Enhydra Octopus Application

Enhydra Octopus Application

by

Enhydra Octopus Application: Enhydra Octopus Application

by

Table of Contents

1. Introduction .....
    Introduction 1
    Purpose 1
    Enhydra Octopus architecture 1
    License 3
2. Installation Guide .....
    System Requirements 4
    Installing Octopus 4
    Building the source 5
        Pre-Build Configuration 5
        Configure options 5
    Building the Enhydra Octopus application 7
        Make Options 7
        Run Octopus tests 8
3. Using Enhydra Octopus .....
    Introduction 9
    Starting OctopusGenerator via GUI 9
    Starting OctopusGenerator as an application 12
        Start OctopusGenerator (start up scripts) 12
        Start OctopusGenerator as library, from Java Application: 15
        Start OctopusGenerator, as an Ant task 17
    Starting OctopusLoader via GUI 20
    Starting OctopusLoader as an application 22
        Starting OctopusLoader as an application, with one parameter 22
        Starting OctopusLoader as an application with more parameters 22
        Starting OctopusLoader as library 24
        Starting OctopusLoader as an Ant task 25
        Using OctopusLoader as a test case 26
    Using Octopus project files 30
4. OctopusGenerator .....
    Introduction 31
    JDBC database as source data 31
        XML files 31
        SQL files 32
        DOML file 33
    DOML file as source data 33
5. OctopusLoader .....
    Introduction 35
    Validation 35
    Logging 35
        Standard logger 36
        Log4j logger 37
        User's logger 37
    Data Cleaning 38
        'If some table fields have value null' 38
        'Cutting off data' 38
        'Cleaning foreign key values' 39
        Log Table 39
    The LoaderJob.olj file 41
    The <loaderJob> root tag 41
    The <restartCounter> tag 43
    The <variables> tag 44
    The <variable> tag 45

The <jdbcDefaultParameters> tag 46
The <jdbcSourceParameters> tag 46
The <jdbcTargetParameters> tag 47
The <sql> tag 48
The <jdbcTargetParameters> tag 49
The <sqlStmt> tag 50
The <definitionInclude> tag 50
The <include> tag 50
The <echo> tag 51
The <copyTable> tag 51
The <importDefinition> tag 52
The <sortColumns> tag 55
The <jdbcParameters> tag 56
The <valueColumns> tag 57
The <transformations> tag 58
The <variableColumns> tag 59
The <userIDColumn> tag 60
The <timeStampColumn> tag 61
The <relationColumns> tag 61
The <constantColumns> tag 62
The <counterColumns> tag 63
The <tables> tag 65
Working with TOS (Together Object Server) format tables 66
6. Transform data with Octopus .....
Transformations 67
Predefined classes for transformations 71
ReplaceData 71
ConcatenateData 72
CurrencyConverter 73
DateFormat 74
Transformations using JavaScript 75
7. ‘Backup’ and ‘Restore’ of database .....
Introduction 78
Database ‘backup’ 78
Databases ‘restore’ 80
8. Configuration files .....
Introduction 81
OctopusDBVendors.xml file 81
Database Conf.xml file 82
9. JDBC Drivers .....
Introduction 87
Using FreeTDS overview 87
Connecting to a Database 87
JDBC 2.0 87
Using Microsoft JDBC Driver overview 88
Connecting to a Database 88
Connection String Properties 88
Using JTurbo driver overview 90
Connecting to a Database 90
Using JDBC-LDAP Bridge overview 90
Connecting to a Database 90
Using C - JDBC driver overview 91
Connecting to a Database 91
Using the csv driver overview 92
Using the xml driver overview 92
10. Java Web Start .....
Introduction 93
Setting up the Web site 93

Configure the Web server to use the Java Web Start MIME type 93
Create a JNLP file for the application 93
The JNLP Element 94
The Information Element 94
The Security Element 94
The Resources Element 94
Signing JAR Files with a Test Certificate 95
The Application-Desc Element 95
Make the application accessible on the Web server 95
Create a link from the Web page to the JNLP file 95
11. Examples .....
Introduction 96
Octopus Generator Examples 96
Octopus Loader Examples 96
How to create input XML File? 96
Examples 104
'Backup' and 'Restore' 105
Example of LDAP Server 105
Example of Using C-JDBC driver 106
12. Tools in Octopus .....
P6spy 109

List of Tables

3.1. Options for Starting OctopusLoader 22
3.2. Table of LoaderTask arguments 25
3.3. Table LoaderTask subelements 26
5.1. Table of logger attributes 36
5.2. Table of <loaderJob> tag attributes 41
5.3. Table of <restartCounter> tag attributes 44
5.4. Table of <variables> tag attributes 44
5.5. Table of <variable> tag attributes 45
5.6. Table of <jdbcSourceParameter> tag attributes (default parameters) 46
5.7. Table of <jdbcTargetParameter> tag attributes (default parameters) 47
5.8. Table of <sql> tag attJributes 48
5.9. Table of <jdbcTargetParameter> tag attributes (<sql> tag) 49
5.10. Table of <include> tag attributes (<sqlStmt> tag) 50
5.11. Table of <include> tag attributes (<definitionInclude> tag) 50
5.12. Table of <copyTable> tag attributes 51
5.13. Table of <importDefinition> tag attributes 52
5.14. Table of <sortColumn> tag attributes 55
5.15. Table of <jdbcSourceParameter> tag attributes (<jdbcParameters> tag) 56
5.16. Table of <jdbcTargetParameter> tag attributes (<jdbcParameters> tag) 57
5.17. Table of <valueColumn> tag attributes 58
5.18. Table of <transformations> tag attributes 59
5.19. Table of <targetColumn> tag attributes 59
5.20. Table of <variableColumn> tag attributes 59
5.21. Table of <userIDColumn> tag attributes 60
5.22. Table of <timeStampColumn> tag attributes 61
5.23. Table of <relationColumn> tag attributes 62
5.24. Table of <constantColumn> tag attributes 63
5.25. Table of <counterColumns> tag attributes 63
5.26. Table of <counterColumn> tag attributes 64
5.27. Table of <subCounterColumn> tag attributes 64
5.28. Table of <table> tag attributes 65
7.1. Table of database backup attributes 78
9.1. Connection URL properties for FreeTDS driver 87
9.2. Connection property table 88
9.3. SQL Server Connection String Properties 88
9.4. Connection property table 90

List of Examples

11.1. Example of simple relations tag 100
11.2. Example of complex relations tag 100

Chapter 1. Introduction

Introduction

Have you ever wanted to transfer data from one JDBC source to another and do some kind of transformation during transfer, like: normalize non-normalized data; create artificial keys; execute SQL statements during, before or after transfer? Have you ever wanted to create a database in the following steps: create tables, load initial data, create indexes, create primary keys, and create foreign keys? If your answer is yes, you need this application: Enhydra Octopus!

Purpose

Enhydra Octopus loads data from a JDBC data source (database) into JDBC data target (database) and it perform many transformations defined in an XML file.

This application can process several different types of databases.

Possible JDBC source/target database types are: MSSQL, MySql, Access, Excel, Csv, PostgreSQL, Qed, InstantDB, XML, BorlandJDataStore, Oracle, HSQL, McKoi, DB2, Sybase and Paradox database.

Also, Enhydra Octopus application has several loading possibilities: creating databases, creating tables, inserting data into an empty database, inserting data in to a non empty database, updating constant columns, updating relation columns, updating columns with system time, updating columns with a user ID, executing every possible SQL statement, creating artificial keys using the Enhydra DODS objected logic, ...

Enhydra Octopus architecture

Enhydra Octopus application is consisted on two parts:

- OctopusGenerator and,
- OctopusLoader.

Figure 1. gives an overview of the Enhydra Octopus architecture.

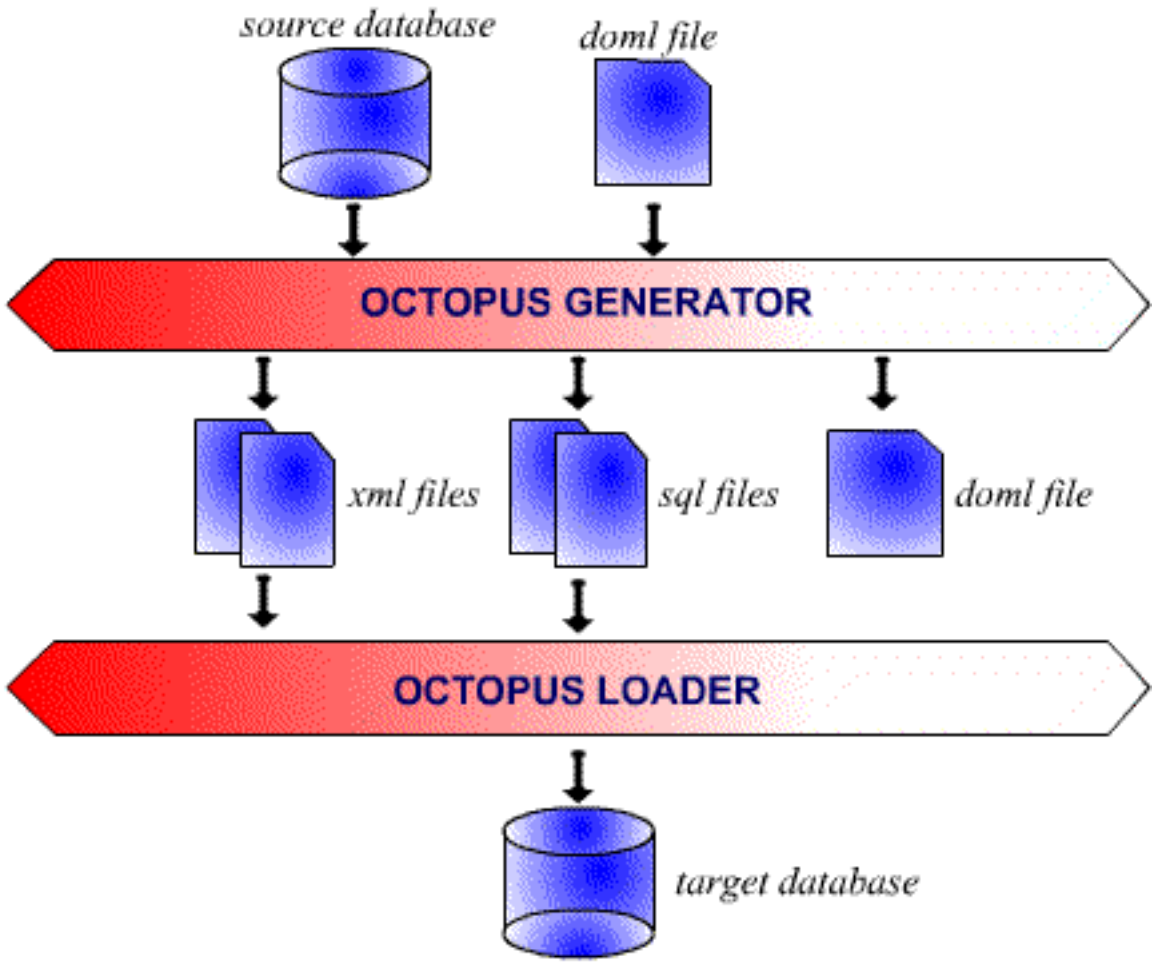

Figure 1. Enhydra Octopus architecture

The main purpose of OctopusGenerator is to create SQL and XML files from source database or source doml file.

SQL files include SQL statements for creating database, tables, primary keys, indexes, and foreign keys. User can choose which of those SQL files will be created.

XML files describe relations between data and between tables (loadJob.xml and importDefinition.xml). In those XML files, transformation rules are written. The number of transformations in one XML loadJob file is not limited.

OctopusLoader is a Java-based Extraction, Transformation, and Loading (ETL) tool for transferring data from JDBC source database to JDBC target database. It may connect to any JDBC data source and perform transformations defined in an XML file.

In order to do that, OctopusLoader needs XML and SQL files created by OctopusGenerator.

OctopusGenerator and OctopusLoader share the same GUI (graphic user interface).

Enhydra Octopus application can also be used for ‘backup’ and ‘restore’ of databases.

Enhydra Octopus is distributed with the following drivers:

- csvjdbc (driver for coma separated files),
- xmljdbc (driver for xml files), and
- freeTds driver (driver for MSSQL server).

License

Chapter 2. Installation Guide

This section describes how to install the Enhydra Octopus.

System Requirements

The application requires the following items:

- Java 2 SDK, Standard Edition (J2SETM SDK) version 1.4.x or greater: java.sun.com [http://java.sun.com/], or
- Java Runtime Environment (JRE) version 1.4.x or greater: java.sun.com [http://java.sun.com/]

Note

There are problems with Java 1.4.0 and GUI - window should be maximized/minimized few times before all fields are properly shown.

Installing Octopus

Octopus can be installed on several different ways:

- **1. Running the setup script**
  - For Windows users:

    The easiest way to install Octopus application is to run the installation file tdt-{version}setup.exe in command prompt, or just double-click on it.
  - For Linux users:

    Unix users must install rpm file tdt-{version}.noarch.rpm. The command for installing this file is:

    ```
    -rpm -i tdt-{version}.noarch.rpm
    ```

- **2. Using the output archive**

  Unzip the distribution bundle into directory of your choice.
  - For Windows users:

    Windows users have to use an utility such as WinZip or WinRar to extract the files from the archive tdt-{version}.zip
  - For Linux users:

    Unix users can proceed as follows:

    ```
    tar xfz /path_to_tdt/tdt-{version}.tar.gz
    ```

- **3. Building Octopus from source distribution**

  Unzip the distribution of source bundle into directory of your choice.

- For Windows users:

  Windows users have to use an utility such as WinZip or WinRar to extract the files from the archive tdt-{version}.src.zip

- For Linux users:

  Unix users can proceed as follows:

  ```
  tar xfz /path_to_tdt/tdt-{version}.src.tar.gz
  ```

  or you can install tdt-{version}.src.rpm file.

After you uncompress the binary files, you must build the source, before you are able to start Octopus application.

Building the source

Pre-Build Configuration

Before building Octopus, you must edit the file "build.properties" in the Octopus/Octopus-src directory. This can be done with configure executable. Properties maven.path and jdk.dir must be set to appropriate values, jdk.dir must point to directory where current JAVA is installed, and maven.path must point to directory where MAVEN is placed. Maven is part of Octopus and is placed in Octopus/maven directory.

Configure options

If you want to edit build.properties file you must run configure command with associated parameters:

```
configure [-release release_number] [-jdkhome jdk_home_dir][-mavenhome maven_home_dir] [-prefix inst
```

Setting JAVA_HOME and MAVEN_HOME variables

JAVA_HOME variable represents the path to directory where the JAVA is installed, and MAVEN_HOME variable represents the path to directory where the MAVEN is placed (Octopus\maven directory). MAVEN source is distributed with Enhydra Octopus source.

- For Windows users:

  Open a command prompt in Octopus-src directory, and type the following command:

  ```
  configure
  ```

  This command will automatically set appropriate values for jdk.dir and for maven.path. The jdk.dir will be pointing to the root directory of your current J2SE SDK installation, and maven.path will be pointing to maven home directory. Default values for other properties will be set automatically also.

- For Unix users:

  Open a shell prompt in Octopus-src directory and type those commands

  ```
  ./configure –jdkhome JAVA_HOME
  ```

  ```
  for e.g ./configure -jdkhome /user/java/j2sdk1.4.1
  ```

After that, jdk.dir variable in build.properties file will be set to appropriate value. For Unix users, maven.path vari-

able does not have to be set.

Setting RELEASE variable

This parameter is used for marking the release of Enhydra Octopus application.

- For Windows users:

  Open a command prompt in Octopus-src directory and type the following command:

  ```
  configure -release release_value
  for e.g configure –release beta1
  ```

- For Unix users:

  Open a shell prompt in Octopus-src directory and type the following command:

  ```
  ./configure –release release_value
  for e.g  ./configure -release beta1
  ```

After that, the release variable in build.properties file will be set.

Setting PREFIX variable

This parameter represents the path to your installation directory. If you want to set this parameter, you must:

- For Windows users:

  Open a command prompt in Octopus-src directory and type the following command:

  ```
  configure –prefix path_to_install_dir
  for e.g configure -prefix C:\Users\InstallForWindows
  ```

- For Unix users:

  Open a shell prompt in Octopus-src directory and type the following command:

  ```
  ./configure –prefix path_to_install_dir
  for e.g ./configure -prefix /usr/home/InstallForLinux
  ```

After that, the prefix variable will be set.

Note

In examples described above we have set configure variables one by one, but they can be set altogether.

HELP

If you want to display option screen:

- For Windows users:

  Open a command prompt in Octopus-src directory and type the following command:

```
configure -help
```

- For Unix users:

  Open a shell prompt in Octopus-src directory and type the following command:

  ```
  ./configure -help
  ```

  After you have finished with configuring, your build.properties file should look something like this (the example below is for Windows users):

  ```
  jdk.dir=c:\j2sdk1.4.1_01
  maven.path=c:\CVS\Octopus\maven
  prefix=c:\CVS\Octopus\OctopusInstall
  release=final
  ```

  Please note that slashes (/) must be used if you are using Unix. If you are using Windows backslashes (\ ) must be used.

