Oracle® Enterprise Data Quality

Integration Guide

11*g* (11.1.1.9)

E55997-01

April 2015

ORACLE®

Oracle Fusion Middleware Integration Guide, 11*g* (11.1.1.9)

E55997-01

Primary Author: Oracle Corporation

Contents

Preface

Describes how to integrate Enterprise Data Quality with external systems and applications.

Audience

This document is intended for advanced users of EDQ and administrators responsible for integrating EDQ with third-party applications.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc.

Access to Oracle Support

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info or visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs if you are hearing impaired.

Related Documents

For more information, see the following documents in the Oracle Enterprise Data Quality documentation set.

EDQ Documentation Library

The following publications are provided to help you install and use EDQ:

- *Oracle Fusion Middleware Release Notes for Enterprise Data Quality*
- *Oracle Fusion Middleware Installing and Configuring Enterprise Data Quality*
- *Oracle Fusion Middleware Administering Enterprise Data Quality*
- *Oracle Fusion Middleware Understanding Enterprise Data Quality*
- *Oracle Fusion Middleware Integrating Enterprise Data Quality With External Systems*
- *Oracle Fusion Middleware Securing Oracle Enterprise Data Quality*
- *Oracle Enterprise Data Quality Address Verification Server Installation and Upgrade Guide*

- *Oracle Enterprise Data Quality Address Verification Server Release Notes*

Find the latest version of these guides and all of the Oracle product documentation at

http://http://docs.oracle.com

Online Help

Online help is provided for all Oracle Enterprise Data Quality user applications. It is accessed in each application by pressing the **F1** key or by clicking the Help icons. The main nodes in the Director project browser have integrated links to help pages. To access them, either select a node and then press **F1**, or right-click on an object in the Project Browser and then select **Help**. The EDQ processors in the Director Tool Palette have integrated help topics, as well. To access them, right-click on a processor on the canvas and then select **Processor Help**, or left-click on a processor on the canvas or tool palette and then press **F1**.

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

1

Integrating with Subversion

This chapter describes how to integrate and use EDQ with the Subversion version control system.

The following sections are included:

1.1 Software Requirements

EDQ supports integration with Subversion 1.6 and 1.7. For more information about Subversion, see the Apache Subversion website found at `http://subversion.apache.org/`.

> **Note:** EDQ currently only supports integration with Subversion 1.6 and 1.7. Attempting to integrate with any other versions will cause an error.

The Subversion server with which EDQ is being integrated must meet these prerequisites:

- Support Hypertext Transfer Protocol (HTTP) and Distributed Authoring and Versioning (DAV) access.
- Require authentication on commit.
- Not require authentication on checkout or update.

When Subversion is integrated with EDQ as a store of configuration information, the following restrictions and limitations apply. Consider the following points before deciding to configure integrated version control using Subversion.

- You cannot update or revert an item that is open in the Director interface or the Subversion server.

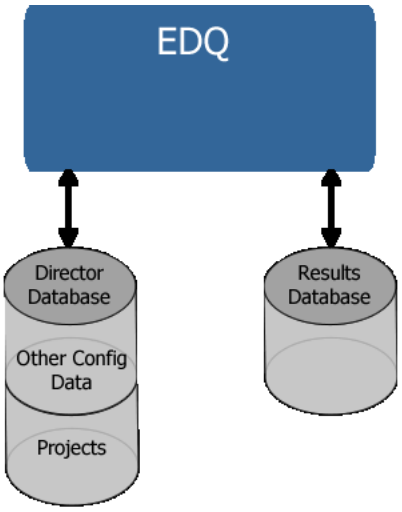
- You cannot rename a project once the project is under version control. This is critical in avoiding duplication of reference processor names in a project.
- Deleting a project does not remove it from the Subversion repository.
- Case insensitive name matching is used.

1.2 Understanding the Integration Architecture

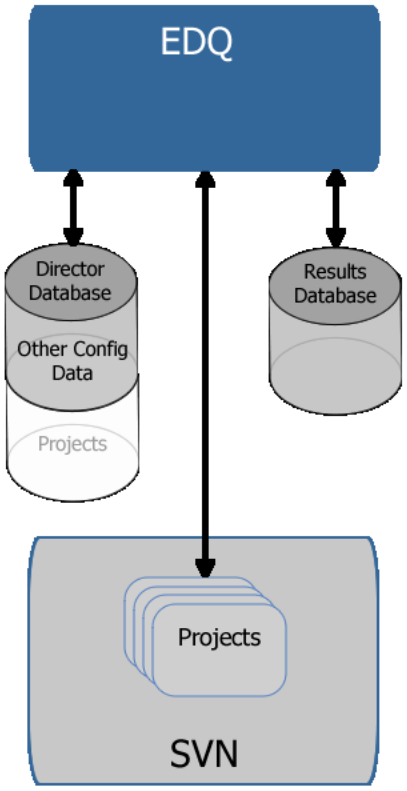
The EDQ server can be configured to be aware of a Subversion server as a store of configuration information.

Note: In this instance, configuration information means information that is managed using the Director UI; for example, projects and system-level data.

In a standard EDQ instance, configuration information, including project information, is stored in the Director database:



The following figure shows an EDQ instance integrated with Subversion:



Note: The Director database is still required because it contains data derived from the file-mastered configuration that has been normalized to allow querying by the applications.

With EDQ configuration files mastered and stored in a Subversion repository, a Subversion client can be used to commit or otherwise access them. Because EDQ includes an embedded Subversion client, Subversion client operations to control configuration changes can be performed directly in Director once the EDQ integration with Subversion has been enabled.

1.3 Setting Up a Repository

The first stage of configuration is to create a workspace directory where the checked out data will be stored:

1. Create a directory on the disk where desired (for example, `C:\MyRepository`) and then add it and commit it to Subversion.
2. Inside the newly created directory, set the following Subversion property:

   ```
   svn propset edq:systemversion 12.1.3:base .
   ```

3. Commit these changes into Subversion. Your workspace now displays these properties:

   ```
   svn proplist -v .
   Properties on '.':
     edq:systemversion
   ```

```
12.1.3:base
```

4. Create the following subdirectories in the newly created directory:
   - Data Stores
   - Hidden Reference Data
   - Images
   - Projects
   - Published Processors
   - Reference Data
5. Add and commit these directories. The repository is now set up correctly for EDQ.

