.9.6.4. Job Definition

A job definition provides information about the configuration for the ERP job scheduler. You use the Job area of the vertical selection dialog box to define a job.



The next sections describe the fields within the Job area.

Active for Execution

This field selects whether the configuration is active or inactive. Select this field to create a job definition for the selected configuration. If you do not select this field, SAPAgent does not generate a job definition for the configuration. However, SAPAgent will include the configuration in the transport to the production system, where it can be activated at a later date for scheduling with the Synchronize with job management selection.

You can also use the **Enable Configuration** and **Disable Configuration** selections in the **Scheduling** menu to activate and deactivate selected configurations. See the section "Configuration Activation" for more information.

Application Server

This field specifies an application server that is to run the job. This is an optional field; if it is not specified, the ERP load balancing subsystem selects the server.

User ID for Job Execution

This field specifies the user ID that is to run the job on the target system. The user ID specified must have access to the Human Resources system and have the authorization required for running background jobs. This is a mandatory field.

Date/Time of Next Execution

This field specifies the date and time at which the job is to be run for the first time. You can select this value from a calendar dialog and an hours dialog using **F4**. This is a mandatory field.

Period of Export

This field specifies the time interval for subsequent executions of the job. You can specify the period between job executions in months (*mm*), days (*dd*), or hours (*hh*). For example, the value 7 in the *dd* field schedules subsequent job execution every seven days from the date/time specified in the **Date/Time of Next Execution** field. The notation: The value 5 in the dd field and 3 in the hh field schedules subsequent job execution every 27 hours from the date/time specified in the **Date/Time of Next Execution** field.

The shortest time period you can select is 1 hour. This is a mandatory field.

Location/Name of Export File

This field specifies the file name or full pathname of the export data file. The path must be to a shared directory so that all instances of the ERP scheduler can access it. If no path is specified, SAPAgent uses the ERP default data directory; see the ERP documentation for details about the default data directory. This is a mandatory field.

You can use the check export file on App server button to check whether the ERP system can create and write to the specified file. If you use this button, the verification process deletes the specified export data file if it already exists.

Omit date/time in filename

This field specifies whether a timestamp in the generated export filename should be omitted or not. By default a timestamp is appended to the filename of the export file, for example if the name is “full_1.TXT” the exported data is written to “full_1*timestamp*.TXT” where *timestamp* is in the format *yyyymmddhhmmss*.

If the timestamp is omitted each export overwrites the last generated export file. No automatic clean-up takes place to delete export files. Delete export files that are no longer in use to save disk space.

Local Name of Export File

This field specifies the file name on the local file system that SAPAgent is to use when it is invoked to copy the export data file from the remote ERP system to the local system. Use the Copy Export File selection in the Extras menu to copy an export data file back to

a local system. This is an optional field.

External Command

This field specifies a program to be executed after the job completes. The root for external commands is:

/usr/sap/*SAP_system_ID***/SYS/exe/run** (UNIX)

*****host_name***\sapmnt***SAP_system_ID***\SYS\exe\run** (Windows)

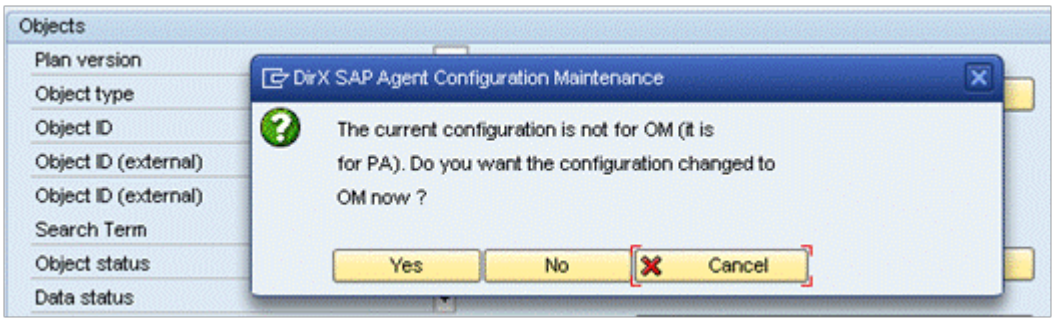
This is an optional field.

Second Parameter for External Command

This field specifies a parameter to be applied to the program specified in the External Command field. This is an optional field. The value specified in this field is applied as a second parameter to the program specified in the External Command field; the first parameter is the path and/or file name of the export data file. (The name of the generated export data file is inserted between the two parts of the external command.)

3.9.6.5. Change Configuration

If an inappropriate processing is selected for an existing configuration, the user is notified and may cancel the operation, or change the configuration (from PA to OM or inverse):



3.9.6.6. Default Configuration

The base configuration consists of the following defaults:

- Daily delta export at 22:00 into the export data file **default_conf.txt** in LDIF format for the user "GEORG"
- The configuration is not activated.
- The vertical selection selects All Personnel numbers with employee status "Active". There are no other exclusions.

The default horizontal selection is shown in the following table:

Tag	Info type	Sub type	Field name	Part of DN	Ref table / field	Description	Comment
BD	0002		GBDAT			Birth date	

Tag	Info type	Sub type	Field name	Part of DN	Ref table / field	Description	Comment
BLD	0032		GEBNR			Building number	
CN	0002		CNAME			Common name	
COM	0032		COM01			Internal communication type	
EXT	0032		TEL01			Internal telephone number	Extension
FTN	0105	5	UsrID			Fax telephone number	UsrID_long
GN	0002		VORNA	X		Given name	
HPA	0006	1	ORT01			Home postal address	
HPC	0006	1	PSTLZ			Home postal code	
HSTA	0006	1	STRAS			Home street address	
HTN	0006	1	TELNR			Home telephone number	
I	0002		INITS			Initials	
NAT	0002		NATIO			Nationality	
NO	0032		NUM01			Internal communication number	
O	0001		BUKRS		T001/ BUTXT	Organization	Company code/ Buchungskreis
OKEY	0001		VDSK1			Organizational key	
OU	0001		ORGEH		T527X/ ORGTX	Organizational unit	
PAREA	0001		WERKS		T500P/ NAME1	Personnel area	i. e. location/divison

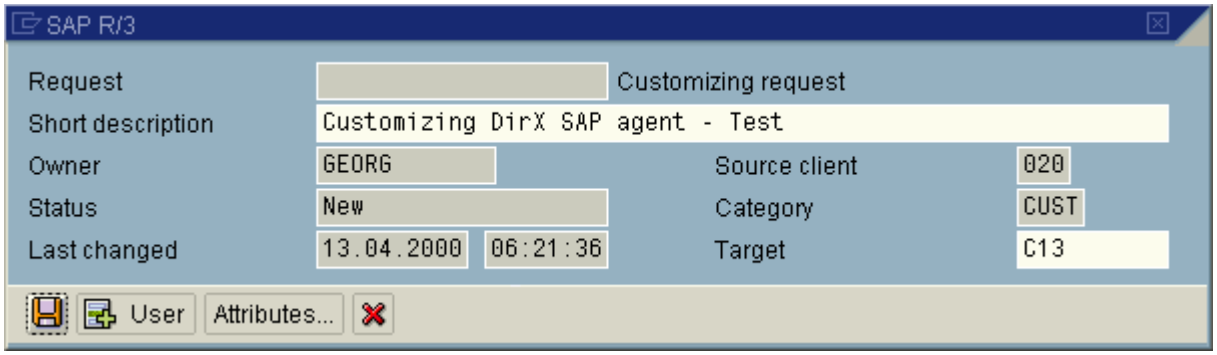
Tag	Info type	Sub type	Field name	Part of DN	Ref table / field	Description	Comment
PNO	0000		PERNR	X		Personnel number	
PSAREA	0001		BTRTL		T001P/ BTEXT	Personnel subarea	i. e. division/location
RMB	0105	0010	UsrID_Long			SMTP address	
ROOM	0032		ZIMNR			Room number	
SAL	0002		ANRED		T522T/ ANRLT	Salutation	
SAPID	0105	1	UsrID			SAPOffice user identifier	UsrID_long
SN	0002		NACHN	X		Surname	
TIT	0002		TITEL		T535N/ TITEL	Title	

3.9.7. Transport from Customizing to Production

SAPAgent assists in the transport of customizing information by storing all relevant objects in a selectable transport. The customizing transport itself must be generated using transaction SE10 or within the pop-up selection dialog. Use the **Extras** menu of the SAPAgent main window to transport configurations.

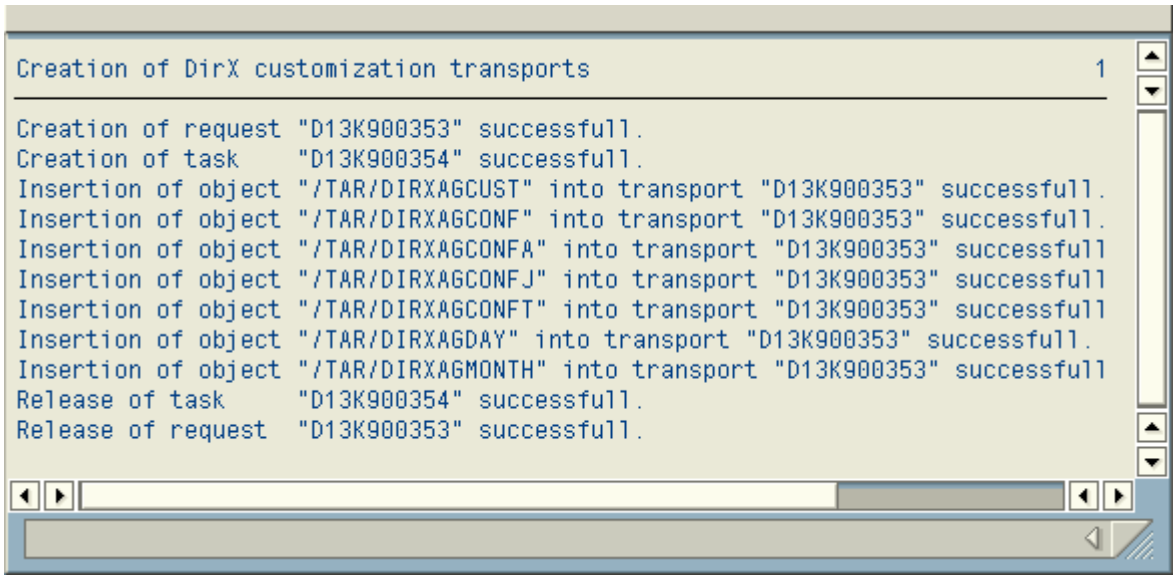
To transport all configurations (and security information) from a development system to a production system:

1. From the **Extras** menu, select **Transport Customizing**
2. On the selection screen, check the Execute button.
3. Use the Execute icon or press **F8** to start the generation of the customizing transport.
4. SAPAgent creates a transport with the attribute CUST. The SAPAgent customizing data will be imported into this transport. In **Short Description**, supply a short description for the transport.



5. A pop-up appears to confirm the release of the transport (if you cancel the release here you can release (or delete) the transport later using transaction SE10).

The transport is exported from ERP to two local files. SAPAgent displays a result dialog box that summarizes the transport export process:



The files generated during this process can now be copied to the production system and imported there using standard ERP methods.

3.9.8. Configuration Activation and Immediate (ad-hoc) Execution

If you have created an inactive configuration, you can set it to an active job definition. Conversely, you can deactivate an active configuration. Use the **Scheduling** menu selection in the SAPAgent main window to activate and deactivate configurations.

To activate a configuration:

1. From the SAPAgent main window, select the configuration.
2. In the **Scheduling** menu, select **Enable Configuration**.

To deactivate a configuration:

1. From the SAPAgent main window, select the configuration.
2. In the **Scheduling** menu, select **Disable Configuration**.

You can also use SAPAgent to perform immediate (or "ad-hoc") execution of a

configuration. To perform an immediate transfer:

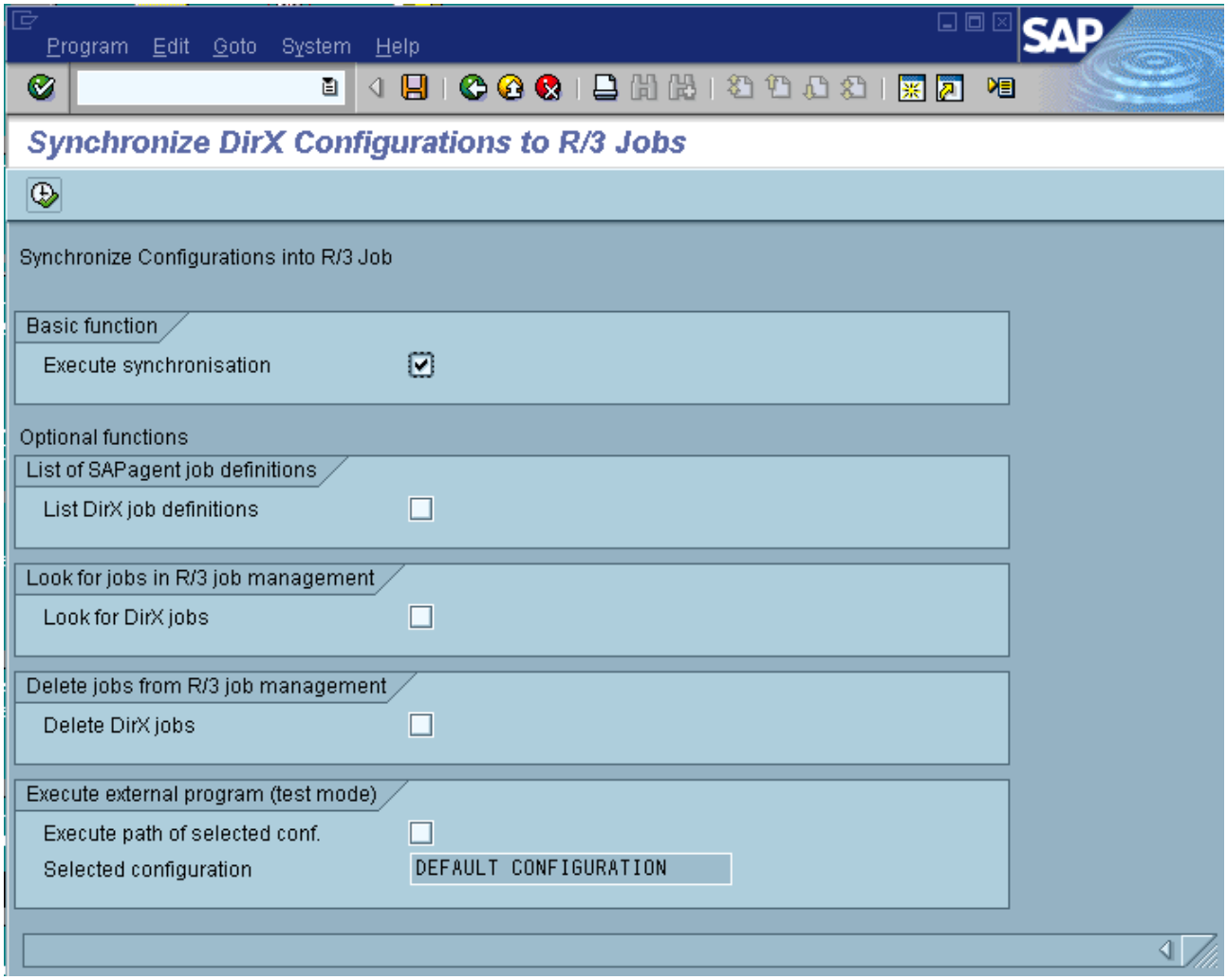
1. From the SAPAgent main window, select the configuration.
2. In the **Scheduling** menu, select **Immediate Export**.

To force a full export immediate transfer for a delta export configuration:

- In the **Extras** menu, select **Force full export**.
- In the Scheduling menu, select **Immediate Export**.

3.9.9. Job Scheduling

Use the **Synchronize DirX Conf's to R/3 Jobs** selection in the **Scheduling** menu of the SAPAgent main window to schedule all of the SAPAgent transfers that you have configured and activated. The following figure shows the job scheduling dialog that SAPAgent displays.



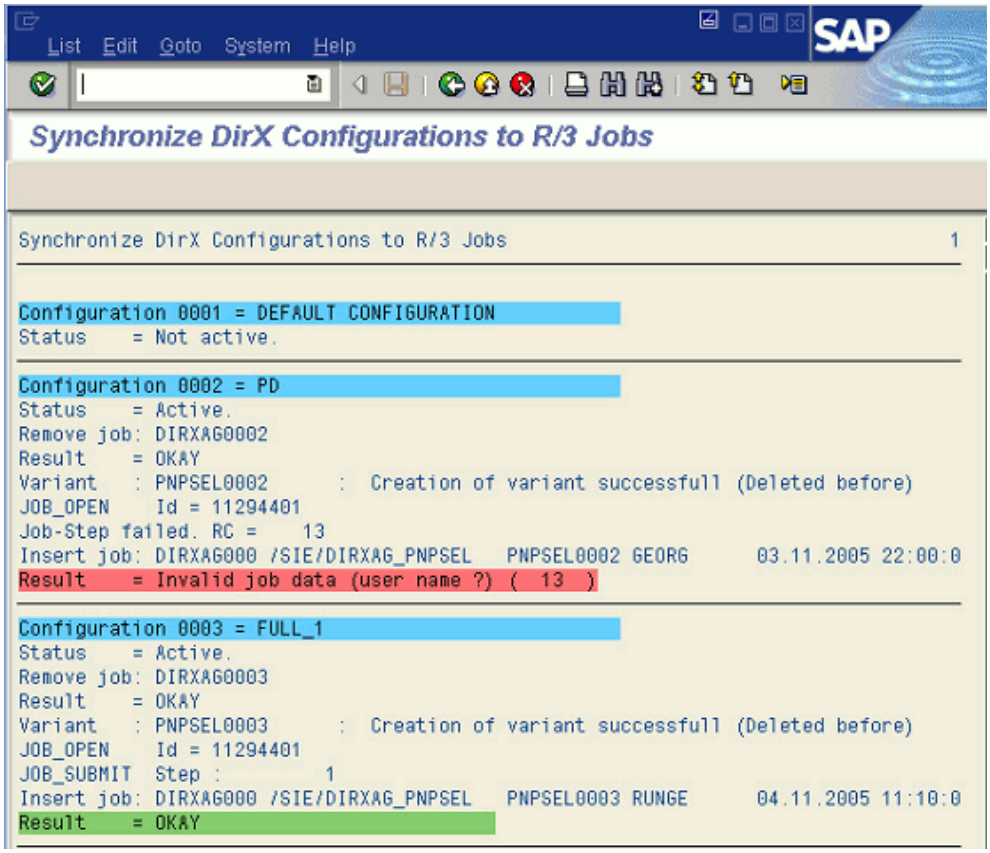
The next sections describe the fields in the job scheduling dialog box.

Execute Synchronization

This field controls the SAPAgent job scheduling function. To activate job scheduling:

1. Check the **Execute Synchronization** field
2. Press **F8** or click the **Execute** button

SAPAgent schedules all active configurations (configurations whose job definitions' **Active for Execution** field are checked) with the ERP job scheduler and generates a report. For example:



SAPAgent automatically creates a name for each job in the format:

DIRX_AG*config_number*

Where *config_number* is a four-digit code that identifies the configuration. The report highlights in red the jobs that SAPAgent was not able to enter into the ERP job management system. Common reasons for being unable to schedule jobs include:

- The start date specified in the job definition's **Date/Time of Next Execution** field is older than the current time
- The user ID specified in the job definition's **User ID for Job Execution** field is not present on the system or does not have the necessary rights to execute background jobs

The generated report name is "/SIE/DIRXAG_PNPSEL", the variant is "PNPSEL*config_number*".

List DirX Job Definitions

This field controls whether SAPAgent creates a list of the configurations it has created. When this field is checked, SAPAgent generates a short description of each active and

inactive configuration that it has created.

Look for Jobs in ERP Job Management

This field controls whether SAPAgent creates a list of all SAPAgent configurations that have been scheduled as ERP jobs. When this field is checked, SAPAgent generates a list of configurations with the following notations:

✓ (green) the configuration is running or has completed without errors

x (red) the configuration has aborted or is incomplete

+ the configuration has been scheduled

Delete DirX Jobs

This field controls whether SAPAgent deletes all of the jobs it has scheduled. When this field is checked, SAPAgent removes all of the ERP jobs it has scheduled. Deleted jobs must be reconfigured with a new start date and must be reactivated before they can be re-scheduled as ERP jobs.

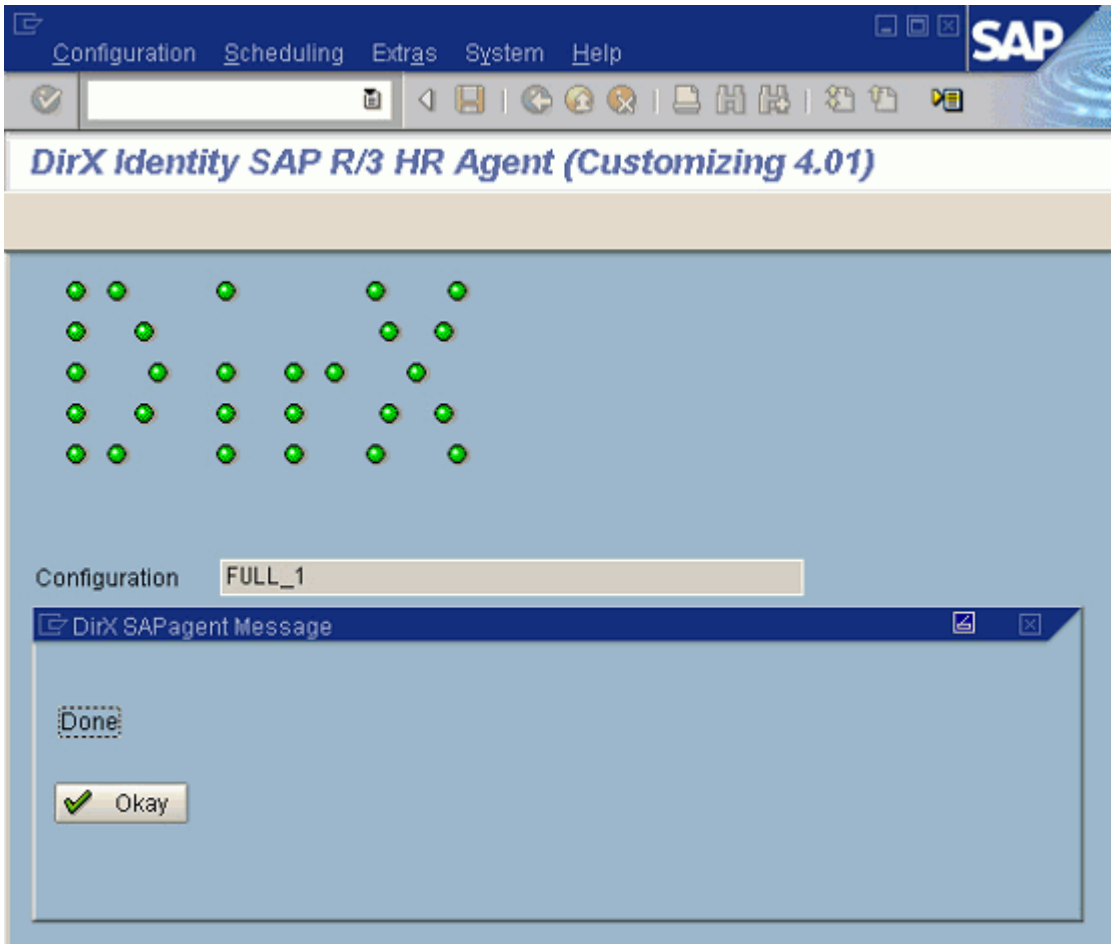
Execute Path of Selected Configuration

This field allows you to test the executable programs you have specified in the job definition. To use this field, select a configuration and check the box.

3.9.10. Export Procedure

A SAPAgent transfer can be scheduled for execution through the ERP job scheduler or for immediate execution. When a transfer, or job, is scheduled for ERP, execution of an export is handled entirely by the ERP job scheduler and takes place even if SAPAgent (transaction code **/sie/dirx_ag**) is not active. You use the **Synchronize DirX Conf's to ERP Jobs** selection in the **Scheduling** menu to manage the scheduling of jobs with ERP. Transfers (jobs) can be also viewed with the standard transaction 'SM37'.

You can also bypass the ERP scheduler and schedule a transfer for immediate execution using the **Immediate export** selection in the **Scheduling** menu. A message indicates the end of the immediate export.



All SAPAgent export operations rely on the exchange of files. SAPAgent follows a protocol during export operation to ensure consistent treatment of these files. Note that it is the ERP *application server* running SAPAgent that is executing the file operations. This fact can cause unexpected results in distributed ERP systems that run multiple application servers. For these types of installations, it may be necessary to:

- Mount specific directories over the network
- Link SAPAgent to a specific application server

Consult with the SAP ERP administrators of the installation to find and implement the appropriate solution(s).

The specification of the transfer directory can be based on physical names (system, for example, /usr/SAPagent/Transfer) or logical names (ERP, maintained using transaction FILE, for example, SAPagentTransferDirectory).

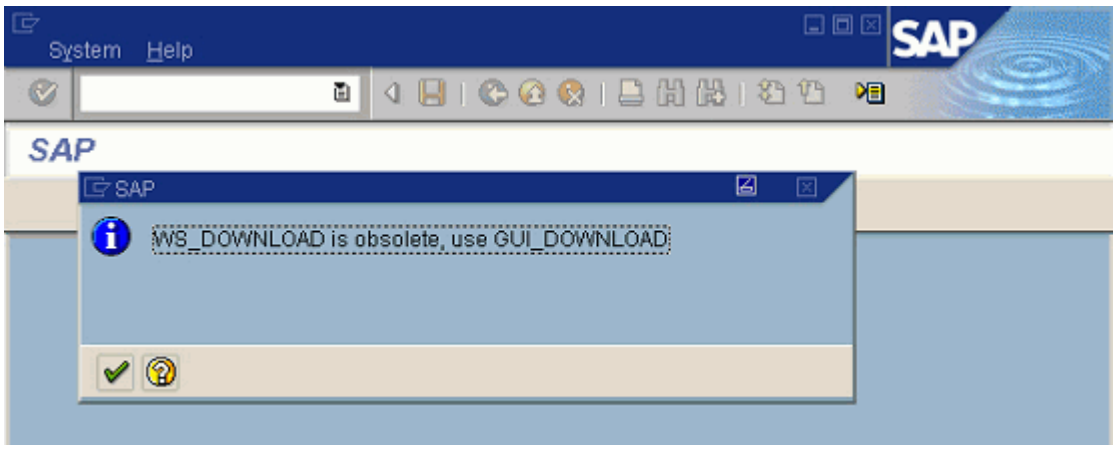
The export procedure proceeds as follows:

1. SAPAgent is scheduled by hand or with the ERP job scheduler
2. SAPAgent creates the export data file with the name *Export_File_NameDateTime*[*.ext*]
3. The ERP job management system calls the optional system command, script or application supplying the filename and a set of parameters.

You can use the **Copy Export File** selection in the **Extras** menu to transfer a generated

export data file to the presentation server (that is, the local PC).