  Next step, after you have configured (edited) the build.properties file, is to build the Enhydra Octopus application.

Building the Enhydra Octopus application

When you uncompress the binary distribution, Octopus directory will be created. In the rest of this documentation this directory will be referred to as OCTOPUS_HOME.

Once you have edited the "build.properties" file discussed above, open the appropriate prompt (command or shell) in the OCTOPUS_HOME/Octopus-src directory and type the following command (this command is the same for Windows and Unix users) :

```
make
```

This command will produce a great deal of output, and it is recommended that you capture this output to a log file for latter reference. The binary output of the build can be found in the following directory OCTOPUS_HOME\Octopus-src\output\Octopus-{version}:

Make Options

Enhydra Octopus compilation and configuration is completely Ant based now. You can give one of the following options to the make command:

make - builds and configures Enhydra Octopus with javadoc and docbook documentation

make buildAll - builds and configures Enhydra Octopus with javadoc and docbook documentation

make install - copies and configures binary files of Enhydra Octopus in 'prefix' directory (parameter in build.properties file)

make buildNoDoc - builds and configures Enhydra Octopus without documentation (it is much faster, and used for testing)

make distributions - builds and configures Enhydra Octopus with javadoc and docbook documentation and creates distribution

make test - run Octopus tests

make clean - clean Octopus output (in order to start a new compilation from scratch)

make buildCsvJdbc - build CsvJdbc subproject. This target can be call only after call to buildNoDoc, buildAll or distributions, because this target use directory tree and jars maked when call these targets.

make buildXmlJdbc - build XmlJdbc subproject. This target can be call only after call to buildNoDoc, buildAll or distributions, because this target use directory tree and jars maked when call these targets.

make buildOctopus - build OctopusLoader subproject. This target can be call only after call to buildNoDoc, buildAll or distributions, because this target use directory tree and jars maked when call these targets.

make buildGenerator - build OctopusGenerator subproject. This target can be call only after call to buildNoDoc, buildAll or distributions, because this target use directory tree and jars maked when call these targets.

make buildXmlutil - build Xmlutil subproject. This target can be call only after call to buildNoDoc, buildAll or distributions, because this target use directory tree and jars maked when call these targets.

make buildFreetds - build Freetds subproject. This target can be call only after call to buildNoDoc, buildAll or distributions, because this target use directory tree and jars maked when call these targets.

make buildOctopusTest - build OctopusTest subproject. This target can be call only after call to buildNoDoc, buildAll or distributions, because this target use directory tree and jars maked when call these targets.

make buildOctopusTask - build OctopusTask subproject. This target can be call only after call to buildNoDoc, buildAll or distributions, because this target use directory tree and jars maked when call these targets.

Run Octopus tests

Enhydra Octopus can be tested with three tests which are included in Enhydra Octopus application. These tests, which are JUnit tests, can be run with 'make test' command. Output of these tests can be found in file OCTOPUS_HOME/Octopus-src/output/Octopus-{version}/report/TEST-test.org.webdocwf.util.loader.RunTests.xml, and also in standard output (in window from which 'make test' command is executed).

Chapter 3. Using Enhydra Octopus

Introduction

As we have said earlier, Enhydra Octopus is consisted of two applications: OctopusGenerator and OctopusLoader application. The easiest way to start both applications is to use GUI (graphic user interface).

Starting OctopusGenerator via GUI

OctopusGenerator is created to generate OctopusLoader loadjob skeletons (and even DODS DOML files!) from an existing database. Many different types of databases can be used (MSSQL, Oracle, DB2, QED, JDBC-ODBC with Excel and Access, MySQL, CSV-files, XML-files,...).

If you have installed Enhydra Octopus application on your machine, just go to “Start Menu”, then choose “Programs”, “tdt {version}”, and “OctopusGenerator”. After that, user frame is presented on the screen.

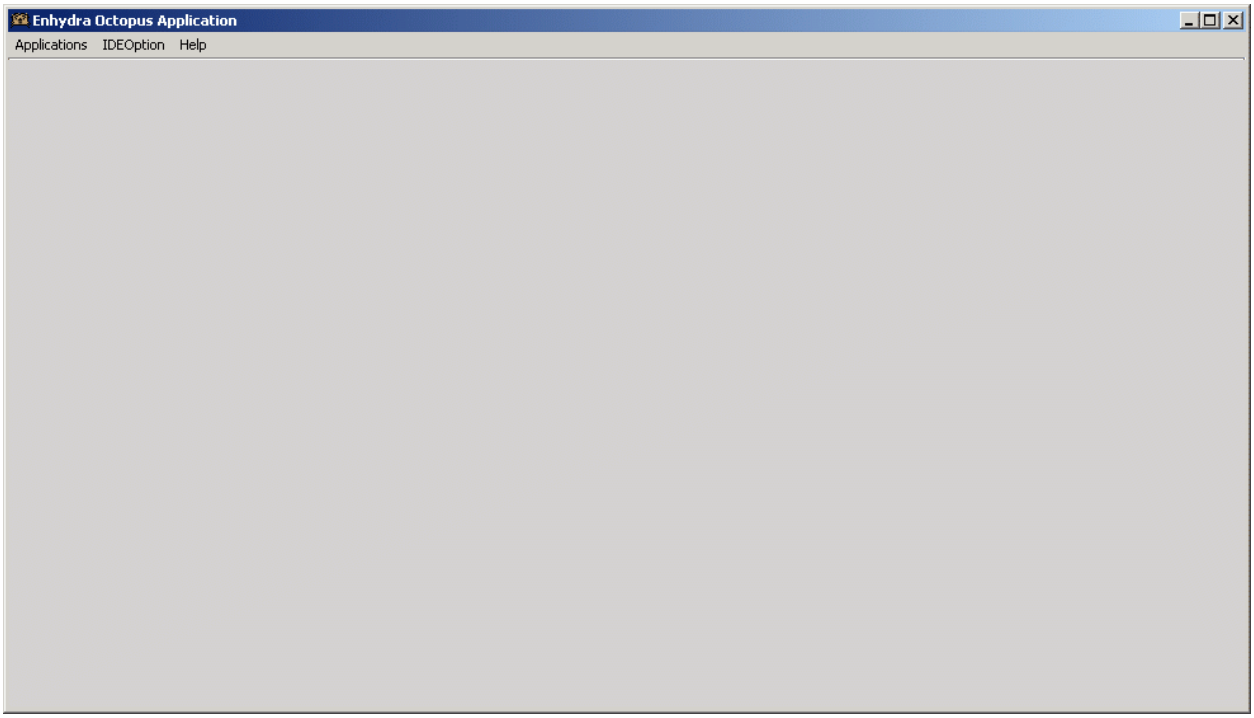

Figure 2.

If you have binary distribution, and you have built it, the only thing you have to do is double-click on OctopusGenerator.jar file, which is placed in OCTOPUS_HOME/Octopus-src/output/Octopus-{version}/bin directory, and is executable file. The same screen is presented in front of you.

After that, if you want to start OctopusGenerator application, you must go to menu and push 'Applications' and then 'New Octopus Generator' button,or right click on mouse button and select 'New Octopus Generator'. New frame is presented on screen.

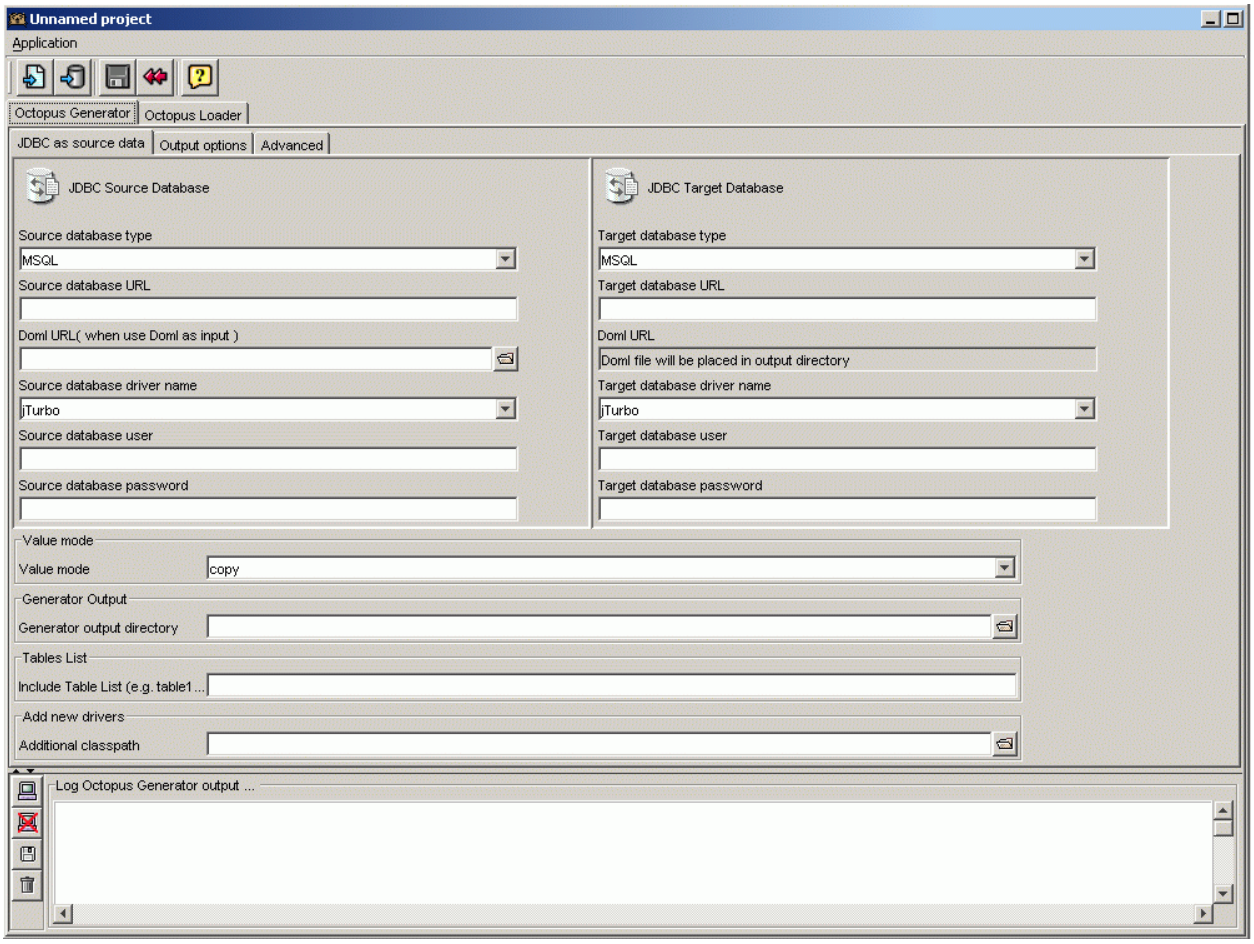

Figure 3.

Now you must enter the parameters into the frame fields that are in front of you ("on JDBC as source data" tab).

- “Source database type” - defines the type of the source database. Possible types are: MSQL, InstantDB, Oracle, Informix, HypersonicSQL, DB2, QED, MySQL, PostgreSQL, McKoi, Octetstring, CJDBC.
- “Target database type” - defines the type of the target database. Possible types are: MSQL, Standard, InstantDB, Oracle, Informix, HypersonicSQL, DB2, QED, MySQL, Csv, Excel, Access, XML, PostgreSQL, McKoi, Octetstring, CJDBC, Sybase, Paradox, I18n.
- “Source database URL” - defines the URL (full path) to source database (e.g.localhost:1433/Together1).
- “Target database URL” – defines the URL (full path) to target database (e.g.localhost:1433/Together1).
- “Doml URL (when use doml as input)” - defines the full path to doml file.
- “Source database driver name” - represents driver name of defined source database. Possible driver names are: jTurbo, microsoft, idb, oracle, ifxjdbc, hsql, db2, quadcap, mm, postgresql, mckoidb, jdbc-ldap, CJDBC.
- “Target database driver name” - represents driver name of defined target database. Possible driver names are: jTurbo, freetds, microsoft, ,standard, idb, oracle, ifxjdbc, hsql, db2, quadcap, mm, csv, jdbc-odbc, xml, postgresql, mckoidb, jdbc-ldap, CJDBC,syb, i18n.
- “Source database user” - defines the user of the source database.

- “Source database password” - defines the user password for the source database.
- “Target database user” - defines the user of the target database.
- “Target database password” - defines the user password for the target database.
- “Value Mode” - is the difference between overwrite and update attributes. Possible values are: Copy and Sync.
- “Generator output directory” - is optional. It represents the directory, where the OctopusGenerator places all created files. If this argument doesn't exist, OctopusGenerator places created files into current directory.
- "Include Table List" - Defines the list of table names which you want to include into Generator process. **Tables must be divided with “;”**.
- “Additional classpaths” – defines the URL to driver jar files which you want to include in OctopusGenerator classpath

  On "Output options" tab:

- “Generate Sql files” – represents the possibility to generate sql files as output files. In sql Options frame, you can choose (check or uncheck) which sql file will be generated. Possible values are:

  - “Drop Tables” – if you check this checkbox, DropTables.sql will be generated as one of the output files. This file contains sql statements for dropping all target tables.
  - "Drop Foreign Keys"- if you check this checkbox, DropIntegrity.sql will be generated as one of the output files. This file contains sql statements for dropping all foreign keys in target tables.
  - “Create Tables” - if you check this checkbox, CreateTables.sql will be generated as one of the output files. This file contains sql statements for creating all target tables.
  - “Create Primary Keys” - if you check this checkbox, CreatePrimary.sql will be generated as one of the output files. This file contains sql statements for creating primary keys in target tables.
  - “Create Foreign Keys” - if you check this checkbox, CreateIntegrity.sql will be generated as one of the output files. This file contains sql statements for creating foreign keys in target tables.
  - “Create Indexes” – if you check this checkbox, CreateIndex.sql will be generated as one of the output files. This file contains sql statements for creating indexes in target tables.
  - “Create SQL Statements for All Vendors” – represents the possibility to generate sql statements for all named database vendors. Only sql statements that are “checked” will be generated.

- “Generate Xml files” – represents the possibility to generate xml files (LoaderJob.olj and ImportdDefinitin.oli) as output files. In xml Options frame, you can choose mode in which this xml files will be generated. Possible modes are:

  - “Optimized mode for all tables” - means that all source tables will be just copied to target tables.
  - “Full mode for all tables” - means that all source tables will be mapped to target tables with all relationships between them.

- “Generate Doml file” – represents the possibility to generate dom file as one of the output files.
- “Package name” - if you choose to generate doml file as one of the output files, you must enter the package name for this file.

- "Logging options" - logging activities during generation process

  - "Log mode" - level of logging. Possible values are:none, normal, full. Default value is normal.
  - "Log file directory" - directory where log file will be placed. Default is working directory.
  - "Log file name" - name for log file. Default is GeneratorLog-YYYY-MM-DD-HH-mm-SS.txt

  On "Advanced" tab:

- "Path to conf files in jar" - Defines the path to jar file in which conf files are placed.

When you enter all needed parameters, just press “start” button.

Starting OctopusGenerator as an application

OctopusGenerator as an application can be started on several ways:

Start OctopusGenerator (start up scripts)

If you want to start OctopusGenerator application with start up scripts (batch file), you have to start OctopusGenerator (cmd or sh) file with associated parameters.

This batch file is placed in bin directory of output of binary distributions.