The preceding steps only need to be performed once per repository. All remaining changes can be made using EDQ.

1.4 Configuring EDQ with Subversion

Subversion must be integrated with a fresh installation of EDQ.

Caution: When an EDQ instance is integrated with Subversion, all pre-existing and other configuration information is lost. To retain this information, you must package and export it first. For further details, see Section 1.4.2, "Retaining Existing Configuration Information."

Note: Oracle recommends that a single workspace be assigned to each instance of EDQ because it is difficult to move between workspaces in a single EDQ instance.

1.4.1 Configuring a New EDQ Installation

To configure a new EDQ installation:

1. Shut down the application server.
2. Check out the workspace from Subversion. It is not necessary to checkout the whole tree; just the workspace directory itself is required.
3. Edit the `director.properties` file in the *`ORACLE_HOME`*`/user_projects\domains\domains\edq_domain\edq\oedq.local.home` directory.
4. Add the following line replacing the directory path with that of the absolute path to your root workspace directory. For example:

   ```
   sccs.workspace = C\:\\MyRepository
   ```

   Note: This example demonstrates the need to escape colon (:) and backslash (\) characters in the path with a backslash. You must also escape space characters in the path with a backslash.

5. Start your EDQ server, and then start Director.

6. Check the top of the `Main0.log` file for an `INFO` message listing the name of the SCCS workspace you added. For example:

   ```
   INFO: 02-Sep-2013 10:05:21: SCCS workspace is C:\MyRepository
   ```

7. If no errors follow this message, EDQ is configured to use Subversion. If there are errors, see Section 1.7, "Troubleshooting Errors," for possible solutions.

1.4.2 Retaining Existing Configuration Information

As previously stated, Subversion must be integrated with a fresh installation of EDQ. Therefore, any pre-existing projects and other configuration items in an EDQ installation must be packaged before integration begins and then imported to the new installation afterwards:

1. Package all configuration items in the current EDQ instance into DXI files.
2. Install a new instance of EDQ with the Subversion integration enabled.
3. Import the DXI files into the new instance, and commit the files to the Subversion workspace.
4. Check that the configuration items are all valid and working correctly.

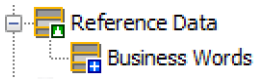
   Note that all passwords for Data Stores must be re-entered after a configuration import.
5. Decommission the previous instance.

1.5 Understanding the Integration Elements

Once EDQ is integrated with Subversion enabled, the following interface elements become visible within the Director application:

- Subversion status icon overlays in Project Browser - There are two icons used to indicate the three possible Subversion statuses of nodes in the Project Browser:
  - No icon - The node (and its sub-nodes) are all up to date.
  - Green icon- This node (and its sub-nodes) have modifications.
  - Blue icon - This node (and its sub-nodes) is new/currently not under Version Control.

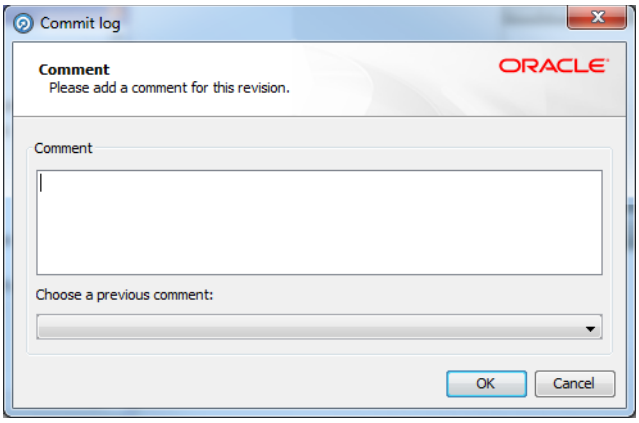
  For example, the following image shows both icons in use. The Reference Data node is modified (green icon) as one of its sub-nodes has changed. A new piece of Reference Data, Business Words, has been added, and is marked with the blue icon:



- Version Control tab - The Properties dialog (displayed by right-clicking on an item in the Project Browser and selecting **Properties**) now contains a Version Control tab that describes the state of the item, when it was last updated, its Subversion revision, and whether it is current.
- New context menu for Version Control - The Project Browser right-click menu now contains a Version Control option. When selected, this displays a sub-menu with Subversion options to update, commit, revert, compare or view the log for the item. These options are recursive. For example, if you perform View Log on a

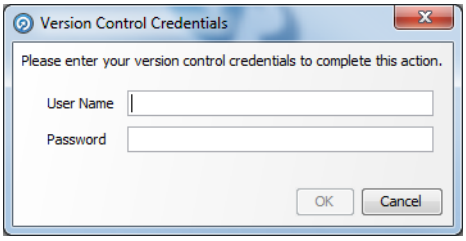
single process then you will see the log for this process only, but if you perform View Log on the Processes node you will see changes for all processes.

- Comment and credentials dialogs on commit - When you commit changes to the repository, Director displays the Commit log dialog:



In this dialog you can enter a comment describing the change in the Comment field. Alternatively, you can automatically populate the field by choosing a comment from the list of comments previously entered in the current session.

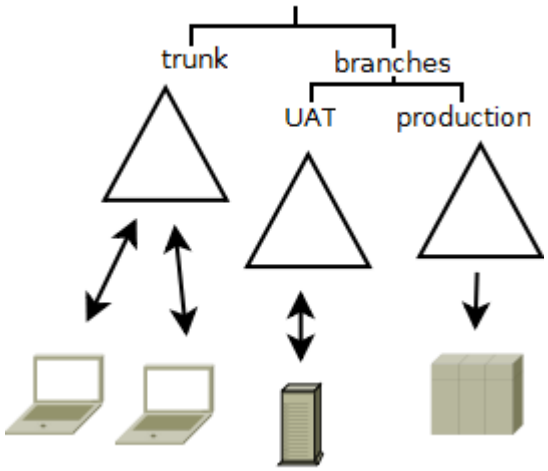
After you click **OK** in the Commit log dialog, Director displays the Version Control Credentials dialog if you have not already provided your credentials in the current session:



In this dialog you enter your user name and password for the Subversion repository and then click OK.