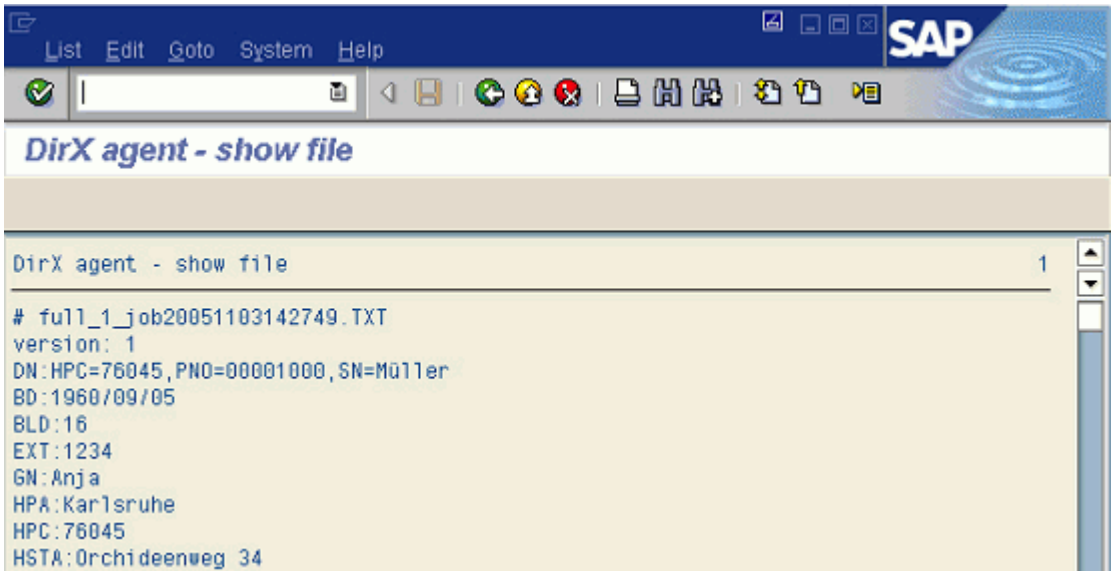
The following informational message box can be ignored (ERP internal issue). Just click on Continue or press Enter to continue.



After the downloading is completed another message box (“Transfer complete”) informs about successful copying.

If you use the SAPAgent (Unicode) be sure to set the Upload/Download Encoding for the SAP GUI to “Default UTF8 for Unicode Systems”. To specify this value select the Unicode system entry in the SAP logon, right click “Properties” and then “Advanced”. Then specify the value in the Advanced Options dialog under “Encoding for Upload/Download”.

You can use the **Show Export File** selection in the **Extras** menu to view a generated export file.



3.9.10.1. Delta Export Procedure

The SAPAgent delta export no longer uses the "Change documents" ("Belegschreibung") mechanism. The delta reference is held in SAPAgent’s internal cluster tables.

Please note the following about SAPAgent delta export:

- The SAPAgent delta mechanism operates on the basis of a daily time pattern. If you perform a full export and a delta export on the same day and changes to the ERP database also occur, the changes are exported twice. Consequently, your scripts to import SAPAgent output to other directories should be designed to handle duplicate change records.
- The ERP system allows you to make updates that become effective at a future date. The SAPAgent delta mechanism does not recognize updates made to the ERP database before the first export that are to become effective sometime after the first export. Instead, any full export will recognize them. Consequently, it is recommended to use a time schedule that consists of a cycle that equals one full export and several subsequent delta exports.

3.9.10.2. Security Features

An HR system operates with highly confidential data and must therefore be maintained under several aspects of security. It should prevent any unauthorized export; at least not under specific allowance and control. In most installations, special guidelines from the public (for example, in Germany: Bundesdatenschutz bestimmungen) and from the company (in Germany: Betriebsvereinbarungen) must be followed.

SAPAgent implements the following features to guarantee the highest possible security:

- Two levels of security: administration and configuration
- Denial of selection of fields of specific types (for example, currency)
- Selection of the maximum set of allowed fields
- Logging of all activities (transfers and changes to configuration and scheduling) to provide a historical record of everything that happened in the system and who was responsible for it.

These security features are configurable (by a system administrator, not by the user) to allow tailoring for the individual needs of an installation.

SAPAgent also includes the following functions to prevent unauthorized export or modification of confidential data:

- Positive list of trusted (ERP) users
- Positive list of infotypes
- Positive list of fields in the trusted users and infotypes structures. Use the "*" identifier to select all fields of one structure.
- Denial of fields of type "CURR" (currency)
- Denial of fields of type "DATE" (date).

These features must be configured by an ERP-knowledgeable administrator using the field/value pairs in the table **/sie/dirxagcust**. SAPAgent does not provide a special user interface for configuring these features.

The following table shows the attributes that control whether a transfer for any configuration will succeed or fail. For any configuration, the entire transfer will fail if one of the attributes shown in the table is violated. The transfer is allowed if none of these attributes is violated.

Attribute Name	Type	Multiple	Description	Example Value
SECLIMITUSER	Boolean		Limit users to the following list	X
SECUSER	Text	X	Name of an allowed user	DIRX
SECLIMITCURR	Boolean		Deny CURR fields	
SECLIMITDATE	Boolean		Deny DATE fields	
SECLIMITFIELD	Boolean		Allow only fields of the following list	
SECFIELD	Text	X	Pair (table, field) allowed for selection. "*" for all fields	0002, vorna

The security features described here are set to default values (all selections are allowed) and can be changed during the customization phase. SAPAgent transfers any changes to these values that are made during customization to the production system in the customizing transport operation.

The security features described here are only valid during SAPAgent execution in a production system; they are not verified during customization. If security is violated when a transfer begins, the entire transfer is cancelled and a log entry is generated.

3.9.10.3. Customer Exits

The SAPAgent includes a set of exits to allow customer specific extension of the built-in functionality. The exits are implemented using the SAP Business Add-Ins ('BAdI') technology. There is a BAdI *definition* for every exit. To use an exit, a BAdI *implementation* (an ABAP objects method) must be developed.

There are the following exits:

- Excluding a person or OM object from export
- Computing a value of a user-defined tag
- Exporting multiple virtual employees

- Creating or modifying a BAdI implementation results in a change request for a transport.
- The SAPAgent requires Unicode enabled BAdI implementations.

3.9.10.3.1. Exits to modify/disable the processing of a person or an OM object

There are two definitions, one for a person export and the other for an OM object export.

For both definitions the following input parameters are passed to the method:

CONFIGURATION
Specifies the configuration identifier (id=number).

CONTENT
A structure of strings that matches the actual configuration. Here is an example, if you use the default configuration:

```
data bd like p0002-gbdat.
data bld like p0032-gebnr.
data cn like p0002-cname.
data com like p0032-com01.
data ext like p0032-tel01.
data ftn like p0105-usrid.
data gn like p0002-vorna.
data hire like p0000-massn.
data hpa like p0006-ort01.
data hpc like p0006-pstlz.
data hsta like p0006-stras.
data htn like p0006-telnr.
data i like p0002-inits.
data nat like p0002-natio.
…
```

In a BAdI implementation you can either specify the content in a structure as given in the example above or you access the structure by specifying offset and length.

EVENT
Is a flag specifying the time when the exit is called while processing each object (a person or an OM object) in an export. Valid values for this flag are:

- **B**: Call the exit before evaluating the object's attribute values. If you can exclude a person or an OM object just on the knowledge of the personnel number or the object identifier specifying this value results in a faster processing.
- **A**: Call the exit after evaluating the object's attribute values. The **CONTENT** parameter delivers a list of all evaluated object's attribute value pairs. The exclusion can then include this information to determine whether an object should be excluded from export or not.

Additionally to the parameters above the following input parameter is passed to the method for a person export:

PERNR
Specifies the personnel structure.

The following input parameter is passed to the method for an OM object export:

OBJEC
Specifies the OM object structure.

The method delivers the following output parameter:

STATUS

Specifies whether an object is excluded from the export (value **1**) or not (all values not equal **1**).

Of course there may be several configurations using exclusion. However, you should activate only one implementation. The configuration class and the configuration identifier specify the actual activated configuration.

Here is an example for a person export method:

```
BAdI Definition:     /SIE/DIRXAG_CHKPERNR
Interface:               /SIE/IF_EX_DIRXAG_CHKPERNR
Method:                CHECK_PERSON
Parameters:
```

Parameter	Type	P...	O...	Typing	Reference type	Defa...	Description
CONFIGURATION	Importing	☐	☐	Type	/SIE/DIRXAG_CONFIGURATION		DirX agent configuration id
PERNR	Importing	☐	☐	Type	PERNR		Standard Selections for HR Master Data Reporting
EVENT	Importing	☐	☐	Type	CHAR01		B=before, A=after
CONTENT	Importing	☐	☐	Type	ANY		content (during 'after' event)
STATUS	Returning	☑	☐	Type	NUM02		Result code (1 = skip person)

Here is an example for an OM object export method:

```
BAdI Definition:     /SIE/DIRXAG_CHKOBJ
Interface:               /SIE/IF_EX_DIRXAG_CHKOBJ
Method:                CHECK_OBJECT
Parameters:
```

Parameter	Type	P...	O...	Typing	Reference type	Def...	Description
CONFIGURATION	Importing	☐	☐	Type	/SIE/DIRXAG_CONFIGURATION		DirX agent configuration id
OBJEC	Importing	☐	☐	Type	OBJEC		Work Area for Database PCH
EVENT	Importing	☐	☐	Type	CHAR01		B=before, A=after
content	Importing	☐	☑	Type	any		content (during 'after' event)
STATUS	Returning	☑	☐	Type	NUM02		Result code (1 = skip object)

3.9.10.3.2. Exits to compute the value of a user-defined tag

There are two definitions: one for a person export and the other for an OM object export.

For both definitions, the following input parameters are passed to the method:

CONFIGURATION
Specifies the configuration identifier (id=number).

TAG NAME
Specifies the tag name.

FIELD NAME
This parameter is not used and therefore is always empty.

FIELD VALUE
The value that the agent computes before calling the exit.

Additionally to the parameters above the following input parameter is passed to the method for a person export:

PERNR
Specifies the personnel structure.

The following input parameter is passed to the method for an OM object export:

OBJEC
Specifies the OM object structure.

The method delivers the following output parameter:

RETVALUE
Returns the computed user-defined tag value.

There may be several user-defined tags per configuration and several configurations using user-defined tags. However, you should activate only one implementation. The implementation class, configuration identifier, and the tag name specify the current activated configuration and tags.

Here is an example for a user-defined tag in a person export method:

```
BAdI Definition:     /SIE/DIRXAG_UF_PERNR
Interface:               /SIE/IF_EX_DIRXAG_USERFUNC
Method:                EVALUATE_PERNR_TAG
Parameters:
```

Parameter	Type	P...	O...	Typing	Reference type	Def...	Description
CONFIGURATION	Importing	☐	☐	Type	/SIE/DIRXAG_CONFIGURATION		DirX agent configuration id
TAG_NAME	Importing	☐	☐	Type	STRING		the tag name
FIELD_NAME	Importing	☐	☐	Type	STRING		the field name
FIELD_VALUE	Importing	☐	☑	Type	STRING		the field value
PERNR	Importing	☐	☐	Type	PERNR		Standard Selections for HR Master Data Reporting
RETVALUE	Returning	☑	☐	Type	STRING		the computed value

Here is an example for a user-defined tag in an OM object export method:

```
BAdI Definition:     /SIE/DIRXAG_UF_OBJ
Interface:               /SIE/IF_EX_DIRXAG_UF_OBJ
Method:                EVALUATE_OBJ_TAG
Parameters:
```

Parameter	Type	P...	O...	Typing	Reference type	Def...	Description
CONFIGURATION	Importing	☐	☐	Type	/SIE/DIRXAG_CONFIGURATION		DirX agent configuration id
TAG_NAME	Importing	☐	☐	Type	STRING		the tag name
FIELD_NAME	Importing	☐	☐	Type	STRING		the field name
FIELD_VALUE	Importing	☐	☑	Type	STRING		the field value
OBJEC	Importing	☐	☐	Type	OBJEC		Work Area for Database PCH
RETVALUE	Returning	☑	☐	Type	STRING		the computed value

To enable a tag as a user exit, specify a **X** in the **Use Exit** field in the **Change View "Tag-Names": Details** dialog box. (See the section "Horizontal (Attribute) Selection" above for details.) The agent evaluates the **Infotype** and **Field name** fields of the tag before calling the exit. It passes the resulting value to the exit.

3.9.10.3.3. Export of multiple virtual employees

This functionality is provided by an additional BAdI:

BAdI Definition: /SIE/DIRXAG_ADDPERNR

Method: GET_ADD_PERSONS

Parameters

This method accepts the following parameters:

Name	Type	Associated Type	Description
CONFIGURATION	Importing	/SIE/DIRXAG_CONFIGURATION	Configuration Key
PERNR	Importing	PERNR	Personnel Number
BEGDA	Importing	BEGDA	Start Date
MASK	Importing	ANY	Structure of configuration
VIRTUAL_PERSONS_PERNR	Exporting	HCM_PERNR_TABLE	Table of personnel numbers
VIRTUAL_PERSONS_CONTENT	Exporting	STANDARD TABLE	Table of virtual persons (like MASK)

On a call of an implementation of the above BAdI, the agent supplies the configuration number (so that the BAdI can distinguish between different configurations), the PERNR object (NOT only the personnel number) of the actual (real) employee and the key date for which the employee's data is being extracted.

The BAdI implementation is then expected to return a table of generated personnel

numbers (Type PERNR_D) in parameter VIRTUAL_PERSONS_PERNR and a corresponding table of virtual employees with each entry of the same structure as MASK in parameter VIRTUAL_PERSONS_CONTENT.

Example Coding

Assuming the following TAG configuration:

TAG-Names

Con...	Tag name	Infotype	Subtype	Field name
9	GN	0002		VORNA
9	HSTA	0006	1	STRAS
9	OU	0001		ORGEH
9	PNO	0000		PERNR
9	SN	0002		NACHN

Here is the corresponding BAdI implementation:

```
method /SIE/IF_EX_DIRXAG_ADDPERNR~GET_ADD_PERSONS.
*
* Sample implementation of BAdI /SIE/DIRXAG_ADDPERNR.
* This implementation adds two virtual employees
* for an existing employee FOR_EMPLOYEE in configuration
IN_CONFIGURATION.
*
* The implementation assumes the data of a virtual employee to be of
structure:
*DATA BEGIN OF SNEW.
*DATA GN LIKE P0002-VORNA.
*DATA HSTA LIKE P0006-STRAS.
*DATA OU LIKE P0001-ORGEH.
*DATA PNO LIKE P0000-PERNR.
*DATA SN LIKE P0002-NACHN.
*DATA END OF SNEW.
*
* This directly corresponds to the 'TAGS' definition of the
configuration.
* An ABAP type with exactly this structure is supplied in parameter
MASK
* and should be used as a reference.
*
constants:
for_employee type pernr_d value '00001001',
in_configuration type /sie/dirxag_configuration value '0009'.
```

```
data:
dref type ref to data.
field-symbols:
<fs1> type any,
<fs2> type any.
clear: virtual_persons_pernr[], virtual_persons_content[].
case configuration.
when in_configuration.
case pernr-pernr.
when for_employee.
* First virtual employee
create data dref like mask. " Allocate memory for data of one virtual
employee
assign dref->* to <fs1>. " and assigned the memory to a field symbol
(pointer).
clear: <fs1>.
assign component 'GN' of structure <fs1> to <fs2>. " Assign pointer
to a specific…
if sy-subrc is initial. " In case of success set data.
<fs2> = 'Joe'.
endif.
append '90000001' to virtual_persons_pernr. " Append the PERN to the
return list.
append <fs1> to virtual_persons_content. " Append the new data line.
* Second virtual employee
create data dref like mask. " Allocate memory for data of one virtual
employee
assign dref->* to <fs1>. " and assigned the memory to a field symbol
(pointer).
clear: <fs1>.
assign component 'HSTA' of structure <fs1> to <fs2>. " Assign pointer
to a specif...
if sy-subrc is initial. " In case of success set data.
<fs2> = 'Road to nowhere'.
endif.
append '90000002' to virtual_persons_pernr. " Append the PERN to the
return list.
append <fs1> to virtual_persons_content. " Append the new data line.
endcase.
endcase.
```

```
endmethod.
```

Example Output

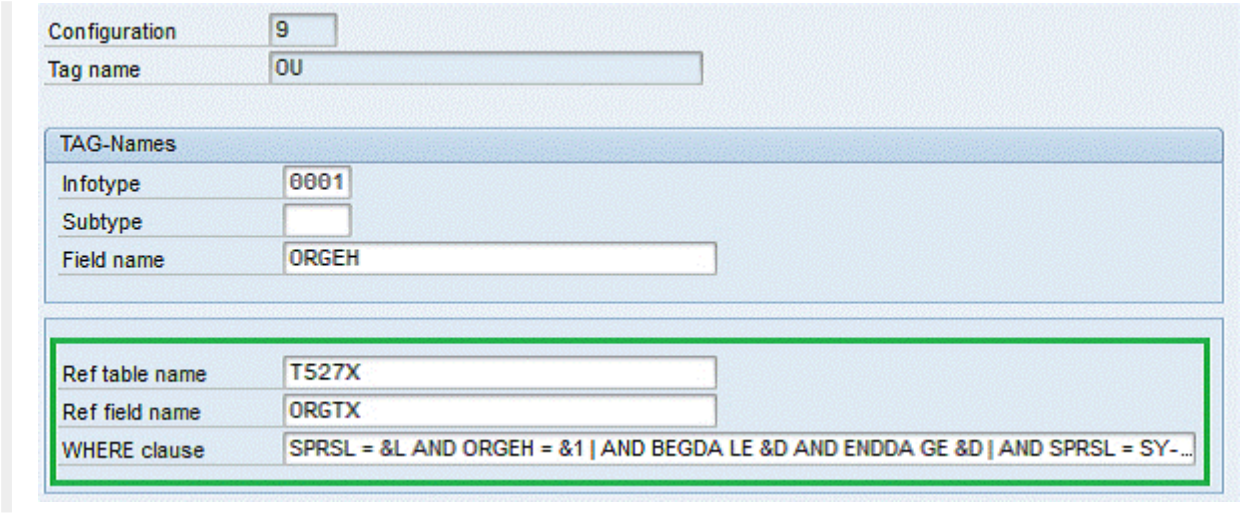
Based on the BAdI implementation sample but with all fields of the two virtual employees filled, this is the resulting export file:

You can see that the two "virtual" persons 90000001 and 90000002 have been inserted between the "real" persons 00001001 and 00001002 with the values assigned to them in the BAdI implementation. The DN has also been constructed by these values.

Output Transformation via Reference Tables

No longer used. BadI returns "text" values.

Looking at the definition of tags for this configuration, the definition of tag "OU" defines a value replacement via the organizational unit's text table (T527X):



This is also performed for virtual persons, so just the key has to be inserted into the virtual person; the SAPAgent automatically transforms the key to a text value via the reference definition. The corresponding coding in the BAdI for virtual person 90000001's OU is:

```
assign component 'OU' of structure <fs1> to <fs2>. " Assign pointer to a specific field.
if sy-subrc is initial.      " In case of success set data.
  <fs2> = '50000031'.
endif.
```

The output file shows the transformed value:

```
DN:GN=Joe,PNO=90000001,SN=Bar
GN:Joe
HSTA:Hauptstraße
OU:Branch Munich
PNO:90000001
SN:Bar
```

Delta Processing

The virtual person records are subject to the SAPAgent's delta processing.

If in the example above, the BAdI implementation would return a different value for any tag:

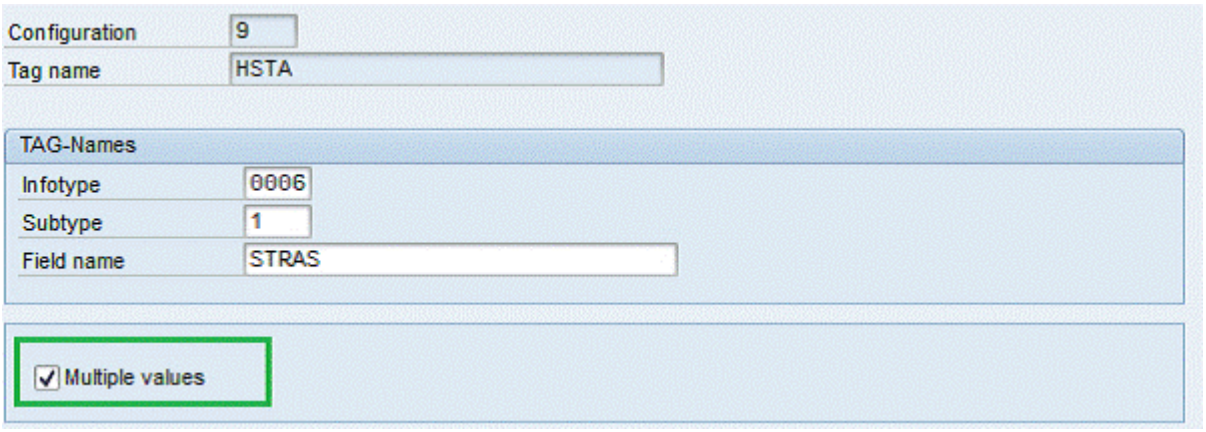
```
  assign component 'HSTA' of structure <fs1> to <fs2>. " Assign pointer to a specific field.
  if sy-subrc is initial.      " In case of success set data.
*    <fs2> = 'Hauptstraße'.
    <fs2> = 'Nebenstraße'.
  endif.
```

A MODIFY record would be created:

```
# DEFAULT_CONF_PA20160506074948.TXT
version: 1
DN:GN=Joe,PNO=90000001,SN=Bar
CHANGETYPE: MODIFY
REPLACE:HSTA
HSTA:Nebenstraße
-
```

Tags with Multiple Values

Tags can be defined to return a list of values instead of a single value:



In these cases, the BAdI implementation must deal with field symbols and operations to internal tables instead of data elements (note the usage of field symbol <fs2t> instead of <fs2> :

Definition:

```
field-symbols:
<fs1> type any,
<fs2> type any,
<fs2t> type standard table.
```

Assigning the internal table and inserting data:

```
assign component 'HSTA' of structure <fs1> to <fs2t>. " Assign
pointer to itab
if sy-subrc is initial. " In case of success set data.
append 'Hauptstraße' to <fs2t>.
append 'Nebenstraße' to <fs2t>.
endif.
```

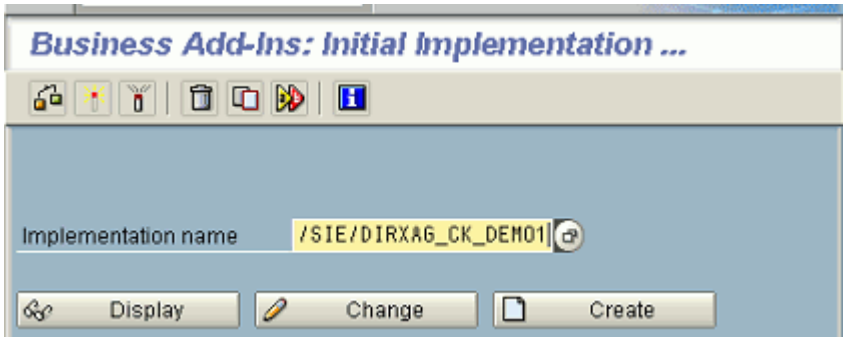
The corresponding output looks like this:

```
DN:GN=Joe,PNO=90000001,SN=Bar
GN:Joe
HSTA:Hauptstraße
HSTA:Nebenstraße
OU:Branch Munich
PNO:90000001
SN:Bar
```

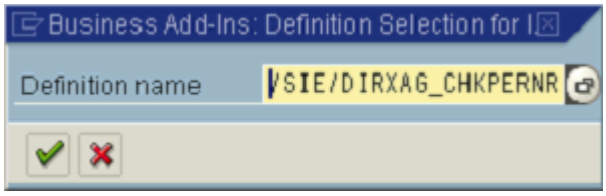
3.9.10.3.4. Case study 1: Creating an exit for a person selection

This case study provides an example how to set up an exit for a person selection. Perform the following steps:

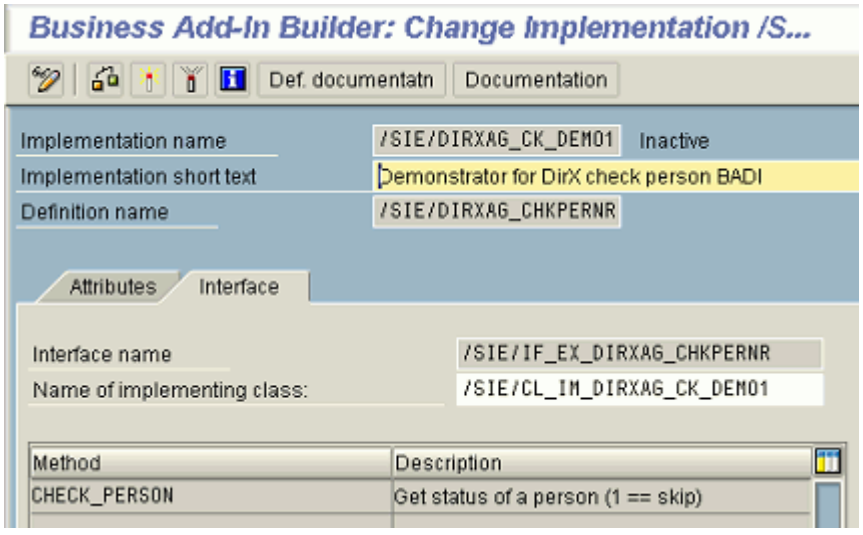
1. Start the BAdI implementation (transaction SE19).
2. Enter a (new) name for the BAdI *implementation* in your customer or in the Z* namespace.