Usage:

```
OctopusGenerator [-st source_type
                  -tt target_type
                  -sdb source_database
                  -tdb target_database
                  -su source_user
                  –sp source_password
                  –tu target_user
                  –tp target_password
                  –sdn source_driver_name
                  –tdn target_driver_name
                  –m value_mode
                  –doml path_to_doml_file
                  -o output_directory
                  –pack package_name
                  –xml generate_xml_files
                  –gdoml generate_doml_file
                  –sqlDT generate_sql_DT_file
                  –sqlDI generate_sql_DI_file
                  –sqlCT generate_sql_CT_file
                  –sqlCPK generate_sql_CPK_file
                  –sqlCFK_generate_sql_CFK_file
                  –sqlCT generate_sql_CI_file
                  –sqlAll generate_sql_ for_all
                  –fm full_mode
                  -rm restart_mode
                  -it includeTableList
                  -cjs path_to_conf_files_in_jar
                  -lm log_mode
                  -l log_directory_name
                  -f log_file_name
                 ]
```

[options]:

- **-st**: defines the type of the source database. This argument is **required**. Possible values for this argument are MSQL, DB2, QED, Oracle, PostgreSQL, McKoi, MySQL , HypersonicSQL, InstantDB, Access and Standard.

  ```
  e.g.: -st MSQL (source database is Microsoft SQL ).
  ```

- **-tt**: defines the type of the target database. This argument is **required**. Possible values for this argument are: MSQL, DB2, QED, Oracle , PostgreSQL, McKoi, MySQL , HypersonicSQL, InstantDB, Access, Csv, XML, Excel, Standard, Sybase, Paradox.

  ```
  e.g.: -tt msql (target database is Microsoft SQL ).
  ```

- **-sdn**: represents driver name of defined source database. This parameter is **optional**. If this argument doesn't exist, application takes first driver name, which is placed in conf file (which belongs to source database). Possible values for this argument are: jTurbo, microsoft, freetds, quadcap, mm, hsql, idb, oracle, db2, ifxjdbc, postgresql, and mckoi.

  ```
  e.g.: -sdn JTurbo
  ```

- **-tdn**: represents driver of defined target database. This parameter is **optional**. If this argument doesn't exist, application takes first driver name, which is placed in conf file (which belongs to target database). Possible values for this argument are: jTurbo, microsoft, freetds, quadcap, mm, hsql, idb, oracle, db2, ifxjdbc, jdbc-odbc, xml, csv, postgresql, and mckoi.

  ```
  e.g.: -tdn quadcap
  ```

- **-sdb**: represents the part of Connection property and defines the name of source database and it's URL. This argument is **required**.

  ```
  e.g.: -sdb localhost:1433\\Together
  ```

- **-tdb**: represents the part of Connection property and defines the name of target database name and it's URL. This argument is **required**.

  ```
  e.g.: -tdb localhost:1433\\newTogether
  ```

- **-m**: defines the difference between overwrite and update attribute. Possible values are '**Copy**' and '**Sync**'. If this argument doesn't exist, default value is 'Copy'.

  ```
  e.g.: -m copy (then the parameter mode=overwrite. This is copying of whole source database to tar
  ```

  ```
  e.g. : -m sync (then the parameter mode=update. This is the synchronization of the data from targ
  ```

- **-su**: defines user name of the source database. If the source database requires user name, you must enter corresponding user name.

  ```
  e.g.: -su sa
  ```

- **-sp**: defines user password for the source database. If the source database requires user password, you must enter corresponding user password.

  ```
  e.g.: -sp (empty)
  ```

- **-tu**: defines user name of the target database. If the target database requires user name, you must enter corresponding user name.

  ```
  e.g.: -tu sa
  ```

- **-tp**: defines user password for the target database. If the target database requires user password, you must enter corresponding user password.

  ```
  e.g.: -tp (empty)
  ```

- **-doml**: defines the full path to doml file. This argument is required if you want to use doml file as source data.

```
e.g.: -doml C:\Doml\DomlTest.doml
```

- **-o**: this parameter is **optional**. It represents the directory, where the Octopus place created files. If this argument does not exist, Octopus places created files into the current directory.

  ```
  e.g.: -o GeneratorOutput (in current directory, you create new directory GeneratorOutput).
  ```

- **-pack**: defines the package name for the generated doml file. If you want to generate doml file as one of the output files, this argument is required.

  ```
  e.g.: -pack org.webdoc.util.loader
  ```

- **-xml**: it represents the possibility to generate xml files (LoaderJob.olj and ImportDefinition.oli) as output files. Possible values are **true** and **false**.

- **-gdoml**: it represents the possibility to generate doml file as one of the output files. Possible values are **true** and **false**.

- **-sqlCT**: it represents the possibility to generate CreateTables.sql file as one of the output files. Possible values are **true** and **false**. This file contains sql statements for creating all target tables.

- **-sqlDT**: it represents the possibility to generate DropTables.sql file as one of the output files. Possible values are **true** and **false**. This file contains sql statements for dropping all target tables.

- **-sqlDI** : it represents the possibility to generate DropIntegrity.sql file as one of the output files. Possible values are **true** and **false**. This file contains sql statements for dropping all foreign keys in target tables.

- **-sqlCPK**: it represents the possibility to generate CreatePrimary.sql file as one of the output files. Possible values are **true** and **false**. This file contains sql statements for creating primary keys in target tables.

- **-sqlCFK**: it represents the possibility to generate CreateIntegrity.sql file as one of the output files. Possible values are **true** and **false**. This file contains sql statements for creating foreign keys in target tables.

- **-slqCI**: it represents the possibility to generate CreateIndex.sql file as one of the output files. Possible values are **true** and **false**. This file contains sql statements for creating indexes in target tables.

- **-sqlAll**: it represents the possibility to generate sql statements for all named database vendors. Possible values are **true** and **false**. Only the sql options, which are given value ‘true’, will be generated.

- **-fm**: defines the mode of output xml files. Possible values are **true** and **false**. If this parameter has value ‘false’, all source tables will be just copied to target tables and if this parameter has value ‘true’, all source tables will be mapped to target tables with all relationships between them.

- **-rm** : defines if you use Octopus generator application for restoring some database. Possible values are **true** and **false**.

- **-it**: defines the list of table names which you want to include into Generator process. Tables must be divided with “**;**”.

- **-lm**: level of logging. Possible values are:none, normal, full. Default value is normal.

- **-l**: directory where log file will be placed. Default is working directory.

- **-f**: name for log file. Default is GeneratorLog-YYYY-MM-DD-HH-mm-SS.txt

- **-cjs** : defines the path to jar file in which conf files are palced. This parameter should be used only if you put conf files in separate jar file.

  ```
  e.g.: -cjs newXml/conf
  ```

Note

The order of the options is not relevant, but the arguments values are case sensitive.

Example of starting OctopusGenerator application with parameters:

```
OctopusGenerator -m copy -o GeneratorOutput -sdb localhost:1433/Together -st MSQL -sdn jTurbo -su sa
-tdb C:\Users\Qed3.0\Test1;create=true -tt QED -tdn quadcap –pack org.webdocwf.util.loader -xml true
-sqlCT true -sqlCPK true -fm true -it table1;table2
```

Also, using native Java VM commands, you can start the OctopusGenerator with the following syntax:

```
java org.webdocwf.util.loader.generator.LoaderGenerator [options]
```

Note

Options are listed above.

Start OctopusGenerator as library, from Java Application:

The OctopusGenerator can be used from every Java application by its public constructor.

```
public LoaderGenerator(
       String source_type,
       String source_database,
       String value_mode,
       String output_directory,
       String source_driver_name,
       String target_driver_name,
       String target_database,
       String target_type,
       String source_user,
       String source_password,
       String target_user,
       String target_password,
       String path_to_doml_file,
       String package_name,
       String generate_drop_table_file,
       String generate_drop_integrity_file,
       String generate_create_table_file,
       String generate_create_primary_kays__file,
       String generate_create_foreign_keys_file,
       String generate_create_indexes_file,
       String generate_sql_for_all_vendors,
       String generate_xml_files,
       String generate_doml_file,
       String full_mode,
       String restart_mode,
       String includeTableList)
throws LoaderException {
```

To start the load process simply include following code in your java application:

```
e.g.

try{
    LoaderGenerator generator=new LoaderGenerator(
                  "MSQL", "localhost:1433/Together", "copy",
                  GeneratorOutput, "jTurbo", "quadcap",
                  C:\Users\TestBaze\Qed3.0\Test1;create=true",
                  "QED", "sa", "", "", "", "",
                  "org.webdoc.util.loader", "true", "true",
                  "true", "true", "true", "true", "true",
                  "true", "true","table1;table2" );

    generator.generate();
```

```
    }catch(LoaderException le){
        le.printStackTrace();
    }
```

Method *generate()* is the main method in *LoaderGenerator* class. It creates xml, sql and doml files.

Public constructor of OctopusGenerator class and it's method *generate()* must be in try - catch block, because public constructor and generate method throws the *LoaderException* exception. *LoaderException* class is part of the Octopus.jar file.

Next public constructor of OctopusGenerator is like this above, but with one more parameter. Named parameter is –cjs (path to jar file in which conf files are placed).

```
public LoaderGenerator(
       String source_type,
       String source_database,
       String value_mode,
       String output_directory,
       String source_driver_name,
       String target_driver_name,
       String target_database,
       String target_type,
       String source_user,
       String source_password,
       String target_user,
       String target_password,
       String path_to_doml_file,
       String package_name,
       String generate_drop_table_file,
       String generate_drop_integrity_file,
       String generate_create_table_file,
       String generate_create_primary_kays__file,
       String generate_create_foreign_keys_file,
       String generate_create_indexes_file,
       String generate_sql_for_all_vendors,
       String generate_xml_files,
       String generate_doml_file,
       String full_mode,
       String restart_mode,
       String includeTableList,
       String path_to_conf_files_in_jar)
throws LoaderException {
```

The next public constructor of *LoaderGenerator* class has 4 parameters. For other parameters, OctopusGenerator uses default values. Also, you can set values for other parameters using set methods in *LoaderGenerator* class, as defined in the API documentation.

```
public LoaderGenerator(
                  String sourceDataBase,
                  String targetDataBase,
                  String sourceType,
                  String targetType)
throws LoaderException {
```

To start the load process simply include following code in your java application:

```
e.g.

try{
    LoaderGenerator generator=new LoaderGenerator(
                         "localhost:1433/Together",
                         "C:\Users\TestBaze\Qed3.0\Test1;create=true""MSQL",
                         "QED");

    generator.generate();

}catch(LoaderException le){
    le.printStackTrace();
```

```
}
```

Next public constructor of OctopusGenerator is like this above, but with one more parameter. Named parameter is –cjs (path to jar file in which conf files are placed).

```
public LoaderGenerator(
                  String sourceDataBase,
                  String targetDataBase,
                  String sourceType,
                  String targetType,
                  String path_to_conf_files_in_jar)
throws LoaderException {
```

Start OctopusGenerator, as an Ant task

OctopusGenerator Task class (*LoadGeneratorTask*) extends jakarta-ant Task class and is used for starting OctopusGenerator application as a jakarta-ant task in build.xml file.

Attributes and sub elements of *LoadGeneratorTask* represent OctopusGenerator parameters.

Tag <taskdef>

Adds a task definition to the current project, so that this new task can be used in the current project. Two attributes are needed: the name that identifies this task uniquely, and the full name of the class (including the packages) that implements this task.

```
e.g.
<taskdef name="LoaderGeneratorTask" classname="org.webdocwf.util.loader.task.LoadGeneratorTask">
```

Wherever path-like values need to be specified, a nested element can be used. This takes the general form of:

```
e.g.
    <classpath>
      <pathelement path="${classpath}"/>
      <pathelement location="lib/OctopusGenerator.jar"/>
    </classpath>
```

The location attribute specifies a single file or directory relative to the project's base directory (or an absolute filename), while the path attribute accepts colon- or semicolon-separated lists of locations. The path attribute is used with predefined paths - in any other case, multiple elements with location attributes should be preferred. OctopusGenerator.jar and Octopus.jar are required jar files for this application.

```
e.g.
    <classpath>
        <pathelement location="lib/antlr.jar"/>
    </classpath>
```

Here we add all drivers needed for building of OctopusGenerator application. If you use the driver, which isn't supported in OctopusGenerator application, you must install it first, and add it into classpath.

Tag <LoaderGeneratorTask>

<LoaderGeneratorTask> represents the name of the java class that implements this task, with associated parameters.

Parameters:

- *sourceType:* defines the type of the source database. This argument is **required**. Possible values for this argument are: MSQL, DB2, QED, Oracle , PostgreSQL, McKoi, MySQL , HypersonicSQL, InstantDB, Access and Standard.
- *targetType:* defines the type of the target database. This argument is **required**. Possible values for this argument are: MSQL, DB2, QED, Oracle , PostgreSQL, McKoi, MySQL , HypersonicSQL, InstantDB, Access, Csv,

XML, Excel, Standard, Sybase, Paradox.

- *sourceDriverName:* represents driver of defined source database. This parameter is **optional**. If this argument doesn't exist, application takes the first driver name, which is placed in conf file (which belongs to source database). Possible values for this argument are: jTurbo, microsoft, freetds, quadcap, mm, hsql, idb, oracle, db2, ifxjdbc, postgresql, and mckoi.

- *targetDriverName:* represents driver of defined target database. This parameter is **optional**. If this argument doesn't exist, application takes the first driver name, which is placed in conf file (which belongs to target database). Possible values for this argument are: jTurbo, microsoft, freetds, quadcap, mm, hsql, idb, oracle, db2, ifxjdbc, postgresql, csv, xml, jdbc-odbc, and mckoi.

- *sourceDataBase:* represents the part of Connection property and defines the name of the source database and it's URL. This argument is **required**.

- *targetDataBase:* represents the part of Connection property and defines the name of the target database and it's URL. This argument is **required**.

- *valueMode:* represents the difference between overwrite and update attribute. Possible values are '**Sync**' and '**Copy**'. If this argument doesn't exist, default value is 'Copy'.

- *sourceUser:* defines user name of the source database. If the source database requires user name, you must enter corresponding user name.

- *sourcePassword:* defines user password for the source database. If the source database requires user password, you must enter corresponding user password.

- *targetUser:* defines user name of the target database. If the target database requires user name, you must enter corresponding user name.

- *targetPassword:* defines user password for the target database. If the target database requires user password, you must enter corresponding user password.

- *generatorOutput* is **optional**. This parameter represents the directory, where the OctopusGenerator places created files. If this argument does not exists, OctopusGenerator places created files in to current directory.

- *packageName:* defines the package name for the generated doml file. If you want to generate the doml file as one of the output files, this argument is **required**.

- *generateXml:* represents the possibility to generate xml files (LoaderJob.olj and ImportDefinition.oli) as output files. Possible values are **true** and **false**.

- *generateDoml:* represents the possibility to generate doml file as one of the output files. Possible values are **true** and **false**.

- *generateCreateTableStmt:* represents the possibility to generate CreateTables.sql file as one of the output files. Possible values are **true** and **false**. This file contains sql statements for creating all target tables.

- *generateDropTableStmt:* represents the possibility to generate DropTables.sql file as one of the output files. Possible values are **true** and **false**. This file contains sql statements for dropping all target tables.

- *generateDropIntegrityStmt:* represents the possibility to generate DropIntegrity.sql file as one of the output files. Possible values are **true** and **false**. This file contains sql statements for dropping all foreign keys in target tables.

- *generateCreatePKStmt:* represents the possibility to generate CreatePrimary.sql file as one of the output files. Possible values are **true** and **false**. This file contains sql statements for creating primary keys in target tables.

- *generateCreateFKStmt:* represents the possibility to generate CreateIntegrity.sql file as one of the output files. Possible values are **true** and **false**. This file contains sql statements for creating foreign keys in target tables.

- *generateCreateIndexStmt:* represents the possibility to generate CreateIndex.sql file as one of the output files. Possible values are **true** and **false**. This file contains sql statements for creating indexes in target tables.
- *generateSqlForAllVendors:* represents the possibility to generate sql statements for all named database vendors. Possible values are **true** and **false**. Only the sql statements which have the value 'true' will be generated.
- *fullMode:* defines the mode of output xml files. Possible values are **true** and **false**. If this parameter has value false, all source tables will be just copied to target tables and if this parameter has value true, all source tables will be mapped to target tables with all relationships between them.
- *pack:* defines the package name for the generated doml file. If you want to generate the doml file as one of the output files, this argument is required.
- *includeTableList*: defines the list of table names which you want to include into Generator process. **Tables must be divided with ";"**
- *confJarStructure:* defines the path to jar file in which conf files are placed. This parameter is optional.
- *logMode:* defines log mode. Possible values are:none,normal, full.Default is normal.
- *logDirName:* defines logging directory where will be placed log files.Default is working directory.
- *logFileName:* defines log file name. Default is GeneratorLog-YYYY-MM-DD-HH-mm-SS.txt.

Required parameters are:

- sourceDataBase
- targetDataBase
- sourceType
- targetType

e.g.

```
    <LoaderGeneratorTask
            sourceType="QED"
            targetType="MSQL"
            sourceDriverName="quadcap"
            targetDriverName="jTurbo"
            sourceDataBase="C:\Database\testDatabase"
            targetDataBase ="localhost:1433/newTogether"
            valueMode ="copy"
            generatorOutput ="GeneratorOutput"
            sourceUser =""
            sourcePassword =""
            targetUser ="sa"
            targetPassword =""
            domlPath =""
            packageName ="org.prozone.octopus"
            generateDropTableStmt="true"
            generateCreateTableStmt="true"
            generateCreatePKStmt="true"
            generateCreateFKStmt="true"
            generateCreateIndexStmt="true"
            generateXml="true"
            generateDoml="true"
            fullMode="true"
            includeTableList="table1;table2"
            octopusHome="C:\Users\OctopusGenerator">
```

```
</LoaderGeneratorTask>
```

Starting OctopusLoader via GUI

If you have installed Enhydra Octopus application on your machine, just go to “Start Menu”, then choose “Programs”, “tdt {version}”, and then “Enhydra Octopus”. After that, user frame is presented on screen.