1.6 Reviewing a Deployment Example

An example deployment is presented here. In this illustration, there is a single Subversion server that holds three copies of the configuration for four EDQ installations:



The copies of the configuration are:

- **trunk** - the traditional location that all development work is performed on. New features of the configuration are developed and saved here.
- **branches and UAT** - this branch represents the copy of the configuration under UAT testing.
- **branches and production** - this branch represents the production copy of the configuration.

The four EDQ installations using the Subversion server for storing their configuration are:

- Two development laptops where design work and maintenance of existing projects are carried out.
- A UAT server for User Acceptance Testing changes.
- A production server for production runs.

In this example deployment, the laptop users develop configuration for individual projects on their own laptops and then commit changes back to the subversion repository on "trunk". Where the developers are co-operating on developing a project they will periodically update their local installation to pick up changes from the other developer.

At some point development reaches a point where it needs to be released to UAT for testing. A release manager then copies the necessary projects from "trunk" to "UAT" on the subversion server.

For example, the following Subversion command may be used:

```
svn cp -m"Release Project X to UAT" http://svn/repos/config/trunk/ProjectX
http://svn/repos/config/branches/UAT
```

The test manager then updates the UAT server's projects to load the new configuration into the EDQ server. Over a period of time testing continues. As issues are found they are fixed in the UAT environment and committed back to the subversion repository.

Once UAT environment has achieved an acceptable test level it is promoted to release. This achieved in much the same way as the release from development to UAT. The

necessary projects are copied across in the version control repository and then the production server is updated to use this configuration.

1.7 Troubleshooting Errors

You may encounter the following errors for which the cause and solution is provided.

Error	Cause and Solution
`Configuration database is not compatible with workspace`	The database has been used with a different workspace. This error usually arises occurs when operations have been performed in EDQ before Subversion version control is enabled.There are two solutions: drop and recreate the Director database or reinstall EDQ.
`Unable to open an ra_local session to URL`	This may occur when trying to commit files to an invalid repository. The EDQ integration is not compatible with file-based repositories (those repositories beginning with `file:///` or `C:\example`). A fully declared http:// path to the repository must be made.

2

Integrating with IBM Global Name Recognition

This chapter describes how to integrate EDQ with IBM Global Name Recognition (GNR).

You can configure EDQ to connect to IBM GNR to facilitate linguistic analysis of names, and linguistically sensitive name searching.

This chapter includes the following sections:

2.1 System Requirements

To enable EDQ connectivity with IBM GNR, you must have the following:

- EDQ 12*c* (12.1.3) installed on 64-bit AIX operating system or Linux operating system running 64-bit Java.
- IBM GNR 4.2.1 (4.2 + 4.2.1 fixpack) or later, including the hotfix based on GNR 4.2.2 (4.2 +4.2.2 fixpack). For more information, see the IBM website at `http://www.ibm.com`.

  EDQ does not make use of any of the web services provided by GNR, so you do not need to configure these during GNR installation.

> **Note:** GNR can only be installed on an EDQ instance if you have the required license agreements with both Oracle and IBM.

2.2 Configuring the EDQ Server

The `LD_LIBRARY_PATH` must be set as required for the installation environment.

The EDQ GNR analytic processors use a shared library (`.so`) in the `lib64` directory of the GNR installation. This directory must be specified in an environment variable passed to the EDQ server.

In a Linux 64-bit environment, the environment variable is `LD_LIBRARY_PATH`; for example:

```
LD_LIBRARY_PATH=$LD_LIBRARY_PATH:gnr-installation-dir/lib64
```

gnr-installation-dir is the GNR installation path; for example, `/opt/GNR/GNM`.

In an AIX environment, the environment variable name is `LIBPATH` instead of `LD_LIBRARY_PATH`.

> **Note:** The environment variable must be available to the application server process.

2.3 Building the Search Library

The GNR Search Processor uses a native library that must be linked with the GNR libraries.

Oracle supplies these files to create the library:

- Two `Makefile` templates, one for each platform, that script the building of the search library
- The `namehunter.o` object module file

When building the library on an AIX system, IBM C++ must be available. When building the library on Linux systems, the GCC C++ compiler must be available and it must be of the same version used to create the GNR libraries, as specified in the IBM GNR documentation.

The `Makefile` template for a 64-bit AIX operating system is as follows:

```
# Build library from object file
# ------------------------------

GNR=/opt/GNR/GNM
CFLAGS=-qmkshrobj
LIBS=-lNameHunter -lNameTransliterator -lsicui18n -lsicuuc -lsicudata
SDK=aix61_64-xlc9-release
LIBDIRS=-L$(GNR)/sdk/$(SDK)/lib -L$(GNR)/sdk/icu4c/$(SDK)/lib

all: libnimrod.so

libnimrod.so: namehunter.o
    xlc++_r -q64 $(CFLAGS) -o $@ $? $(LIBDIRS) -lNameHunter -lNameTransliterator
-lsicui18n -lsicuuc -lsicudata
```

The `Makefile` template for 64-bit Linux operating system is as follows:

```
# Build 64-bit library from object file
# -------------------------------------

GNR=/opt/GNR/GNM
CFLAGS=-shared -fPIC

all: libnimrod.so

libnimrod.so: namehunter.o
    g++ -m64 $(CFLAGS) -o $@ $? -L$(GNR)/sdk/rhel4_64-gcc34-release/lib
-L$(GNR)/sdk/icu4c/rhel4_64-gcc34-release/lib -lNameHunter -lNameTransliterator
-lsicui18n -lsicuuc -lsicudata
```

Before running the `Makefile` script for your platform, ensure that the value of `GNR` in the `Makefile` template is set correctly, according to the GNR installation directory

location. On the AIX operating system, also ensure that the value of `SDK` is set correctly, according to the system architecture.

After running the edited `Makefile`, the newly created `libnimrod.so` shared library file can be installed anywhere and can be copied to other systems with GNR installs.

2.4 Configuring the GNR Connector

The EDQ GNR connector requires three types of configuration files to integrate it with GNR:

- The `gnr.properties` properties file in the EDQ installation
- The `nameworks.config` configuration file in the GNR installation
- Search configuration files in the EDQ installation

2.4.1 Creating the EDQ GNR Properties File

The `gnr.properties` file and the `gnr` subdirectory that contains it must be manually created and placed in the `gnr` subdirectory of the EDQ configuration directory. It must contain the following properties:

`gnr.install`
The GNR installation path. This is the path to the directory containing the following GNR subdirectories:

- `bin`
- `bin64`
- `data` (which contains the GNR data files)
- `lib`
- `lib64`

`analytics.config`
The absolute location of the `nameworks.config` configuration file in the GNR installation.