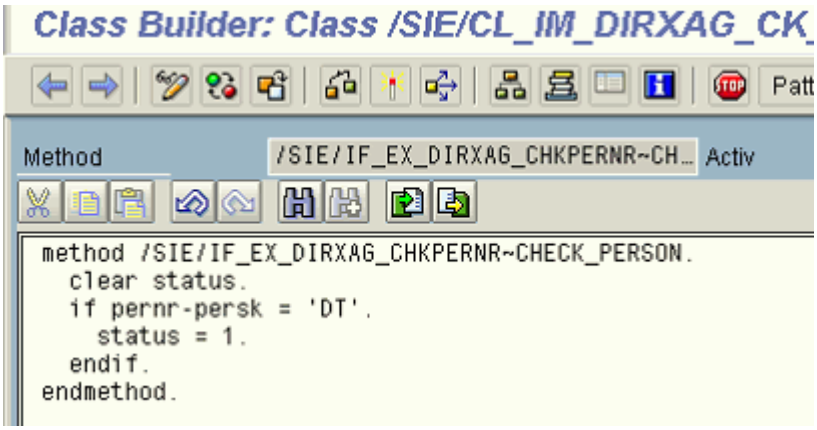
3. Select the correct BAdI definition (here '/SIE/DIRXAG_CHKPERNR'):



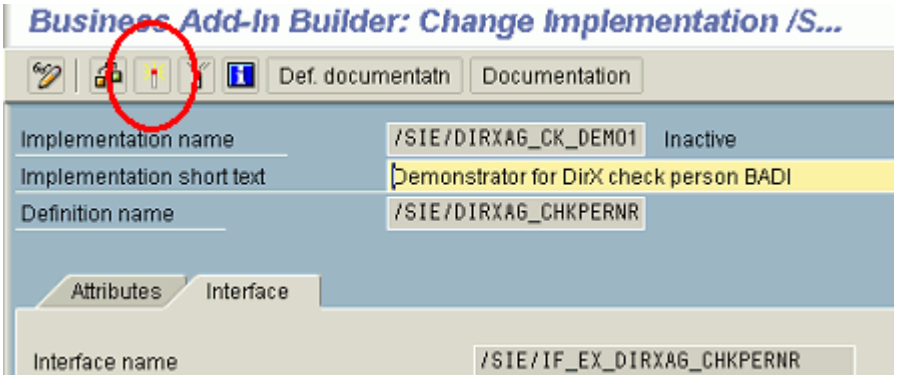
4. Enter a description and (optionally) modify the name of the implementing class, then select method 'CHECK_PERSON'



5. Implement the method (the example here disables persons of employee subgroup 'DT')



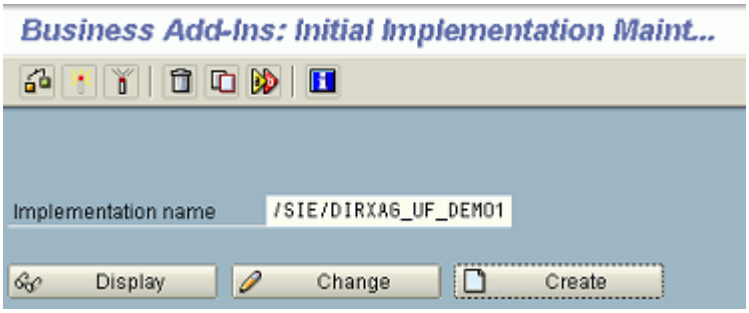
6. Activate the BAdI implementation



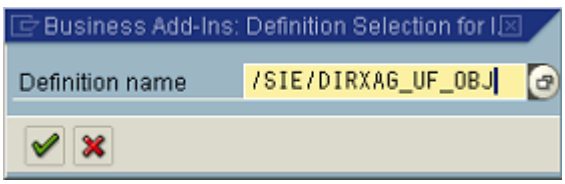
3.9.10.3.5. Case study 2: Creating an exit for user defined tag evaluation (here OM)

This case study provides an example how to set up an exit for user-defined tag evaluation of an OM object. Perform the following steps:

1. Start BAdI implementation (transaction SE19)
2. Enter a (new) name for the BAdI *implementation* in your customer or the Z* namespace

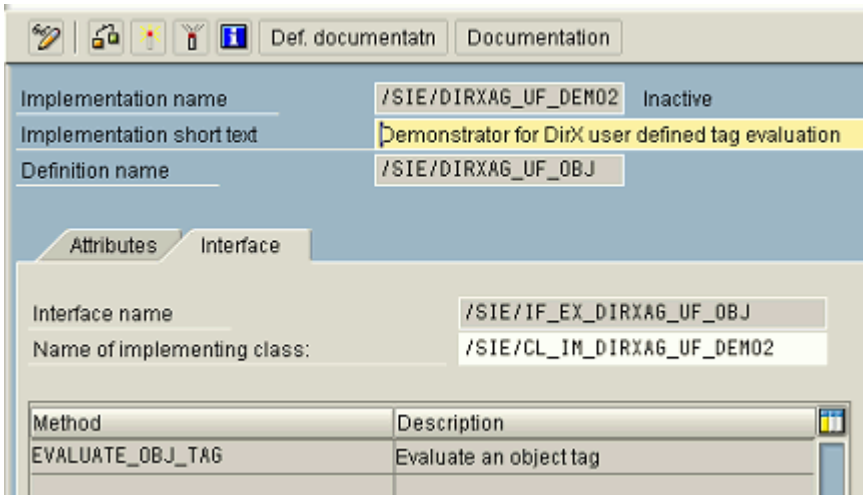


3. Select the above BAdI definition ('/SIE/DIRXAG_USERFUNC'):

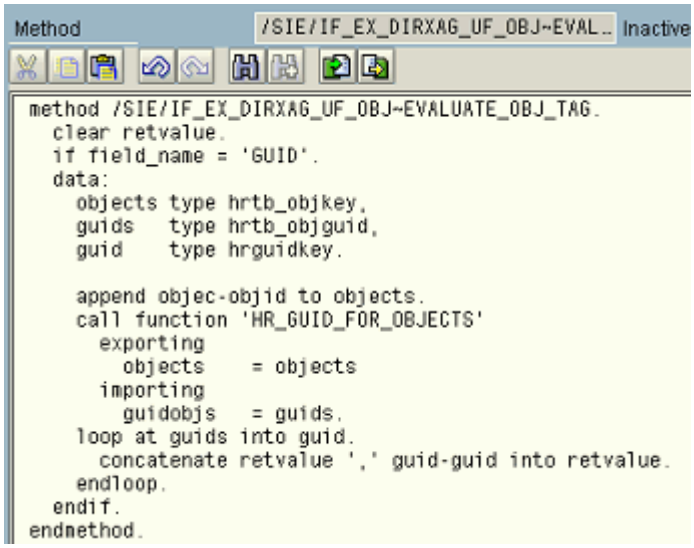


4. Enter a description and (optionally) modify the name of the implementing class, then

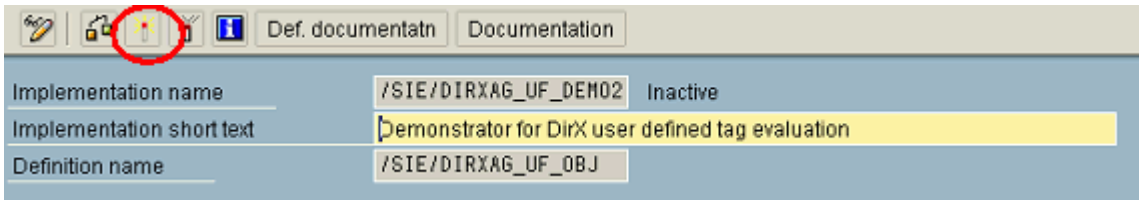
select method ‘EVALUATE_OBJ_TAG’



5. Implement the method (the example here concatenates the GUIDs of an object)



6. Activate the BAdI implementation



3.9.10.3.6. Case study 3: Defining a tag to report a user’s "hire" date

This case study provides an example how to set up an exit for defining a user-defined tag to report a user’s hire date. It is implemented using the same procedure as case study 2 above.

The code below demonstrates the exit implementation to fill the tag "HIRE" in configuration "1". The hire-semantic is customer specific and therefore this example may not work in every installation.

The administrator must enter the values for the **Infotype** (value 0000) and **Field name** fields (value MASSN). The agent evaluates these values. The retrieved value is not used in this exit.

```
method /SIE/IF_EX_DIRXAG_USERFUNC~EVALUATE_PERNR_TAG.
if configuration >= 0.
  if tag_name = 'HIRE'.
  data:
    pphifi       type table of phifi,
    pp0000       type table of p0000,
    pp0001       type table of p0001,
    phifi        like line of pphifi,
    hiredate(10) type c.
    retvalue = 'unknown'.
      CALL FUNCTION 'HR_READ_INFOTYPE'
      EXPORTING
        pernr                = pernr-pernr
        infty                = '0000'
      tables
        infty_tab            = pp0000
     EXCEPTIONS
       INFTY_NOT_FOUND       = 1
       OTHERS                = 2.
    IF sy-subrc <> 0.
      retvalue = 'no 0000'.
    ENDIF.
      CALL FUNCTION 'HR_READ_INFOTYPE'
      EXPORTING
        pernr                = pernr-pernr
        infty                = '0001'
      tables
        infty_tab            = pp0001
     EXCEPTIONS
       INFTY_NOT_FOUND       = 1
       OTHERS                = 2.
    IF sy-subrc <> 0.
      retvalue = 'no 0001'.
    ENDIF.
      CALL FUNCTION 'RP_HIRE_FIRE'
      TABLES
        pphifi          = pphifi
```

```
        pp0000          = pp0000
        pp0001          = pp0001.
    loop at pphifi into phifi.
      if not phifi-hires is initial.
        write phifi-begda to hiredate.
        move hiredate to retvalue.
      endif.
    endloop.
  endif.
endif.
endmethod.
```

3.9.10.4. Configuring OM Extracts

This section provides detailed information on how to export OM extracts.

The SAP Agent allows including OM information in its exports. Beacaure of the complexity of PD/OM, it is not easy to understand the capability of this functionality. As a result, the section describes it based on examples.

Suppose you have the following OM structure:

Staff assignments (structure)	Code	ID	Relationship text	C	Valid from	Valid to
myroot	myroot	O 12040000			19.11.1800	Unlimited
mysubroot1	mysubroot1	O 12040001	Is line supervisor of		19.11.1800	Unlimited
mypos1	mypos1	S 12041001	Incorporates		19.11.1800	Unlimited
mysubroot2	mysubroot2	O 12040002	Is line supervisor of		19.11.1800	Unlimited
mypos2	mypos2	S 12041002	Incorporates		19.11.1800	Unlimited
Captain Future	Future	P 00000002	Holder		01.02.2007	Unlimited

Figure 18. OM Structure

You can use transaction PPOME to view the hierarchy of the organizational unit, the related objects and some of the associated info types. The screenshot shown here is from a PPOME display. You can use PPOME (or the older PP01) to create the hierarchy shown in the screenshot to allow you to replay the following examples.

The SAPAgent provides two mechanisms to export information from the PD component:

- Objects related to an employee
- Objects selected directly from PD

They will be explained in the following sections.

3.9.10.4.1. Objects Related to an Employee

This is used in the "tag" details configuration, where it allows to maintain an evaluation path to go from an employee's OM relevant information (common are ORGEH or PLANS in infotype 0001) to the details of that OM object.

Example:

Assume in a standard PA export the following definition of tag POS:

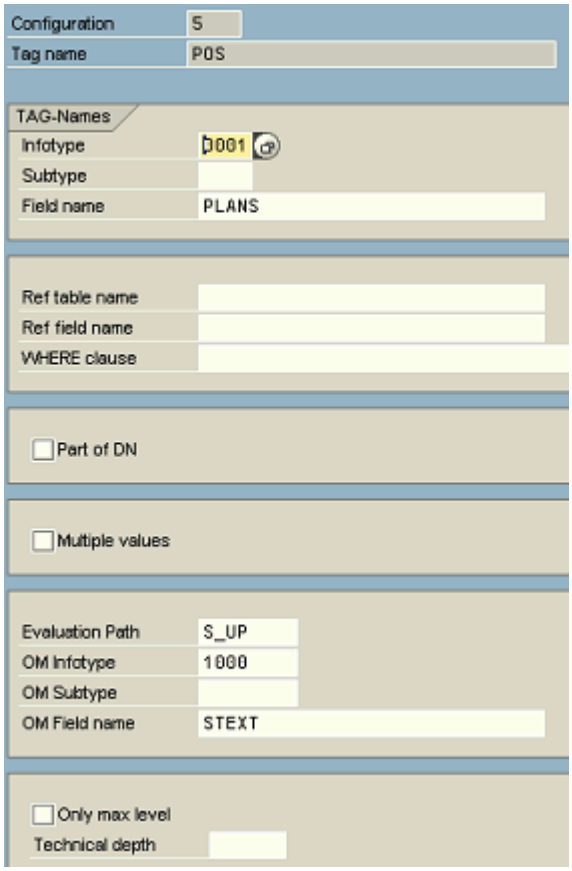


Figure 19. Tag POS

Then, for the selected employee's every infotype 0001 record, the content of the position (field PLANS) is taken and OM's evaluation path 'S_UP' is used to define the owning organizational unit of this position, and from this organizational unit's object definition (infotype 1000) the description (field STEXT) is returned.

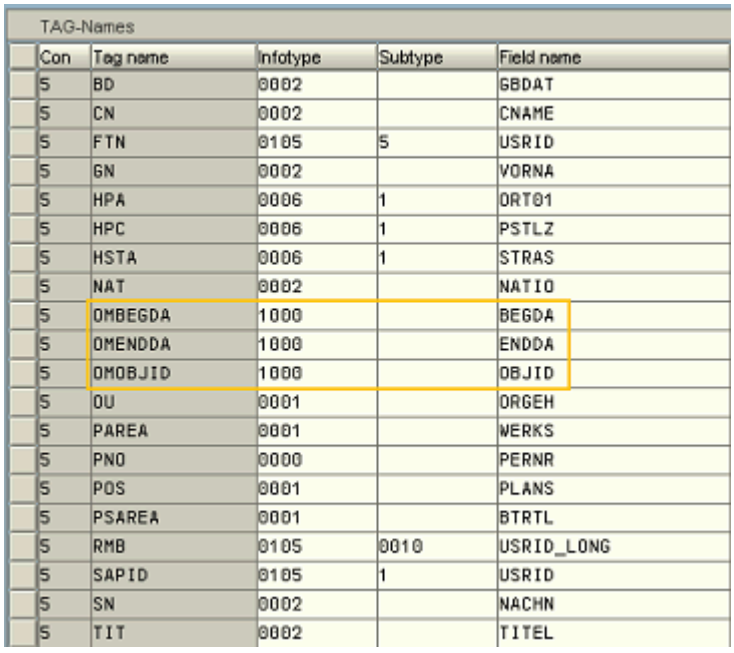
Yielding in an output like:

```
POS:mysubroot2
POS:myroot
```

It can be seen that all the related OUs of the position of the employee are listed. If only the root is needed, then 'Only max level' should be specified; if only the direct OU of the position is needed, then a 'Technical depth' of 2 should be specified in the tag's configuration.

3.9.10.4.2. Objects Selected Directly from PD

For a SAPAgent export defined on above OU hierarchy this would be a valid "tag" configuration:



TAG-Names

Con	Tag name	Infotype	Subtype	Field name
S	BD	0002		GBDAT
S	CN	0002		CNAME
S	FTN	0105	5	USRID
S	GN	0002		VORNA
S	HPA	0006	1	ORT01
S	HPC	0006	1	PSTLZ
S	HSTA	0006	1	STRAS
S	NAT	0002		NATIO
S	OMBEGDA	1000		BEGDA
S	OMENDDA	1000		ENDDA
S	OMOBJID	1000		OBJID
S	OU	0001		ORGEH
S	PAREA	0001		WERKS
S	PNO	0000		PERNR
S	POS	0001		PLANS
S	PSAREA	0001		BTRTL
S	RMB	0105	0010	USRID_LONG
S	SAPID	0105	1	USRID
S	SN	0002		NACHN
S	TIT	0002		TITEL

Figure 20. Tag Configuration

Especially take note, when looking at the details of for example the OMOBJID tag, it is not necessary (that is it would not work as expected) to mark OM specifications additionally here:

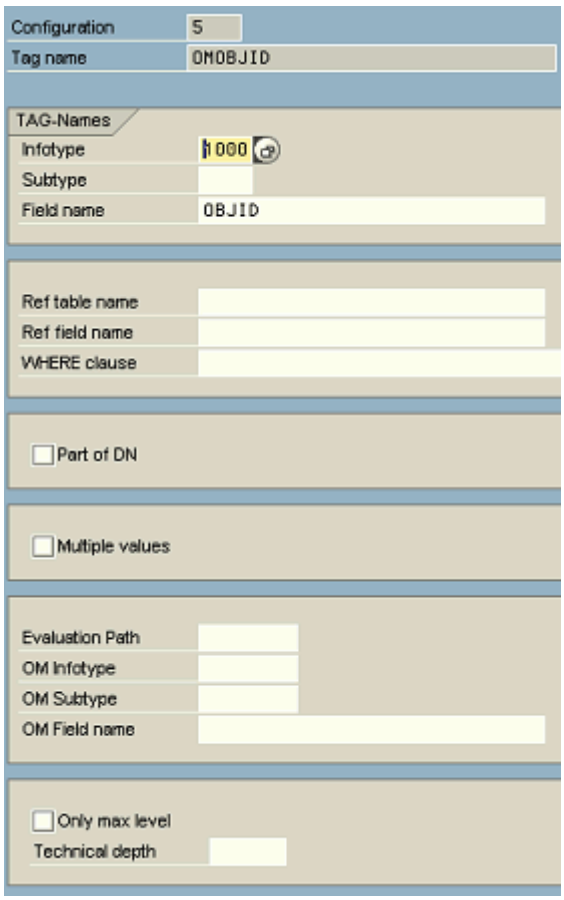


Figure 21. OM OBJID Tag

Shown below is a valid "attributes" configuration:

It uses the (common) "O-S-P" evaluation path (selectable via the F4 help in the SAPAgent's tag detail configuration) to travel from a given OU via (even multiple sub OUs) to a position and then to the associated employee:

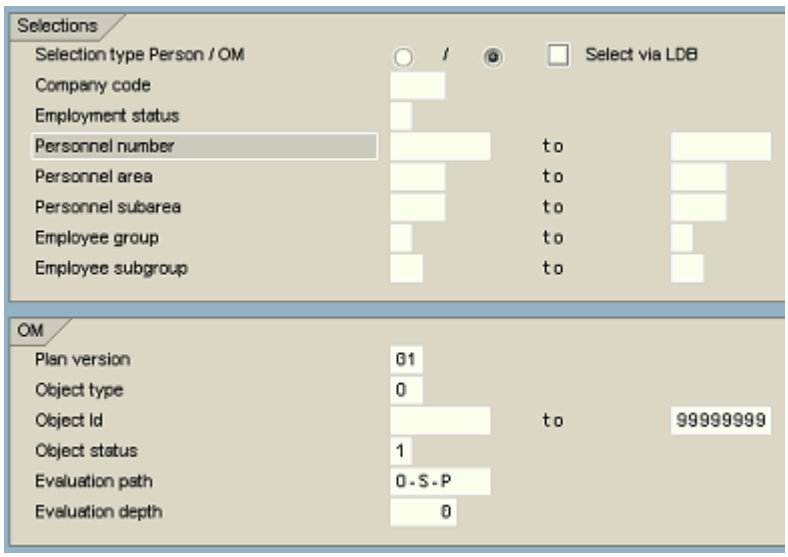


Figure 22. Figure : Attributes Configuration

However, you may wonder why you get the same (one) employee multiple (2) times in your export file. This is so because it will be found beneath organizational units **myroot** and **mysubroot1**.

Therefore a more elegant way would be to restrict only to the root(s) of the organizational units:

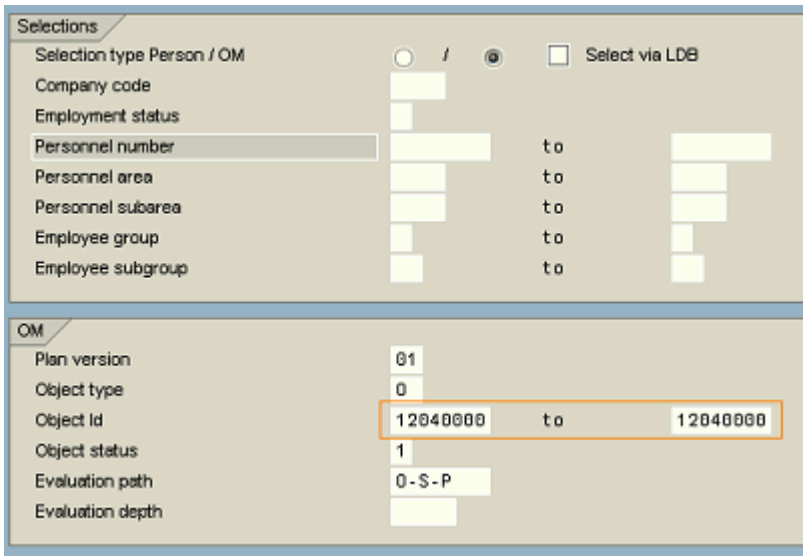


Figure 23. Attributes Configuration for Organizational Units as Root

The complete export would then look like this:

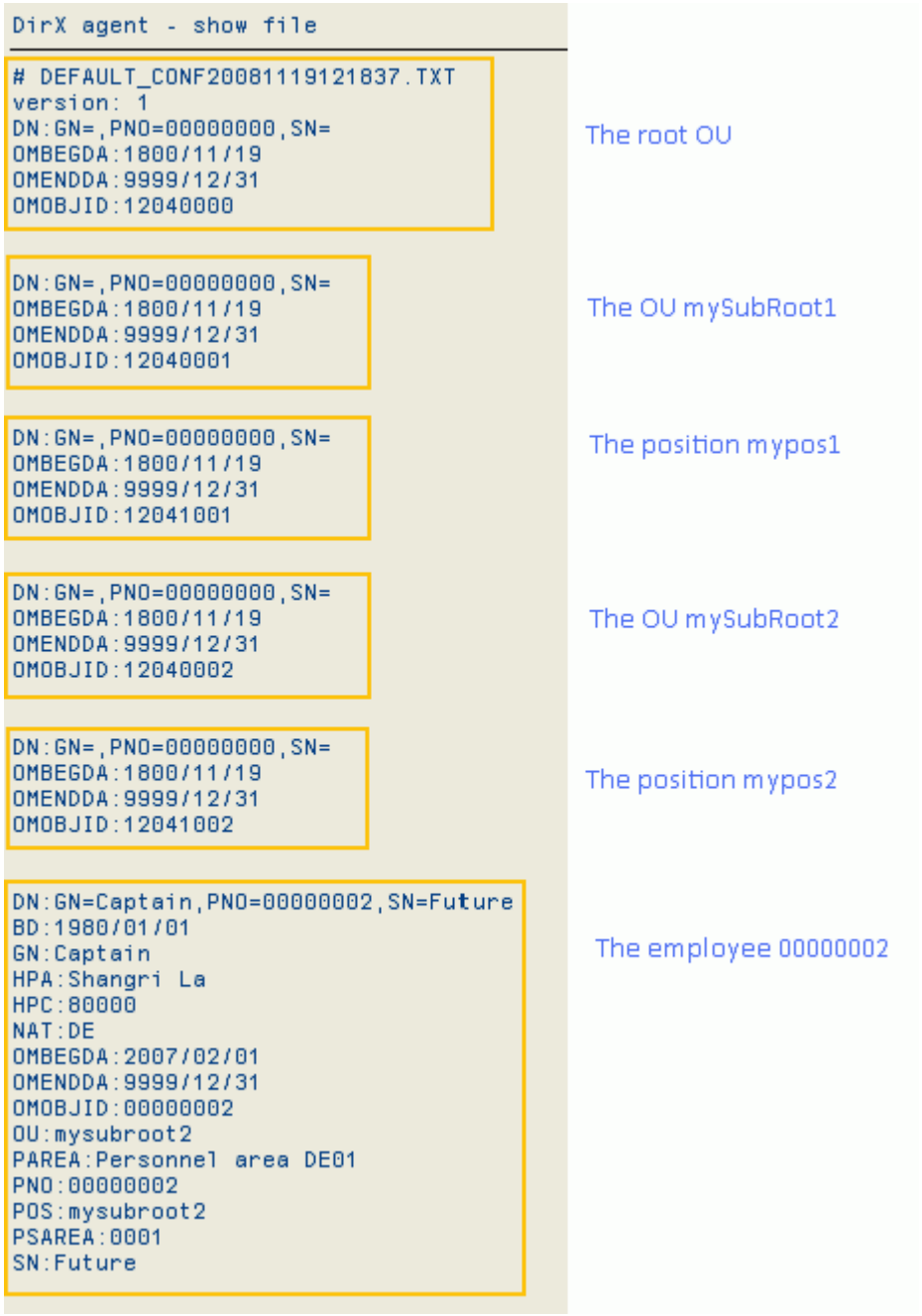


Figure 24. OM Export

3.9.11. Export File Formats

SAPAgent generates the following export data file formats:

- CSV format
- LDIF content format
- LDIF change format
- LDIF change format with modified changetype:modify records

CSV and LDIF content are used for full exports, and LDIF change format is used for delta exports.

3.9.11.1. CSV Format

CSV format is used for full exports that are to be processed with Microsoft Excel. SAPAgent full export data files in CSV format have the following characteristics:

- The first line of the file contains the column names.
- All attributes (items) of an entry (object) are distinguished by a separator character; this separator character can be customized. All attributes are embedded into a pair of double quotes ("") to guarantee that MS Excel can read the information even if there are special characters embedded; if an attribute contains a double-quote character, the character is replaced by a """ sequence.
- Special characters are output in C notation using the backslash (\) escape character. A backslash itself is transformed into a \\ sequence.
- Entries are separated by different lines using a CR/LF end-of-line marker. The first line names the attributes; the following lines contain the entries.
- Multiple attribute values are held in one field separated by a second separator character.
- Subtypes are treated equally; the subtype field specifies the content.

Here is an example of SAPAgent CSV format where separator 1 is the comma (,) and separator 2 is the semicolon (;):

```
UserName,Comment,FullName,UserID
"Test1",,"Hugo Test1,"1234"
"Test2","second";"and last","Hugo Test2","1235"
```

3.9.11.2. LDIF Content and Change Formats

SAPAgent LDIF content and change formats have the following general characteristics:

- All attributes (items) of an entry (object) are placed in separate lines. Each line consists of the attribute's name and value, separated by a colon (':').
- Empty lines separate entries.
- Multiple instances of the same entry appear on separate lines with the same tag.
- Any group of key attributes related to one person is identified by a distinguished name (**dn**) line. When present in an LDIF change file, it is marked by a "changetype" value. The changetype on the DN is not supported. The format of the distinguished name line is:

  dn: cn=ID

  The primary key is always the personnel number but can be extended by other key attributes (as specified during horizontal selection). For example:

  dn: PNR=00000027, SN=Winter

SAPAgent supports the following change types:

- add - All attributes (tags/values) are supplied for the new entry.
- modify - Any attribute/value (tag /value) can follow. If prefixed with "add", the attribute will be included and the attribute/value (tag/value) pair follows in the next line. If prefixed with "delete" the entire attribute will be deleted. If prefixed with "replace", the attribute (tag) is replaced with the new value (equivalent to a "delete" then an "add").
- delete - The entry (object) is to be deleted.
- modrdn - The entry's relative distinguished name has changed (with checked "Create MODRDN" in section "The Other Attributes Area" above).