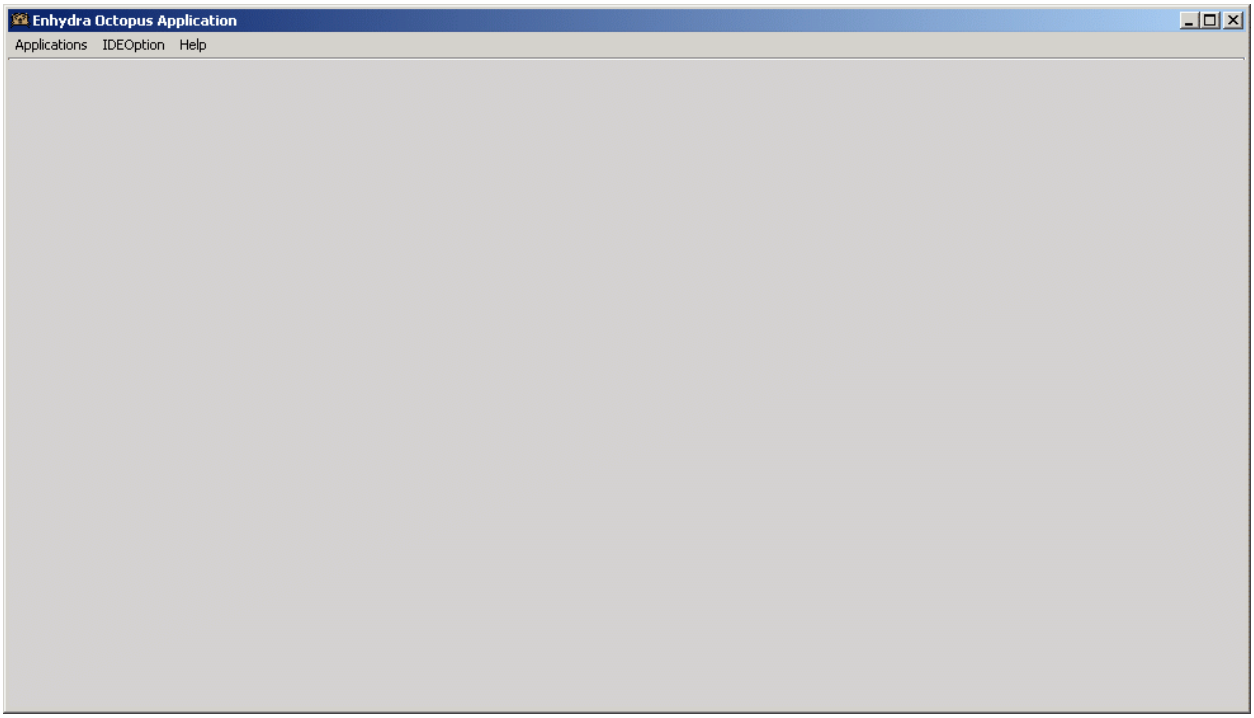

Figure 4.

If you have binary distribution, and you have built it, the only thing you have to do is double-click on OctopusGenerator.jar file, which is placed in OCTOPUS_HOME/Octopus-src/output/Octopus-{version}/bin directory, and is executable file. The same screen is presented in front of you.

After that, if you want to start OctopusLoader application, you must go to menu and push "Applications" and then "New Octopus Loader" button, or click on right mouse button and select "New Octopus Loader". New frame is presented on screen.

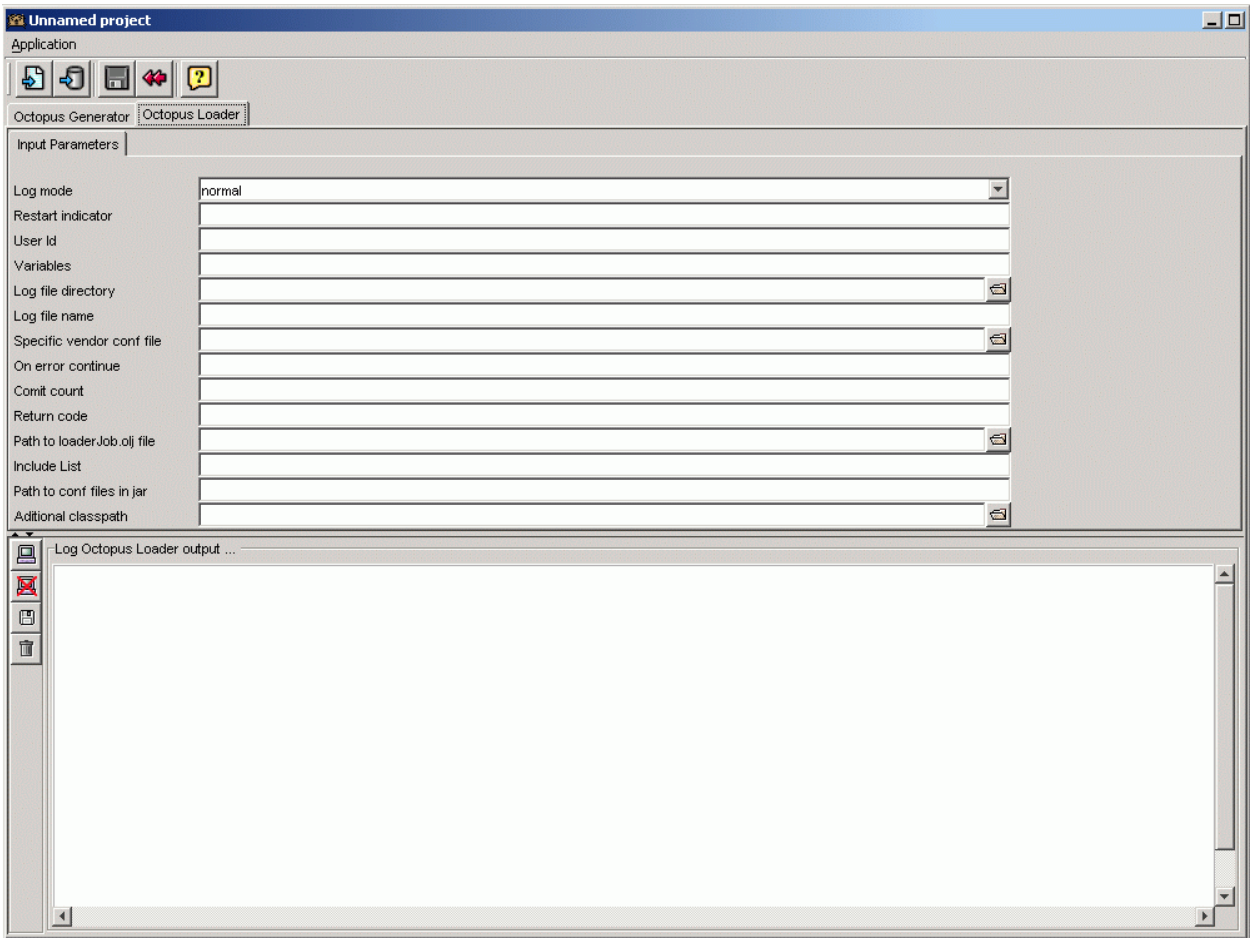

Figure 5.

Now you must enter the parameters into frame fields that are in front of you.

- “Log mode” - defines the default log mode. There are three possible log modes:
  - “none” - writes only that the application is started, the number of the import definitions, and time when the application has been finished;
  - "normal" - writes main events (number of import jobs, the names of the importDefinitions, time when the application was started/ended and the number of committed rows) and
  - "full" - writes detailed information about the loading process.
- “Restart indicator” – represents the possibility to continue the process where it was stopped (if it was stopped in the middle of the process) without losing or multiplying data.
- “User Id” - defines the current UserID used in UserID value columns.
- “Variables” - defines variables used in variable columns.
- “Log file directory” - defines the path to log file directory. The default log file directory is the current working directory.

- “Log file name” - defines the log file name. The default log file name is "LoaderLog-YYYY-MM-DD-HH-mm-SS.txt".
- “Specific vendor conf file” - defines the filename of the XML DB-vendor configuration file. The default is "OctopusDBVendors.xml".
- “On error continue” – enables the loading process to continue if an error in SQL statements or import jobs occurs. Possible values are true or false.
- “Additional classpath” - extends the classpath with additional paths.
- “Commit count” - defines the default commit count. System default is "100".
- “Return code” - defines the default java.exe return code to the environment, if some element of load jobs fails.
- “Path to loaderJob.olj file” – defines the path to loaderJob.olj file, main file for loading process.
- “Include list” - defines the tables, which will be processed. If you enter this parameter, only tables in this field will be processed.
- "Path to conf files in jar" - defines the path to jar file in which conf files are placed

When you enter all needed parameters, just press “Start” button.

Starting OctopusLoader as an application

OctopusLoader as an application can be started on several ways:

Starting OctopusLoader as an application, with one parameter

The only one parameter used is URL to XML file, for example:

```
java org.webdocwf.util.loader.Loader MyLoadJob.xml
```

where, *org.webdocwf.util.loader* is a package name to which *Loader* class belongs, and MyLoadJob.xml is input XML file.

Note

For other parameters, application use default values.

Starting OctopusLoader as an application with more parameters

Using native Java VM commands, you can start the OctopusLoader with the following syntax:

```
java org.webdocwf.util.loader.Loader [options] loadJob_xml_filename
```

"loadJob_xml_filename" is the name of the XML file which contains all processing instructions.

Table 3.1. Options for Starting OctopusLoader

-m	defines the default log mode. Possible values are "none", "normal" (this is the default) and "full".	-m none

-r	starts the Octopus in restart mode after abnormal termination in a prior execution.	-r
-u	defines the current UserID used in UserID value columns.	-u r004d\meier
-v	defines variables used in variable columns.	-v varname1=varval1;varname2=varval2
-l	defines the log file directory. The default is the current working directory.	-l c:\my_directory_for_logfiles
-f	defines the log file name. The default is "LoaderLog-YYYY-MM-DD-HH-mm-SS.txt".	-f c:\somewhere\down\..\mydir2\mylog.txt
-d	the filename of the XML DB-vendor configuration file. The default is "OctopusDBVendors.xml".	-d MyVendor.xml
-e	defines to set the default of "onErrorContinue" to "true". Is false otherwise.	-e
-p	Extend the classpath with additional paths.	-p c:\lib\csvjdbc.jar;c:\lib\JTurbo.jar
-c	Sets the default commit count. System default is "100".	-c 1000
-rc	Sets the default return code to the environment if some element of load jobs fails	-rc 10
-it	Sets the table names which will be processed	-it table1;table2;table3
-cjs	sets the path to jar file in which conf files are placed. This parameter should be used only if you put conf files in separate jar file.	-cjs newXml\conf

Note

The order of the options is not relevant.

An example of complete start command:

```
java -cp c:\lib\wdw.jar;c:\lib\xmljdbc.jar
org.webdocwf.util.loader.Loader -m none
                                -u myID
                                -v myvar1=value1;myvar2=value2;myvar3=value3
                                -l c:\mylogs
                                -f loadlogs\mylog.txt
                                -d MyDB.xml
                                -p c:\lib\csvjdbc.jar
                                -c 1000 MyLoadJob.xml
                                -rc 1
```

If you want to start OctopusLoader application with start up scripts (batch file), you have to start Loader (cmd or sh) file with associated parameters.

This batch file is placed in bin directory of output of binary distributions.

```
Loader [options]
```

```
e.g Loader -u myID MyLoadJob.xml
```

Be sure to include necessary entries in the classpath for extra JDBC drivers, which are used in your loader jobs, or simply use the -p parameter (e.g –p C:\JTurbo\jturbo.jar).

Starting OctopusLoader as library

The OctopusLoader application can be used from every Java application by it's public constructor.

```
public Loader(String loadJobFileName, String mode, String userID, String logDirName,
String logFileName, boolean restartIndicator, Map variableValues, String vendorFileName, boolean onE
int commitCount, int returnCode)
```

To start the load process simply include the following code in your java application:

```
e.g.
Loader myOctopus=new Loader(
                  "MyLoadJob.xml",
                  Loader.LOGMODE_NONE,
                  "myID",
                  null,
                  null,
                  false,
                  null,
                  null,
                  true,
                  null,
                  0,
                  1);

   myOctopus.load();
```

Method load() is the main method in *Loader* class.

The log modes are defined as public String constants in the *Loader* class. Their names are:

- Loader.LOGMODE_NONE,
- Loader.LOGMODE_NORMAL,
- Loader.LOGMODE_FULL.

In next public constructor is added new parameter 'pathToConfFilesInJar' adn it's look like this:

```
public Loader(String loadJobFileName, String mode, String userID, String logDirName,
String logFileName, boolean restartIndicator, Map variableValues, String vendorFileName, boolean onE
int commitCount, int returnCode, String pathToConfFilesInJar)
```

The next public constructor is:

```
public Loader(String loadJobFileName, String mode, String userID, String logDirName,
String logFileName, boolean restartIndicator, Map variableValues, String vendorFileName, boolean onE
int commitCount, int returnCode, String[] includeTableList)
```

In next public constructor is added new parameter 'pathToConfFilesInJar' adn it's look like this:

```
public Loader(String loadJobFileName, String mode, String userID, String logDirName,
String logFileName, boolean restartIndicator, Map variableValues, String vendorFileName, boolean onE
int commitCount, int returnCode, String[] includeTableList, String pathToConfFilesInJar)
```

The next public constructor of *Loader* class has only one parameter. For other parameters, *Loader* class uses default values. Also, you can set values for other parameters using set methods in *Loader* class, as defined in the API documentation.

```
public Loader(String loadJobFileName)
```

To start the load process simply include the following code in your java application:

```
e.g.
Loader myOctopus=new Loader("MyLoadJob.xml");

myOctopus.load();
```

The next public constructor is :

```
public Loader(String loadJobFileName, String pathToConfFilesInJar)
```

Starting OctopusLoader as an Ant task

Octopus Task class (*LoaderTask*) extends jakarta-ant Task class and is used for starting OctopusLoader application as a jakarta-ant task in build.xml file.

Attributes and sub elements of *LoaderTask* class represents OctopusLoader parameters.

Tag <taskdef>

Adds a task definition to the current project, so that this new task can be used in the current project. Two attributes are needed, the name that identifies this task uniquely, and the full name of the class (including the packages) that implements this task.

```
e.g.

<taskdef name=" LoaderTask" classname=" org.webdocwf.util.loader.task.LoaderTask">
```

Tag < LoaderTask >

< LoaderTask > represents the name of the java class that implements this task, with associated parameters.

Table 3.2. Table of LoaderTask arguments

argument	**meaning**	**sample**
loadJob	defines input loaderJob xml file with rules for loading data from source to target database	loadJob="c:\test\loadJobExample.xml"
mode	defines the default logMode. Possible values are "none", "normal" (this is the default) and "full".	mode=" none"
restartIndicator	starts the Octopus in restart mode after abnormal termination in a prior execution.	restartIndicator="true"
userID	defines the current UserID used in UserID value columns.	userID=" r004d\meier"
logDir	defines the logfile directory. The default is the current working directory.	logDir="c:\my_directory_for_logfiles"
logFile	defines the logfile name. The default is "LoaderLog-YYYY-MM-DD-HH-mm-SS.txt".	logFile="c:\somewhere\down\..\mydir2\mylog.txt"
vendorFile	the filename of the XML DB-vendor	vendorFile="MyVendor.xml"

argument	meaning	sample
	configuration file. The default is "OctopusDBVendors.xml".	
onErrorContinue	defines to set the default of "onErrorContinue" to "true". Is false otherwise.	onErrorContinue="true"
commitCount	Sets the default commit count. System default is "100".	commitCount="1000"
pathToConfFile	sets the path to jar file in which conf files are placed. This parameter should be used only if you put conf files in separate jar file.	pathToConfFile=newXml\conf
tableNames	Sets the table names which will be processed	tableNames=table1;table2;table3

Table 3.3. Table LoaderTask subelements

subelement	consists of subelements	argument	meaning
variables	variable	name, value	"name" is name of variable that is used in Octopus variable columns (loaderJob xml file). "value" is value of variable that is used in Octopus variable columns (loaderJob xml file).
additionalPaths	additionalPath	path	File to add in classpath

```
e.g.
<taskdef name="LoaderTask" classname="org.webdocwf.util.loader.task.LoaderTask"/>
  <LoaderTask
           mode="full"
           restartIndicator="yes"
           userID="admin"
           logDir="c:\my_directory_for_logfiles"
           logFile="c:\somewhere\down\..\mydir2\mylog.txt"
           vendorFile="MyVendor.xml"
           onErrorContinue="yes"
           commitCount="1000"
           loadJob="GRECOOMDSBROWSER/ObjectLoader/LoadOMDSProject.xml">
      <additionalPaths>
          <additionalPath path="../lib/xercesImpl.jar"/>
          <additionalPath path="../lib/Octopus.jar"/>
          <additionalPath path="../lib/freetds_jdbc.jar"/>
      </additionalPaths>
      <variables>
          <variable name="name" value="varval"/>
      </variables>
  </LoaderTask>
```

Using OctopusLoader as a test case

OctopusLoader can be used for test case also.

Core classes

The following text walks you through the core classes that make Octopus as a JUnit extension.

Database Operation (org.webdocwf.util.loader.test)

It is an abstract class and you will find yourself using implementations or subclasses.

The Database Operation abstract class defines operations performed on a target database before and after each test. Again, the framework provides several implementations:

- DO_NOTHING
- CreateDatabaseOperation
- DropDatabaseOperation
- LoaderOperation

The DatabaseOperation class declaration:

```
public abstract class DatabaseOperation
{
    public static final String NONE = "NONE";
    public static final String CREATE = "CREATE";
    public static final String DROP = "DROP";
    public static final String LOADER = "LOADER";
    public static DatabaseOperation DO_NOTHING = new DummyAction();


    /**
     * Executes this operation on the specified database using the specified
     * connection to the database.
     *
     * @param conn the database connection.
     */
    public abstract void execute(Connection conn) throws SQLException;


/**
     * Returns type of database operation
     *
     */
    public abstract String getDatabaseOperationType();

}
```

DatabaseOperation.DO_NOTHING

This operation does absolutely nothing with the target database.