`search.jnilib`
The absolute location of the `libnimrod.so` shared library, which was built using the `Makefile` template.

`nameworks.config`
During GNR installation, a `nameworks.config` file is created and stored in the GNR `data` directory.

The critical part of the `nameworks.config` file is the reference files section:

```
[Reference Files]
NameSifter=/opt/GNR/GNM/data/SifterRules.ibm
```

The `NameSifter` value must refer to the `SifterRules.ibm` file in the GNR installation.

2.5 Creating the Search Configuration Files

Search configuration files are located in the `gnr/search` subdirectory of the EDQ configuration directory. They are read by the connector and used to set parameters for the Search function.

A sample search configuration file named `search.config` is available in the `support/data/search` subdirectory of the EDQ installation. To create a search configuration file, copy this sample file to the `gnr/search` subdirectory of the EDQ configuration directory and edit the copy to suit your needs.

2.5.1 Support for GNR 3.2 and GNR 4.2 in Search Configuration Files

The search configuration format changed slightly from GNR 3.2 to GNR 4.2, and the EDQ GNR connector supports both versions as far as possible. It also processes data for Organization searches.

The basic differences between the search configuration files in GNR 3.2 and GNR 4.2 are:

- GNR 4.2 specifies the parameter files (for example, tags and variants) in the `[hunter]` section. GNR 3.2 uses the `[search]` section. The EDQ GNR connector looks in the `[hunter]` section first then the `[search]` section.
- The tag and variant files in GNR 4.2 are specified by keys such as `ibmTaqFile` and `custTaqFile`. In GNR 3.2, just `taqFile` is used. The EDQEDQ GNR connector looks for `taqFile`, `ibmTaqFile` and `custTaqFile` and loads each if found. The same rules are used for variant and terms files.
- The generic reg file is set by a specific `genericRegFile` setting; in GNR 3.2 this always defaults to the `anglo` reg file.
- Some settings have been added to the `[parms]` sections, and others have been removed.

3

Integrating with Experian QAS

This chapter describes how to integrate EDQ with Experian QAS API and migrate earlier versions of the QAS integration to EDQ versions 8.1.3 or later.

This chapter includes the following sections:

3.1 Software Requirements

EDQ includes a connector to the Experian QAS Batch API. You must have an installed version of the Experian QAS Batch API software appropriate for your platform. For more information, see the Experian Data Quality website found at `http://www.qas.com/`.

3.2 Integrating EDQ with Experian QAS

Integration of EDQ with Experian QAS Batch is carried out by editing the `qas.properties` file distributed with EDQ. The `qas.properties` file is located in the *ORACLE_HOME*`/user_projects\domains\domains\edq_domain\edq\oedq.home\qas` directory.

Once both EDQ and the QAS Batch software have been installed, edit the properties in the `qas.properties` as required for your integration. The properties specified in the file are as follows:

Property name	Description	Default Value
`qas.install.path`	The location of the Experian QAS Batch installation.	`C:\\Program Files (x86)\\QAS\\QuickAddress Batch API`
`qas.qaworld.ini`	The path to the `QAWorld.ini` file to use. This allows you to create and edit copies of the original `QAWorld.ini` file distributed with Experian QAS Batch API. If no value is specified for this property, it uses the `QAWorld.ini` file within the Experian QAS Batch API installation.	None

Property name	Description	Default Value
max.number.connections	The maximum number of connections that EDQ will create to the API. This should **not** be set to a value greater than 32. Note: There is a known threading and memory issue with Experian QAS Batch API versions 6.85, 6.89 and 6.95, where if maximum usage of 32 instances and 8 threads is reached, the Batch API may crash. This can be avoided by setting this property to a value between 18 and 22 inclusive.	32
connection.pool.timeout	This property specifies the number of milliseconds that a connection can be idle for before it will be closed by the pool management functionality. If this is set to -1, idle connections will not be closed.	60,000
connection.pool.timer.interval	This property specifies how often, in milliseconds, the connection pool will be scanned for idle connections.	60,000
default.layout	The default layout to use. If no layout with the specified name is available in the specified QAWorld.ini file, this property will be ignored.	GBR

3.3 Migrating QAS integrations

Some earlier versions of EDQ (versions prior to 8.1.3) were shipped with a customized version of QAWorld.ini that was used instead of the version contained within Experian QAS. It is not possible to migrate automatically from these versions of the Experian QAS integration to the later versions. To migrate an earlier Experian QAS integration, you must:

- Locate the local copy of QAWorld.ini and copy any custom layouts specified in this file into the version of QAWorld.ini contained within Experian QAS.
- Update the settings in qas.properties See Section 3.2, "Integrating EDQ with Experian QAS."

In addition, any existing processes and results books that make use of QAS processors must be updated as follows:

- Open each configured QAS processor and rename the output attributes to match those in the new QAWorld.ini file.
- Open any results books built on results grids from QAS processors and re-map the fields to the new output attribute names.

Refer to the following table for the affected releases for each version of EDQ. All versions of EDQ (previously known as dn:Director) prior to 7.2 are affected:

Version	Releases affected
EDQ (dn:Director) 7.2	Release 7.2.9 and all earlier 7.2 releases

Version	Releases affected
EDQ (dn:Director) 8.0	Release 8.0.21 and all earlier 8.0 releases
EDQ (dn:Director) 8.1	Release 8.1.2 and all earlier 8.1 releases

4

Integrating with Capscan Matchcode

This document describes how to integrate address verification and cleansing features from GBGroup Capscan Matchcode with EDQ. This documentation is intended for system administrators responsible for installing and maintaining EDQ applications.

This chapter includes the following sections:

4.1 Software Requirements

You must have the Capscan Matchcode software installed on a system that is accessible to the EDQ Server. For more information, see the GBGroup Matchcode website found at `http://www.gbgplc.com/products/matchcode/`.