If „All tags in Modify-Record“ in section "The Other Attributes Area" above is checked then the changetype:modify record does not follow the LDIF change format rules but looks like an changetype:add record with tag/value pairs:

```
DN:HPC=80338,PNO=00001995,SN=Schmidt
CHANGETYPE: MODIFY
ADRBEG:2005/10/28
ADREND:2005/10/31
BD:1944/08/12
GN:Hans-Peter
HPA:München
HPC:81200
HSTA:Franz-Josef-Strauß-Str. 1
O:IDES AG
OU:Sales Office 1000 Frankfurt
PAREA:Frankfurt
PNO:00001995
SN:Schmidt
```

Here is an example of SAPAgent LDIF content format:

```
version: 1
dn: GN=Dieter, PNO=00000002, SN=Meyer\\Walter
BD: 1954/10/31
BLD:
CN:
COM:
EXT:
FTN:
GN: Dieter
HPA: Frankfurt,Main
```

```
HPC: 60318
HSTA: An der großen Brücke 5
HTN:
I:
```

Here is an example of SAPAgent LDIF change format:

```
version: 1
dn: GN=Dieter, PNO=00000002, SN=Meyer\\Walter
changetype: modify
delete:HPA
-
delete:HPC
-
delete:HSTA
-

dn: GN=Dieter, PNO=00000002, SN=Meyer\\Walter
changetype: delete
-
```

3.9.12. Logging

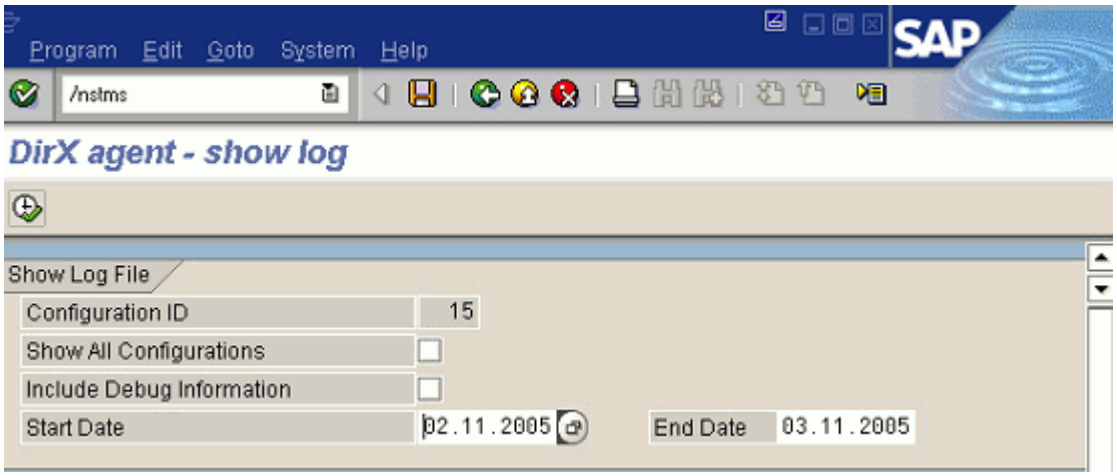
SAPAgent logs the following master information for every activity:

- Unique Key
- Activity
- Date/time-stamp
- Executing User-Id

SAPAgent logs the following information for export operations:

- Success
- Filename
- Selection criteria
- Number of records affected, type of affection
- Number of records in error

Use the **Show Log Information** selection in the **Extras** menu to display log information. SAPAgent displays the Log view dialog:



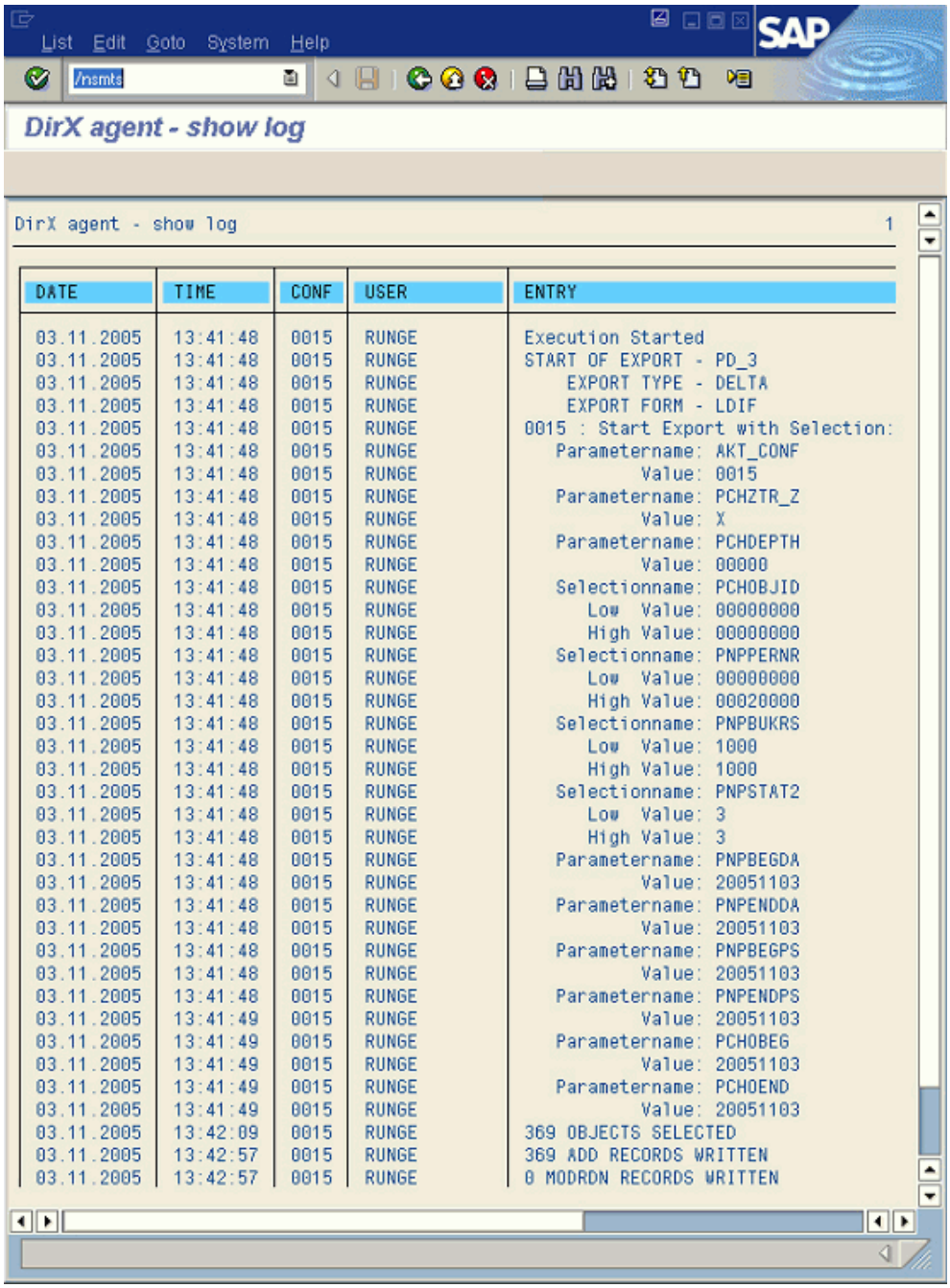
The Log view dialog box contains the following fields:

Show all configurations - By default, SAPAgent displays log information from the currently selected configuration. Check this field to display log information from all configurations.

Include Debug Information - By default, SAPAgent displays log information about normal operation. Check this field to include advanced information about SAPAgent internal processing in the log file display. This information can be helpful during the evaluation of error situations.

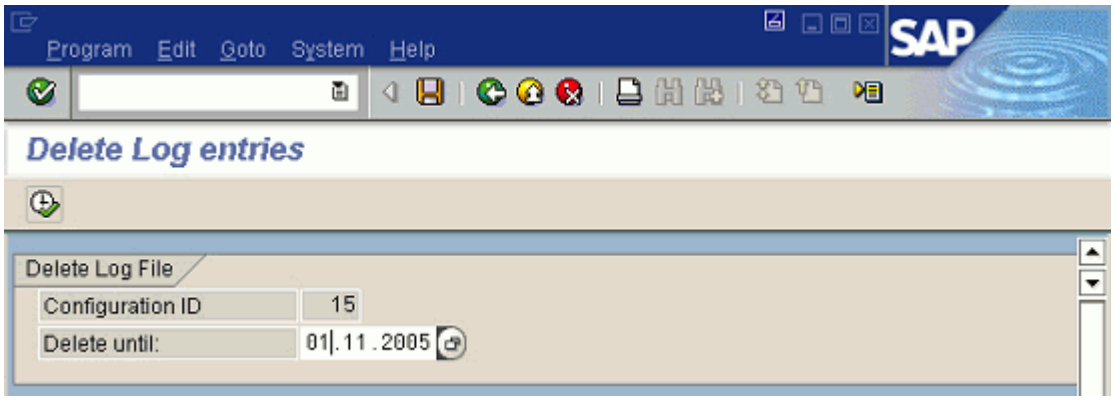
Start Date and **End Date** - Use the **Start Date** and **End Date** fields to display log information that was generated during a specific period of time.

Click the Execute button or press F8 in the Log View dialog to generate a log file display. Here is an example of a log file display:



To delete log file entries up to a specified date:

1. In the **Extras** menu, select **Delete Log Information**. The **Delete Log entries** dialog appears.

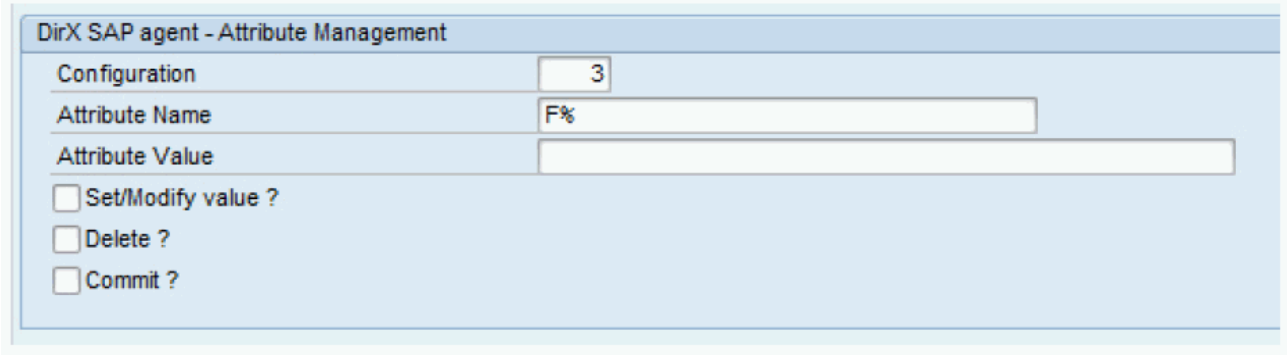


2. In **Delete until**, select the date before which all log file entries are to be deleted.

3.9.13. Manually Inspecting and Maintaining Attributes

The SAP Agent saves customizing (and other items) in the table /SIE/DIRXAGCONFA. The agent typically maintains this table automatically in the background. However, during problem analysis it is helpful to inspect or even modify entries manually. Report /SIE/DIRXAG_ATTRIBUTES is intended for such situations. Here are some examples.

View attributes with names starting with F of interface 3:

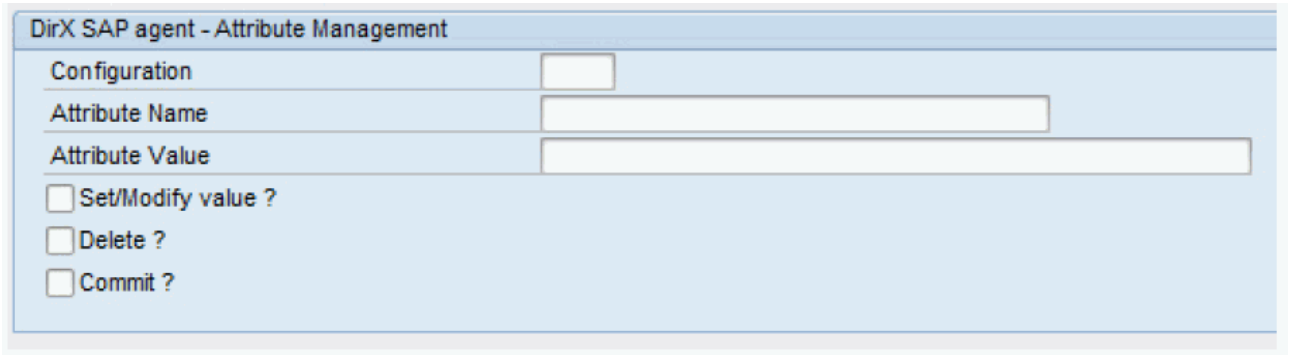


Here is the sample output:

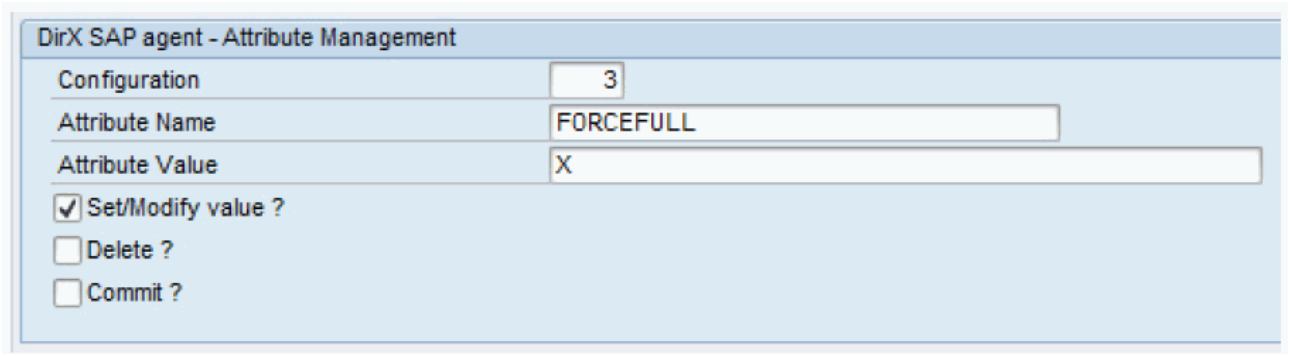
```
DirX agent - Attributes Management


Attribute Management:
0003 FILEFORMAT                          LDIF
0003 FILENAME                            DEFAULT_CONF_OM.TXT
0003 FILETYPE                            L
0003 FORCEFULL
0003 FULLDATE                            20131126
0003 FUTURED                             0000
0003 FUTURED2                            0000
```

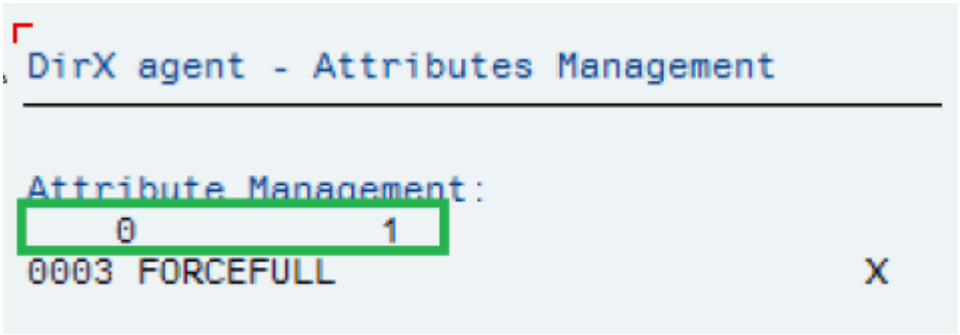
View all attributes:



Set the value of an existing attribute or create a new attribute in interface 3 in test mode (no commit):

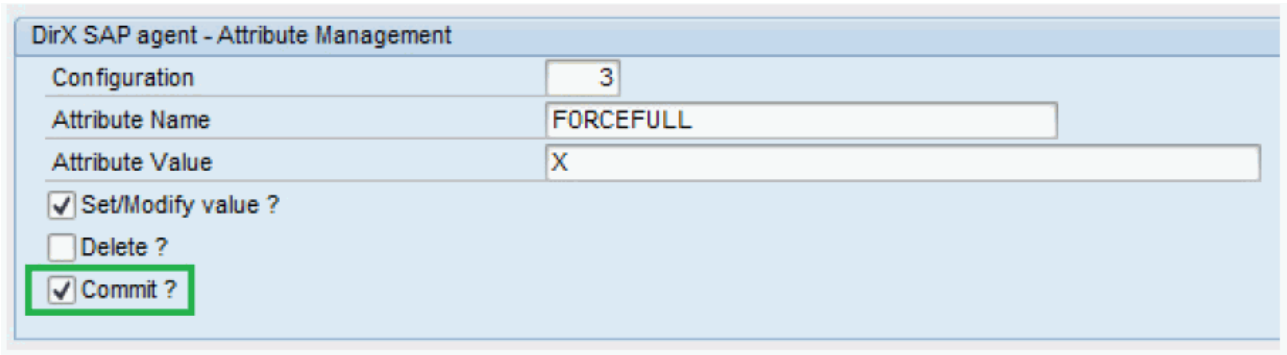


Here is the sample output:



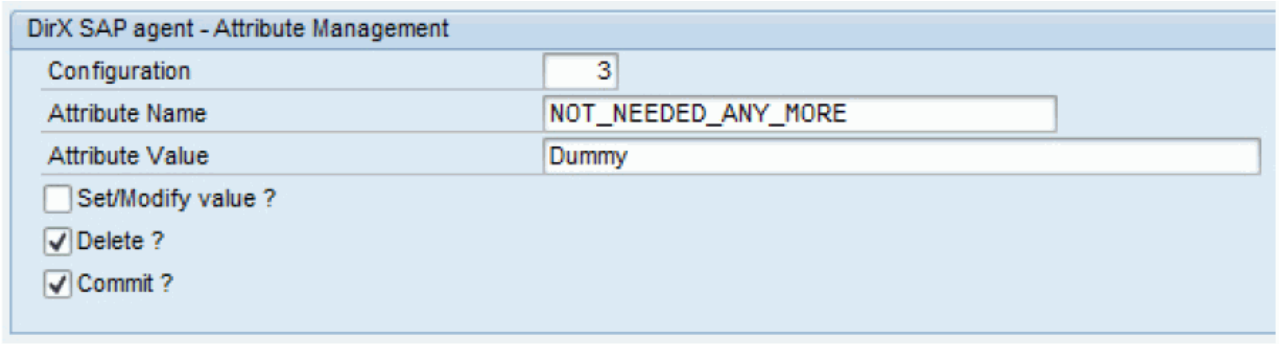
A status line with two numbers is output for all operations that modify data. The first number indicates the execution result, with 0 indicating success and any other value indicating a problem. The second number indicates the number of related records, usually 1. The resulting record(s) are listed after the status line.

Set the value of an existing attribute or create a new attribute in interface 3 in real mode (with commit to the database):



Here is the sample output:

Delete an existing entry:



Here is the sample output:

If the above deletion run were to be repeated, the result would be:

```
DirX agent - Attributes Management

Attribute Management:
     4              0
Changes Commited.
```

Because no matching entry to be deleted could be found (the first number **4** indicates an error; the second number **0** indicates no records were changed.

Extra care should be taken when using this tool. Also note that changes applied by this tool are not recorded in a customizing transport.

3.10. SAP ECC UM Agent

SAP-ECC-UMAgent is the DirX Identity agent that handles the synchronization of SAP user entries from a SAP ECC database with the Identity Store.

SAP-ECC-UMAgent is implemented in Java. SAP-ECC-UMAgent supports ECC 6.0, SAP S/4HANA (1709 FPS1) on-premise and higher and runs with all NetWeaver (ABAP stack) platforms that are supported by the SAP Java Connector and by DirX Identity. The underlying interface is upwards compatible from SAP R/3 to SAP ECC.

The employed "USER" BAPI methods are also generally applicable to the SAP Central User Administration (CUA).

With SAP S/4HANA the user management has been extended to include Business User Management. With that you can update the business user through multiple channels. There is a customization view available that defines the maintenance source of the workplace address attributes of a (business) user. If the source is set to User Management then the entire user attributes can be managed by the SAP ECC UMAgent/Connector. Attributes in this view where the source is not set to user management can only be read by the SAP ECC UMAgent/Connector. (See SAP note 2570961 for more details.)

SAP-ECC-UMAgent can:

- Perform a full export of users, roles (activity groups) and profiles from an SAP ECC system, including the references to all assigned roles and profiles for users.
- Perform a delta import of users into an ECC system, including creation of users, modification of user attributes, modification of a user's role and profile assignments and deletion of users.
- Generate a trace file (for tracing, reporting which objects were processed and the operations that failed)

Prerequisites

The SAP-ECC-UMAgent requires:

- The SAP Java Connector (JCo) - to be installed on the machine where the UMAgent (that is, DirX Identity server) should run. The SAP Java Connector is a toolkit that allows a Java application to communicate with any SAP system. Unfortunately, the redistribution of the SAP JCo is not allowed, but the Java Connector can be downloaded free of charge at SAP's Support Portal (https://support.sap.com → Products → Connectors → SAP Java Connector). If you do not have a login for the SAP Support Portal, ask your SAP administrator or request the data by SAP.

You must install the 64bit JCo.

For the latest information on JCo 3.1.4 or higher, see the release notes or SAP note 2786882 "SAP JCo 3.1 release and support strategy".

On Windows platforms, JCo 3.1 requires the Visual Studio 2013 C/C runtime libraries to be installed on the system. If not present, download and install the 64bit "Visual C 2013 Redistributable Package" from the Microsoft knowledge base article https://support.microsoft.com/en-us/help/4032938. See the latest JCo documentation and/or SAP note 2786882.

Pay attention to the installation notes that come with the JCo distribution.

If you want to use the SAP-ECC-UMAgent for real-time provisioning or in the password synchronization scenario, you must copy the *sapjco3.jar* file **additionally** into the folder _
install_path_*/ids-j/confdb/jobs/framework/lib*.
For Windows and Linux platforms, you must remove the file **sapjco.jar** from the folder _
install_path_*/ids-j/confdb/jobs/framework/lib*,
if it still exists there.

(Windows only): Please do not copy the sapjco3.dll into the *windows-dir*\system32* directory. This could break the operability of other JCo versions that are already installed on the same system. Furthermore, you would risk that the current installation also would not work anymore if the sapjco3.dll is replaced in the *windows-dir*\system32* directory in the future. Instead, you must add the *sapjco3-install-path* to the PATH environment variable.
Additionally pay attention to set the system environment CLASSPATH variable with the full pathname including filename of sapjco3.jar.

- In order to run properly, the account that the agent uses to connect must have the rights for general user management (create, edit, display, lock/unlock, and delete user; S_USER_GRP for user groups (can be limited to certain user groups), S_USER_AGR for roles and S_USER_PRO for profiles) and the right to read the following tables (general authorization object is S_TABU_DIS):

non-CUA environment: USR10, USR11, AGR_DEFINE, AGR_TEXTS
CUA environment: USRSYSPRF, USRSYSPRFT, USRSYSACT, USRSYSACTT. We recommend defining a new authorization group for these tables as you can assign only authorization groups to the field DICBERCLS of S_TABU_DIS and far too many tables are in the relevant authorization groups SC and SS. With S_TABU_NAM the accessible tables can be defined.
Additionally, the UM connector retrieves information from the ECC system's data dictionary. In order to do this, the account that the agent uses needs the following access rights granted:
(Authorization Object: S_RFC, ACTVT: 16, FUGR)
ECC Release Function Groups:
RFC1, SDIFRUNTIME, SG00, SRFC, SYST, SYSU.
For requesting the data, the agent uses RFC_READ_TABLE that belongs to function group SDTX. You must grant the access rights (authorization object: S_RFC, ACTVT: 16, FUGR) also to SDTX.
For changing the productive password and not using SNC (option **setProductivePwdAtModDirectly** is set to false), the agent uses SUSR_USER_CHANGE_PASSWORD_RFC that belongs to function group SUSO. You must grant the access rights (authorization object: S_RFC, ACTVT: 16, FUGR) also to SUSO.

- The agent uses the port 33*xx* if it is connecting to an application server without any gateway server or SAP router, where *xx* stands for the SAP gateway number / system number to connect to the SAP system (eventually a firewall administration is necessary). (See the section "General Notes" below for details.)
- If you want to export the lock status of a user (attribute ISLOCKED.xxx), the following SAP support packages are required: Release 6.10: SP 43, Release 6.20: SP 51, Release 6.40: SP 12, Release 7.00: SP 01 (see SAP Note 826050 for more information).
- The SAP-ECC-UMAgent uses the Java runtime environment (JRE) for DirX identity which is located in *dxi_java_home*.

Secure Connection

You can use Secure Network Communications (SNC) and the "SAP Cryptographic Library" to secure the connection to the SAP system application server. The "SAP Cryptographic Library" is available in the SAP Service Marketplace for software downloading (http://service.sap.com/download; then follow the link to "SAP Cryptographic Software"). If you do not have a login for the SAP Service Marketplace, ask your SAP administrator or request the data from SAP. SNC is called in the native code of JCo. Therefore, you must download and use it for your operating system.

See the section “Installing and Configuring SNC Connections” for more information.

Restrictions

The agent implementation has the following restrictions:

- SAP roles and profiles cannot be created or modified (due to missing appropriate interfaces).

- Productive password update on CUA systems:
- No productive password update is propagated onto CUA child systems unless you use the option **setProductivePwdAtModDirectly** set to true in conjunction with SNC. If you are using a CUA system in conjunction with an SAP Enterprise Portal, this is normally not an issue.
  Workaround: In a password scenario without the use of SAP Enterprise Portal, each CUA child system should be configured as a single target system.
- In conjunction with SAP Enterprise Portal: Password update is only possible when the password authentication store is either the CUA central system or an LDAP directory.
- Search on profiles on a CUA system: only the fields “profn”, “subsystem” and “typ” can be exported.
- The search for user lock status needs extra SAP support packages.

Changes

The agent now uses the SAP module function BAPI_USER_GETLIST for searches to export users and no longer accesses the SAP tables USR02 (non-CUA system) and USZBVSYS (CUA system) directly. As a result, the lock status can only be exported via the ISLOCKED attributes (see the support package referenced above). USR02.UFLAG (non-CUA system) and USZBVSYS.STATUS USZBVSYS.SUBSYSTEM (CUA system) are no longer supported. This is a change to previous versions.

The agent exports only valid role assignments of a user. Roles assigned in the SAP system that are only valid in the past or in the future are not exported. This is a change to previous versions. If you still want the old behavior, you must set the attribute “onlyValidRoles” to false. You only get correct results if the client is in the same time zone (including daylight savings time) as the ECC server.

The agent now uses the parameter SELF_REGISTER in the function module BAPI_USER_CREATE1 for creating a user with a productive password per default. The old mechanism in 2 steps (initial password and then setting the productive password via SUSR_USER_CHANGE_PASSWORD_RFC) is still available. This is a change to previous versions.

Overview

The following figures illustrate the components of the SAP-ECC-UMAgent export and import operations.