CreateDatabaseOperation

This operation creates new database using SQL commands "CREATE DATABASE <database_name>", where <database_name> is defined by method getDatabaseName(). Default value is "LoaderTest", and if you want to change that value, you have to ovewrite this method in class that extends LoaderTestCase class. Sql commands will be executed using Connection to the target database defined by getConnection() method.

DropDatabaseOperation

This operation deletes database using SQL commands "DROP DATABASE <database_name>", where <database_name> is defined by method getDatabaseName(). Default value is "LoaderTest", and if you want to change that value, you have to overwrite this method in class that extends LoaderTestCase class.) . Sql commands will be executed using Connection to the target database defined by getConnection() method.

LoaderOperation

This operation loads data from source tables into target tables using OctopusLoader. The transformation commands are based on a special XML DTD/scheme. The number of transformations in one XML loadJob file is not limited: creating databases, creating tables, inserting data into an empty database, inserting data in to an existing database, updating columns with constant values, updating relations, updating columns with system-date/time, updating columns with a current user ID, executing every possible SQL statement, creating artificial keys using the Enhydra DODS objectid logic, ...

How to create a test case to setup your database

To write a test case, please follow the steps described below.

***Step 1**: Create your database*

Your test will obviously need some data to work with. This means you must create database and load data from source tables into target tables. You can create database using CreateDatabaseOperation or executing SQL statement using Octopus and its sql sub elements in loaderJob configuration file.

Step 2: *Extend the LoaderTestCase class*

Now you need to create a test class. The easiest way to use OctopusLoader as a test case is to make your test class extend the LoaderTestCase class. LoaderTestCase extends the JUnit TestCase class. Two template methods are required to be implemented: getConnection() to return a connection to your database and getLoader() to return the instance of Loader class which will be used for loading data.

The following example is an example of implementation that returns a connection to a MSSQL database and an instance of Loader class, which uses LoadOMDSProjectTest.xml file.

```
public class LoaderTest extends LoaderTestCase
{
    public LoaderTest(String name)
    {
        super(name);
    }

    public Connection getConnection() throws Exception
    {

    Class driverClass = Class.forName("com.internetcds.jdbc.tds.Driver");
    Connection jdbcConnection = DriverManager.getConnection(
                "jdbc:freetds:sqlserver://localhost:1433/master", "sa", "");

    return jdbcConnection;
    }

    /**
     * Override method getDatabaseName() to rename test database.
     */
    public String getDatabaseName() throws Exception
     {
       return "OMDSTest";
     }

    public Loader getLoader() throws Exception
    {
    Loader loadJob= new Loader("LoadOMDSProjectTest.xml");
    loadJob.setUserID("admin");

    return loadJob;
    }
}
```

Note

Please read Octopus documentation, for using setXXX Octopus methods.

Step 3: *(Optional) Implement getSetUpOperation() and getTearDownOperation() methods*

By default, LoaderTestCase performs a CreateDatabaseOperation and execute OctopusLoader with xml file defined by getLoader() method, before executing each test and performs DropDatabaseOperation afterwards. You can modify this behavior by overwriting getSetUpOperation() and getTearDownOperation().

The following example demonstrates how you can easily overwrite the operation executed before or after your test.

```
public class SampleTest extends DatabaseTestCase
{
    public DatabaseOperation[] getSetUpOperation() throws Exception
    {
    //Executing loaderJob defined in xml file.
    DatabaseOperation[] dbOperation = new DatabaseOperation[1];
    dbOperation[0]=new LoaderOperation(getLoader());

    return dbOperation;
    }


    public DatabaseOperation[] getTearDownOperation() throws Exception
    {
     //Do nothing...
     DatabaseOperation[] dbOperation = new DatabaseOperation[1];
     dbOperation[0]=DatabaseOperation.DO_NOTHING;

     return dbOperation;
     }

    You can use more DatabaseOperations in startUp or at the end of test.
    /**
     * Returns the database operations executed in test setup.
     */
    public DatabaseOperation[] getSetUpOperation() throws Exception
    {
    //Creating test database.....
    DatabaseOperation[] dbOperation = new DatabaseOperation[4];
    dbOperation[0]=new CreateDatabaseOperation(getDatabaseName());

    //Creating tables.....
    dbOperation[1]=new LoaderOperation(getLoader());

    //Inserting data.....
    Loader loadJob1= new Loader("LoadOMDSProjectInsertData.xml");
    loadJob1.setUserID("admin");
    dbOperation[2]=new LoaderOperation(loadJob1);

   //Creating indexes, foreign keys.....
   Loader loadJob2= new Loader("LoadOMDSProjectCreateIndex.xml");
   dbOperation[3]=new LoaderOperation(loadJob2);

   return dbOperation;
   }

}
```

Note

In this example, before the start of the test, four database operations would be executed.

Step 4: *Implement your testXXX() methods*

Implement your test methods as you normally would with JUnit. Your database is now initialized before and cleaned-up after each test methods according to what you did in previous steps.

Using Octopus project files

Octopus project file is file with extension .opf which is property file with all settings that user typed in GUI - both for Generator and for Loader.

Example of Octopus project file :

```
#  Octopus project file.

ProjectName=Sample project

#  Loader section

Loader.AdditionalPaths=C:\\csvjdbc\\csvjdbc.jar
Loader.CommitCount=
Loader.ConfJarStructure=
Loader.IncludeTables=
Loader.LogFileDir=
Loader.LogFileName=
Loader.LogMode=normal
Loader.ErrorConntinue=
Loader.PathToXml=C:\\output\\LoaderJob.olj
Loader.RestartIndicator=
Loader.ReturnCode=
Loader.UserId=
Loader.Variables=
Loader.VendorConf=

#  Generator section

Generator.AdditionalPaths=C:\\csvjdbc\\csvjdbc.jar
Generator.ConfJarStructure=
Generator.DomlGenerate=false
Generator.DomlPath=
Generator.FullMode=false
Generator.GeneratorOutput=c:\\output
Generator.IncludeTableList=
Generator.PackageName=
Generator.SourceDatabase=DiscRack
Generator.SourceDriverName=csv
Generator.SourcePassword=
Generator.SourceType=Csv
Generator.SourceUser=
Generator.SqlGenerate=true
Generator.SqlStmtCreateFK=true
Generator.SqlStmtCreateIndex=true
Generator.SqlStmtCreatePK=true
Generator.SqlStmtCreateTable=true
Generator.SqlStmtDropIntegrity=true
Generator.SqlStmtDropTable=true
Generator.SqlStmtForAllVendors=false
Generator.TargetDatabase=DiscRackBackup
Generator.TargetDriverName=csv
Generator.TargetPassword=
Generator.TargetType=Csv
Generator.TargetUser=
Generator.ValueMode=copy
Generator.XmlGenerate=true
```

Octopus project can be saved if you choose from project frame's menu : Application -> Save and choose location where .opf file will be saved. This project can be opened if you choose from main menu : Project -> Open existing Octopus project and choose .opf file.

Also, if you have installed Enhydra Octopus on Windows platform (installed setup.exe), when double-click on .opf file, Enhydra Octopus application will be started and selected project will be open.

Chapter 4. OctopusGenerator

Introduction

The OctopusGenerator is created to generate OctopusLoader loadjob skeletons (and even DODS DOML files!) from an existing JDBC data source.

OctopusGenerator supports many different types of databases: MSQL, DB2, QED, Oracle, PostgreSQL, McKoi, MySQL, HypersonicSQL, InstantDB, Access, Csv, XML, Excel, Standard, Sybase and Paradox.

All supported databases, are described in configuration files: OctopusDBVendors.xml and database_typeConf.xml.

Note

Details about configuration files can be located in "Configuration files" chapter.

As source data, OctopusGenerator application supports all listed databases and doml file (DODS DOML file).

JDBC database as source data

Once, when OctopusGenerator is connected to source database, it reads all META DATA which describes relationships between tables and columns in this tables. All relationships that are read will be stored in appropriate files, which are going to be generated. Generated files are divided into XML, SQL and doml files.

XML files

In LoaderJob.olj file, which is the main file for loading process, OctopusGenerator will store information about:

- jdbc parameters for source and target database (driver name, driver class, user and password),
- sql files (which sql files will be generated and path to them), and
- import definitions (path to importDefinition.oli file).

e.g (for jdbc parameters)

```
    <jdbcDefaultParameters>
        <jdbcSourceParameters dbVendor="MSQL" driverName="jTurbo">
            <jdbcSourceParameter name="Password" value=""/>
            <jdbcSourceParameter name="Connection.Url" value="jdbc:JTurbo://localhost:1433/Together1
            <jdbcSourceParameter name="JdbcDriver" value="com.newatlanta.jturbo.driver.Driver"/>
            <jdbcSourceParameter name="User" value="sa"/>
        </jdbcSourceParameters>
        <jdbcTargetParameters dbVendor="MSQL" driverName="jTurbo">
            <jdbcTargetParameter name="Password" value=""/>
            <jdbcTargetParameter name="Connection.Url" value="jdbc:JTurbo://localhost:1433/newTogeth
            <jdbcTargetParameter name="JdbcDriver" value="com. newatlanta.jturbo.driver.Driver"/>
            <jdbcTargetParameter name="User" value="sa"/>
        </jdbcTargetParameters>
    </jdbcDefaultParameters>
```

e.g (for sql files)

```
    <sql commit="true" logMode="normal" name="CreateTables" onErrorContinue="false">
        <sqlStmt>
            <include href="sql/CreateTables.sql" parse="text"/>
        </sqlStmt>
    </sql>
```

```
e.g (for import definition file)

    <definitionInclude>
        <include href="xml/ImportDefinition.xml" parse="xml"/>
    </definitionInclude>
```

Note

Detailed information about tags listed above, can be founded in next chapter: OctopusLoader application.

In importDefinition.oli file, OctopusGenerator will store information about source and target tables, source and target columns and relationships between them, needed for copying source database to target database, which OctopusLoader application will do.

```
<definitionInclude>
    <importDefinition logMode="normal" name="ATTRIBUTEBOOLVALUE" objectIDIncrementid="20" tableName=
        <valueColumns>
            <valueColumn sourceColumnName="ATTRIBUTEDEFINITIONOID" targetColumnName="ATTRIBUTEDEFINI
            <valueColumn sourceColumnName="BOOLVALUE" targetColumnName="BOOLVALUE" targetTableID="0"
            <valueColumn sourceColumnName="GENERICOBJECTOID" targetColumnName="GENERICOBJECTOID" tar
            <valueColumn sourceColumnName="oid" targetColumnName="oid" targetTableID="0" targetTable
        </valueColumns>
        <constantColumns>
            <constantColumn constantValue="0" targetColumnName="version" targetTableID="0" targetTab
        </constantColumns>
        <tables>
            <table insert="true" oidLogic="false" tableID="0" tableMode="Query" tableName="ATTRIBUTE
        </tables>
    </importDefinition>
</definitionInclude>

(or e.g of copyTable tag)

<copyTable name="PERSISTENTTREE" sourceTableName="PERSISTENTTREE" targetTableName="PERSISTENTTREE" l
```

SQL files

In SQL files, OctopusGenerator will store information about:

- tables and their columns that will be created as target tables,
- primary keys that will be created in target tables,
- foreign keys that will be created in target tables, and
- indexes that also will be created in target tables.

All those information OctopusGenerator application will read from source database.

SQL files that can be created are:

- DropTables.sql (this file contains sql statements for dropping all target tables)
- DropIntegrity.sql (this file contains sql statements for dropping all foreign keys in target tables)
- CreateTables.sql (this file contains sql statements for creating all target tables)
- CreateIndex.sql (this file contains sql statements for creating indexes in target tables)
- CreatePrimary.sql (this file contains sql statements for creating primary keys in target tables)

- CreateIntegrity.sql (this file contains sql statements for creating foreing keys in target tables)

OctopusGenerator has possibility to generate all named sql files for all database vendors, which is support.

Depending on the source database, OctopusGenerator can create some or all of those SQL files.

```
(e.g. of drop table statement)

Drop table ATTRIBUTEBOOLVALUE;

(e.g. of drop foreing key)
ALTER TABLE Disc DROP CONSTRAINT FK_Disc_1


(e.g. of create table statement)

Create table ATTRIBUTEBOOLVALUE
(
ATTRIBUTEDEFINITIONOID  DECIMAL(19,0)  NOT NULL    ,
BOOLVALUE  TINYINT NOT NULL    ,
GENERICOBJECTOID  DECIMAL(19,0)  NOT NULL    ,
oid  DECIMAL(19,0)  NOT NULL    ,
version  INT NOT NULL   );

(e.g. of create index statement)

CREATE UNIQUE INDEX I3_ATTRIBUTEBOOLVALUE ON ATTRIBUTEBOOLVALUE(GENERICOBJECTOID,ATTRIBUTEDEFINITION

(e.g. of create primary key statement)

ALTER TABLE  ATTRIBUTEBOOLVALUE ADD CONSTRAINT ATTRIBUTEBOOLVALUE_oid PRIMARY KEY(oid) ;

(e.g. of create foreign key statement)

ALTER TABLE Disc ADD CONSTRAINT FK_Disc_1 FOREIGN KEY (owner) REFERENCES Person (oid) ;
```

DOML file

If you choose to generate doml file as one of the output files, OctopusGenerator application create doml file with same name as the type of target database. This file will be placed in output_dir/Doml directory (output_dir represents one of the input parameters(-o)).

DOML file as source data

Doml file is DODS (Data Object Design Studio) file, which is XSL based tool for communication with databases.

If we use doml file as source data, OctopusGenerator will read type of the database and all META DATA which describes tables, primary keys, foreign keys and indexes from that file. Location of the source database, type and name of the driver, user and password - if requested, application reads from input parameters.

If OctopusGenerator uses doml file as source data, the same XML and SQL files will be generated as if OctopusGenerator use JDBC database as source data. Generated files are:

- XML files
  - LoaderJob.olj
  - importDefinition.oli
- SQL files

- DropTables.sql
- DropIntegrity.sql
- CreateTables.sql
- CreatePrimary.sql
- CreateIntegrity.sql
- CreateIndex.sql

Note

If you use DODS doml file as source data, you can’t generate doml file as one of the output files.

Chapter 5. OctopusLoader

Introduction

OctopusLoader is a Java-based Extraction, Transformation, and Loading (ETL) tool. It may be connected to any JDBC data source and perform transformations defined in an XML file.

Generating OIDs for new objects supports DODS data models. Natural keys can be used to insert/update existing data and create relationships with OIDs.

An OctopusLoader job is written in XML file corresponding to the Enhydra Octopus DTD/scheme. XML file is the key part for loading criteria. XML tags define how the loading job will be done.

Validation

All XML, SQL and DTD files, which will be used in OctopusLoader jobs, are included in one "main XML" file, and that file will be validated. If validating error occurs, parser will print line number in real XML file. DTDs and XML schemes are placed in JAR file and in OCTOPUS_HOME\XmlTransform\xml directory. Which one will be used depends on whether OCTOPUS_HOME, Java property, is defined or not.

Note

Java property can be defined by using -D option while starting Java application

```
e.g.
java -DOCTOPUS_HOME=C:\Octopus
```

If OCTOPUS_HOME is defined, DTDs and XML schemes from OCTOPUS_HOME\XmlTransform\xml\ will be used for validation. Otherwise, DTDs and XML schemes from Octopus.jar file will be used. If validation error occurs, Loader class will throw LoaderException exception.

Logging

OctopusLoader application is enabled to log its output with three types of logger's:

- you can use '**standard**' logger,
- you can use '**log4j**' logger, or
- you can implement/use **new** logger ('users logger').

Every of named loggers must extend **Logger** class which is placed in **org.webdocwf.util.loader.logging package**, and must implement its abstract methods.

Abstract methods which must be implements are:

- abstract public void **configure**(String confFilePath),
- abstract public boolean **isEnabled**(int level),
- abstract public boolean **isEnabled**(String level),
- abstract public int **getLevel**(String level),

- abstract public void **write**(int level, String msg),
- abstract public void **write**(String level, String msg),
- abstract public void **write**(int level, String msg, Throwable throwable),
- abstract public void **write**(String level, String msg, Throwable throwable),
- abstract public boolean[] **getEnabledLogLevels**(),
- abstract public void **setEnabledLogLevels**(String logMode),
- abstract public boolean **setMessage**(String key, String value),
- abstract public String **getMessage**(String key), and
- abstract public boolean **writeEcho** (String strLogTxt).

Note

For detail information about this class and its methods, see Java API documentation.

Implementation of **standard** logger, which is default logger for Octopus application, is added into Octopus application, and it's placed in **org.webdocwf.util.loader.logging package**.

Implementation of **log4j** logger is distributed separately from standard logger in log4j.jar file.

Depending on which *logger* you want to use, you must set "logger" attributes to appropriate values.

Table 5.1. Table of logger attributes

attribute	description
logClassName	represents full name of logger class, which implements Octopus *Logger interface*
pathToLoggerConf	defines path (URL) to logger configuration file. This path should be relative from LoaderJob.olj file or should be absolute.

Standard logger

If you want to use *standard* logger to log Octopus output, you fist need to set *logClassName* attribute in <loadJob> tag to appropriate value:

```
<loadJob ... logClassName="org.webdocwf.util.loader.logging.StandardLogger" ...>

</loadJob>
```

org.webdocwf.util.loader.logging.StandardLogger class extend *Logger* class from *org.webdocwf.util.loader.logging* package, and implement its abstract methods.