4.2 Integrating the Capscan Matchcode Libraries into EDQ

EDQ includes a connector to the Capscan Matchcode API. EDQ includes a connector to the Capscan Matchcode API from GBGroup. This API provides address verification and cleansing features. This API provides address verification and cleansing features. Integrate the Capscan Matchcode Libraries into EDQ as follows:

1. Copy the `capscan.jar` client API file from the Capscan Matchcode installation to the *`ORACLE_HOME`*`/user_projects\domains\edq_domain\servers\edq_server1\tmp\_WL_user\edq\iz3lfy\war\WEB-INF\widgetjars` directory.

   The location of the `capscan.jar` file in a Capscan Matchcode installation depends on the installation platform; for example, it is located in the `Capscan\SDK\Matchcode client API\Java` directory on Windows.

   > **Note:** You can copy the `capscan.jar` file to a directory other than the default directory given in this step. If you do so, you must edit the `capscan.jar` property of the `capscan.properties` file to specify the location of the file. If you specify a relative path, the path must be relative to one of the directories in the EDQ configuration path.

2. Edit `capscan.properties` file in the *`ORACLE_HOME`*`/user_projects\domains\domains\edq_domain\edq\oedq.home\capscan` directory.

3. Edit the `server.host` property to refer to the system where Capscan Matchcode is running.
4. Restart your EDQ Server.

4.3 Customizing the Matchcode API

Various aspects of the Capscan Matchcode API behavior can be controlled using the `capscan.properties` file. It allows you to set the following properties:

`capscan.jar`
The location of the `capscan.jar` client API file in the EDQ installation. If you specify a relative path, the path must be relative to one of the directories in the EDQ configuration path. The default value is the relative path `capscan/capscan.jar`.

`server.host`
The IP address of the machine where Capscan Matchcode is running.

`connection.timeout`
A timeout period, in seconds, after which the Capscan Matchcode API will abort the search and return (a timeout period of zero indicates that there is no time limit on searches).

`connection.type`
The connection mode to use when communicating with the Capscan Matchcode API. The default connection mode is `CONNECTIONLESS`. The remaining options are:

- `CONNORIENTED`
- `STATELESS`
- `WEBCONNECTION`
- `ONDEMAND`

For information about these connection modes, refer to the Capscan Matchcode API documentation.

`number.capscan.connections`
The number of connections that EDQ should make to the Capscan Matchcode API.

`number.threads`
The number of threads that should be used when communicating with the Capscan Matchcode API.

The default contents of the `capscan.properties` file are as follows:

```
# This configuration file is configuring the CapScan processor
# to be able to communicate with the CapScan server
#
# Capscan server name
server.host = 127.0.0.1
#
# Connection timeout in seconds (0 means no time out)
connection.timeout = 30
#
# The connection type to make to the CapScan server.
# Possible values are:
#
# CONNORIENTED
# CONNECTIONLESS
```

```
# STATELESS
# WEBCONNECTION
# ONDEMAND
connection.type = CONNECTIONLESS
#
# The number of connections the director server should
# make to the CapScan server
number.capscan.connections = 1
#
# The number of threads that should be used to
# communicate with the CapScan server
number.threads = 1
```

5

Using the EDQ Command Line Interface

This chapter describes how to use the EDQ command line interface.

This chapter includes the following sections:

The EDQ command line interface, `jmxtools.jar`, provides access to a number of EDQ facilities.

5.1 Running the Command Line Interface

The EDQ command line interface is distributed as a self contained `.jar` file in the tools directory, and is executed by the following command line invocation:

`java -jar jmxtools.jar` *`commandname arguments`*

The commands and arguments are described in the following section.

5.2 Understanding the Commands and Arguments

The command line interface can run a number of commands and provides functionality including:

- Running jobs
- Listing and dropping orphaned results tables
- Showing user session logs
- Shutting down real-time jobs
- Checking the EDQ version number

The following sections provide a full guide to the commands, arguments and options available.

5.2.1 `runjob`

The `runjob` command runs a named job in the same way as if running the job using the Director UI. The `runjob` command takes the following arguments:

Argument	Use
-job *job_name*	Specifies the name of the job to run.
-project *project_name*	Specifies the name of the project that contains the job.
-u *user_name*	Specifies the user name to use to connect to the EDQ server. The user must have permission to run jobs and must have permission to the project containing the job.
-p *password*	Specifies the connecting user's password. If the -p option is not set, EDQ will prompt the user for the password.
-nolockwait	Indicates that if any of the resources used by the job are locked, the job should not wait for them to become available. Instead, it should terminate with a failure code and return control to the command line. The -nolockwait argument takes no extra values.
-nowait	Indicates that the command line should not wait for the job to complete. The -nowait argument takes no extra values.
server:port	Specifies the server and port of the JMX (management) interface.

5.2.2 runopsjob

The runopsjob command runs a named job in the same way as if running the job using the Server Console user interface. This provides additional functionality to the runjob command, specifically the use of Run Labels and Run Profiles. Run Labels may be used to store results separately from other runs of the same job. Run Profiles may be used to override externalized configuration settings at runtime.

The runopsjob command takes the following arguments:

Argument	Use
-job *job_name*	Specifies the name of the job to run.
-project *project_name*	Specifies the name of the project that contains the job
-u *user_name*	Specifies the user name to use to connect to the EDQ server. The user must have permission to run jobs and must have permission to the project containing the job.
-p *password*	Specifies the connecting user's password. If the -p option is not set, EDQ will prompt the user for the password.
-nolockwait	Indicates that if any of the resources used by the job are locked, the job should not wait for them to become available. Instead, it should terminate with a failure code and return control to the command line. The -nolockwait argument takes no extra values.
-nowait	Indicates that the command line should not wait for the job to complete. The -nowait argument takes no extra values.
-runlabel *run_label_name*	Specifies the name of the run label under which you wish to store staged output results. Note that this will override any run label that is specified in a run profile or by -D runlabel = *run_label_name*.
-props *run_profile_name*	Specifies the full path to a run profile properties file containing override settings for externalized configuration options in the job.

Argument	Use
`-D externalized_option=value`	Allows you to override specific externalized options for the job individually. The syntax for the externalized options and values is the same as used in run profile properties files. Note that characters will be interpreted by the command line, so some characters will need to be escaped according to the shell conventions of your environment. Also note that any individually specified externalized option settings will override any settings for the same option if these are specified in a run profile used in the same run.
server:port	Specifies the server and port of the JMX (management) interface.