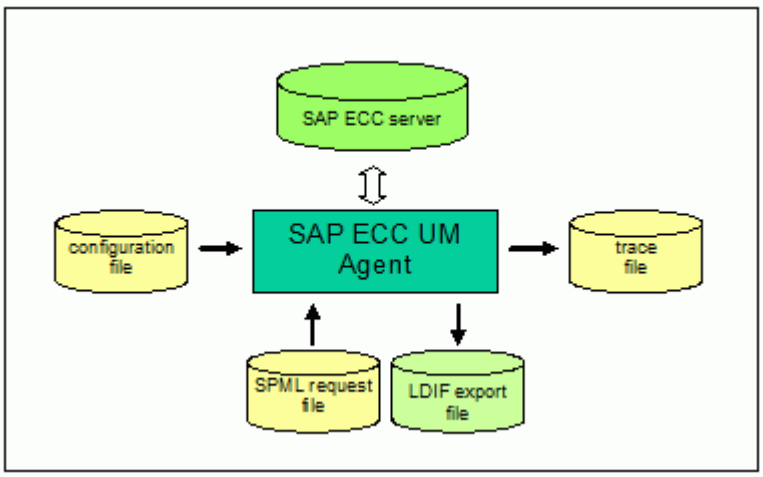


Figure 25. SAP-ECC-UM-Agent Export Components

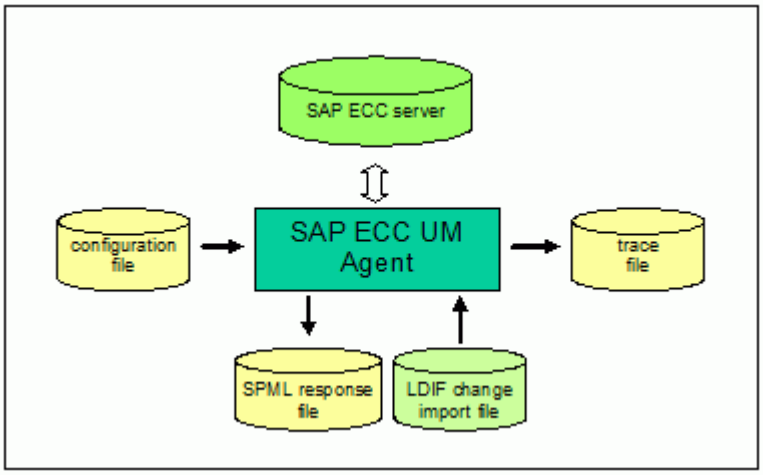


Figure 26. SAP-ECC-UM-Agent Import Components

This section describes:

- SAP-ECC-UMAgent command line format for export and import operations
- SAP-ECC-UMAgent configuration files for export and import operations
- The export data file format that SAP-ECC-UMAgent generates
- The import data file format that SAP-ECC-UMAgent recognizes
- The search request file format that SAP-ECC-UMAgent recognizes
- General Notes

3.10.1. Command Line Format

The command line format to invoke SAP-ECC-UMAgent is as follows:

SAPUMAgent.bat *configuration_file*

3.10.1.1. Parameters

configuration_file

Specifies the name of the file that contains the specifications for the export or import procedure. With the exception of the search criteria in export mode (which are described in a separate Service Provisioning Markup Language (SPML) file), all parameters of SAP-ECC-UMAgent operation are defined in the agent's configuration file, in XML format.

The following table describes the codes provided when SAP-ECC-UMAgent finishes running:

Exit Code	Description
0	SAP-ECC-UMAgent completed successfully.
1	SAP-ECC-UMAgent completed with errors, which are described in the specified *tracefile* unless this file cannot be created due to a file exception error.
60	SAP-ECC-UMAgent completed with warnings, which are described in the specified *tracefile*.

3.10.2. Configuration File Formats

SAP-ECC-UMAgent uses the following configuration files:

- ECC UM export configuration file - controls the export of data from a SAP R/3 system
- ECC UM import configuration file - controls the import of data into an a SAP R/3 system

Templates of these configuration files are provided with the Agent installation. The filenames are:

- **ImportConfig.xml** (to import user and user to role and profile assignments)
- **ExportConfig.xml** (to export users or profiles or roles)
- **SearchRequest.xml** (contains the search request to select the objects for export)

In general, you have to customize these files to support the requirements of your SAP R/3 system import and export operations.

This section also describes the general structure of a configuration file.

3.10.2.1. General Structure of a Configuration File

A SAP-ECC-UMAgent configuration file is in XML format.

The SAP-ECC-UMAgent is composed of multiple sub-units (connectors), which are configured in the configuration file. Different types of connectors are used for export and import. Consequently, you must not change the general structure of SAP-ECC-UMAgent import/export configuration files. Instead, you configure some well-defined attribute values to the specific environment in which the agent runs.

Tags

The configuration files contain the tags *job*, *connector*, *logging* and *connection*.

- *job* - Defines the file's document tag, with connector sub-tags
- *connector* - Configures the properties of one connector, has connection and/or logging sub tags
- *connection* - Configures connection parameters, for example, filename for a reader/writer or host/port/credentials for a network connector
- *logging* - Configures the logging properties of a connector

Attributes

A *connector* tag can have the following attributes:

- *name* - The connector's name
- *role* - One of reader, controller, connector, RequestCryptTransformer, or responseWriter
- *className* - The name of the Java class that implements the connector

The connection parameters of the specific connectors are described in their *connection* sub-tags.

Each *connection* tag has the attribute

- *type* - The type of connection (file format, protocol)

Readers and response writers are configured by the attribute

- *filename* - The pathname of the input or output file.

The SAP_ECC_UM connection is configured by the attributes

- *logonVariant* – indicator for logon variants:
- 0 = no load balancing (direct connection to the SAP instance)
- 1 = no load balancing but via gateway
- 2 = with load balancing

User logon properties:

- *user* - The logon user for binding to the ECC system
- *password* - The logon user password
- *client* - The client number (3 digits)
- *language* - The language that is used in response messages from the ECC system, if not defined the default user language is used.

Configuration for physical connection:

Direct connection to SAP instance:

- *server* - The host name or IP address of the ECC application server
- *systemID,systemIDgateway* - The system identification number (2 digits) or the name

of the SAP system

- *gwhost* - Host name of the SAP gateway
- *gwserv* - Service number of the SAP gateway

Load balancing connection to a group of SAP instances:

- *server* - The host name / IP address of the message server
- *r3SystemName* - Name of the SAP system
- *group* - Name of the group of application servers
- *msserv* - SAP message server port, optional for a logon balancing connection

SAP router string can be used in both cases if the SAP system is behind a SAP router:

- *saprouter* - SAP router string

SNC configuration:

- *snc_mode* - Specifies SNC mode (true or false)
- *snc_lib* - The path and file name of the cryptographic library
- *snc_partnername* - The application server's SNC name
- *snc_myname* - The client's SNC name

Destination configuration:

- *poolCapacity* - (formerly *maxConnections*) The maximum number of idle connections kept open by the destination. A value of 0 has the effect that there is no connection pooling (default 3).
- *peakLimit* - The maximum number of active connections that can be created for a destination simultaneously (default 6).

SAP tracing:

- *RFC_TRACE* - A boolean switch to enable/disable RFC trace.
- *CPIC_TRACE* - Enable/disable CPIC trace (0..3).

Behavioral configuration:

- *accesstoCUA* - A boolean switch to specify whether the target system is a single ECC system or a ECC CUA system. The default is **false**.
- *combinedRoleProfileSubsystem* – A boolean switch to specify whether (true) or not (false) combined role#subsystem or profile#subsystem names is used. The default is **false**.
- *blankValues* – A boolean switch to specify whether (true) or not (false) blank values in attributes are exported. The default is **false**.
- *trim* – A boolean switch to specify whether (true) or not (false) values in attributes are trimmed (delete blanks at the beginning and the end of a value). The default is **true**.

- *onlyValidRoles* - A boolean switch to specify whether (true) or not (false) role assignments that are valid on the day of the export are exported (false; meaning also in the past or future). The default is **true**.
- *dontuseCacheResults* - A boolean switch to specify whether (true) or not (false) internal BAPI_USER_GET_DETAIL calls set the import parameter CACHE_RESULTS to " " (blank). The default of the SAP interface method is "X". The default is **false**.
- *searchSapServiceUser* - A boolean switch to specify whether (true) or not (false) certain SAP system accounts like "SAP*" should be exported. The default is **false**.
- *directlyAssignedRolesOnly* - A boolean switch to specify whether (true) or not (false) only directly assigned roles are exported. If false, the export result contains all single roles for a composite role (indirectly assigned roles). So there is no difference visible in directly assigned single roles and indirectly single roles via a composite role. The default is **false**.
- *setProductivePwdAtAddDirectly* - A boolean switch to specify whether (true) or not (false) the internal BAPI_USER_CREATE1 uses the SELF_REGISTER flag to set a productive password in one step. The default is **true**.
- *setProductivePwdAtModDirectly* - A boolean switch to specify whether (true) or not (false) the internal BAPI_USER_CHANGE uses the PRODUCTIVE_PWD flag to set a productive password in one step. Notice that this requires an SNC connection. The default is **false**.
- *tryLoginAsUser* – A boolean switch to specify whether (true) or not (false) a login as the account with the given new password is processed as part of a productive password modify operation. The default is **true**.
- *doCommit* - A boolean switch to specify whether (true) or not (false) a BAPI_TRANSACTION_COMMIT call is executed in add and modify requests. This is necessary if you want certain user changes be reported in change log records. The default is **false**.
- *useCombinedAttributeForParameter* - A boolean switch to specify whether (true) or not (false) combined values is used (key=value pair) for the PARAMETER1 table. This is necessary if you want to set multiple key-value pairs. The key is assigned to the field PARAMTER1.PARID and the value is assigned to the field PARAMETER1.PARVA. The default is **false**.
- *combinedAttributeForParameter* - The name of the attribute from which or to which a combined key-value pair is read or written. There is no default (the attribute is only relevant if *useCombinedAttributeForParameter* is set to **true**).
- *useAdditionalRoleParameters* - A boolean switch to specify whether (true) or not (false) additional SAP role parameters are accepted when both CUA and *combinedRoleProfileSubsystem* are set. By default, only “dxrRole.NAME” is allowed. When this switch is set to **true**, “dxrRole.TO_DAT” and “dxrRole.FROM_DAT” are also allowed. Note that you must provide an equal number of NAME and other values, also with the same changetype (add, replace). The default is **false**.
- *doUserLockedRetry* – The number of retries an operation (Add, Modify, Delete) is repeat if the user is currently locked. When modifications via BAPI calls are performed in sequence, it can happen that the user to modify is still locked by a previous call (asynchronous handling of operations). To manage this problem, use this parameter in

conjunction with the *doUserLockedInterval* parameter. The default is **15**. If set to **0**, no retry is done.

- *doUserLockedInterval* – The number of milliseconds between retries. The default is **200** milliseconds.

The agent's logging is configured in the controller's logging tag by the attributes:

- *level* - The integers 0-9, where 0 indicates no logging and 9 indicates full logging
- *filename* - The name of the trace file

3.10.2.2. Export Configuration File Format

The export configuration file has the format defined in the general section. The following template describes its configuration. The attribute values that you can configure are shown in bold (blue) italic, for example, ***level***:

```
<?xml version="1.0" encoding="UTF-8" ?>
<job>
 <connector name="Default Controller" version="0.1" role="controller"
className="siemens.dxm.connector.framework.DefaultControllerStandalon
e">
    <logging level="level" filename="tracefilename" />
</connector>
<connector role="reader" name="SPML file reader"
className="siemens.dxm.connector.framework.SpmlFileReader">
    <connection type="SPML" filename="SPMLinputfile" />
  </connector>
  <connector role="connector"
className="siemens.dxm.connector.sapUM.sapUMuser"
name="SAP UM Agent" version=”2.00”>
    <connection type="SAP_ECC_UM"
        user="account"
        password="password"
        server="server">
      <property name=”client” value=”client number”/>
      <property name=”systemID” value=”system number”/>
      <property name=”systemIDgateway” value=”system number”/>
<property name=”gwhost” value=”gateway server”/>
<property name=”gwserv” value=”gateway service number”/>
<property name=”group” value=”group name”/>
      <property name=”r3systemName” value=”ECC System name”/>
<property name=”logonVariant” value=”0 or 1 or 2”/>
      <property name=”language” value=”language ISO code”/>
```

```
      <property name="accesstoCUA" value="false or true"/>
<property name="combinedRoleProfileSubsystem"
         value="false or true"/>
<property name="maxConnections" value="number"/>
<property name="blankValues" value="false or true"/>
      <property name="snc_mode" value="false or true"/>
      <property name="snc_lib" value="path"/>
      <property name="snc_partnername" value="p:distinguished name"/>
     </connection>
  </connector>
  <connector role="responseWriter" name="LDIF File writer"
classNam="siemens.dxm.connector.framework.LdifFileWriter">
    <connection type="LDIF" filename="outputFile" />
  </connector>
 </job>
```

level

level specifies how much information the messages in the trace files provide. The value is an integer in the range 0 to 5 and 9.

Level	Type of Messages Logged
0	none
1	FatalError and Error
2	FatalError, Error and Warning
3	FatalError, Error and Warning
4	FatalError, Error and Warning
5	FatalError, Error, Warning and Trace
9	FatalError, Error, Warning and Trace (and additional HTML files)

tracefilename

tracefilename specifies the pathname of the trace file.

SPMLinputfile

SPMLinputfile specifies the pathname of the Service Provisioning Markup Language (SPML) file that contains the search request.

className

ClassName specifies which object type is processed:

siemens.dxm.connector.sapUM.sapUMuser for users,
siemens.dxm.connector.sapUM.sapUMactgroups for roles,
siemens.dxm.connector.sapUM.sapUMprofile for profiles.

siemens.dxm.connector.sapUM.sapUM as a common class. In this case a prefix "USER:", "ROLE:", or "PROFILE:" must be provided in the SPML identifier part.

account

account specifies the account to be used for connecting to the ECC system.

password

password specifies the password to be used for connecting to the ECC system.

client

client specifies the client number to be used for connecting to the ECC system.

language

language specifies the language that is used in ECC response messages (SAP ECC Language ID in accordance with ISO 639).

logonVariant

logonVariant indicates the variant of the connection:

- **0** = no load balancing: connect to an application server or via SAP router
- **1** = no load balancing but via gateway: connect to a gateway server
- **2** = with load balancing: connect to a message server

server

server specifies the host name or the IP address of the application server (logonVariant 0 or 1) or the host name/ IP address of the message server (logonVariant 2).

The host name and the service name of the application or message server must be defined in the *hosts* and *services* files:

- logonVariant 0 or 1: <service name> = **sapdp**<system number>
- logonVariant 2: <service name> = **sapms**<ECC system name>.

systemID, systemIDgateway

systemID,systemIDgateway specifies the system number to be used for connecting to the ECC system (logonVariant **0** or **1**).

gwhost

gwhost specifies the host name or the IP address of the SAP gateway server (logonVariant **1**).

The host name and the service name of the SAP gateway must be defined in the *hosts* and *services* files. If GWHOST and GWSERV are not specified the service name of the SAP gateway must be defined in the services file (<service name> = **sapgw**<system number>).

gwserv

gwserv specifies the gateway service number (logonVariant **1**). For example: "sapgw00".

r3SystemName

r3SystemName is the name (system ID) of the ECC system (logonVariant **2**).

group

group specifies the name of the group of application servers (logonVariant **2**).

msserv

msserv specifies the SAP message server port, optional for a logon balancing connection.

sapRouter

sapRouter specifies a string for connection to systems behind a SAP Router. SAP Router string contains the chain of SAP Routers and its port numbers and has the form:

(/H/*host***)+**

poolCapacity

poolCapacity specifies the maximum number of idle connections kept open by the destination. A value of **0** has the effect that there is no connection pooling (default value is **3**)

peakLimit

peakLimit specifies the maximum number of active connections that can be created for a destination simultaneously (default value is **6**)

snc-mode

snc_mode specifies whether or not SNC is used.

snc_lib

snc_lib specifies the path and file name of the SAP Cryptographic library.

snc_partnername

snc_partnername specifies the application server's SNC name.

snc_myname

snc_myname specifies the client's SNC name.

combinedRoleProfileSubsystem

combinedRoleProfileSubsystem is a boolean switch to specify whether the agent should process combined "<role name>#<subsystem>" or "<profile name>#<subsystem>" names for the assignment of roles or profiles in a CUA environment.

accessToCUA

accessToCUA specifies whether the target system is a single ECC system or a ECC CUA system.

blankValues

blankValues specifies whether (true) or not (false) blank values in attributes should be

exported. The default is **false**.

trim

trim specifies whether (true) or not (false) values in attributes should be trimmed; that is blanks at the beginning and the end of a value are deleted. The default is **true**.

onlyValidRoles

onlyValidRoles specifies whether (true) or not (false; meaning also in the past or future) role assignments that are valid at the day of the export is exported. The default is **true**.

searchSAPServiceUser

searchSAPServiceUser specifies whether (true) or not (false) certain SAP system accounts like "SAP*" is exported in a search request. The default is **false**.

dontUseCacheResults

dontUseCacheResults specifies that for the internal used BAPI Call BAPI_USER_GET_DETAIL in modify or search requests the import parameter CACHE_RESULTS is set to " " (blank). See SAP note 1101858 for more details. The default is **false**.

directlyAssignedRolesOnly

directlyAssignedRolesOnly - A boolean switch to specify whether (true) or not (false) only directly assigned roles are exported. If false, the export result contains all single roles for a composite role (indirectly assigned roles). So there is no difference visible in directly assigned single roles and indirectly single roles via a composite role. The default is **false**.

setProductivePwdAtAddDirectly

setProductivePwdAtAddDirectly - A boolean switch to specify whether (true) or not (false) the internal BAPI_USER_CREATE1 uses the SELF_REGISTER flag to set a productive password in one step. The default is **true**.

setProductivePwdAtModDirectly

setProductivePwdAtModDirectly - A boolean switch to specify whether (true) or not (false) the internal BAPI_USER_CHANGE uses the PRODUCTIVE_PWD flag to set a productive password in one step. Notice that this requires an SNC connection. The default is **false**.

tryLoginAsUser

tryLoginAsUser - A boolean switch to specify whether (true) or not (false) a login as the account with the given new password is processed as part of a productive password modify operation. The default is **true**.

doCommit

doCommit - A boolean switch to specify whether (true) or not (false) a BAPI_TRANSACTION_COMMIT call is executed in add and modify requests. This is necessary if you want certain user changes be reported in change log records. The default is **false**.

useCombinedAttributeForParameter

useCombinedAttributeForParameter – A boolean switch to specify whether (true) or not (false) combined values are used (key=value pair) for the PARAMETER1 table. This is necessary if you want to set multiple key-value pairs. The key is assigned to the PARAMTER1.PARID field and the value is assigned to the PARAMETER1.PARVA field. The default is **false**.

combinedAttributeForParameter

combinedAttributeForParameter – The name of the attribute from which or to which a combined key-value pair is read or written. There is no default (the attribute is only relevant if *useCombinedAttributeForParameter* is set to **true**).

useAdditionalRoleParameters

useAdditionalRoleParameters - A boolean switch to specify whether (true) or not (false) additional SAP role parameters are accepted when both CUA and *combinedRoleProfileSubsystem* are set. By default, only “dxrRole.NAME” is allowed. When this switch is set to **true**, “dxrRole.TO_DAT” and “dxrRole.FROM_DAT” are also allowed. The default is **false**.

Use the following attributes to control password generation:

minLength

minLength specifies the minimum number of characters. The default value is **8**.

maxLength

maxLength specifies the maximum number of characters. The default value is **8**.

minUpperChar

minUpperChar specifies the minimum number of capital letters. The default value is **4**.

minLowerChar

minLowerChar specifies the minimum number of lower-case letters. The default value is **0**.

minNumeric

minNumeric specifies the minimum number of digits. The default value is **1**.

minNonAlphaNum

minNonAlphaNum specifies the minimum number of non-alphanumeric characters. The default value is **1**.

minSpecialChar

minSpecialChar specifies the minimum number of special characters. The default value is **0**.

prohibitChars

prohibitChars specifies the characters that are prohibited.

3.10.2.3. Search Request File Format

The objects to be exported are defined in a Service Provisioning Markup Language (SPML) search request. SPML is an XML format. The following template describes its configuration. The attribute values that can be configured are shown in bold (blue) italic, e.g., ***attribute1***:

```
<?xml version="1.0" ?>
  <spml:searchRequest xmlns="urn:oasis:names:tc:SPML:1:0"
xmlns:spml="urn:oasis:names:tc:SPML:1:0" requestID="search_01">
  <spml:searchBase
type="urn:oasis:names:tc:SPML:1:0#UserIDAndOrDomainName">
     <spml:id>prefix</spml:id>
 </spml:searchBase>
 <spml:filter>
     filter_expression
 </spml:filter>
 <spml:attributes>
         <attribute name="attribute1" />
         <attribute name="attribute2" />
    </spml:attributes>
  </spml:searchRequest>
```

searchbase

searchbase specifies the type of objects to be returned by the search: "USER:", "ROLE:", or "PROFILE:". In case of users a sapusername can be added to do a search on one account ("USER:_sapusername_").

searchbase is optional. Default is type user.

attributes

attributes specifies the attributes ***attribute1, attribute2,...*** to be returned by the search.

filter_expression

filter_expression specifies the search filter in SPML syntax. Only ApproximateMatch and ExtensibleMatch are prohibited. You can also have filter criteria on ECC user, activitygroup and profile names. In this case, use “USERNAME”, ”AGR_NAME”, and “PROFN” respectively as attribute names. See the section "Framework-based Agents" above for more information about SPML filters.

See the section "Framework-based Agents" above for more information.

3.10.2.4. Filter Expression in BAPI USER GETLIST

The SPML filter expression is mapped to the SAP filter expression for the module BAPI_USER_GETLIST if it is appropriate. If the mapping is not possible, then a filter is not sent to the SAP system and the filtering takes place in the agent on the client side.

SAP only supports a subset of searchable user attributes and allows multiple attributes in the filter expression. An attribute can appear more than once; in this case, however, only the following is allowed: a selection using the same attribute linked with 'OR' and a selection using different attributes with 'AND'. An SPML filter linking the same attribute with ‘AND’ is not possible.

The mapping accepts an AND selection of the same attribute and the first selection operator is “GreaterEqual” and the second is “LessEqual”. This will be mapped in one SAP selection using “Between”. Similarly, “LessEqual” and “GreaterEqual” will be mapped into “NotBetween”. Note that the upper limit is exclusive, not inclusive. This is the only exception where you can give the same attribute linked with ‘AND’. For the upper limit, give the next higher value. It is also recommended to use only upper-case values.

Example: Searching for users that begin with ‘A’ and ‘B’:

```
-<and>
    <greaterOrEqual name="ADDRESS.LASTNAME">
         <value>A</value>
    </greaterOrEqual>
    <lessOrEqual name="ADDRESS.LASTNAME">
         <value>C</value>
    </lessOrEqual>
</and>
```

This will be mapped to the following internal selection range table:

PARAMETER	FIELD	SIGN	OPTION	LOW	HIGH
ADDRESS	LASTNAME	I	BT	A	C

“I” is the INCLUDE sign, “BT” is BETWEEN.

The following table is taken from the current SAP documentation at the time of this writing and is subject to change by SAP. It shows which user attributes can be used:

PARAMETER	FIELD	Permitted LOW Values
USERNAME		<user name>
LOGONDATA	GLTGV, GLTGB, USTYP, CLASS, ACCNT, TZONE, CODVN, UFLAG (not in BAPI_USER_GET_DETAIL, which shows this information in parameter ISLOCKED)	(in accordance with the field type)
DEFAULTS	SPLD, SPLG, SPDB, SPDA, DATFM, DCPFM, LANGU, KOSTL, START_MENU, TIMEFM	(in accordance with the field type)
REF_USER	REF_USER	<user name>

PARAMETER	FIELD	Permitted LOW Values
ALIAS	USERALIAS	<user alias>
PROFILES	BAPIPROF	<profile name>
LOCPROFILES	SUBSYSTEM, PROFILE	(in accordance with the field type)
ACTIVITYGROUPS	AGR_NAME, FROM_DAT, TO_DAT	(in accordance with the field type)
LOCACTGROUPS	SUBSYSTEM, AGR_NAME, FROM_DAT, TO_DAT	(in accordance with the field type)
ADDRESS	FIRSTNAME, LASTNAME, DEPARTMENT, INHOUSE_ML, FUNCTION, BUILDING_P, BUILDING, ROOM_NO_P, TEL1_EXT, TEL1_NUMBR, FAX_EXTENS, FAX_NUMBER, E_MAIL	(in accordance with the field type)
COMPANY	COMPANY	<cross-system key of company address>
LASTMODIFIED	MODDATE, MODTIME	'L'
ISLOCKED	LOCAL_LOCK, GLOB_LOCK, WRNG_LOGON, NO_USER_PW	(in accordance with the field type)
SYSTEM	SUBSYSTEM	<logical system name>

The UMAgent does not check the attributes. The expression is sent to the SAP system. If an attribute is not allowed, an error return code is returned, which results in a warning on the client side and no search result is returned.

3.10.2.5. Import Configuration File Format

The import configuration file has the format defined in the general section. The following template describes its configuration. The attribute values that you can configure are shown in bold (blue) italic, for example ***level***:

```
<?xml version="1.0" encoding="UTF-8" ?>
<job>
  <connector name="Default Controller" version="0.1"
role="controller"
className="siemens.dxm.connector.framework.DefaultControllerStandalon
e">
    <logging level="level" filename="tracefilename" />
  </connector>
```

```
  <connector role="reader" name="LDIF change file reader"
className="siemens.dxm.connector.framework.LdifChangeReader">
    <connection type="LDIF change" filename="inputFilename" />
       <property name=”IdentifierType”
value=”urn:oasis:names:tc:SPML:1:0#UserIDAndOrDomainName”/>
       <property name=”ExtractRDN” value=”true or false”/>
       <property name=”IncludingNamingAttribute” value=”true or
false”/>
    </connector>
  <connector role="connector"
className="siemens.dxm.connector.sapUM.sapUMuser"
name="SAP UM Agent" version=”2.00”>
    <connection type="SAP_ECC_UM"
        user="account"
        password="password"
        server="server">
  <property name=”client” value=”client number”/>
  <property name=”systemID” value=”system number”/>
  <property name=”systemIDgateway” value=”system number”/>
  <property name=”gwhost” value=”gateway server”/>
  <property name=”gwserv” value=”gateway service number”/>
  <property name=”group” value=”group name”/>
  <property name=”r3SystemName” value=”ECC System name”/>
  <property name=”logonVariant” value=”0 or 1 or 2”/>
    <property name=”language” value=”language ISO code”/>
  <property name=”accesstoCUA” value=”false or true”/>
  <property name=”combinedRoleProfileSubsystem” value=”false or
true”/>
  <property name=”maxConnections” value=”number”/>
  <property name=”blankValues” value=”false or true”/>
  <property name="snc_mode" value="false or true"/>
  <property name="snc_lib" value="path>"/>
   <property name="snc_partnername" value="p:distinguished_name"/>
    </connection>
  </connector>
  <connector role="responseWriter" name="SPML File writer"
className="siemens.dxm.connector.framework.SpmlFileWriter">
    <connection type="SPML" filename="responseFilename" />
  </connector>
</job>
```

Here follows a description of the fields that are different to their description in the export configuration.

inputfileName

inputfileName specifies the pathname of the LDIF change file that contains the data for import.

ExtractRDN

ExtractRDN specifies whether the DN or RDN is used. If false, the DN is unescaped and will be used as the identifier. If true, the RDN is extracted with or without naming attributes (see next field IncludingNamingAttribute) and unescaped.

IncludingNamingAttribute

IncludingNamingAttribute specifies whether or not naming attributes are included in the identifier.

responseFilename

responseFilename specifies the name of the Service Provisioning Markup Language (SPML) response file that contains the responses to the add, modify, and delete requests.

3.10.3. Export Data File Format

A search request creates an export file in LDIF content format that contains the search result.

The identifiers of the users (i. e. attribute USERNAME.BAPINAME), roles (i. e. attribute AGR_DEFINE.AGR_NAME or USRSYSACT.AGR_NAME) and profiles (i. e. attribute USR10.PROFN or USRSYSPRF.PROFN) are converted to LDAP distinguished name (DN) syntax.