After that, you may set log directory (directory where the log file will be placed), and log file name, or you may use defaults.

Log dir and log file name you set when you start OctopusLoader as an application with more parameters.

Note

pathToLoggerConf attributes doesn't to be set, if you use standard logger.

Log4j logger

In Octopus application, implementation of Logger interface for log4j logger, is distributed separatly, in log4jlogger.jar file. This file is placed in OCTOPUS_HOME/lib directory.

If you want to use *log4j* logger to log Octopus output, you fist need to set *logClassName* and *pathToLoggerConf* attributes in <loadJob> tag to appropriate values:

```
<loadJob ... logClassName="org.webdocwf.util.loader.logging.Log4jLogger" pathToLoggerConf="URL_to_lo

</loadJob>
```

org.webdocwf.util.loader.logging.Log4jLogger class extend *Logger* class from *org.webdocwf.util.loader.logging* package, and implement its abstract methods.

In configuration file for log4j logger, you may set log directory and log file name, and you also may set medium in which logger will write to.

```
e.g. of configuration file for Log4j logger

<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE log4j:configuration SYSTEM "log4j.dtd">

<log4j:configuration xmlns:log4j='http://jakarta.apache.org/log4j/'>
    <appender name="ROLL" class="org.apache.log4j.RollingFileAppender">
        <param name="File" value="ObjectLoader/octopus.log"/>
        <param name="MaxFileSize" value="10MB"/>
        <param name="MaxBackupIndex" value="2"/>
        <layout class="org.apache.log4j.PatternLayout">
            <param name="ConversionPattern" value="%d{ISO8601}: %c : %m%n"/>
        </layout>
    </appender>
    <appender name="Console" class="org.apache.log4j.ConsoleAppender">
        <!-- <param name="Threshold" value="info"/> -->
        <layout class="org.apache.log4j.PatternLayout">
            <param name="ConversionPattern" value="%d{ISO8601}: %c : %m%n"/>
        </layout>
    </appender>

    <logger name="OctopusLogger" additivity="false">
        <level value="info"/>
        <appender-ref ref="Console"/>
        <appender-ref ref="ROLL"/>
    </logger>

    <root>
        <level value="debug"/>
        <appender-ref ref="ROLL"/>
        <appender-ref ref="Console"/>
   </root>

</log4j:configuration>
```

Note

If you want to use log4j logger you must include log4jlogger.jar file in application's classpath

User's logger

If you want to use your own logger to log Octopus application output, you must write new 'user' logger class.

This class must extend **Logger** class, which is placed in *org.webdocwf.util.loader.logging package*, and must implement all abstract methods from **Logger** class.

Note

Detail information about Logger class you can find in Java API for Octopus application

To be able to use this implementation of *Logger* class, you must include named implementation into Octopus application **classpath**, and you must set 'logger attributes' to appropriate values.

```
e.g.

<loadJob ... logClassName="org.user.logger.Logger" pathToLoggerConf="URL_to_logger_conf_file" ...>

</loadJob>
```

Whether you use ‘standard’, ‘log4j’ or some new logger, OctopusLoader will allow **only three** possible log modes. This mode defines which steps of the loading process will be written in log file.

Possible values are:

- '*none*' – writes only exceptions if they occur, and this mode return application exit code,
- '*normal*' - writes start and finish of the application and each process, job name, number of committed rows, duration of each job and number of import jobs (sql) will be notified also, and
- '*full*' - writes detailed information about the loading process into the log file.

The system default is 'normal'.

Data Cleaning

Data cleaning is advanced feature implemented in OctopusLoader application. Data Cleaning enables OctopusLoader application to work more efficiently. Data cleaning feature has several methods:

'If some table fields have value null'

This method enables OctopusLoader application to replace 'null' value with column's default value. If you want to use this method you must set the following attributes to appropriate values:

- *defaultValue* attribute in <valueColumn> tag must be set to appropriate value. If this attribute exists in this column, value 'null' in valueColumn will be replaced with the value of this attribute.
- *defaultValue* attribute in <userIDColumn> tag must be set to appropriate value. If this attribute exists in this column, value 'null' in userIDColumn will be replaced with the value of this attribute.
- *defaultValue* attribute in <constantColumn> tag must be set to appropriate value. If this attribute exists in this column, value 'null' in constantColumn will be replaced with the value of this attribute.
- *defaultValue* attribute in <variableColumn> tag must be set to appropriate value. If this attribute exists in this column, value 'null' in constantColumn will be replaced with the value of this attribute.

'Cutting off data'

If application tries to insert/update some value in table field, which has length greater than it is allowed, then that

value will be truncated to appropriate length. If you want to use this method you must set the following attributes to appropriate values:

- *dataCutOff* attribute in <loaderJob> tag must be set to true, or
- *dataCutOff* attribute in <importDefinition> tag must be set to true.

dataCutOff attribute in <importDefinition> tag has more relevance than the same attribute in <loaderJob> tag.

'Cut off data' method is related to <valueColumn>, <userIDColumn>, <variableColumn> and <constantColumn> tags.

'Cleaning foreign key values'

This method enables OctopusLoader application to replace invalid foreign key value with relations column's default value, or with 'null' value. If you want to use this method you must set the following attribute to appropriate value:

- *defaultValue* attribute in <relationColumn> tag must be set to appropriate value.

Default value represents an SQL statement that retrieves default value for relation column.

defaultValue statement has the following form:

```
SELECT [source_column_name] FROM [relation_source_table_name] WHERE [relation_key_source_column_name
```

```
e.g. of defaultValue statement
```

```
SELECT oid FROM PERSISTENTTREE WHERE KEYVALUE='HOME'
```

Log Table

If "data cleaning" feature is used, every event, which took place while loading process was going on, will be written in log file and in log table, if it is created.

Creating LOGTABLE

To create log table, please follow the steps described below, and set the named attributes to appropriate values:

***Step 1**:*

- in *logTableName* attribute of <loaderJob> tag, you specify the name of log table,
- in *logColumnName* attribute of <loaderJob> tag, you specify log table's column name,
- in *logRowNumber* attribute of <loaderJob> tag, you specify log table's column name,
- in *logOriginalValue* attribute of <loaderJob> tag, you specify log table's column name,
- in *logNewValue* attribute of <loaderJob> tag, you specify log table's column name,
- in *logImportDefinitionName* attribute of <loaderJob> tag, you specify log table's column name,
- in *logOperationName* attribute of <loaderJob> tag, you specify log table's column name,
- in *logTypeName* attribute of <loaderJob> tag, you specify log table's column name,

The meaning of log table attributes

logTableName attribute represents the name of log table,

logTable attribute represents the name of the currently processed table, and this attribute will be set automaticaly (not by user).

logColumnName attribute represents the name of the column, in which some replacements have been made,

logRowNumber attribute represents the number of the row in the table, in which some replacements have been made,

logOriginalValue attribute represents original value of data, which have been replaced,

logNewValue attribute represents new value of data, which are placed instead of old (original) value of data,

logImportDefinitionName attribute represents the name of 'importDefinition',

logOperationName attribute represents the name of the data cleaning methods (cut off, replaced null...),

logTypeName attribute represents the type of the OctopusLoader operation (insert, update...),

***Step 2**:*

In order to be able to use log table, you must create it first. Creation is done by writing sql statements and adding them in CreateTables.sql files.

```
e.g of creating the LogTable

create table LOGTABLE
 (
LOGIMPORTDEFINITIONNAME VARCHAR(254)    ,
LOGOPERATIONNAME VARCHAR(254)    ,
LOGTYPENAME VARCHAR(254)        ,
LOGTABLE VARCHAR(254)    ,
LOGCOLUMNNAME VARCHAR(254)    ,
LOGROWNUMBER VARCHAR(254)    ,
LOGORIGINALVALUE VARCHAR(254)    ,
LOGNEWVALUE VARCHAR(254),
LOGFAILEDSTATEMENT VARCHAR(254),
LOGTIME VARCHAR(254)
);
```

If you don't specify LogTable tag's, default values are used, and they are:

- LOGTABLENAME
- LOGCOLUMNNAME
- LOGROWNUMBER
- LOGORIGINALVALUE
- LOGNEWVALUE
- LOGIMPORTDEFINITIONNAME
- LOGOPERATIONNAME
- LOGTYPENAME

All LogTable attributes in loaderJob tag must match specified column names in sql statement for creating log table.

If the above mentioned attributes are not specified and you still want to use log table, specified column names in sql statement for creating log table must match default values for log table attributes.

Note

For more details about data cleaning see example DataCleaningExample in OctopusLoader application.

The LoaderJob.olj file

The LoaderJob.olj file is main file for Octopus application. In that file is placed crucial information needed for correct working of application.

The <loaderJob> root tag

The main XML file (LoaderJob.olj), for OctopusLoader application, has root tag <loaderJob>. This tag must be unique in corresponding XML file, and it has many attributes, which are described in the table below.

Table 5.2. Table of <loaderJob> tag attributes

attribute	default for	description	param
logMode	<importDefinition> and <sql>	defines the default logMode. Possible values are: "normal", "none" and "full". The system default is "normal".	-m
objectIDIncrement	<importDefinition>	Defines the incremental value for global unique objectID's as used by Enhydra DODS. Default is 20.	none
objectIDTableName	<importDefinition>	Defines the table name for global unique objectID's as used by Enhydra DODS. Default is "objectid".	none
objectIDColumnName	<importDefinition>	Defines the column name for global unique objectID's as used by Enhydra DODS. Default is "next".	none
objectIDNameColumnName	<importDefinition>	Defines the column name for OID name defined by TOS. If this value is set, Loader uses OID logic as used by TOS.	none
objectIDNameColumnValue	<importDefinition>	Defines the column name for OID values defined by TOS. If this value is set, Loader uses OID logic as used by TOS. Type of this column is VARCHAR.	none
onErrorContinue	<sql> and importDefini<tion>	Defines whether the load job should continue if an error in a SQL statement or import jobs occurs. Default is "false"	-e

commit	<sql>	Defines if SQL statement blocks should be committed separately. Default is "false"	none
userID	none	Defines the value for userID columns.	-u
logDir	none	Defines the target directory for the logfile. Default is the current working directory	-l
logFile	none	Defines the logfile name. Default is "LoaderLog-YYYY-MM-DD-HH-mm-ss.txt"	-f
vendorConfig	none	Defines the name of the DB vendor configuration XML file. Default is "OctopusDBVendors.xml"	-d
returnCode	none	Defines the default java.exe return code to the environment if some element of loadjob fails.	-rc
objectIDAutoCreate	<importDefinition>	Defines if the objectID table should be created automatically if not already present. Default is "false".	none
objectIDStartValue	<importDefinition>	Defines the start value for objectIDs ONLY if the objectID table is created automatically. Default is "1"	none
commitCount	<loaderJob>, importDefinition> and <copyTable>	Defines the default commit count. System default is "100"	-c
oidLogic	<importDefinition> and <table>	Defines if tables use the DODS oid logic. The system default is "true".	none
tableMode	<importDefinition> and <table>	Defines the default access method for tables. The system default is "Query".	none
dataCutOff	<importDefinition>	Defines if you want to use cut off methods of ‘data cleaning’ feature. The system default is "false".	none
logTableName	<loaderJob>	Defines if you want to use Log Table in data cleaning method . If you want to use LogTable, you must specify table name in this attribute.	none
logColumnName	<loaderJob>	Defines name of the column in the LogTable table. If you don't specify this name, default value is LOGCOLUMNNAME.	none

logRowNumber	<loaderJob>		Defines name of the column in the LogTable table. If you don't specify this name, default value is LOGROWNUMBER.	none
logOriginalValue	<loaderJob>		Defines name of the column in the LogTable table. If you don't specify this name, default value is LOGORIGINALVALUE.	none
logNewValue	<loaderJob>		Defines name of the column in the LogTable table. If you don't specify this name, default value is LOGNEWVALUE.	none
logImportDefinitionName	<loaderJob>		Defines name of the column in the LogTable table. If you don't specify this name, default value is LOGIMPORTDEFINITIONNAME.	none
logOperationName	<loaderJob>		Defines name of the column in the LogTable table. If you don't specify this name, default value is LOGOPERATIONNAME.	none
logTypeName	<loaderJob>		Defines name of the column in the LogTable. If you don't specify this name, default value is LOGTYPENAME.	none
logClassName	<loadJob>		represents the full name of logger class	none
pathToLoggerConf	<loadJob>		defines the URL to logger configuration file	none
setCursorName	<laoderJob>, <importDefinition>		Defines if we want to use setCursorName method	none
setFetchSize	<laoderJob>, <importDefinition>		Defines if we want to use setFetchSize method	none

Attributes of the loaderJob tag are overwritten by the corresponding parameters of the application/constructor!

```
e.g. of a <loaderJob> tag:
<loaderJob logMode="full" objectIDIncrement="5" objectIDTableName="myAdminObjectID" objectIDColumnNa
    onErrorContinue="false" commit="true" userID="r004d\meier" logDir="c:\mylogs" logFile="myFile.tx
    objectIDAutoCreate="true" objectIDStartValue="150000" commitCount="1000" oidLogic="false" tableM
    logTable="LOGTABLE">
...

</loaderJob>
```

LoaderJob element may have several sub elements: <restartCounter>, <variables>, <jdbcDefaultParameters>, <importDefinition>, <importDefinitionInclude>, <copyTable>, <sql> and <echo>

The <restartCounter> tag

Defines table and column name of the restart counter. This is an *optional* element/feature of the OctopusLoader. Jobs are only restartable if this element is specified. Additionally importDefinitions job must have a defined value of sort order to be able to restart.

<restartCounter> element has 4 attributes: tableName, importDefinitionColumnName, restartCounterColumnName and restartAutoCreate. All attributes are optional.

Table 5.3. Table of <restartCounter> tag attributes

attribute	description
tableName	Table name in target database where restart column value is. The system default is "LOADERRESTART".
importDefinitionColumnName	Column name in the tableName where the name of importDefinition is. The system default is "IMPORTDEFINITION".
restartCounterColumnName	Column name in the tableName where the value of restartCounter for determined importDefinition is. The system default is "RESTARTCOUNTER".
restartAutoCreate	The system default is "false".

Note

If a loaderJob is started without the "-r" parameter all entries in this table will be deleted.

```
e.g. of a restartCounter

<loaderJob ...>
   <restartCounter tableName="MyRestart" importDefinitionColumnName="MyName"
      restartCounterColumnName="MyCounter" restartAutoCreate="true"/>
   ...
</loaderJob>
```

The <variables> tag

This element together with it's subelements defines logical variables which are going to be used in constantColumns, SQL-statements, source-data and JDBC parameters.

Table 5.4. Table of <variables> tag attributes

attribute	description
prefix	Default value for attribute prefix in each <variable> tag.
suffix	Default value for attribute suffix in each <variable> tag.
override	Default value for attribute override in each <variable> tag.
replaceInConstants	Default value for attribute replaceInConstants in each <variable> tag.
replaceInSQL	Default value for attribute replaceInSQL in each vari-<able> tag.
replaceInData	Default value for attribute replaceInData in each vari-<able> tag.
replaceInJDBC	Default value for attribute replaceInJDBC in each vari-

able> tag.

The system defaults are: prefix="%" suffix="%" override="true" replaceInConstants="false" replaceInSQL="false" replaceInData="false" replaceInJDBC="false".

The <variable> tag

All occurrences of prefix+name+suffix will be replaced with value or the override value specified in the application/constructor parameters (if the override="true"). The replace will take place where replaceIn... attribute is equal "false".

The system defaults are: prefix="%" suffix="%" override="true" replaceInConstants="false" replaceInSQL="false" replaceInData="false" replaceInJDBC="false".

Table 5.5. Table of <variable> tag attributes

attribute	description	default value
name	See Note.	
value	See Note.	
prefix	See Note.	%
suffix	See Note.	%
override	See Note.	true
replaceInConstants	The replace take place in Constants.	false
replaceInSQL	The replaces take place in SQL.	false
replaceInData	The replaces take place in Data.	false
replaceInJDBC	The replaces take place in JDBC.	false
defaultValue	This attribute is used in DataCleaning method	optional

Note: prefix+name+suffix will be replaced with value or the override value specified in the application/constructor parameters (if the override="true"). name is required parameter.

```
e.g.
<loaderJob ...>
   ...
   <variables prefix="!" suffix="!" override="false" replaceInConstants="true" replaceInSQL="false"
      replaceInData="true" replaceInJDBC="true">
      <variable name="myvar1" value="myVal1" prefix="$" suffix="$" override="true" replaceInConstant
         replaceInSQL="true" replaceInData="false" replaceInJDBC="false"/>
      <variable name="myvar2" prefix="$" suffix="$"/>
      <variable name="myvar3" value="myVal3" defaultValue="myVal"/>
   </variables>
   ...
</loaderJob>
```

In this example the following will be done:

- All occurrences of "$myvar1$" will be replaced with "myVal1" or the override value specified in the application/constructor parameters. The replace will take place in SQL statements only.
- All occurrences of "$myvar2$" will be replaced with value specified in the application/constructor parameters. The replace will take place in constant columns, source data and JDBC parameters. If there is no value specified in the application/constructor parameters for this variable name, an exception will occur. A NULL value for my-

var2 can be specified by "-v myvar2=;myvar3=myval3"

- All occurrences of "!myvar3!" will be replaced with value "myVal3" which can NOT be overridden by parameters. Conflicts between variable definitions and parameters lead to a runtime exception ! The replace will take place in constant columns, source data and JDBC parameters.