5.2.3 droporphans

The `droporphans` command is used to remove any orphaned results tables that may be created when processes are terminated unexpectedly. It should not be run when any jobs or processes are running on the EDQ server.

The `droporphans` command takes the following arguments:

Option	Use
`-u user name`	Specifies the user name to use to connect to the EDQ server. The user must have permission to cancel jobs and must have permission to the project containing the job.
`-p password`	Specifies the connecting user's password. If the `-p` option is not set, EDQ will prompt the user for the password.
server:port	Specifies the server and port of the JMX (management) interface.

5.2.4 listorphans

The `listorphans` command is used to identify any orphaned results tables. The `listorphans` command takes the same arguments as the `droporphans` command.

5.2.5 scriptorphans

The `scriptorphans` command creates a list of SQL commands for dropping orphaned results tables. This is useful if you want to review exactly which commands will run on the Results database when you drop tables, or if you want to drop the tables yourself manually.

5.2.6 list

The `list` command lists all the available commands.

5.2.7 showlogs

The `showlogs` command starts a small graphical user interface application that allows user session logs to be retrieved.

5.2.8 shutdown

The `shutdown` command shuts down all real-time jobs. These are jobs that are running from real-time record providers (web services or Java Message Service).

The `shutdown` command takes the following arguments:

Option	Use
-u *user name*	Specifies the user name to use to connect to the EDQ server. The user must have permission to cancel jobs and must have permission to the project containing the job.
-p *password*	Specifies the connecting user's password. If the -p option is not set, EDQ will prompt the user for the password.
-nowait	Indicates that the command line should not wait for the job to complete. The -nowait argument takes no extra values.
server:port	Specifies the server and port of the JMX (management) interface.

5.2.9 version

The version command is used to identify the version of the currently installed instance of EDQ.

Enter the following at the command line:

```
java -jar jmxtools.jar version
```

The version number is returned.

5.3 Reviewing Examples

This section lists several possible invocations of the command line interface:

5.3.1 Listing All the Available Commands

The following invocation of the command line interface lists all of the available commands:

```
java -jar jmxtools.jar -list
```

The output is as follows:

```
Available launch names:
<Job tools>
runjob Run named job
shutdown Shutdown realtime jobs
runopsjob Run named job in operations mode

<Logging>
showlogs Show session logs

<Database Tools>
listorphans List orphaned results tables
droporphans Drop orphaned results tables
scriptorphans Create script for dropping orphaned results tables

<System Information>
version Display version number of tools
```

5.3.2 Listing the Available Parameters for a Command

If the command line interface is invoked by specifying a command without the corresponding parameters, it outputs detailed help for the command. For example, to get detailed help on the runjob command, invoke the command line interface as follows:

java -jar jmxtools.jar runjob

The output is as follows:

```
Usage: runjob -job jobname -project project [-u user] [-p pw] [-nowait]
[-nolockwait] [-sslprops props | -ssltrust store] server:port
```

5.3.3 Running a Named Job

This example illustrates how to run a named job in a named project on a specific EDQ instance (as specified by machine name and port).

To run a job called “rulecheck” in a project called “Audit” on the local machine with a JMX server on port 8090 using a user named "dnadmin" , the command is as follows:

```
java -jar jmxtools.jar runjob -job rulecheck -project audit -u dnadmin
localhost:8090
```

The application prompts the user to enter the password for the `dnadmin` user.

5.3.4 Running a Named Job in Operations Mode

This example illustrates how to run a named job in 'operations mode' in a Windows environment. In operations mode, there is access to the Run Label and Run Profile capabilities so that the configuration of the job can be specified dynamically, and so that the results of the job can be stored by Run Label.

To run a job called "profiling" in a project called "MDM" on a server called "prod01", with a run label of "Nov2011" and a run profile file called `File1.properties`, with a JMX server on port 8090, the command is as follows:

```
java -jar jmxtools.jar runopsjob -job profiling -project MDM -runlabel
Nov2011 -props c:\ProgramData\Oracle\"Enterprise Data Quality\oedq_local_
home\File1.properties" -u dnadmin prod01:8090
```

6

Configuring Additional Database Connections

This chapter describes how you can configure additional database connections for use in EDQ Director.

- Using JNDI to Connect to Data Stores
- Using TNS to Connect to Data Stores
- Using LDAP to Connect to Data Stores

The standard options for Director to connect to data stores are described in the online help. Once implemented, these options appear in the Data Store Configuration step of the New Data Store wizard in Director. For help with using this wizard, see the Director online help.

6.1 Using JNDI to Connect to Data Stores

You can configure EDQ to use a Java Naming and Directory Interface (JNDI) data store connection.

1. Define the JNDI data store. JNDI is provided by the hosting application server. For more information about defining JNDI data sources in Oracle WebLogic Server, see "Using DataSource Resource Definitions" in *Oracle Fusion Middleware Developing JDBC Applications for Oracle WebLogic Server*
2. In the EDQ data store wizard, specify JNDI as the type of data store, and then specify the JNDI name.

6.2 Using TNS to Connect to Data Stores

You can configure EDQ to use an Oracle Transparent Network Substrate (TNS) data store connection. To use this connection method, you specify a name from a `tnsnames.ora` file as the data source when using the data sources wizard. Only the `tnsnames.ora` file is needed. No other Oracle client software is needed.

To configure EDQ to connect through TNS

1. Set the `oracle.net.tns_admin` Java system property to a local directory that contains the `tnsnames.ora` file.
2. Create a file named `jvm.properties` in your EDQ local configuration directory (`oedq_local_home` by default) and add an entry similar to the following: `-d oracle.net.tns_admin = c:\temp`). This property may have been set already in the application server when EDQ was installed.

For more information about the `tnsnames.ora` file, see "Configuring the Local Naming Method" in *Oracle Database Net Services Administrator's Guide*.

6.3 Using LDAP to Connect to Data Stores

You can configure EDQ to use an Oracle Lightweight Direct Access Protocol (LDAP) data store connection by setting the required Java system properties. These properties are:

`dn.oracle.directory.servers = ldap://`*`servername:port`*

`dn.oracle.default.admin.context = dc=`*`domaincontext1`*`,dc=`*`domaincontext2`*

The first property gives the location of your LDAP servers. The second property sets the context within the LDAP tree. Together, these properties enable EDQ to construct an Oracle and LDAP JDBC connection string, which looks similar to:

`jdbc:oracle:thin:@ldap://`*`servername:port`*`/unicode,cn=`*`Oraclecontext`*`,dc=`*`domaincontext1`*`,dc=`*`domaincontext2`*

7

Configuring EDQ to Process XML Data Files

This chapter describes how EDQ can be configured to read and write XML data files.