3.10.4. Import Data File Format

The import data file format recognized by the SAP ECC UM agent is LDIF change file format. The data has to be provided in UTF-8 character set (or US-ASCII), not in ISO8859-1 (Latin-1).

The supported change types are add, modify, and delete; modifyDN is not supported.

Example:

1. A user with user name **ps045293** for the person Paul Simon with telephone number 45293 and role SAP_ALL_DISPLAY valid from 03/11/01 is created. Mandatory attributes for creating a user are lastname (and user name). Password is also mandatory if no SNC parameters are set:

```
dn: cn=ps045293
changetype:add
ADDRESS.LASTNAME:Simon
```

```
ADDRESS.TEL1_NUMBR:45293
PASSWORD.BAPIPWD:hugohugo
ADDRESS.FIRSTNAME:Paul
ACTIVITYGROUPS.AGR_NAME:SAP_ALL_DISPLAY
ACTIVITYGROUPS.FROM_DAT:2003-11-01
ACTIVITYGROUPS.TO_DAT:9999-12-31
```

2. Another role is assigned to the user **ps045293**:

```
dn: cn=ps045293
changetype:modify
add: ACTIVITYGROUPS.AGR_NAME
ACTIVITYGROUPS.AGR_NAME:SAP_WORKPLACE_USER
-
```

3. The user **rk044251** is deleted:

```
dn: cn=rk044251
changetype:delete
```

3.10.5. Installing and Configuring SNC Connections

The SAP Cryptographic Library installation package contains the following relevant files:

- The SAP Cryptographic Library (**sapcrypto.dll** for Windows or **libsapcrypto.so** for Linux - you need the 64bit package)
- The configuration tool **sapgenpse.exe** (**sapgenpse** for Linux)

Installation Procedure

1. Extract the contents of the SAP Cryptographic Library installation package.
2. Copy the library and the configuration tool **sapgenpse.exe** to a local directory, for example to *install_path*/**SAPCryptolib**.
3. Check the file permissions. The user under which the SAP -ECC-UMAgent runs must be able to execute the library's functions.
4. Create the sub-directory **sec** in this directory. This is also the directory where the user's PSE and credentials are created.
5. Set the (system) environment variable **SECUDIR** to the **sec** sub-directory.
6. On Linux, in the file **SapUM.sh** in the installation folder, set the environment variable **LD_LIBRARY_PATH** accordingly.

When using the SAP Cryptographic Library as the security product for SNC, the SAP-ECC-

UMAgent user must possess a Personal Security Environment (PSE). This PSE contains the user's public-key information, which includes its private key, its public-key certificate, and the list of public-key certificates that it trusts.

To create the SNC PSE for the user, in which the SAP-ECC-UMAgent runs, use the command line tool **sapgenpse.exe** as shown below. As an alternative, you can use a single PSE for both, the application server and the SAP-ECC-UMAgent. In this case, copy the application server's SNC PSE to the user's **SECUDIR** directory.

Use the following command to create the PSE:

sapgenpse get_pse -p *PSE_Name* **-x** *PIN Distinguished_Name*

Example 2. sapgenpse

The following command creates the *PSE_Name* file **UM.pse** that is protected with the *PIN* **umpin**. When using this PSE, the SAP-ECC-UMAgent user has the *Distinguished_Name* **CN=sapum, O=MyCompany, C=DE**.

sapgenpse gen_pse -p UM.pse -x umpin "CN=sapum, O=MyCompany, C=DE"

The UMAgent must have active credentials at run-time to be able to access its PSE. Therefore, use the configuration tool's command **seclogin** to "open" the PSE.

Use the following command to open the user's PSE and create credentials:

sapgenpse seclogin -p *PSE_Name* **-x** *PIN* **-O** *user_ID*

If you run the services under the LocalSystem account - which is the default on Windows - then use **-O SYSTEM**.

Example 3. sapgenpse

sapgenpse seclogin -p UM.pse -x umpin -O SYSTEM

To be able to communicate using SNC, the SAP system application server must be able to identify the SAP-ECC-UMAgent and vice versa. This identification process takes place using the information stored in the user's PSE. Therefore, to make sure that the two servers can identify each other, you can either use a single PSE for both sides, or you can create individual ones. If you use individual PSEs, you must exchange the public-key certificates so that they can identify each other You have two possibilities: you can use these certificates without signing them by a Certificate Authority (CA), or you can create certificate requests and send them to your CA to sign them, and then import the signed certificates you receive from the CA.

The identification process steps without using any CA are:

1. Export the SAP-ECC-UMAgent's public-key certificate using the configuration tool's command **export_own_cert**:

   sapgenpse export_own_cert -o *output_file* **-p** *PSE_Name* **-x** *PIN*

2. Import the SAP-ECC-UMAgent's public-key certificate into the application server's SNC PSE:

   If the application server is an SAP Web Application Server with Release 6.20 or later, then you can use the trust manager (transaction STRUST) to import the certificate. Otherwise, use the configuration tool's command **maintain_pk**:

   sapgenpse maintain_pk -a *cert_file* **-p** *PSE_Name* **-x** *PIN*

3. Export the application server's public-key certificate:

   If the application server is an SAP Web Application Server >= Release 6.20, then you can use the trust manager. Otherwise, use the configuration tool's command **export_own_cert.**

4. Import the application server's public-key certificate into the SAP-ECC-UMAgent's SNC PSE using the configuration tool's command **maintain_pk**.

5. In User Maintenance (transaction SU01) in the SAP system, assign the user's SNC name that you entered in generating the PSE on the SNC tab page. The SNC name is **not** the same as the Distinguished Name you used when creating the PSE. The SNC name has the syntax *p:*Distinguished_Name*.
   Take care to specify the distinguished name in the same way wherever it occurs. The operations are case sensitive.

   Other mandatory settings on the server-side are:

   In the table USRACLEXT of the server, the user name and user's SNC name must be maintained (user in which the SAP-ECC-UMAgent process runs).

   The following system parameters must be set:

   - **snc/accept_insecure_rfc** to **1**
   - **snc/permit_insecure_start** to **1** for all application servers

   If you want to use signed certificates, perform these steps after performing step 1 and before performing step 2 above:

   - Export a certificate request file from your PSE with the command:

     sapgenpse gen_pse -p *PSE_Name* **-x** *PIN* **-onlyreq -r** *certfile.p10*

     The request file is in PKCS#10 format. Send this to your CA to sign.
   - Import the certificate request response and the CA certificate into your PSE with the command:

     sapgenpse import_own_cert -p *PSE_Name* **-x** *PIN* **-c** *user_cert.crt* **-r** *ca_cert.crt*

     where *user_cert.crt* is the signed user certificate and *ca_cert.crt* is the certificate

from the CA. If there are any intermediate certificates from the CA, you can add them with additional **-r** options.

Similar steps must be performed on the SAP server side.

Example 4. Commands

1. Exporting the SAP UMAgent's public-key certificate.

   sapgenpse export_own_cert -o UM.crt -p UM.pse -x umpin

2. Importing the UMAgent's public-key certificate into the application server's SNC PSE.

   sapgenpse maintain_pk -a UM.crt -p SAPSNC.pse -x sappin

3. Export the application server's public-key certificate.

   sapgenpse export_own_cert -o SAPSNC.crt -p SAPSNC.pse -x sappin

4. Import the application server's public-key certificate into the UMAgent's SNC PSE.

   sapgenpse maintain_pk -a SAPSNC.crt -p UM.pse -x umpin

For the SAP-ECC-UMAgent connection, set the following SNC-related mandatory parameters:

SNC mode
whether (**true**) or not (**false**) SNC is active.

SNC library path
the path and file name of the SAP Cryptographic library; for example, *install_path*/SAPCryptolib/sapcrypto.dll.

Partner's SNC name
the application server's name; for example, p:CN=ABC, O=MyCompany, C=US. (Case-sensitive, as written in the certificate! Note the prefix "p:")

3.10.6. General Notes

The UMAgent uses SAP's Business Application Programming Interface (BAPI) to import and export user data. As a result, you must use the BAPI table and attribute names, not the SAP ECC internal table names. The BAPI data view can be viewed on an ECC system using the transaction BAPI. The business object is USER.

You can view attribute names for roles or profiles if you select the method "ActgroupsAssign" or "ProfilesAssign" and then the table "Activitygroups" or "Profiles" (non-CUA). For CUA, the methods are "LocActgroupsAssign" or "LocProfilesAssign" and then the table "Activitygroups" or "Profiles". The CUA methods can only be viewed in release 640, although they exist in previous releases, too.

3.10.6.1. Distinguished Names

On the directory side, the DN (distinguished name) is used to identify an entry. There is no DN in the SAP user management. For ECC users, the user name is the key. You can find it in the SAP ECC table USR05 as field BNAME. The agent maps the BAPI attribute USERNAME.BAPINAME to a DN. If you want to export the user name, you must have the DN in your list of selected attributes. In an export, the user name is written as a DN attribute.

3.10.6.2. Attribute Configuration File

An attribute configuration file defines the attributes that are present in a particular connected directory. The example attribute configuration file of the SAP ECC connected directory uses only a subset of valid attributes in the SAP ECC user management. You can see the content of this file in the DirX Identity Manager in the connectivity global view default scenario when selecting **Configure** from the context-sensitive menu of the **Scenario** → **Default** → **Target Scheduled** → SAP ECC/UM container (field **Attribute Config**). You find more attributes in the file *inst_path***/schema/dirx/dirxabbr-ext.DirXmetahub.SAP-UM**.

3.10.6.3. Import/Export Date Values

To import a date, use the format *yyyymmdd*. The exported format is *yyyy-mm-dd*.

3.10.6.4. Distribution in a CUA Environment

In a CUA environment, it depends on the current distribution parameter settings for the user master records where attributes can be maintained:

- In the central system
- Locally in the child system
- In the child system with automatic redistribution to the central system and the other CUA child systems

The DirX Identity Provisioning scenario only supports a distribution model where SAP roles and/or profiles are distributed globally. You can display and maintain the distribution model within transaction **SCUM**.

Synchronizing User Groups
User groups can be created and distributed if you change the system behavior of the central system and the target system by setting a switch in the PRGN_CUST Customizing table. See SAP note 395841 for further information.

3.10.6.5. Lock/Unlock

To lock or unlock a user, the agent uses the (pseudo-)attribute dxrTSState to determine whether a user must be locked or unlocked:

Operation / dxrTSState	ENABLED	DISABLED
AddRequest	user is created without any lock	user is created with lock
ModifyRequest	user is unlocked	user is locked
DeleteRequest	not applicable	not applicable

Other values of the attribute dxrTSState are ignored.

In a CUA environment, the lock/unlock is done only globally and it depends on the current distribution parameters set for the lock data whether global locking is possible.

3.10.6.6. Export Lock Status

In a search, the lock status can be retrieved via the attributes ISLOCKED.WRNG_LOGON, ISLOCKED.LOCAL_LOCK, ISLOCKED.GLOB_LOCK, and ISLOCKED.NO_USER_PW with the following meaning:

'Wrng_Logon'
The password logon is locked by incorrect user logons.

'Local_Lock'
The logon to this client is locked for the user.

'Glob_Lock'
Logon in all systems of Central User Administration is locked for the user ('global').

'No_User_Pw'
For this user, the option for password-based logon is deactivated.

All attributes are of type CHAR(1) with values **L** means locked or **U** means unlocked.

In a CUA environment the lock status in a child system can not be retrieved.

3.10.6.7. Export Users

The search for users in a CUA environment exports even users that have no child system assigned. This has changed from previous releases due to the BAPI_USER_GETLIST search.

The following system accounts are not exported: “SAP*”, “DDIC”, “EarlyWatch”, “BCUSER”, and “SAPCPIC” unless the configuration parameter searchSAPServiceUser is set to TRUE.

3.10.6.8. Export User to Child System Relationship

In a CUA environment the child systems on which a user has an account can be exported in the attribute SYSTEMS.SUBSYSTEM.

3.10.6.9. Password Synchronization

In Version 8.3, a new mode to synchronize passwords has been introduced that simplifies the password synchronization of productive passwords. Both allow using the underlying BAPI methods USER.CREATE1 or USER.CHANGE to set a productive password in one step. It is configured by the configuration parameters **setProductivePwdAtAddDirectly** (default: true) and/or **setProductivePwdAtModDirectly** (default: false).

New procedure:

Using the parameter SELF_REGISTER in the BAPI method USER.CREATE1 a productive password can be set. No other constraints exist. Therefore, the default value for **setProductivePwdAtAddDirectly** is true.

Using the parameter PRODUCTIVE_PWD in the BAPI method USER.CHANGE a productive password can be set. However, using this parameter needs a secured connection via SAP's SNC to the application server. Therefore the default value for **setProductivePwdAtModDirectly** is false.

Old procedure:

Parameter **setProductivePwdAtAddDirectly** set to false and **setProductivePwdAtModDirectly** set to false:

The BAPI methods USER.CREATE1 or USER.CHANGE are only used in the mode to set an initial password. The agent uses the RFC-enabled function SUSR_USER_CHANGE_PASSWORD_RFC to change or set a productive password. If a password must be set, the agent first tries to log in as the user to the SAP system with the new password. If the login is successful, no password update is done. This attempt can be configured through the **tryLoginAsUser** option. If not, the agent calls USER.CREATE1 or USER.CHANGE to set an internally-generated dummy password and then calls the above function to set the productive password.

Either one or both function calls can fail. If the first fails no change has been made. If the second fails the user is protected via an unknown password. Either the password synchronization has to be processed again or only an administrative person can resolve the issue.

SAP does not perform productive password replication in a CUA environment. Password synchronization will therefore only work if the central CUA system is also the authentication server in the ECC system landscape (which is generally the case in conjunction with an Enterprise Portal). Otherwise the password is just changed on the central CUA system but not on the child systems.

The agent offers a set of configuration parameters to configure the password generation for the internally-generated dummy password (minLength, maxLength, minUpperChar, and so on).

The login attempt configuration option is intended for the password

scenario where users change their password in SAP and then in the Web Center. In this case, the second password set fails, which sets the failed login counter on the SAP side. To avoid this situation, this login attempt is configurable.

To set the password via the RFC-enabled function, the user must be in the unlocked state (through failed logins). Therefore, if the function calls fail with the SAP message id **190**, the user is unlocked and the password set is processed again.

For both procedures:
The change of a password on the central CUA system fails if the user has no role or profile assignment on this central ECC system. To overcome this situation an empty role can be assigned as follows:

`dxrRole.Name:#``<_ACTIVITYGROUPS.SUBSYSTEM_>`

or

`ACTIVIGROUPS.SUBSYSTEM:``<_ACTIVITYGROUPS.SUBSYSTEM_>`

(See section “Role or Profile Assignments in CUA Environment” below for more information.) The name of the subsystem must be the one of the central CUA system.

3.10.6.10. Password Reset

The agent supports an administrative password reset and change password at next log on. The agent uses the boolean (pseudo-)attribute dxrPwdReset to decide if a productive or initial password must be set. The attribute works in both scenarios in conjunction with the attribute PASSWORD.BAPIPWD.

If dxrPwdReset is TRUE then only an initial password is set via USER.CREATE1 or USER.CHANGE. If dxrPwdReset is FALSE or not set (default behavior) then the productive password is set as described above.

Example for administrative password reset or change password at next log on:

dxrPwdReset: TRUE
PASSWORD.BAPIPWD: *Administrator_Password*

Example for set productive password:

dxrPwdReset: FALSE
PASSWORD.BAPIPWD: *value_of_dxmPassword*

The attribute dxmPassword holds the current password of the user.

3.10.6.11. Role or Profile Assignments in CUA Environment

The agent has been extended to ease the role assignments in a CUA environment. The

agent accepts the (pseudo-)attributes “dxrRole.NAME” / “dxrProfile.NAME” for this purpose. The values describe the role/profile name concatenated with the subsystem. Delimiter is the pound symbol (#).The values have therefore the following syntax: “<ACTIVITYGROUPS.AGR_NAME>#<ACTIVITYGROUPS.SUBSYSTEM>”

To use this feature use the combinedRoleProfileSubsystem switch in the configuration file.

3.10.6.12. Setting Additional Options in Realtime Workflows

If you want to set additional options for the connector like **directlyAssignedRolesOnly** for realtime workflows perform the following steps:

- In DirX Identity Manager → **Connectivity** → **Expert View**, browse to the workflow and expand the workflow to see all contained entries.
- Select **join** → **ts** and perform the **Goto DataView** operation from the context-sensitive menu.
- Export the value of the attribute **dxmContent** as a xml file.
- Open the file with an editor and insert the option in the connection tag section of the connector for example:

  <property name="directlyAssignedRolesOnly" value="true"/>
- Import the changed xml file into the attribute **dxmContent**.
- Go back to the **Expert View**, select the workflow and perform the **Load IDS-J Configuration** from the context-sensitive menu.

This is the way for non-clustered realtime workflows. See the description of clustered workflows on how to set additional options in that case.

3.10.6.13. Special Cases When Changing Data

When changing data, consider the following special cases:

- **Address**: You can maintain certain address data in the **Address** structure or alternatively in tables. For example, data such as telephone number, fax and e-mail address can be maintained in the tables **AddTel**, **AddFax**, and **AddSmtp** respectively.

We recommend maintaining the information in the tables instead of in the **Address** structure for the following reasons:

- You can store multiple entries in the tables. The **Address** structure only contains one entry for each of these fields.
- The telephone and fax numbers are stored in international format in the tables, but not in the **Address** structure.
- If you change data in the **Address** structure, any entries in the corresponding table will be lost.
- **Communication data**: When changing communication data (Add<Xxx> parameters), you need to consider the following fields:

- **CONSNUMBER**: To differentiate between multiple entries for communication data, use the sequence number that is stored in the field CONSNUMBER. To change a specific entry, enter the entry's sequence number in this field. If you want to add an entry, specify a sequence number that is higher than that for any existing entry.
- **R_3_USER**: This field applies to the telephone numbers. It indicates the type of telephone connection and if the number used is the standard number. The following applies:
- <blank>: The telephone number is a land-line telephone.
- 1: The telephone number is the standard land-line telephone.
- 2: The telephone number is a mobile telephone.
- 3: The telephone number is the standard mobile telephone.
- **STD_NO**: Only one telephone number appears as the standard telephone number in the Address structure. Therefore, use this field to indicate that the telephone number (land-line or mobile) for this entry is the overall standard telephone number that appears the **Address** structure.
- **STD_RECIP**: This field indicates whether the corresponding telephone number can be used for short messages (SMS). If this is the case, then the number is copied to the communication data used for paging services.

Not all fields are used by all of the communication data parameters.

If you want to modify for example the extension you must provide at least CONSNUMBER, STD, NO, and TELEPHONE all together because the LDIF change replace operation clears the whole row in the ADDTEL table.

- **Company Location**: The company location address is stored with the business object **AddressOrg** and not the object **USER**. Therefore, when specifying or changing the company location with the BAPI_USER_CHANGE, you can only specify or assign an existing company location.

3.11. SAP NetWeaver UM Agent

SAP NetWeaver UM Agent is the DirX Identity agent that handles the import and export of users and user to role assignments as well as the export of portal roles from an SAP NetWeaver/Enterprise portal.

The SAP NetWeaver UM agent is implemented in Java, uses the Identity Connector Integration Framework and uses a SOAP service to bind to the portal server.

The SAP NetWeaver UM Agent or Connector requires SAP NetWeaver '04 SPS 14 or SAP NetWeaver Portal or higher.

SAP NetWeaver UM Agent can:

- Perform a full export of portal roles and groups from an SAP NetWeaver portal using SPML search filters.

- Perform a delta import of users into an SAP NetWeaver portal, including the creation of new users, creation of user-to-role assignments, modification of users and user-to-role assignments and deletion of users.
- Perform a full export of users from an SAP NetWeaver portal using SPML search filters.
- Generate a trace file (for tracing, reporting which objects were processed and the operations that failed).

The following figures illustrate the components of the SAP NetWeaver UM Agent export and import operations.

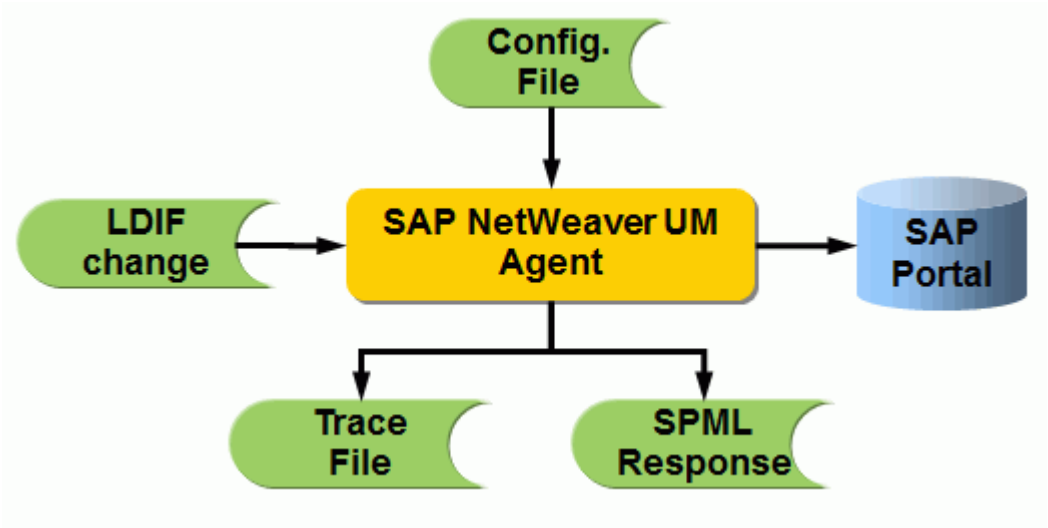


Figure 27. SAP NetWeaver UM Agent Export Components

During startup, the agent reads its configuration file that defines the internal structure and parameters. Next, it reads the SPML request file that defines the amount of entries to be read from the SAP Enterprise portal, reads it and writes it to an LDIF content file. In parallel, it creates a trace file.

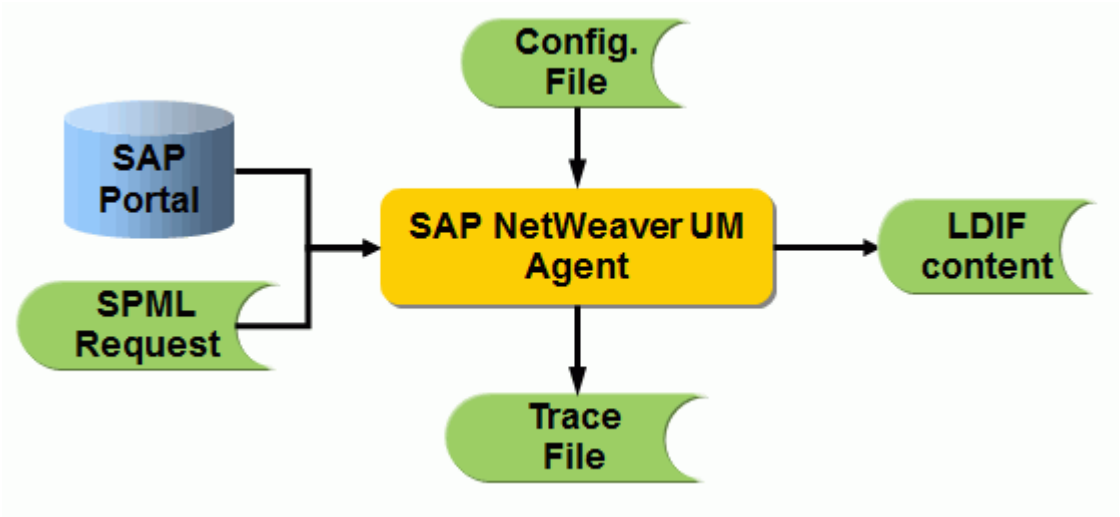


Figure 28. SAP NetWeaver UM Agent Import Components

During startup, the agent reads its configuration file that defines the internal structure and parameters. Next, it reads an LDIF change file and imports it to the SAP NetWeaver portal database. In parallel, it creates a trace file and an SPML response file. The response file

contains information about whether an entry operation was successful. This information can be used for example to update the state information in DirX Identity.

This section describes:

- SAP NetWeaver UM Agent configuration files for export and import operations
- The export data file format that SAP NetWeaver UM Agent generates
- The import data file format that SAP NetWeaver UM Agent recognizes

3.11.1. Configuration File Formats

SAP-NetWeaver-UMAgent uses the following configuration files:

- NetWeaver UM export configuration file - controls the export of data from a SAP NetWeaver Portal
- NetWeaver UM import configuration file - controls the import of data into an a SAP NetWeaver Portal

Templates of these configuration files are provided with the Agent installation. The filenames are:

- **Configuration.xml** (to import and export objects)
- **SearchRequest.xml** (contains the search request to select the objects for export)

In general, you must customize these files to support the requirements of your SAP NetWeaver Portal import and export operations.

The following sections describe the extensions to the standard file formats.

3.11.1.1. Configuration File Extensions

The standard parameters are described in the Configuration File Formats section of the Framework-based Agents chapter. This section describes only the extra parameters.

The **SOAP connection** is configured by the attributes

- *Url* - The URL of the Web service, in the format **http://***host:port***/spml/spmlservice**
- *user* - The user password (basic authentication only)
- *password* - The password for basic authentication
- *trustStore* - the path to the trust store file containing the certificate of the server to be used for SSL/TLS server side authentication.
- *trustStorePassword* - the password that is needed to read the server certificate from the trust store.
- *keyStore* - the path to the key store file containing the private key or certificate to be used for SSL/TLS client authentication.
- *keyStorePassword* - the password that is needed to read the key from the key store.

- *keyStoreAlias* - the alias name to identify the private key in the key store.

The following attributes control internal dummy password generation:

- *minLength* - specifies the minimum number of characters. The default value is **8**.
- *maxLength* - specifies the maximum number of characters. The default value is **8**.
- *minUpperChar* - specifies the minimum number of capital letters. The default value is **4**.
- *minNumeric* - specifies the minimum number of digits. The default value is **1**.
- *minNonAlphaNum* - specifies the minimum number of non alpha numeric characters. The default value is **1**.
- *minSpecialChar* - specifies the minimum number of special characters. The default value is **0**.

SSL connection is defined by the protocol value **https**. In this case you must provide the public certificate of the SAP NetWeaver server in the truststore as mentioned above (see trustStore, keyStore).

A successful authentication requires that the user is assigned the UME action **MANAGE_ALL_COMPANIES**. Only this type of users is allowed to call the SAP SPML provider. The assignment must be done now via the UME admininistration UIs (useradmin webapplication). Use a role already containing the action or create a new one and assign the action **MANAGE_ALL_COMPANIES**. Then assign this role to all users shall be able to use the SPML provider.

3.11.1.2. Search Request Format

The standard parameters are described completely in the Search Request File Format section of the Framework-based Agents chapter. There are no extra parameters to configure.

Hints to configure the parameters for SAP NetWeaver UM:

searchBase - specifies the base object for the search, for example “USER.PRIVATE_DATASOURCE.un:” for users.

filter - for example set it to 'objectclass' and 'sapuser'.

3.11.2. Data File Formats

The next sections describe the used data file formats for import and export of data.

3.11.2.1. Import Data File Format

The SAP NetWeaver UM agent recognizes import data files in LDIF change file format. The supported change types are add, modify, and delete; modifyDN is not supported.