The system defaults are: prefix="%" suffix="%" override="true" replaceInConstants="false" replaceInSQL="false" replaceInData="false" replaceInJDBC="false"

The <jdbcDefaultParameters> tag

This tag defines default JDBC parameters, which enable connection to source and target database. It can occur only once. <jdbcParameters> tag has two child tags: <jdbcSourceParameters> and <jdbcTargetParameters>. This tag doesn't have any attribute.

The <jdbcSourceParameters> tag

<jdbcSourceParameters> tag defines JDBC source parameters. This tag may occur only once. This tag has two attributes - dbVendor and driverName. dbVendor attribute is name of database or database type, driverName attribute is the name of used driver. Depending on that value, OctopusLoader reads driver parameters for that database type from OctopusLoader configuration file.

Possible values for attribute dbVendor as well as possible source database types are : mssql (MSSQL server), mysql (MySlq database), excel (Source table in Excel format), cvs (separators: ';' or ',')), access (Access database), PostgreSQL, Qed, instantDB, Hsqldb(HSQL database), Oracle, JDataStore (BorlandJDataStore database), XML and DB2, and McKoi. Default value of dbVendor is csv. Default value of driverName is csv too. Child element of this tag is <jdbcSourceParameter>.

The <jdbcSourceParameter> tag

<jdbcSourceParameter> tag defines JDBC parameters: JDBC driver, connection URL, user and password. This tag occurs at least once (for connection URL). <jdbcSourceParameter> tag for JDBC driver is optional and its default values OctopusLoader reads from configuration file. User and password are optional.

Table 5.6. Table of <jdbcSourceParameter> tag attributes (default parameters)

attribute	description
name	Name of the JDBC parameters for source database. Possible values are: JdbcDriver, Connection.Url, User and Password.
value	Value of the appropriate parameter, depends on the value of the attribute name: -if name="JdbcDriver", attribute value is JDBC driver; -if name="Connection.Url", attribute value is connection URL, -if name="User", attribute value is user name, tag with this attribute name is optional, -if name="Password", attribute value is password, tag with this attribute name is optional.

Note

If the named database is on file system (e.g. c:\Users\Database), connection url should be relative from LoaderJob.olj file of should be absolute.

The <jdbcTargetParameters> tag

<jdbcTargetParameters< tag defines JDBC target parameters. This tag may occur only once. This tag has two optional attributes - dbVendor and driverName. dbVEndor attribute is name of database or database type, driverName. Depending on that value Octopus reads driver parameters for that database type from Octopus configuration file.

Possible values for attribute dbVendor as well as possible source database types are : mssql (MSSQL server), mysql (MySlq database), excel (Source table in Excel format), cvs (separators: ';' or ',')), access (Access database), PostgreSQL, Qed, instantDB, Hsqldb(HSQL database), Oracle, JDataStore (BorlandJDataStore database), DB2, McKoi, Sybase, Paradox. Default value of dbVendor is mssql. Default value of driverName is jTurbo. Child element of this tag is <jdbcTargetParameter>>.

The <jdbcTargetParameter> tag

<jdbcTargetParameter> tag defines JDBC parameters: JDBC driver, connection URL, user and password. This tag occurs at least once (for connection URL). JdbcTargetParameter for JDBC driver is optional and its default values OctopusLoader reads from configuration file. User and password are optional.

If there are no values for user and password, application is automatically asking for those values later.

Table 5.7. Table of <jdbcTargetParameter> tag attributes (default parameters)

attribute	description
name	Name of the JDBC parameters for target database. Possible values are: JdbcDriver, Connection.Url, User and Password.
value	Value of the appropriate parameter, depends on the value of the attribute name: -if name="JdbcDriver", attribute value is JDBC driver, -if name="Connection.Url", attribute value is connection URL, -if name="User", attribute value is user name, tag with this attribute name is optional, -if name="Password", attribute value is password, tag with this attribute name is optional.

Note

If the named database is on file system (e.g. c:\Users\Database), connection url should be relative from LoaderJob.olj file of should be absolute.

```
e.g. of JDBC default parameters

<loaderJob ...>
   ...
   <jdbcDefaultParameters>
      <jdbcSourceParameters>
         <jdbcSourceParameter name="JdbcDriver" value="org.myDriver.JDBCDriver"/>
         <jdbcSourceParameter name="Connection.Url" value="jdbc:myDriver:MySourceDatabase"/>
         <jdbcSourceParameter name="User" value="myDBUserName"/>
         <jdbcSourceParameter name="Password" value="myDBPassword"/>
      </jdbcSourceParameters>
      <jdbcTargetParameters>
         <jdbcTargetParameter name="JdbcDriver" value="org.myDriver2.JDBCDriver2"/>
         <jdbcTargetParameter name="Connection.Url" value="jdbc:myDriver2:MyTargetDatabase"/>
         <jdbcTargetParameter name="User" value="myDBUserName2"/>
         <jdbcTargetParameter name="Password" value=""/>
      </jdbcTargetParameters>
   </jdbcDefaultParameters>
   ...
</loaderJob>
```

If there is no specified password you will be asked for it during runtime.

All values can use the 'variable-replace' functionality defined in the variables tag.

The <sql> tag

This tag defines SQL statements executing in target database. Target database parameters are in the jdbcTargetPa-<rameters> tag.

<sql> tag is optional, and there could be more in loaderJob.

<sql> tag has four attributes: name, logMode, onErrorContinue and commit; and it also has two child tags: jdbc-<TargetParameters> and <sqlStmt>.

Table 5.8. Table of <sql> tag attJributes

attribute	description
name	Name of the <sql> tag. This attribute is required.
logMode	This attribute is optional. This mode defines which steps of the loading process will be written in log file. Possible values: *none*, *normal* and *full*. -'**none**' – writes only exceptions if they occur, and this mode return application exit code.; -'**normal**' - writes start and finish of the application and each importDefinition process, importDefinition name, number of committed rows, duration of each job and number of import jobs (sql) will be notified also; -'**full**' - writes detailed information about the loading process into the log file. The system default is 'normal'.
onErrorContinue	Possible values are **true** or **false.** When value is true, if an error occurs, error message will be shown and applications will continue. When value is false, if an error occurs, error message will be shown and application will be finished. This attribute is optional. The system default is false.
commit	Possible values are **true** or **false**. If value is true, SQL statement will be committed immediately after their execution, else it will be committed at the end of the application. This attribute is optional. The system default is true.
returnCode	Represents the default java.exe return code to the environment if some element of loaderJob fails. This attribute is optional. The system default is 1.

```
e.g. of the <sql> tag

<loaderJob ...>
   ...
   <sql logMode="none" onErrorContinue="true" commit="true">
      <jdbcTargetParameters>
         <jdbcTargetParameter name="JdbcDriver" value="org.myDriver2.JDBCDriver2"/>
         <jdbcTargetParameter name="Connection.Url" value="jdbc:myDriver2:MyTargetDatabase"/>
         <jdbcTargetParameter name="User" value="myDBUserName2"/>
         <jdbcTargetParameter name="Password" value=""/>
      </jdbcTargetParameters>
      <sqlStmt>
         CREATE DATABASE Together;
```

```
         CREATE TABLE tab1 (col1 varchar(250));
      </sqlStmt>
   </sql>
   ...
   <sql logMode="full" onErrorContinue="false" commit="false">
      <jdbcTargetParameters>
         <jdbcTargetParameter name="JdbcDriver" value="org.myDriver2.JDBCDriver2"/>
         <jdbcTargetParameter name="Connection.Url" value="jdbc:myDriver2:MyTargetDatabase"/>
         <jdbcTargetParameter name="User" value="myDBUserName2"/>
         <jdbcTargetParameter name="Password" value=""/>
      </jdbcTargetParameters>
      <sqlStmt>
         <include href="c:\mySQLFile1" parse="text"/>
      </sqlStmt>
   </sql>
   ...
</loaderJob>
```

The <jdbcTargetParameters> tag

<jdbcTargetParameters> defines JDBC source parameters. This tag may occur only once. This tag has two optional attributes - dbVendor and driverName. dbVEndor is name of database or database type, driverName.

Depending of that value Octopus reads driver parameters for that database type from Octopus configuration file - OctopusDBVendors.xml. Possible values for attribute dbVendor as well possible source database types are : mssql (MSSQL server), mysql (MySlq database), excel (Source table in Excel format), cvs (separators: ';' or ',')), access (Access database), PostgreSQL, Qed, instantDB, Hsqldb(HSQL database), Oracle, JDataStore (BorlandJDataStore database), XML, DB2, Sybase, Paradox.

Default value of dbVendor is mssql. Default value of driverName is jTurbo. Child element of this tag is jdbcTar-<getParameter>.

The <jdbcTargetParameter> tag

<jdbcTargetParameter> tag defines JDBC parameters: JDBC driver, connection URL, user and password. This tag occurs at least once (for connection URL). JdbcTargetParameter for JDBC driver is optional and its default values Octopus reads from OctopusDbVendors.xml configuration file. User and password are optional. If there are not values for user and password, application is automatically asking for those values later.

Table 5.9. Table of <jdbcTargetParameter> tag attributes (<sql> tag)

attribute	description
name	Name of the JDBC parameters for target database. Possible values are: JdbcDriver, Connection.Url, User and Password.
value	Value of the appropriate parameter, depends on the value of the attribute name: -if name="JdbcDriver", attribute value is JDBC driver, -if name="Connection.Url", attribute value is connection URL, -if name="User", attribute value is user name, tag with this attribute name is optional; -if name="Password", attribute value is password, tag with this attribute name is optional.

Note

If the named database is on file system (e.g. c:\Users\Database), connection url should be relative from LoaderJob.olj file of should be absolute.

The <sqlStmt> tag

<sqlStmt> defines SQL Statement. This tag is required, and it may occur only once. <sqlStmt> tag doesn't have any attribute. It has text (sql statement). Child of this tag is <include> tag.

The <include> tag

<include> tag defines file to be included. This tag is optional, and it may occur only once. <include> tag has two attributes: href and parse. This tag doesn't have any child tag.

Table 5.10. Table of <include> tag attributes (<sqlStmt> tag)

attribute	description
href	path to file which is going to be included. This path should be relative form LoaderJob.olj file or should be absolute.
parse	file's type (xml or text)

If parse = "xml", OctopusLoader substitutes included element with another XML file.

If parse = "text", Octopus substitutes included element with another text file with SQL commands.

```
e.g of the <sqlStmt> tag:

<sqlStmt>
    CREATE DATABASE Together;
</sqlStmt>

  or:

<sqlStmt>
    <include href="SQLproba.sql" parse="text"/>
</sqlStmt>
```

The <definitionInclude> tag

<definitionInclude> tag defines another xml file with import definitions to be included. This tag is *optional*, and it may occur more than once.

<definitionInclude> tag has one child element (<include>) with two attributes: *href* and *parse*. Tag <include> is required and may occur only once.

The <include> tag

<include> tag defines file to be included. This tag is required, and it may occur only once. <include> tag has two attributes: href and parse. This tag doesn't have any child tag.

Table 5.11. Table of <include> tag attributes (<definitionInclude> tag)

attribute	description
href	path to file which is going to be included. This path should be relative form LoaderJob.olj file or should be absolute.
parse	file's type (xml or text)

```
e.g of the definitionInclude tag

<loaderJob ...>
   ...
   <definitionInclude>
      <include href="c:\MyImportDefinitions.oli" parse="xml"/>
   </definitionInclude>
   ...
</loaderJob>
```

The included file must have the following structure/content:

```
e.g. of an included definition file

<definitionInclude>
   <importDefinition...>
   ...
   </importDefinition...>
   <importDefinition...>
   ...
   </importDefinition...>
...
</definitionInclude>
```

The <echo> tag

Echo element is implemented in parallel with importDefinitions and sql's. This element has only one attribute: message. This attribute is required.

OctopusLoader writes this messages in log output (screen and log file) in logMode 'normal' and logMode 'full'.

```
e.g of an echo element

<echo message ="Log message..."/>
```

The <copyTable> tag

CopyTable element is implemented into OctopusLoader application in order to enable simple copying of one named table from source database to target database.

Table 5.12. Table of <copyTable> tag attributes

attribute	description
name	Name of the copyTable job. The name attribute of copyTable must be unique within one loaderJob in order to make the restart functionality work correctly. This attribute is **required**.
sourceTableName	Defines name of the source table which will be copied into target table.
targetTableName	Defines name of the target table
logMode	This mode defines which steps of the loading process will be written in log file. Possible values: none (Loader.LOGMODE_NONE), normal (Loader.LOGMODE_NORMAL) and full (Loader.LOGMODE_FULL). -'**none**' - writes only exceptions if they occur, and this mode return application exit code.; -'**normal**' - writes start and finish of the application and each copyTable process, copyTable job name, number of committed rows, duration of each job

	and number of import jobs (sql) will be notified also; -'**full**' - writes detailed information about the loading process into the log file. For 'normal' and 'full' modes, time start/end and the name of the job will be written. This attribute is optional. The system default is Loader.LOGMODE_NORMAL
ObjectIDIncrement	This number defines the OID increment value. This attribute is optional and the default is 20.
oidLogic	Defines the default if tables use the DODS oid logic. The system default is "true".
onErrorContinue	Defines if the load job should continue if an error in a copyTable job occurs. Default is "false"
commitCount	Number of rows in source table after which the target database will be committed. This attribute is optional. If the attribute doesn't exist, the default (from loaderJob) will be used.
setCursorName	Defines if the setCursorName method is going to be used in this job.
setFetchSize	Defines if the setFetchSize method is going to be used in this job.

```
e.g. of the <copyTable> tag

<copyTable logMode="normal" name="PERSISTENTTREE" objectIDIncrement="20" oidLogic="false"
onErrorContinue="false" sourceTableName="PERSISTENTTREE" targetTableName="PERSISTENTTREE"/>
```

The <importDefinition> tag

<importDefinition> tag defines loading process. It is parent tag to other tags connected with the same loading process. It is *optional* tag.

<importDefinition> tag has 8 child tags: <sortColumns>, <jdbcParameters>, <valueColumns>, <variableColumns>, <relationColumns>, <constantColumns>, <counterColumns> and <tables>.

Table 5.13. Table of <importDefinition> tag attributes

attribute	description
name	Name of the import definition. The name attribute of importDefinitions must be unique within one loaderJob in order to make the restart functionality work correctly. This attribute is **required**.
tableName	Name of the source table in the source database. This attribute is optional. If it not exists, you must set selectStatement for loading data using SQL statement
selectStatement	SQL statement for loading data from source tables. This attribute is optional. Required if table name doesn't exist.
commitCount	Number of rows in source table after which the target database will be committed. This attribute is optional. If the attribute doesn't exist, the default (from loaderJob) will be used.
logMode	This mode defines which steps of the loading process will be written in log file. Possible values: none (Loader.LOGMODE_NONE), normal

	(Loader.LOGMODE_NORMAL) and full (Loader.LOGMODE_FULL). -'**none**' - writes only exceptions if they occur, and this mode return application exit code.; -'**normal**' - writes start and finish of the application and each importDefinition process, importDefinition name, number of committed rows, duration of each job and number of import jobs (sql) will be notified also; '**full**' - writes detailed information about the loading process into the log file. For 'normal' and 'full' modes, time start/end and the name of the job will be written. This attribute is optional. The system default is Loader.LOGMODE_NORMAL
objectIDIncrement	This number defines the OID increment value. This attribute is optional and the default is 20.
objectIDTableName	Defines table name where is oid number. This attribute is optional. Default value is objectid.
objectIDColumnName	Defines column name where is oid number. This attribute is optional. Default value is 'next'.
objectIDNameColumnName	Defines the column name for OID name defined by TOS. If this value is set, Loader uses OID logic as used by TOS. This attribute is optional.
objectIDNameColumnValue	Defines the column name for OID values defined by TOS. If this value is set, Loader uses OID logic as used by TOS. Type of this column is VARCHAR. This attribute is optional.
returnCode	Defines the default java.exe return code to the environment if some element of loadjob fails.
dataCutOff	Defines if we want to use 'data cut off' methods of 'Data Cleaning' feature.
onErrorContinue	Value is true or false. When value is true, if an error occurs, error message is shown (log file and log table) and application continues. When value is false, if an error occurs, error message is shown (log file and log table) and application will be finished. This attribute is optional. The system default is false.
setCursorName	Defines if the setCursorName method is going to be used in this job.
setFetchSize	Defines if the setFetchSize method is going to be used in this job.