This chapter includes the following sections:

- Using Simple XML Data Stores
- Using XML and Stylesheet Data Stores

You can use XML data files in snapshots to read and write the data contained in the file. A snapshot is a staged copy of data in a data store that is used in one or more processes. EDQ provides two types of data stores for working with XML data files: Simple XML and XML and Stylesheet. Both are available for server-side and client-side data stores.

7.1 Using Simple XML Data Stores

Simple XML data stores can read and write XML files that have a simple 2-level structure in which the top level tag represents the entity and the lower level tags represent the attributes of the entity. XML files exported from Microsoft Access are an example.

Following is an example of a simple XML file format that could be used with EDQ:

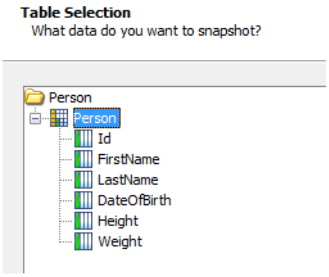
```
<dataroot>
  <Person>
    <Id>1</Id>
    <FirstName>Fred</FirstName>
    <LastName>Bloggs</LastName>
    <DateOfBirth>1972-01-31T00:00:00.000+0000</DateOfBirth>
    <Weight>85</Weight>
  </Person>
  <Person>
    <Id>2</Id>
    <FirstName>Jane</FirstName>
    <LastName>Smith</LastName>
    <DateOfBirth>1985-07-16T00:00:00.000+0100</DateOfBirth>
    <Weight>63</Weight>
  </Person>
</dataroot>
```

7.1.1 Reading Simple XML Files

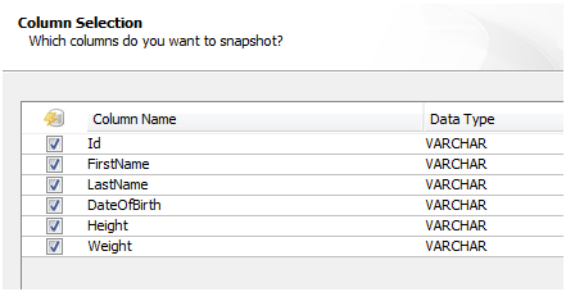
When EDQ reads Simple XML files the following occurs:

- The root element name is not used, so it can be anything.

- The record element name appears as the table name in the Table Selection page of the Snapshot Wizard dialog.



- The lower level element names appear as the column names in the Column Selection page of the Snapshot Wizard and therefore become EDQ attribute names.



7.1.2 Writing Simple XML Files

When generating Simple XML files using an EDQ export to the data store, the name of the data store defines the record XML element name. The element `Person` in the example in Section 7.1 shows how this appears in the XML.

The XML element names of the lower level tags are taken from the EDQ attribute names. EDQ names are encoded to ensure that invalid XML is not generated. For example, space characters in names are replaced by the character sequence `_x0020_`, so an EDQ attribute named `Date Of Birth` would generate XML elements in the following format:

```
<Date_x0020_Of_x0020_Birth>
```

7.2 Using XML and Stylesheet Data Stores

When there is a requirement to work with XML of a different structure than that of Simple XML, then you use the XML and Stylesheet data stores.

These data stores read and write XML conforming to the DN-XML schema and optionally allow the use of a custom stylesheet to:

- Transform XML from a custom XML format to DN-XML during data snapshot
- Transform XML from DN-XML to a custom XML format during data export

For more information about XML stylesheets, see the W3C website found at `http://www.w3.org/Style/XSL/` and `http://www.w3.org/standards/xml`.

7.2.1 Using DN-XML

DN-XML is the format by which custom XML can be processed by EDQ.

An example of DN-XML is as follows:

```
<dn:data xmlns:dn="http://www.datanomic.com/2008/dnx">
  <dn:record>
    <dn:value name="Id" type="string"/>
    <dn:value name="FirstName" type="string"/>
    <dn:value name="LastName" type="string"/>
    <dn:value name="DateOfBirth" type="date"/>
    <dn:value name="Height" type="number"/>
    <dn:value name="Weight" type="number"/>
  </dn:record>
  <dn:record>
    <dn:value name="Id">1</dn:value>
    <dn:value name="FirstName">Fred</dn:value>
    <dn:value name="LastName">Bloggs</dn:value>
    <dn:value name="DateOfBirth">1972-01-31</dn:value>
    <dn:value name="Height">1.85</dn:value>
    <dn:value name="Weight">85</dn:value>
  </dn:record>
  <dn:record>
    <dn:value name="Id">2</dn:value>
    <dn:value name="FirstName">Jane</dn:value>
    <dn:value name="LastName">Smith</dn:value>
    <dn:value name="DateOfBirth">1985-07-16</dn:value>
    <dn:value name="Height">1.65</dn:value>
    <dn:value name="Weight">63</dn:value>
  </dn:record>
</dn:data>
```

This is the equivalent DN-XML for the example given in Section 7.1, "Using Simple XML Data Stores."

Note that the EDQ attribute names are defined differently in DN-XML compared with Simple XML. Because DN-XML uses attribute content to specify EDQ attribute names, it is possible to create EDQ attributes with spaces and other special characters in their names.

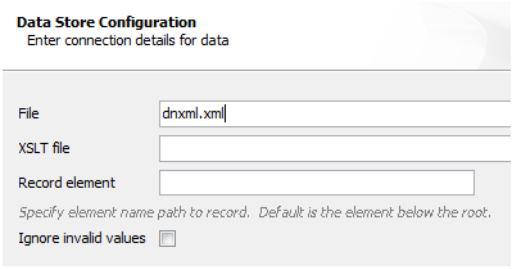
In the previous example, the `<dn:record skip="true">` XML element and its contents allows the definition of the structure of the source including the field names and their data types. All other record elements define a row of data in EDQ. This is analogous to the header row in a comma-separated values file. The following data types are permitted:

- string
- date
- number

> **Note:** Date values in DN-XML files should be specified in the XSD date format (ISO 8601). For example, '2008-10-31T15:07:38.6875000-05:00' or without the time component simply as '2008-10-31'.