Example 5. Adding a user to the portal

User Marco Bellosa is added to the portal.

```
version:1
dn:mbello75
changetype:add
firstname:Marco
islocked:False
lastname:Bellosa
logonname:mbello75
password:x#mb7564
objectclass:sapuser
```

Example 6. Modifying a user

User mbello75 is modified. Its description is changed, and the user gjx32406 is removed from the role.

```
dn: mbello75
changetype: modify
delete: title
-
add: title
title: Big boss
-
```

Example 7. Deleting a user

User mbello75 is deleted from the portal.

```
dn: mbello75
changetype: delete
```

3.11.2.2. Export Data File Format

As a result of the search request, an export file in full LDIF format is created, containing the search result.

Example:

```
dn: mbello75
objectclass: sapuser
firstname:Marco
islocked:False
lastname:Bellosa
logonname:mbello75
```

3.11.3. Setting Up a Secure Connection to SAP NetWeaver

This section describes in more detail how to set up a secure SSL/TLS connection to the SAP NetWeaver Portal based on server or client side authentication.

1. Server Authentication

   This type of connection requires

   - a key store located at the server containing the servers private-key / certificate pair
   - a trust store located at the client containing the server's certificate.

   Follow the instructions below to accomplish this.

   Server-side actions

   Open the "Visual Administrator" (this is a command line tool provided by the SAPEP6 portal) and select the "Key Storage" service (left pane, tree-view).

   Select the view "service_ssl" which shows a list of ssl-entries in the "Entries" pane. By default, the entry "ssl-credentials-cert" holds the server certificate being used in SSL handshake negotiations. Note that the certificate's "common name" MUST be the name of the host where the portal is installed. Otherwise, the client will NOT accept this certificate (SSL checks if the certificate's "common name" matches the host name).

   If there is no suitable certificate available, create a SSL credential pair by pressing the "create" button. As mentioned above, the "common name" MUST be the host name.

   Export the certificate to the filesystem (see "export" button).

   If you have created a new (self signed) SSL credential, select the Service "SSL Provider). The property page "Server Identity" shows the currently active SSL credential. Add your credential (see "add" button).

   Client-side actions

   Trust store:

   To import the server's certificate into the trust store, copy this certificate to the SOAP-client's file system. If the server's certificate is self-signed, this can be accomplished in two ways:

a. copy the formerly exported certificate (see 1.1) or

b. open the Internet Explorer (Window platform only), browse to https://<HOST>:53001.

The browser will warn you about the untrusted (self-signed) certificate and provide an "install certificate" button. Install the certificate in one of Window's certificate stores and copy it to your file system (drag&drop!).

Now go to the DirX Identity installation directory (for example C:\Program Files\DirX\Identity) and open the subfolder security/java. Open a DOS window and enter the command:

keytool -import -noprompt -trustcacerts -alias sapep6 -file portal.cert -keystore truststore -storepass changeme

where "portal.cert" contains the server's certificate.

DirX Identity configuration:

Open the DirX Identity Manager, select the **Connectivity** pane, and go to the **Expert View**.

Navigate to the service object which describes your portal service (for example, configuration/services/my-company/SAPEPUM).

Change the server name from "**http://*host***:53000/spml/spmlservice*" to "**https://*host***:53001/spml/spmlservice*". Note that the port may differ. In general, if your http-port is *PORT*, the https-port will be *PORT*+1*.

Open the associated "System" object in **Expert View** → **configuration/systems** folder). In the **security** property page, add the path to the trust store (for example, C:\Program Files\Atos\DirX Identity\java\security\truststore) and provide the password (for example, changeme).

Eventually, go to the respective password synchronization workflow (for example, workflows/my-company/password synchronization /setPassword in SAP) and check the SOAP SSL-Button in the **Set Password Activity** property page.

2. Client Authentication

This type of connection requires

- a key store located at the server containing the servers private-key / certificate pair
- a trust store located at the server containing the clients's CA authority
- a key store located at the client containing the client's private-key / certificate pair
- a trust store located at the client containing the server's certificate

Follow the instructions below to accomplish this.

Server-side actions

If your client certificate had been issued by a trusted CA (certification authority), you should have the CA's certificate. Open the "Visual Administrator", navigate to the service "SSL Provider" and open the "Client Authentication" property page. Select "Require client certificate" and add the CA's certificate to the list of "Trusted Certification Authorities.

If you don't have a trusted client certificate, things get a little bit more complicated:

- Navigate to the service "Key Storage", select the view "TrustedCAs".
- Create a new "trusted" private-key / certificate entry.
- Go to the "service_ssl" view and create a SSL credential using this trusted certificate (see "select CA key" button).
- Export this ssl credential (i.e. private-key / certificate pair) to a PKCS12 key store (see export, type p12).
- Copy the PKCS12 file (usually ending with .p12) to the client machine.
- Navigate to the service "SSL Provider" and open the "Client Authentication" property page.
- Select "Require client certificate" and add the trusted CA certificate (see above) to the list of "Trusted Certification Authorities.

Client-side actions

Trust store:

See above.

Key store:

Typically, a CA authority provides the client's private-key / certificate along with its own certificates in a PKCS12 file.

To import this PKCS12 file into the key store, copy it to the SOAP client's file system.

Now go to the DirX Indentity installation directory (for example, C:\Program Files\Atos\DirX Identity) and open the subfolder security/java. Open a DOS window and enter the command:

keytool -import -noprompt -alias sapep6 -file portal.p12 -keystore keystore -storepass changeme

DirX Identity configuration

Open the DirX Identity Manager, select the Connectivity pane, go to the Expert View.

Navigate to the service object which describes your portal service (for example, configuration/services/my-company/SAPEPUM).

Open the associated "System" object (see Expert-View, configuration/systems folder). In the "security" property page, add the path to the key store (for example C:\Program Files\Atos\DirX Identity\java\security\keystore) and provide the password (for example, changeme).

Navigate to the "Connected Directory" representing your SAPEP6 Portal (for example, Connected Directories/my-company/Provisioning/SAPEPUM)

Select the bindprofile SAPEPUMprofile and provide the "Key Store Alias" (for example, sapep6, see keytool command, -alias).

3.11.4. Password Synchronization

The SPML add request only sets an initial password. The agent has been extended to set a “productive” password. If a password has to be set, the agent first sends a SPML add request to set an internally generated dummy password and then sends a SPML modify request to set the “productive” password.

Either one or both function calls can fail. If the first fails no change has been made. If the second fails the user is protected by an unknown password. Only an administrative person can resolve the issue.

A modify should use the attributes oldpassword and password. No password is generated therefore in this case.

3.11.5. Password Reset

The agent supports an administrative password reset and change password at next log on. The agent uses the boolean (pseudo-)attribute **dxrPwdReset** to decide if a productive or initial password must be set. The attribute works only in case of creating a user (SPML add request) in conjunction with the attribute password.

If **dxrPwdReset** is TRUE then only an initial password is set. If **dxrPwdReset** is FALSE or not set (default behavior) then the productive password is set as described above.

Example for an administrative password reset:

```
dxrPwdReset: TRUE
PASSWORD.BAPIPWD: Administrator_Password
```

Example for a change password at next log on:

```
dxrPwdReset: TRUE
PASSWORD.BAPIPWD: value_of_dxmPassword
```

The attribute **dxmPassword** holds the current password of the user.

3.12. SiPass Agent

SiPass agent is the DirX Identity agent that handles the synchronization of SiPass user entries between DirX Identity and a SiPass database.

SiPass agent is implemented in C#. SiPass agent supports SiPass 2.11 SP2 and higher and runs with all platforms that are supported by the SiPass Human Resources Interface which is based on the Windows COM interface.

SiPass agent can:

- Perform a full export of card holders and workgroups from a SiPass system.
- Perform a delta import of users into a SiPass system, including creation, modification and deletion of new users and assignment or removal of users to workgroups.
- Generate a trace file that reports the processed objects and the results of the corresponding operations.

Prerequisites

The SiPass agent has the following requirements:

- The .NET framework to be correctly installed on the machine where the agent (that is, DirX Identity server) should run. The .NET framework can be downloaded from Microsoft's web pages (http://www.microsoft.com).
- To run properly, the account that the agent uses to connect must have the access rights for general user management (read, modify, and delete user).
- The agent must run on the SiPass server machine. Install a secondary C++-based Server on this machine to satisfy this condition.
- The agent may require additional files copied in the *install_path***/bin** directory. This may be needed when working with SiPass versions newer than 2.3. These files are indirectly referenced by the SiPass agent and can be located in the SiPass installation directory. In this case the .NET framework writes error messages describing missing files to standard output when running the agent. For example version 2.5 requires copies of the files **GenuineChannels.dll** and **SiPass.ClientServices.Interfaces.dll**.

Overview

The following figures illustrate the components of the SiPass agent export and import operations.

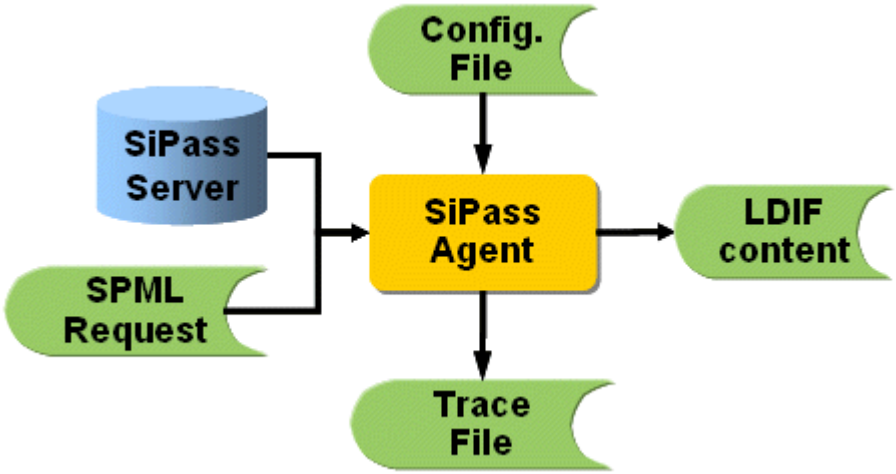


Figure 29. SiPass Agent Export Components

SiPass agent reads its configuration from the configuration file, the search request from the SPML request file and performs the search in the SiPass server. An LDIF content file contains the output information, a trace file contains the trace information.

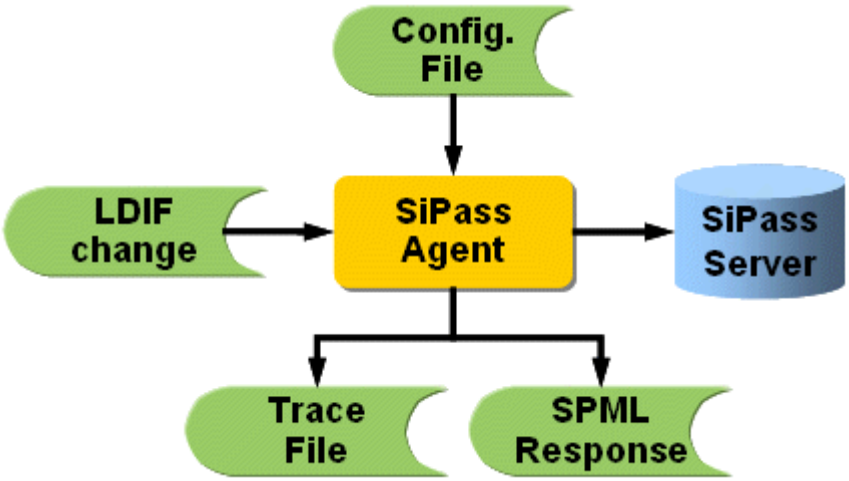


Figure 30. SiPass Agent Import Components

SiPass agent reads its configuration from the configuration file, the input information from an LDIF change file and performs the necessary operations in the SiPass server. A trace file and an SPML response file are generated.

This section describes:

- General information
- SiPass agent command line format for export and import operations
- SiPass agent configuration files for export and import operations
- The export and import data file formats
- The search request file format that SiPass agent recognizes

3.12.1. General Notes

On the directory side, the DN (distinguished name) is used to identify an entry. There is no DN in the SiPass user management. For SiPass users, the reference number is the key (stored by default in dxrName attribute). Therefore, the DN attribute from incoming requests is internally mapped to the reference number during the import action. If you want to export the Reference attribute, you must have this attribute name in your list of selected attributes.

SiPass agent does currently not support encryption of bind profiles nor user attributes (set the switch 'Disable Encryption' in the relevant bind profile).

3.12.2. Command Line Format

The command line format to invoke SiPass agent is:

SiPassAgent.bat *action configuration_file* [**-f** *filter_file*]

3.12.2.1. Parameters

action

Specifies the action type to be preformed. The supported actions are:

- **Export**
- **Import**

configuration_file

Specifies the name of the file that contains the configuration of the export or import procedure. All parameters of SiPass agent operation are defined in the agent's configuration file in XML format.

-f ***filter_file***

This is a mandatory attribute for **Export** action. It specifies the name of the file that contains the specification of the search criteria in for export mode (which are described in a separate Service Provisioning Markup Language (SPML) file).

3.12.3. Exit Codes

The following table describes the codes provided when SiPass agent finishes running:

Exit Code	Description
0	Agent completed successfully.
10	SiPass agent completed with fatal errors, which are described in the specified *tracefile* unless this file cannot be created due to a file exception error, in such a case error is logged to the standard output (system console).
60	SiPass agent completed with warnings or non-fatal errors, which are described in the specified *tracefile*.

3.12.4. Configuration Files

SiPass agent uses the following configuration files:

- SiPass export configuration file - controls the export of data from a SiPass system
- SiPass import configuration file - controls the import of data into an a SiPass system

This section also describes the general structure of a configuration file.

3.12.4.1. General Structure of a Configuration File

A SiPass agent configuration file is in XML format.

The SiPass agent is composed of multiple sub-units, that are configured in the configuration file. Different types of sub-units are used for export and import. Consequently, you must not change the general structure of SiPass agent import/export configuration files. Instead, you configure some well-defined attribute values to the specific environment in which the meta agent runs.

3.12.4.1.1. Tags

The configuration files contain the tags job, connector, logging and connection.

- **job** - Defines the file's document tag, with connector sub-tags
- **connector** - Configures the properties of one connector, has connection and/or logging sub-tags
- **connection** - Configures connection parameters, for example, filename for a reader/writer or host/credentials for a network connector
- **logging** - Configures the logging properties of a connector

3.12.4.1.2. Attributes

A connector tag can have the following attributes:

- **name** - The connector's name
- **role** - One of reader, controller, connector or responseWriter

The **connection parameters** of the specific connectors are described in their connection sub-tags.

Each connection tag has the attribute

- **type** - The type of connection (file format, protocol)

Readers and response writers are configured by the attribute

- **filename** - The pathname of the input or output file.

A **connection to the target system** is configured by some attributes that might differ dependent on the target system. Typical attributes are:

- **server** - The host name or IP address of the SiPass system
- **user** - The user for binding to the SiPass system
- **password** - The user password

The agent's **logging** is configured in the controller's logging tag by the attributes:

- **level** -The integers **0**-**9**, where **0** indicates no logging and **9** indicates full logging
  0 - none
  1 - FatalError and Error
  2 - FatalError, Error and Warning
  3 - FatalError, Error and Warning
  4 - FatalError, Error and Warning
  5 - FatalError, Error, Warning and Trace
  6 - FatalError, Error, Warning and Trace
  7 - FatalError, Error, Warning and Trace
  8 - FatalError, Error, Warning and Trace
  9 - FatalError, Error, Warning and Trace (and additional HTML files)
- **filename** -The name of the trace file

3.12.4.2. Example of an Export Configuration File

The export configuration file has the format defined above The following generic example describes shows the general layout. The attribute values that can be configured are shown in bold italic, e.g. ***level***:

```
<?xml version="1.0" encoding="UTF-8" ?>
<job>
  <connector name="Default Controller" role="controller">
    <logging level="level" filename="traceFileName"/>
  </connector>
  <connector name="SiPass Connector" role="connector">
    <connection type="SiPass"
      user="account"
      password="password"
      server="server">
      <property name="Export_Date_Format" value="validSiPassDate"/>
    </connection>
  </connector>
  <connector name="LDIF File Writer" role="responseWriter">
    <connection type="LDIF" filename="outputFile" />
  </connector>
</job>
```

3.12.4.3. Specific Parameters of the SiPass Export Configuration File

This section describes the specific information for the SiPass agent for the export operation. For the general structure of this file and standard parameters see the section Configuration File Formats in the General Configuration section.

The SiPass agent export operation is configured by the attributes shown in the following table:

Connector	Parameter	Description
Default Controller	level	The logging level (for details, see the General Structure of a Configuration File).
	filename	The trace file name.
SiPass Connector	user	The user for binding to the SiPass system.
	password	The user password.
	server	The host name or the IP address of the SiPass system.
	Export_Date_Format	The date format used internally in SiPass DB. The SiPass agent needs to know what kind of date format can be obtained from the SiPass API in order to be able to automatically convert this value to DirX Identity native date format (generalized time format). To get a validSiPassDate only a permutation of the string “dmy“ (day/month/year) with separator “/“ or “.“ can be used. For example format string “m/d/y“ means that the month is followed by the day and the year. Use trace level 9 to set this format string to the correct values.
LDIF File Writer	filename	The name of the LDIF response file that contains the exported card holder or workgroup entries.

3.12.4.4. Example of an Import Configuration File

The import configuration file has the format defined in the general section. The following template describes its configuration. The attribute values that you can configure are shown in bold italic, e.g. ***level***:

```
<?xml version="1.0" encoding="UTF-8" ?>
<job>
  <connector name="Default Controller" role="controller">
    <logging level="level" filename="traceFileName"/>
  </connector>
  <connector name="LDIF-change File Reader" role="reader">
    <connection filename="inputFileName"/>
  </connector>
  <connector name="SiPass Connector" role="connector">
```

```
    <connection type="SiPass"
      user="account"
      password="password"
      server="server">
      <property name="Exact_Add_Action" value="true or false"/>
      <property name="Exact_Modify_Action" value="true or false"/>
      <property name="Import_Date_Format" value="validSiPassDate"/>
    </connection>
  </connector>
</job>
```

3.12.4.5. Specific Parameters of the SiPass Import Configuration File

This section describes the specific information for the SiPass agent for the import operation. For general structure of this file and standard parameters see the section Configuration File Formats in the General Configuration section.

The SiPass agent import operation is configured by the following attributes:

Connector	Parameter	Description
Default Controller	level	The logging level (for details, see the General Structure of a Configuration File).
	filename	The trace file name.
LDIF-change File Reader	filename	The name of the LDIF-change formatted input file.
SiPass Connector	user	The user for binding to the SiPass system.
	password	The user password.
	server	The host name or the IP address of the SiPass system.
	Exact_Add_Action	Only an add action is allowed for incoming add requests if this switch is set to true. If set to false, a modify operation can be performed alternatively if a card holder with the specified reference number already exists in the SiPass system.
	Exact_Modify_Action	Only a modify action is allowed for incoming modify requests if this switch is set to true. If set to false an add operation is automatically performed if no user with the specified reference number could be found in the SiPass system.

Connector	Parameter	Description
	Import_Date_Format	The date format used internally in the SiPass DB. The SiPass agent needs to know what kind of date format is expected by the SiPass API in order to be able to automatically convert strings with DirX Identity native date format (generalized time format) to valid SiPass date format. This conversion is done only if an incoming generalized time format for SiPass attributes StartDate and EndDate is detected. validSiPassDate can be only a permutation of the string “dmy“ (day/month/year) with separator “/“ or “.“.For example format string “m/d/y“ means that the month is followed by the day and the year. Use trace level 9 in order to set this format string to the correct values.

3.12.5. Data File Formats

A search request creates an export file in LDIF content format that contains the search result. Note that the identifiers of the card holders (employees) are Reference attributes. Workgroups are always exported with their name and void status.

3.12.5.1. Import Data File Format

The import data file format recognized by the SiPass meta agent is LDIF-change file format. The data has to be provided in UTF-8 character set (or US-ASCII), not in ISO8859-1 (Latin-1).

The supported change types are add, modify, and delete; modifyDN is not supported.

The SiPass agent does not process multi-value attributes. An error is issued if a multi-value attribute is detected and the whole LDIF-entry with such an attribute is ignored.

During add operations, the SiPass system requires the mandatory attributes CardNumber, FirstName, LastName, Reference number, StartDate and EndDate.

If StartDate is not provided in the input file, it is set to the current date.

If EndDate is not provided in the input file, it is set to the current date plus 100 years.

If the Workgroup attribute or the corresponding value is missing or invalid the user cannot be added.

3.12.5.1.1. Examples

This section provides some examples of input data files.

Example 8. Create User

```
dn:5367
changetype:add
CardNumber:367
```

```
FirstName:Aliza
LastName:Dasrath
StartDate:30/3/2005
EndDate:20050531121745Z
Workgroup:Developers
```

A user with the reference number 5367 for the person Aliza Dasrath with card number 367 valid from 30/03/05 is created. Mandatory attributes for creating a user are CardNumber, FirstName, LastName and Reference number (internally produced from the DN attribute).

The mandatory attributes StartDate, EndDate are automatically generated by the SiPass agent.

Example 9. Change User

```
dn:5367
changetype:modify
replace:Workgroup
Workgroup:ProjectLeaders
-
```

A new workgroup is assigned to the card holder with reference number 5367.

Example 10. Delete User

```
dn:5367
changetype:delete
```

The card holder with the reference number 5367 will be deleted.

3.12.6. Search Request File

The search request file defines the search to be performed in the SiPass server. It is SPML compliant.

3.12.6.1. Example of a Search Request File Format

The objects to be exported are defined in a Service Provisioning Markup Language (SPML) search request. SPML is a specialized XML format. The following template describes its configuration. The attribute values that can be configured are shown in bold (blue) italic, e.g., ***typeOfExportedObject***:

```
<?xml version="1.0" encoding="UTF-8" ?>
<spml:searchRequest xmlns:dsml="urn:oasis:names:tc:DSML:2:0:core"
xmlns:spml="urn:oasis:names:tc:SPML:1:0" requestID="search-01">
  <spml:filter>
    <dsml:substring name="objecttype">
      <dsml:value>
        typeOfExportedObject
      </dsml:value>
    </dsml:substring>
  </spml:filter>
  <spml:attributes>
    <attribute name="firstAttribute"/>
    ...
    <attribute name="lastAttribute"/>
  </spml:attributes>
</spml:searchRequest>
```

spml:filter

smpl:filter specifies what kind of SiPass object shall be exported. The value ***typeOfExportedObject*** can be either “employee“ for exporting SiPass card holders or “workgroup“ for exporting workgroups.

spml:attributes

smpl:attributes specifies the attributes ***firstAttribute ... lastAttribute*** to be returned by the search (only necessary for exporting card holders).

4. Event Listeners and Triggers

DirX Identity event listeners and triggers capture or generate events in applications or target systems and transfer them to the Messaging server. Event managers can subscribe to these events and process them.

4.1. Microsoft Windows Password Listener

The Windows Password Listener for Microsoft Windows environments captures passwords in clear text from the Windows Domain Controller and transfers them to the messaging service, where the User Password Event Manager picks them up for further processing. The Windows Password Listener captures only passwords of user accounts. For more information on password management, see the chapter "Password Management" in the *DirX Identity Connectivity Administration Guide*.

4.1.1. Architecture

The Windows Password Listener consists of two components:

- **Windows Password Listener Plug-in** - A small component that is integrated as a dynamic link library (DLL) into the Windows domain controller's Local Security Authority (LSA), to which it is registered as a password filter.
- **Windows Password Listener Service** - A separate component that runs independently and thus cannot disturb the Windows domain controller operation.

Both components transfer all information in Unicode / utf-8 code set.

The following figure illustrates the architecture of the Windows Password Listener.

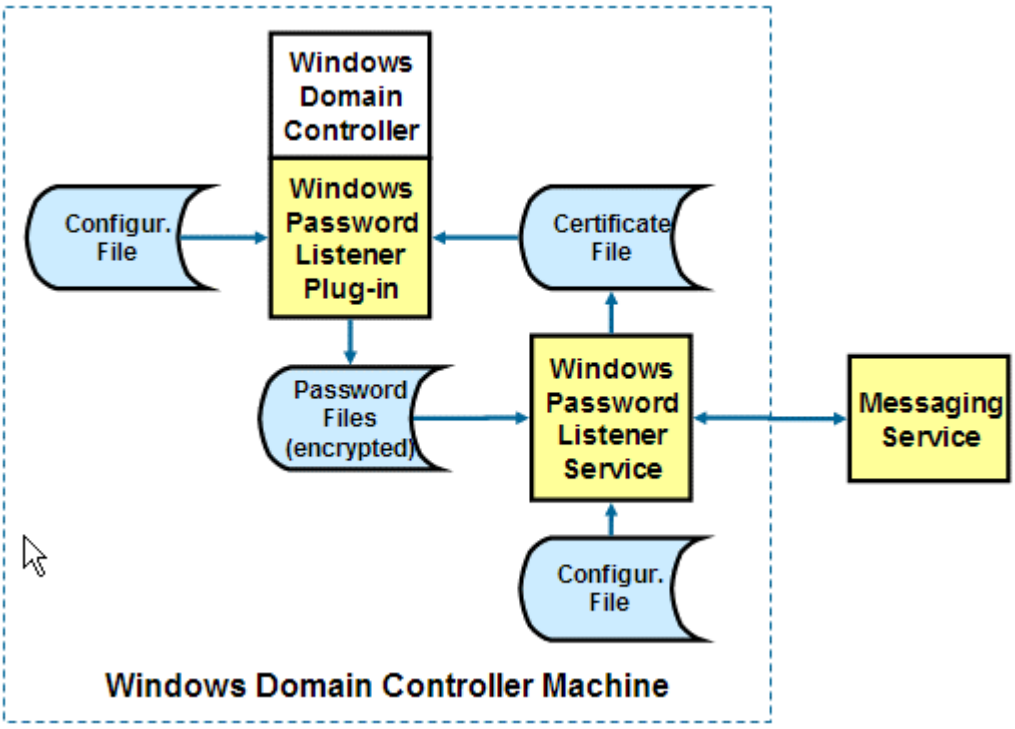


Figure 31. Windows Password Listener Architecture

4.1.1.1. Windows Password Listener Plug-In

The Windows Password Listener plug-in is a DLL that is based on the Microsoft Security Management Password Filters functions. Specifically, it uses the **InitializeChangeNotify** and **PasswordChangeNotify** functions, as follows:

- During Windows domain controller start-up, the Password Listener plug-in calls the function InitializeChangeNotify to initialize the client.
- Each time a password change occurs, the Password Listener plug-in calls the function PasswordChangeNotify. The client reads the certificate file that contains the public key, encrypts the password and writes it together with the account name into a password file. If the certificate file does not exist the password is encrypted with a default internal public key. As soon as a certificate file has been retrieved from the Java-based Server on Windows Password Listener service start-up, the password files are decrypted by the service with a default internal private key, encrypted with the certificate from the file and then sent to the Java-based Server. The name of these password files are *timestamp_rel_id*, for example 20040629111230487_1107.

This interface generates a notification of any password change event.