Within a data record, value elements are used to specify EDQ attribute values for the record. The name attribute is used to specify the EDQ attribute in question and the text content of the attribute specifies the value for that EDQ attribute. For example, the XML fragment, `<dn:value name="FirstName">Fred</dn:value>`, assigns the value 'Fred' to the EDQ attribute 'FirstName'.

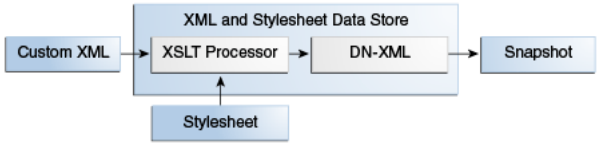
DN-XML files can be read in to EDQ by creating an XML and Stylesheet data store and specifying the location of the XML source file; the XSLT file option should be left blank:



Similarly, EDQ can write DN-XML files by exporting data to an XML and Stylesheet data store with the XSLT option left blank.

7.2.2 Reading Custom XML Files

XML files in custom formats can be read by EDQ using the XML and Stylesheet data store configured to use a custom XML stylesheet (XSLT) to transform from the custom schema to the DN-XML schema during data snapshotting.



Following is an example custom XML file that could be read into EDQ:

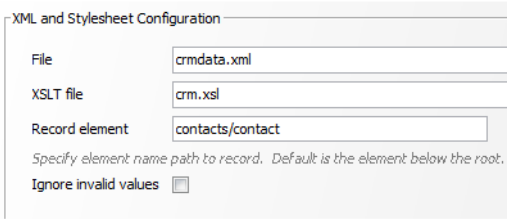
```
<crmdata>
  <contacts>
    <contact id="1">
      <name>
        <firstname>Fred</firstname>
        <surname>Bloggs</surname>
      </name>
      <dob>1972-01-31</dob>
      <properties>
        <property name="height" value="1.85"/>
        <property name="weight" value="85"/>
      </properties>
    </contact>
    <contact id="2">
      <name>
        <firstname>Jane</firstname>
        <surname>Smith</surname>
      </name>
      <dob>1985-07-16</dob>
      <properties>
        <property name="height" value="1.68"/>
        <property name="weight" value="63"/>
      </properties>
    </contact>
  <contacts>
</crmdata>
```

The following XML stylesheet demonstrates one way that the preceding example custom XML can be transformed into a suitable DN-XML format:

```
<xsl:stylesheet version="1.0" xmlns:dn="http://www.datanomic.com/2008/dnx"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:fn="http://www.w3.org/2005/02/xpath-functions">

  <xsl:output method="xml"/>

    <xsl:template match="/">
        <dn:data>

          <!-- Write out the header record -->
          <dn:record skip="true">
              <dn:value name="Id" type="string"/>
              <dn:value name="FirstName" type="string"/>
              <dn:value name="LastName" type="string"/>
              <dn:value name="DateOfBirth" type="date"/>
              <dn:value name="Height" type="number"/>
              <dn:value name="Weight" type="number"/>
          </dn:record>

         <!-- Get each contact record -->
         <xsl:apply-templates select="/crmdata/contacts/contact"/>

      </dn:data>
    </xsl:template>

    <xsl:template match="contact">

      <!-- Write out a data record -->
      <dn:record>
        <dn:value name="Id"><xsl:value-of select="@id"/></dn:value>
        <dn:value name="FirstName"><xsl:value-of select="name/firstname"/></dn:value>
        <dn:value name="LastName"><xsl:value-of select="name/surname"/></dn:value>
        <dn:value name="DateOfBirth"><xsl:value-of select="dob"/></dn:value>
        <dn:value name="Height">
         <xsl:value-of select="properties/property[@name='height']/@value"/>
        </dn:value>
        <dn:value name="Weight">
         <xsl:value-of select="properties/property[@name='weight']/@value"/>
        </dn:value>
     </dn:record>

    </xsl:template>

  </xsl:stylesheet>
```

7.2.2.1 Configuring the Data Store

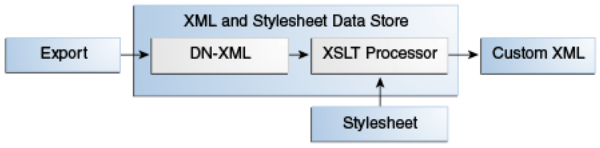
The data can be read in to EDQ by creating an XML and Stylesheet data store and specifying the location of the XML source file and the XSLT file (stylesheet).



EDQ reads the source XML file in chunks for efficiency breaking up the file on record boundaries. By default EDQ uses the element immediately below the root as the record element. If this is not the case in the source XML file then an XPath-style expression to the record element from the root must be specified.

7.2.3 Writing Custom XML Files

XML files in custom formats can be written by EDQ using the XML and Stylesheet data store configured to use a custom XSLT to transform from the DN-XML schema to the custom target schema the during data export.



Following is an example target custom XML format that needs to be generated by EDQ:

```
<Report>
  <Person Id="1" FullName="Fred Bloggs"/>
  <Person Id="2" FullName="Jane Smith"/>
</Report>
```

The following XML stylesheet demonstrates one way in which the DN-XML format can be transformed into the target custom XML format:

```
<xsl:stylesheet version="1.0"
  xmlns:dn="http://www.datanomic.com/2008/dnx"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:fn="http://www.w3.org/2005/02/xpath-functions">

  <xsl:output method="xml"/>

  <xsl:template match="/">
    <Report>
      <xsl:apply-templates select="/dn:data/dn:record"/>
    </Report>
  </xsl:template>

  <xsl:template match="dn:record">
    <Person>
      <xsl:attribute name="Id">
        <xsl:value-of select="dn:value[@name = 'Id']"/>
      </xsl:attribute>
      <xsl:attribute name="FullName">
        <xsl:value-of select="dn:value[@name = 'FirstName']"/>
        <xsl:text> </xsl:text>
        <xsl:value-of select="dn:value[@name = 'LastName']"/>
```

```
      </xsl:attribute>
    </Person>
  </xsl:template>

</xsl:stylesheet>
```

7.2.3.1 Configuring the Data Store

The data can be written by EDQ by creating an XML and Stylesheet data store and specifying the destination for the custom XML file and XSLT (stylesheet) file.