4.1.1.2. Windows Password Listener Service

During start-up, the Windows Password Listener service reads its configuration file (**options.ini**) and tries to connect to the messaging service specified in ServerAddress and ServerPort. It requests the current certificate (public key for encryption) and writes it into a file to be used by the Windows password listener plug-in (CertFileName). During start-up, it also requests the list of messaging services from a Java-based Server with a ConfigurationHandler adaptor configured and active and writes them into the file specified

in MsgSrvFile (default = **msgServers.ini**). The file is always updated when a current list of messaging services is received from the Java-based Server. The Java-based Server repeats sending it at an interval configured in the specific attribute **broadcastinterval** of the ConfigurationHandler adaptor. The MsgSrvFile is deleted on update and upgrade installations to make sure that the service tries to connect to the messaging service specified during installation and not to the first - possibly outdated one-listed in the MsgSrvFile.

If at some time later the Windows Password Listener service loses the connection to the original messaging service, it tries to connect to one of the messaging services listed in MsgSrvFile processing one after another. The timing behavior for building up a connection to a messaging service can be controlled with the parameters Timeout, Repeat and RetryInterval. Once a connection to another messaging service succeeds this messaging service and its host name, messaging port, SSL and client authentication flag is written to the MsgSrvFile as the first one in the list of messaging services. These parameters are also written back to the options.ini file, so that on the next Windows Password Listener service restart this messaging service is tried first to connect to.

The Windows Password Listener service checks for existing password files in the directory **DirectoryPasswordFiles**, reads them, adds the additional information forest name, domain name, computer name and the "User must change password during next login" flag depending on whether the password was reset by an administrator (flag is set to 1) or by the user itself (flag is set to 0) and then sends each password change request via the messaging service to the Java-based Server. If successful, it deletes the password file and then waits for further password changes.

If the Password Listener service, when processing a password file, recognizes that the assigned user no longer exists in Active Directory or that access to the user is denied, it writes a warning message to the Event Viewer and moves the password file to the folder **PasswordFiles_moved** residing parallel to the **PasswordFiles** folder.

The operating system automatically notifies the Windows Password Listener service when a password file is created. The service then reads the password file, processes it and deletes it if send successfully.

4.1.2. Configuration File Format

The Windows Password Listener plug-in and service components are controlled by different configuration files. The next sections identify these files and describe the parameters in them that control the plug-in or service. Most of these parameters are set automatically by the installation procedure.

4.1.2.1. Windows Password Listener Plug-In Configuration File

The **libdxmEventListenerAds.ini** file contains parameters that control the operation of the Windows password listener plug-in. The file is located in the directory

*install_path***conf**

Settings

ListenerActive

controls whether the event listener can be deactivated without rebooting the Windows domain controller machine. A value of **1** means active, a value of **0** means inactive (default: 1). This flag is checked during each password change operation.

Trace

defines the trace level for the information written to the Event Viewer and the file defined in the LogFile parameter of the File Definitions section:
0 - no trace (default)
1 - write warnings to Event Viewer
2 - write warnings and infos to Event Viewer
3 - write warnings and infos to Event Viewer and more detailed tracing to LogFile

File Definitions

DirectoryPasswordFiles

the directory where the intermediate encrypted password files are stored (default: *install_path**\PasswordFiles*)

CertFileName

the name of the file that contains the current certificate to be used for encryption of the passwords (default: *install_path***CertFile\Certificate.txt**)

LogFile

the path and name of the file where trace information is written to (see the Trace parameter above). The default location is: *install_path***log\dxmEventListener.log**.

Encryption Parameters

EncrLogLevel

the log level for the encryption module. The value is an integer in the range 1 (minimal logging) to 10 (maximum logging) (default: 1)

EncrLogFile

the location where the encryption module writes log information (default: *install_path***log\dxmEventListener_dxc_crypt.log**).

EncrRandomFile - the random file that the encryption mechanism uses (default: *install_path***log\dxmEventListener_dxc_rand.dat**).

WaitTime - the time (in milliseconds) that the plug-in is to wait for the certificate (default: 5000 ms).

This section is not intended to be used by the customer directly. Support personnel will instruct the customer to use this feature in case of specific errors. Normally all error information is reported to the Windows Event Viewer.

4.1.2.2. Windows Password Listener Service Configuration File

The **options.ini** file contains parameters that control the operation of the Windows Password Listener service. The file is located in the directory

*install_path***conf**

Settings

Trace

defines the trace level for the information written to the Event Viewer and the file defined in the LogFile parameter of the File Definitions section:*
0* - no trace (default)*
1* - write warnings to **Event Viewer**
2 - write warnings and infos to **Event Viewer**
3 - write warnings and infos to Event Viewer and more detailed tracing to LogFile

Message Server

ServerAddress

the messaging service address.

ServerPort

the messaging service port.

ServerDisplayName

the display name of the Java-based Server that contains the messaging service. It is used as the section name of a messaging service in the MsgSrvFile.

PollingTime

the time in seconds that the service is to wait for messages at the messaging interface (default: 5 seconds).

MessageLifeTime

the expiration time of sent messages (default: 86400 seconds = 1 day). The messaging service automatically deletes timed-out messages.

WaitTime

the time (in milliseconds) that the service is to wait for access to a semaphore to avoid deadlocks (default: 5000 milliseconds).

Timeout

the time (in seconds) that the service is to wait for a connection when attempting to connect to the messaging service during start-up. (default: 30 seconds).

Repeat

the number of connection retries the service is to make to the messaging service when the **Timeout** parameter is reached (default: 10).

RetryInterval

interval to retry a connect operation to the messaging service after a temporary

problem (for example network was not available). Default is 15 minutes.

UseSSL

if set to 1, the messaging service works with SSL (default: 0).

Messages

SubscriberQueue

the name of the message queue that is to receive messages (default: Dxm.event.QUEUE)

CommandPrefix

the name of the command prefix (default: dxm.event)

Stream

the internal queue (default: Dxm.event.STREAM)

SendPasswordTopicSuffix

the suffix of the topic to which password change requests are sent (default: **pwd.changed**, which is combined with the CommandPrefix to result in **dxm.event.pwd.changed**).

DomainTopicPrefix

the domain name prefix (default: **My-Company**) of the topic to which password change requests are sent. By default this key is set into comment. You should uncomment it and specify your own domain. Then the password change requests are combined with the domain name prefix resulting in *domain_name**.dxm.event.pwd.changed*, for example, **My-Company.dxm.event.pwd.changed**. In this case, a Java-based Server configured to this domain and which has the flag **Include domain into topic** set receives these requests.

SendReqConfTopic

the topic to which configuration requests are sent. Configuration requests are sent with the appropriate flag for getting either the certificate or the list of configured messaging services. If a domain topic prefix is specified, it is prefixed to this request configuration topic.

ReceiveConfTopic

the topic to which a durable subscriber is created from where the requested or updated certificate or messaging service list is read. If a domain topic prefix is specified, it is prefixed to this response configuration topic.

ExpirationTime

the time at which the sent messages expire (at which the messaging service automatically deletes it). This parameter is used for both password and certificate requests (default: 24 hours). Setting an expiration time prevents sent messages from remaining indefinitely in the system.

File Definitions

CertFileName

the location where the certificate file is to be stored (default: *install_path**\CertFile\Certificate.txt*).

DirectoryPasswordFiles

the location where the encrypted password change request files are written (default: *install_path***PasswordFiles**).

LogFile

the path and name of the file where trace information is written to (see the Trace parameter above). The default location is: *install_path***log\dxmWinPwListener.log**.

MsgSrvFile

the path and name of the file to which the list of messaging services is written at start-up. The default location is: *install_path***conf\msgServers.ini**.

Encryption Parameters

EncrLogLevel

the log level for the encryption module. The value is an integer in the range 1 (minimal logging) to 10 (maximum logging) (default: 1).

EncrLogFile

the location where the encryption module writes log information (default: *install_path***log\dxmWinPwListener_dxc_crypt.log**).

EncrRandomFile

the random file that the encryption mechanism uses (default: *install_path***conf\dxmWinPwListener_dxc_rand.dat**).

+ NOTE: This section is not intended to be used by the customer directly. Support personnel will instruct the customer to use this feature in case of specific errors. Normally, all error information is reported to the Windows Event Viewer.

4.1.3. Error Handling

The Windows Password Listener can handle the following error situations:

- Windows Password Listener plug-in - if the certificate file is not available or readable, the Windows password listener plug-in encrypts incoming passwords with the default internal public certificate.
- Windows Password Listener plug-in - because DirX Identity can only partially work with the full Unicode character set, the Windows Password Listener blocks password changes for account names containing Unicode characters that cannot be transformed to Latin 1 character set. The plug-in writes a message to the Windows Event Viewer (Application Log).
- Windows Password Listener service - the configuration file is not available or not readable or it contains incorrect parameters. The service writes an error message to the

Windows Event Viewer (Application Log).

- Windows Password Listener service - the certificate cannot be retrieved from the Java-based Server during startup of the Windows Password Listener service (due to an unavailable messaging service or a non-existing certificate in the Connectivity LDAP directory or an inactive ConfigurationHandler adaptor). In this case, the certificate file cannot be written and the service cannot process password change requests. The Windows Password Listener service checks regularly for the certificate's availability. When the certificate can be retrieved, the Windows Password Listener service starts to process the existing password files: it decrypts each password with the default internal private key, encrypts it with the retrieved public key and sends it to the Java server via the messaging service.

4.2. Web Event Trigger

The Web event trigger contains Java classes that encrypt an attribute (RSACipher) and publish events (SharedEventPublisher or CumulativeEventPublisher). You can integrate these classes into your Web application to build password maintenance clients. In scenarios where each message does not need to be evaluated, we recommend that you use CumulativeEventPublisher.

To store a password in a dxrUser entry, use the helper class PasswordSupport. This class ensures the integrity of the attributes userPassword, dxmPassword and dxmPwdLastChange.

Alternatively, you can use the Web event trigger to send just an event (no data change is performed). In this case, it is not necessary to read a certificate (public key) from the directory.

We provide some test clients that allow you to perform tests without a Web server environment (for example performance tests with a well defined load profile).

Note: The use of the Java classes is not restricted to Web applications; they can be used in any Java application.

A typical application of the classes in a Web application for a user password change is:

- The user requests to change his password.
- The user must enter the old password and enter the new password twice.
- The connection to the LDAP server should change to an SSL connection, and the connection to the Web server should be HTTPS-based.
- The Web application performs an authentication with the user DN and the old password to the LDAP directory.
- The Web application reads the public key from the server_admin account in the configuration directory, initializes the cipher class [RSACipher.init()] and encrypts the new password [RSACipher.encrypt()].
- Next, the Web application performs the necessary changes at the user entry (the prerequisite is that the user has enough access rights to change his password in the

directory). It sets:

- The userPassword attribute (be sure that the directory server performs a hash operation) to the new password value (use the java class PasswordSupport to store userPassword and dxmPassword).
- The dxmPassword attribute to the encrypted new password value (see java class PasswordSupport).
- The dxmOprTriggerOrigin attribute to a value not equal to any Active Directory domain (for example 'WebEventTrigger').
- The dxmADsResetUserPassword attribute to FALSE.

all these attributes must be set in one LDAP operation to guarantee data consistency (see java class PasswordSupport).

- Finally, it sends an event message to the messaging service to inform the event manager [fireEvent()].

A typical application of the classes in a Web application for an administrative password reset is:

- The user requests to reset the password from the administrator (this is a process that is out of scope for DirX Identity).
- The administrator authenticates to the LDAP directory with an administrative account (if he has not already done so).
- The connection to the LDAP server should be a SSL connection.
- The administrator searches for the user entry.
- The administrator initiates a password reset. This action is an application- and customer-specific procedure that can be chosen freely (for example, the administrator clicks a Reset Password button and the default password is automatically calculated).
- The application reads the public key from the server_admin account in the configuration directory, initializes the cipher class [RSACipher.init()] and encrypts the default password [RSACipher.encrypt()].
- The application next performs the necessary changes at the user entry (the prerequisite is that the administrator has enough access rights to change his password in the directory). It sets
- The userPassword attribute (be sure that the directory server performs a hash operation) to the default password value.
- The dxmPassword attribute to the encrypted default password value.
- The dxmOprTriggerOrigin attribute to a value that is not equal to any Active Directory domain (for example, 'WebEventTrigger').
- The dxmADsResetUserPassword attribute to TRUE.

all these attributes must be set in one LDAP operation to guarantee data consistency.

- The application sends an event message to the message server to inform the event manager [fireEvent()].
- The application sends the user an e-mail with the value of the default password.

4.2.1. Web Event Trigger Java Classes

Web event trigger Java classes include Java classes for encryption, event management, and message publishing. The next sections briefly describe these classes. See the Java classes documentation contained on your DirX Identity DVD for a more detailed description of these classes:

Documentation\DirXIdentity\eventing_docu.zip

4.2.1.1. Java Classes for Encryption

The RSACipher class provides RSA encryption and decryption facilities. It is initialized with an X.509 certificate or a private key [RSACipher.init()], takes a clear text parameter and encrypts it [RSACipher.encrypt()] or decrypts a byte buffer and returns a string [RSACipher.decrypt()].

For the detailed interface description, see the Java documentation "RSACipher.html" on your DirX Identity DVD. It contains a source code fragment that shows how to construct the class, read and hand over the certificate [RSACipher.init()] and encrypt some text [RSACipher.encrypt()].

4.2.1.2. Java Classes for Event Management

Two classes are provided for event management: SharedEventPublisher publishes each message. CumulativeEventPublisher sends one message per type in a given time interval.

4.2.1.2.1. Java Class SharedEventPublisher

This class publishes messages to the ActiveMQ message broker. The thread-safe implementation uses just one connection to the messaging service.

The Java documentation "EventPublisher.html" contains a code snippet that shows how to initialize an event publisher and send a message.

The constructor needs the topic to publish to and an initial context parameter to connect to the messaging service. The method fireEvent() takes an LDAP entry and a reason string. It writes the attributes of the LDAP entry into the "entry" field, the reason parameter into the reason field of the message and publishes it to the given topic.

This class is designed for a multi-threading environment and can be shared among several threads. It uses a separate worker thread that sends the events to the messaging service. The method fireEvent() simply delegates the operation to this worker using a backlog list. Thus it never blocks and the client application continues immediately.

For the interface documentation, see the Java documentation "SharedEventPublisher.html" on your DirX Identity DVD.

4.2.1.2.2. Java Class CumulativeEventPublisher

This class cumulates JMS messages using their type (for example, "passwordChanged") and publishes only ONE message within a given time interval to the messaging service. This strategy keeps message traffic and CPU consumption to a minimum. It can be applied when the event handler does not need to evaluate each message.

The method setTimerInterval() takes the timer as a long value and re-initializes it. The method fireEvent() takes an LDAP entry and a reason and publishes it to the configured messaging service.

The timer interval should be coordinated with the timer used by the event handler.

For detailed information, see the Java documentation "CumulativeEventPublisher.html" on your DirX Identity DVD.

4.2.1.2.3. Java Class PasswordSupport

Apart from setting userPassword and dxmPassword, PasswordSupport.storePassword calculates dxmPwdLastChange from the localtime and the ldap-server's time. Note that these timestamps may differ, since the local machine and the server machine may be located in different time zones. Even if both machines are within the same time zone, times may be out of sync.

DxmPwdLastChange MUST be a servertime, local time would be meaningless. The calculation of the servertime from the localtime is done in several steps:

- Firstly, storePassword creates a modification set comprising userPassword and the encrypted dxmPassword and modifies the respective LDAPEnty. This very entry is re-read to get the modifyTimestamp. The delta of servertime and localtime is calculated as "servertime - localtime".
- Secondly, after the delta has been calculated, consecutive storePassword calls will calculate the servertime by "servertime = localtime + delta" and modify userPassword, dxmPassword AND dxmPwdLastChange in one ldap-modify operation. The entry will be re-read in order to get the modification timestamp, from which a new delta will be calculated. If this new delta differs significantly (ie 2s) from the old delta, dxmPwdLastChange will be re-written accordingly.

4.2.1.3. Jar File Deployment

Running your Web event trigger application requires the deployment of some Java classes.

4.2.1.3.1. Jar files to be placed in the web application (i.e. tomcat)

Folder *WEBAPP*/web_inf/lib:

storage:

dxcCrypto.jar Crypto functionality

dxcLogging.jar Logging

bcprov-jdk14-116.jar (encryption support)

DirXjdiscoverAPI.jar (used by storage.jar)

storage.jar (used by dxmStorage.jar, generic data storage)

dxmStorage.jar (Connectivity configuration data storage)

ldapjdk.jar (LDAP sdk)

js.jar (JavaScript from Rhino project)

misc:

crimson.jar 3rd party: XML parsing

jaxp.jar 3rd party: XML Parsing

4.2.1.3.2. Jars to be placed into the endorsed directory (i.e. tomcat: common/endorsed)

dxmStorageURL.jar (implements storage:// URLs)

storage.jar

4.2.2. Web Event Trigger Test Clients

DirX Identity provides several clients that allow you to simulate Web and WPL event trigger applications:

- Data encryption client
- Stress test client
- WET Password Generator client using PasswordSuppport
- WPL Simulator client (simulates Windows Password Listener compatible events)

The following sections describe how to configure and run these clients. Test clients are also available that simulate password changes.

4.2.2.1. The Data Encryption Client

The data encryption client simulates the change of data through an end-user Web client. It reads a definable set of entries, takes one of the attributes (if it exists), encrypts it and writes another attribute (or the same attribute). If the change is successful, it creates an event.

Configure the data encryption client with **client.cfg** file, then run the script **runClient.bat**. Configuration parameters are:

Common parameters:

- **trace** - sets the trace details (use values between 1 and 3)
- **tf** (or equivalently: **tfile**) - the trace file where the trace output will be written. If absent, no trace output will be written.

Configuration database parameters:

- **confdb.host** - the host name where the configuration database resides

- **confdb.port** - the port of the configuration database
- **confdb.user** - the user to authenticate to the configuration database
- **confdb.password** - the password
- **confdb.certdn** - the DN of the entry that keeps the public key (the certificate) for data encryption.

Messaging service parameters:

- **messageServer** - set the messaging service type:*
  ATS* - deprecated
- **mq.appid** - The test client's unique identifier (required by JMS)
- **mq.queuename** - the name of the activeMQ queue
  The default value in **client.cfg** is *domain*.dxm.event.pwd.changed*. Make sure that the domain name is included in the queue name if the flag **Include Domain into topic** is set at the at the domain object. Otherwise drop the prefix "*domain*.*".
- **mq.expirationtime** - the time the sent messages will expire. Default is one day (24 h).

Event body definition:

- **source.application** - the web application name (written to dxmOprOrigin by the event manager)
- **source.type** - the source type
- **source.resource** - the resource, for example the AdsDomain.
- **source.cluster** - the cluster, for example the AdsForest
- **source.computername** - the name of the computer
- **source.DNSdomainname** - the name of the DNSdomain
- **attr.username** - the user name

The fields from **source.resource** down to **attr.username** can be used to distribute the messages between multiple event managers. Normally the user name is the best attribute for statistical distribution.

Data server parameters:

- **host** - the host name of the database where the user entries reside
- **port** - the port number
- **user** - the user for authentication
- **password** - the password
- **basedn** - the base DN at which to read the entries to encrypt
- **searchfilter** - the search filter
- **cleartextName** - the name of the attribute from which to read the clear text. If this attribute does not exist at an entry, an event is not generated.
- **cyphertextName** - the name of the attribute at which to write the encrypted value.

Note: if this line is commented, encryption does not take place.

4.2.2.2. The Stress Test Client

The stress test client creates events for a stress test of the event server. Configure this client with the file **stress.cfg** and then run the script **runStress.bat**. The configuration parameters are:

Common parameters:

- **trace** - sets the level of trace information (use values between 1 and 3)
- **tf** (or equivalently: **tfile**) - the trace file where the trace output will be written. If absent, no trace output will be written.

Messaging service parameters:

- **messageServer** - set the messaging service type:*
  ATS* - deprecated
- **mq.appid** - The test client's unique identifier (required by JMS)
- **mq.queuename** - the name of the activeMQ queue
  The default value in **stress.cfg** is *domain**.dxm.event.pwd.changed*. Make sure that the domain name is included in the queue name if the flag **Include Domain into topic** is set at the at the domain object. Otherwise drop the prefix "*domain**.*".
- **mq.expirationtime** - the time the sent messages expire. Default is one day (24 h).

Event body definition:

- **source.application** - the Web application name (written to dxmOprOrigin by the event manager).
- **source.type** - the source type.
- **source.resource** - the resource, for example the AdsDomain.
- **source.cluster** - the cluster, for example the AdsForest.
- **source.computername** - the name of the computer.
- **source.DNSdomainname** - the name of the DNSdomain.
- **attr.username** - the user name.

The fields from **source.resource** down to **attr.username** can be used to distribute the messages between multiple event managers. Normally the user name is the best attribute for statistical distribution.

Generator parameters:

- **cycles** - the number of test cycles.
- **clients** - the number of clients per cycle.
- **events** - the number of events sent by a client.
- **sleep** - the number of seconds to sleep after a cycle has been processed.

4.2.2.3. The WET Password Generator client

The WebEvent Trigger Password Generator client allows you to generate password events that are identical to events coming from the Web event trigger. Based on a definable collection of users in your user directory, this client creates random passwords either in sequence (as the search result is retrieved) or randomly. The fields in the sent message are definable. The password is written to the user entry in the directory and not part of the message.

Configure the data encryption client with **password.cfg** file, then run the script **runPassword.bat**. Configuration parameters are:

Common parameters:

- **trace** - sets the trace details (use values between 1 and 3).
- **tf** (or equivalently: **tfile**) - the trace file where the trace output will be written. If absent, no trace output will be written.

Configuration database parameters:

- **confdb.host** - the host name where the configuration database resides.
- **confdb.port** - the port of the configuration database.
- **confdb.user** - the user to authenticate to the configuration database.
- **confdb.password** - the password.
- **confdb.certdn** - the DN of the entry that keeps the public key (the certificate) for data encryption.

Messaging service parameters:

- **messageServer** - set the messaging service type:*
  ATS* - deprecated
- **mq.appid** - The test client's unique identifier (required by the JMS).
- **mq.queuename** - the name of the activeMQ queue
  The default value in **stress.cfg** is *domain**.dxm.event.pwd.changed*. Make sure that the domain name is included in the queue name if the flag **Include Domain into topic** is set at the at the domain object. Otherwise drop the prefix "*domain**.*".
- **mq.expirationtime** - the duration after which sent messages expire. Default is one day (24 h).
- **source.application** - the Web application name (written to dxmOprOrigin by the event manager).

Event body definitions:

- **source.type** - the source type.
- **source.resource** - the resource; for example, the AdsDomain.
- **source.cluster** - the cluster, for example the AdsForest.

- **source.computername** - the name of the computer.
- **source.DNSdomainname** - the name of the DNSdomain.
- **attr.username** - the user name.

The fields from **source.resource** down to **attr.username** can be used to distribute the messages between multiple event managers. Normally the user name is the best attribute for statistical distribution.

Data server parameters:

- **host** - the host name of the database where the user entries reside.
- **port** - the port number.
- **user** - the user for authentication.
- **password** - the password.
- **basedn** - the base DN at which to read the entries to encrypt.
- **searchfilter** - the search filter.
- **sizelimit** - for the LDAP request.
- **timelimit** - for the LDAP request.

Generator parameters:

- **random** - this parameter has two modes:*
  0* - during each cycle, it reads all entries from the directory that are specified in the search and generates a password change message for each of it (if it finds 100 entries, exactly 100 password change messages are created). The sequence depends on the sequence of the search result.*
  >0* - during each cycle it generates this number of password changes by selecting the entries randomly from the directory.
- **cycles** - the number of generation cycles to be performed.
- **sleep** - the number of seconds to wait between cycles (0 means no wait time).

4.2.2.4. The WPL Simulator client

The Windows Password Listener simulator client allows you to generate password events that are identical to events coming from the Windows Password Listener. Based on a definable collection of users in your user directory, this client creates random passwords either in sequence (as the search result is retrieved) or per random. The fields in the sent message are definable. The encrypted password is always part of the message.

Configure the data encryption client with **adsSimulator.cfg** file, then run the script **runADSSimulator.bat**. Configuration parameters are:

Common parameters:

- **trace** - sets the trace details (use values between 1 and 3).
- **tf** (or equivalently: **tfile**) - the trace file where the trace output will be written. If absent,

no trace output will be written.

Configuration database parameters:

- **confdb.host** - the host name where the configuration database resides.
- **confdb.port** - the port of the configuration database.
- **confdb.user** - the user to authenticate to the configuration database.
- **confdb.password** - the password.
- **confdb.certdn** - the DN of the entry that keeps the public key (the certificate) for data encryption.

Messaging service parameters:

- **messageServer** - set the messaging service type:
  ATS - deprecated
- **mq.appid** - the test client's unique identifier (required by JMS).
- **mq.queuename** - the name of the activeMQ queue
  The default value in **adsSimulator.cfg** is *domain*.**dxm.event.pwd.changed**. Make sure that the domain name is included in the queue name if the flag **Include Domain into topic** is set at the domain object. Otherwise, drop the prefix "*domain*".
- **mq.expirationtime** (optional) - the duration after which the sent messages expire. Default is one day (24 h).

Event body definition:

- **event.body** - allows you to define a text file that contains a message template. This message template can contain placeholders in the form ${<user_ldap_attribute>} that are substituted during runtime.

Data server parameters:

- **host** - the host name of the database where the user entries reside.
- **port** - the port number.
- **user** - the user for authentication.
- **password** - the password.
- **basedn** - the base DN at which to read the entries to encrypt.
- **searchfilter** - the search filter.
- **sizelimit** - for the LDAP request.
- **timelimit** - for the LDAP request.

Generator parameters

- **random** - this parameter has two modes:*
  0* - during each cycle, it reads all entries from the directory that are specified in the search and generates a password change message for each of it (if it finds 100 entries,

exactly 100 password change messages are created). The sequence depends on the sequence of the search result.*
>0* - during each cycle it generates this number of password changes by selecting the entries randomly from the directory.

- **cycles** - the number of generation cycles to be performed.
- **sleep** - the number of seconds to wait between cycles (0 means no wait time).

Password policies:

This section allows you to define password policies to assure correct password creation. These policies are currently available:

- **pwd.minLength** - For a definition, see the *DirX Identity Provisioning Administration Guide* (default is 0).
- **pwd.maxLength** - For a definition, see the *DirX Identity Provisioning Administration Guide* (default is 0).
- **pwd.minUpperChar** - For a definition, see the *DirX Identity Provisioning Administration Guide* (default is 0).
- **pwd.minNumeric** - For a definition, see the *DirX Identity Provisioning Administration Guide* (default is 0).
- **pwd.minNonAlphaNum** - For a definition, see the *DirX Identity Provisioning Administration Guide* (default is 0).
- **pwd.minSpecialChar** - For a definition, see the *DirX Identity Provisioning Administration Guide* (default is 0).

You can also set a suffix for the generated passwords:

- **pwdsuffix** - Suffix for the generated passwords (the default is no suffix). For automatic testing, you can create fixed passwords together with minLength and maxLength. Example: set both values to 4 and the suffix to 'dirx'. This action generates fixed passwords for all persons. Be sure to switch of password history in this case.

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

EVIDEN

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.