e.g of the <importDefinition> tag

```
<loaderJob ...>
   ...
   <importDefinition name="MyDef1" tableName="MySourceTable1" logMode="full" objectIDIncrement="2" o
      objectIDColumnName="MyNext" objectIDAutoCreate="true" objectIDStartValue="500" commitCount="10
      dataCleaning="true" onErrorContinue="true" dataCutOff="true">
      <sortColumns>
         <sortColumn entry="ColName1"/>
         <sortColumn entry="ColName2"/>
      </sortColumns>
      <jdbcParameters>
         <jdbcSourceParameters>
            <jdbcSourceParameter name="JdbcDriver" value="org.myDriver.JDBCDriver"/>
            <jdbcSourceParameter name="Connection.Url" value="jdbc:myDriver:MySourceDatabase"/>
            <jdbcSourceParameter name="User" value="myDBUserName"/>
            <jdbcSourceParameter name="Password" value="myDBPassword"/>
```

```
      </jdbcSourceParameters>
      <jdbcTargetParameters>
         <jdbcTargetParameter name="JdbcDriver" value="org.myDriver2.JDBCDriver2"/>
         <jdbcTargetParameter name="Connection.Url" value="jdbc:myDriver2:MyTargetDatabase"/>
         <jdbcTargetParameter name="User" value="myDBUserName2"/>
         <jdbcTargetParameter name="debug" value="on"/>
      </jdbcTargetParameters>
   </jdbcParameters>
   <valueColumns>
      <valueColumn sourceColumnName="Col19" targetTableName="TargetTab1" targetTableID="0"
         targetColumnName"Col17" valueMode="Key"/>
      <valueColumn sourceColumnName="Col20" targetTableName="TargetTab1" targetTableID="0"
         targetColumnName"Col18" valueMode="Update"/>
      <valueColumn sourceColumnName="Col21" targetTableName="TargetTab1" targetTableID="0"
         targetColumnName"Col19" valueMode="Overwrite"/>
      <valueColumn sourceColumnName="Col22" targetTableName="TargetTab1" targetTableID="0"
         targetColumnName"Col20" valueMode="SetIfCreated"/>
      <valueColumn sourceColumnName="Col22" targetTableName="TargetTab1" targetTableID="0"
         targetColumnName"Col21" valueMode="SetNull"/>
      <valueColumn sourceColumnName="Col31" targetTableName="TargetTab2" targetTableID="0"
         targetColumnName"Col17" valueMode="Key"/>
      <valueColumn sourceColumnName="Col32" targetTableName="TargetTab3" targetTableID="0"
         targetColumnName"Col18" valueMode="Key"/>
      <valueColumn sourceColumnName="Col33" targetTableName="TargetTab4" targetTableID="0"
         targetColumnName"Col19" valueMode="Key"/>
      <valueColumn sourceColumnName="Col34" targetTableName="TargetTab5" targetTableID="0"
         targetColumnName"Col20" valueMode="Key"/>
      <valueColumn sourceColumnName="Col35" targetTableName="TargetTab5" targetTableID="1"
         targetColumnName"Col21" valueMode="Key"/>
   </valueColumns>
   <variableColumns>
      <variableColumn name="var1" targetTableName="TargetTab1" targetTableID="0"
         targetColumnName"Col11" valueMode="Key"/>
      <variableColumn name="var2" targetTableName="TargetTab1" targetTableID="0"
         targetColumnName"Col12" valueMode="Update"/>
      <variableColumn name="var3" targetTableName="TargetTab1" targetTableID="0"
         targetColumnName"Col13" valueMode="Overwrite"/>
      <variableColumn name="var4" targetTableName="TargetTab1" targetTableID="0"
         targetColumnName"Col14" valueMode="SetIfCreated"/>
      <variableColumn name="var5" targetTableName="TargetTab1" targetTableID="0"
         targetColumnName"Col15" valueMode="SetNull"/>
      <userIDColumn targetTableName="TargetTab1" targetTableID="0" targetColumnName"Col22"
         valueMode="Key"/>
      <userIDColumn targetTableName="TargetTab1" targetTableID="0" targetColumnName"Col23"
         valueMode="Update"/>
      <userIDColumn targetTableName="TargetTab1" targetTableID="0" targetColumnName"Col24"
         valueMode="Overwrite"/>
      <userIDColumn targetTableName="TargetTab1" targetTableID="0" targetColumnName"Col25"
         valueMode="SetIfCreated"/>
      <userIDColumn targetTableName="TargetTab1" targetTableID="0" targetColumnName"Col26"
         valueMode="SetNull"/>
      <timeStampColumn targetTableName="TargetTab1" targetTableID="0" targetColumnName"Col32"
         valueMode="Key"/>
      <timeStampColumn targetTableName="TargetTab1" targetTableID="0" targetColumnName"Col33"
         valueMode="Update"/>
      <timeStampColumn targetTableName="TargetTab1" targetTableID="0" targetColumnName"Col34"
         valueMode="Overwrite"/>
      <timeStampColumn targetTableName="TargetTab1" targetTableID="0" targetColumnName"Col35"
         valueMode="SetIfCreated"/>
   </variableColumns>
   <constantColumns>
      <constantColumn constantValue="myVal%var12345%" targetTableName="TargetTab1"
         targetTableID="0" targetColumnName"Col47" valueMode="Key"/>
      <constantColumn constantValue="myVal2" targetTableName="TargetTab1" targetTableID="0"
         targetColumnName"Col48" valueMode="Update"/>
      <constantColumn constantValue="myVal3" targetTableName="TargetTab1" targetTableID="0"
         targetColumnName"Col49" valueMode="Overwrite"/>
      <constantColumn constantValue="myVal4" targetTableName="TargetTab1" targetTableID="0"
         targetColumnName"Col50" valueMode="SetIfCreated"/>
      <constantColumn targetTableName="TargetTab1" targetTableID="0" targetColumnName"Col51"
         valueMode="Overwrite"/>
      <constantColumn targetTableName="TargetTab1" targetTableID="0" targetColumnName"Col52"
```

```
            valueMode="SetIfCreated"/>
         </constantColumns>
         <counterColumns counterTableName="counterTable" counterNameColumn=Col01 counterValueColumn="va
            <counterColumn counterName="counter1" counterStartValue="100" counterIncrement="1" targetTa
               targetTableID="0" targetColumnName"Col88" valueMode="SetIfCreated" counterStartValueRese
            <counterColumn counterName="counter2" counterStartValue="1" counterIncrement="10" targetTab
               targetTableID="0" targetColumnName"Col89" valueMode="SetIfCreated" counterStartValueRese
            <subCounterColumn constantValue="myVal3" targetTableName="TargetTab1" targetTableID="0"
               targetColumnName="Col49" valueMode="Overwrite"/>
               <subCounterKeyColumn keyColumnName="Col99"/>
            </subCounterColumn>
         </counterColumns>
         <relationColumns>
            <relationColumn relationSourceTableName="TargetTab2" relationSourceTableID="0"
               relationSourceColumnName="Key21"  targetTableName="TargetTab1" targetTableID="0"
               targetColumnName"Col61" relationMode="Key"/>
            <relationColumn relationSourceTableName="TargetTab2" relationSourceTableID="0"
               relationSourceColumnName="Key22"  targetTableName="TargetTab1" targetTableID="0"
               targetColumnName"Col62" relationMode="Key"/>
            <relationColumn relationSourceTableName="TargetTab3" relationSourceTableID="0"
               targetTableName="TargetTab1" targetTableID="0"
               targetColumnName"Col63" relationMode="Update"/>
            <relationColumn relationSourceTableName="TargetTab4" relationSourceTableID="0"
               targetTableName="TargetTab1" targetTableID="0"
               targetColumnName"Col64" relationMode="Overwrite"/>
            <relationColumn relationSourceTableName="TargetTab5" relationSourceTableID="0"
               targetTableName="TargetTab1" targetTableID="0"
               targetColumnName"Col65" relationMode="SetIfCreated"/>
            <relationColumn relationSourceTableName="TargetTab5" relationSourceTableID="1"
               targetTableName="TargetTab1" targetTableID="0"
               targetColumnName"Col66" relationMode="SetNull"/>
         </relationColumns>
         <tables>
            <table tableName="TargetTab1" tableID="0" insert="true" tableMode="Query" oidLogic="true"/>
            <table tableName="TargetTab2" tableID="0" insert="false" tableMode="Cache" oidLogic="false"
            <table tableName="TargetTab3" tableID="0" insert="false" tableMode="Cache" oidLogic="true"/
            <table tableName="TargetTab4" tableID="0" insert="false" tableMode="Cache" oidLogic="true"/
            <table tableName="TargetTab5" tableID="0" insert="false" tableMode="Cache" oidLogic="true"/
            <table tableName="TargetTab5" tableID="1" insert="false" tableMode="Cache" oidLogic="true"/
         </tables>
      </importDefinition>
      ...
</loaderJob>
```

The <sortColumns> tag

<sortColumns> tag defines columns of the source table which are used as a sort criteria. This is child tag of the im-<portDefinition> tag. The <sortColumn> tag is optional, and it may occur only once. It has one child tag: sortCol-<umn>.

The <sortColumn> tag

<sortColumn> tag defines column of source table, which is used for sorting source data. There could be more sort-<Column> tags. It has one attribute: entry, and it hasn't any child tag.

Table 5.14. Table of <sortColumn> tag attributes

attribute	description
entry	column name which will be used for sort column in the loading process

```
e.g of the <sortColumn> tag

<sortColumn entry="KeyValue"/>
```

Also, OctopusLoader checks if sort columns in source table have unique values. If source table has more rows in sort columns with equal values OctopusLoader throws an Exception.

The <jdbcParameters> tag

<jdbcParameters> tag defines JDBC parameters which store information needed for connection to source and target database. It can occur only once.

<jdbcParameters> tag has two child tags: <jdbcSourceParameters> and <jdbcTargetParameters>. This tag doesn't have any attribute.

The <jdbcSourceParameters> tag

<jdbcSourceParameters> tag defines JDBC source parameters. This tag may occur only once. This tag has two attributes - dbVendor and driverName. dbVEndor is name of database or database type, driverName is the name of driver. Depending on that value OctopusLoader reads driver parameters for that database type from Octopus configuration file.

Possible values for attribute dbVendor as well as possible source database types are : mssql (MSSQL server), mysql (MySlq database), excel (Source table in Excel format), Csv (separators: ';' or ',')), access (Access database), PostgreSQL, Qed, instantDB, Hsqldb(HSQL database), Oracle, JDataStore (BorlandJDataStore database),DB2 and McKoi.

Default value of dbVendor is Csv. Default value of driverName is csv too. Child element of this tag is jdbcSour-<ceParameter>.

The <jdbcSourceParameter> tag

<jdbcSourceParameter> tag defines JDBC parameters: JDBC driver, connection URL, user and password. This tag occurs at least once (for connection URL).

JdbcSourceParameter for JDBC driver is optional. User and password are optional, too.

Table 5.15. Table of <jdbcSourceParameter> tag attributes (<jdbcParameters> tag)

attribute	description
name	Name of the JDBC parameters for source database. Possible values are: JdbcDriver, Connection.Url, User and Password.
value	Value of the appropriate parameter, depends on the value of the attribute name: -if name="JdbcDriver", attribute value is JDBC driver; -if name="Connection.Url", attribute value is connection URL; -if name="User", attribute value is user name, tag with this attribute name is optional; -if name="Password", attribute value is password, tag with this attribute name is optional.

Note

If the named database is placed on file system (e.g. c:\Users\Database), connection url should be relative from LoaderJob.olj file of should be absolute.

```
e.g. of the <jdbcSourceParameter> tag

<jdbcSourceParameter name="JdbcDriver" value="sun.jdbc.odbc.JdbcOdbcDriver"/>
```

```
<jdbcSourceParameter name="Connection.Url" value="jdbc:odbc:Loader"/>
<jdbcSourceParameter name="User" value="sa"/>
<jdbcSourceParameter name="Password" value="as"/>
```

The <jdbcTargetParameters> tag

<jdbcTargetParameters> tag defines JDBC source parameters. This tag may occur only once. This tag has two optional attributes - dbVendor and driverName. dbVEndor is name of database or database type, driverName is the name of driver. Depending of that value Octopus reads driver parameters for that database type from Octopus configuration file.

Possible values for attribute dbVendor as well possible source database types are : mssql (MSSQL server), mysql (MySlq database), excel (Source table in Excel format), cvs (separators: ';' or ',')), access (Access database), PostgreSQL, Qed, instantDB, Hsqldb(HSQL database), Oracle, JDataStore (BorlandJDataStore database), DB2, McKoi, Sybase, Paradox.

Default value of dbVendor is mssql. Default value of driverName is jTurbo. Child element of this tag is jdbcTar-<getParameter>.

The <jdbcTargetParameter> tag

<jdbcTargetParameter> tag defines JDBC parameters: JDBC driver, connection URL, user and password. This tag occurs at least once (for connection URL). JdbcTargetParameter for JDBC driver is optional and its default values Octopus reads from configuration file. User and password are optional.

Table 5.16. Table of <jdbcTargetParameter> tag attributes (<jdbcParameters> tag)

attribute	description
name	Name of the JDBC parameters for target database. Possible values are: JdbcDriver, Connection.Url, User and Password.
value	Value of the appropriate parameter, depends on the value of the attribute name: -if name="JdbcDriver", attribute value is JDBC driver; -if name="Connection.Url", attribute value is connection URL; -if name="User", attribute value is user name, tag with this attribute name is optional; -if name="Password", attribute value is password, tag with this attribute name is optional.

Note

If the named database is placed on file system (e.g. c:\Users\Database), connection url should be relative from LoaderJob.olj file of should be absolute.

`e.g. of the <jdbcTargetParameter> tag`

```
<jdbcTargetParameter name="JdbcDriver" value="sun.jdbc.odbc.JdbcOdbcDriver"/>
<jdbcTargetParameter name="Connection.Url" value="jdbc:odbc:Loader"/>
<jdbcTargetParameter name="User" value="sa"/>
<jdbcTargetParameter name="Password" value="as"/>
```

The <valueColumns> tag

<valueColumns> tag defines connections between source and target columns in source and target databases. This tag has child tag <valueColumn>. This tag is optional.

The <valueColumn> tag

<valueColumn> tag defines the name of columns in source and target table as well as value mode between them. This tag must occur one or more times.

Table 5.17. Table of <valueColumn> tag attributes

attribute	description
sourceColumnName	The name of the column in table in source database. This attribute is **required**.
targetTableName	The name of the table in target database. This attribute is **required**.
targetColumnName	The name of the column in table in target database. This attribute is **required**.
targetTableID	This is the number that defines logical target table. It is possible to have more logical target tables than actual target tables. This attribute can take values: 0, 1, 2, ... This attribute is required. This attribute defines more logical tables at the same physical table (use same table several times).
valueMode	Defines updating mode. Possible modes are: Key, Overwrite, Update, SetIfCreated and SetNull. This attribute is **required**. 'Key mode' - defines key column of the target table used for selection of rows; 'Overwrite mode' - overwrites the value of the target column in appropriate row or insert the value in a new row; 'Update mode' - updates the value in empty column of the appropriate row; 'SetIfCreated mode' - the value is written only if a new row is inserted; 'SetNull mode' – the value is written only if the new value is NULL.
defaultValue	This tag is optional. If we want to use 'If some value of table fields is null' method of 'Data Cleaning' advanced features, we must set this tag to appropriate value.
toHex	Possible values are **true** or **false**. If toHex attribute is set to true, BLOB object in this column will be transformed into String object.
fromHex	Possible values are **true** or **false**. If fromHex attribute is set to true, String object in this column will be transformed into BLOB object.

```
e.g. of the <valueColumn> tag

<valueColumn sourceColumnName="LONGNAME" targetTableName="PERSISTENTTREETYPE" targetColumnName="LONG
 targetTableID="0" valueMode="Overwrite"/>
<valueColumn sourceColumnName="KEYVALUE" targetTableName="PERSISTENTTREETYPE" targetColumnName="KEYV
 targetTableID="0" valueMode=" Key"/>
<valueColumn sourceColumnName="DATASTRING01DESC" targetTableName="PERSISTENTTREETYPE" targetColumnNa
 targetTableID="0" valueMode="Update"/>
<valueColumn sourceColumnName="XMLNAME" targetTableName="PERSISTENTTREETYPE"
 targetColumnName="XMLNAME" targetTableID="0" valueMode="SetIfCreated" defaultValue="defColVal"/>
```

The <transformations> tag

<transformations> tag defines all transformations that culd be used to transform data in this import definition section.

Child tag is: <transformation>.

The <transformation> tag

This tag defines transformation which will be executed on defined source column(s), and also target table and column(s) in which transformed data will be stored.

Table 5.18. Table of <transformations> tag attributes

attribute	description
name	Name of transformation
transformatorClassName	Name of class which will transform data. (eg. org.webdocwf.util.loader.TransformMyData)
transformatorConfig	Parameter which can be send to class TransformMyData

Child tags are:<sourceColumns> - which contains source column names that will be transformed,<targetColumns> - target table names in which transformed data will be inserted.

Child tag for <sourceColumns> is <sourceColumn>. Tag <sourceColumn> has attribute "name" which is name of source column from table defined for this <importDefinition> tag. Also this this source column name can not be defined in <valueColumns> tag for this <importDefinition>.

Child tag for <targetColumns> is <targetColumn>.

Table 5.19. Table of <targetColumn> tag attributes

attribute	description
name	Name of target column
tableName	Name of target table
tableID	Logical ID for this table
valueMode	Value mode for transformations (Update,SetNull,Overwrite,Key)

The <variableColumns> tag

<variableColumns> tag defines target columns with current time value and user ID value. The updating mode is defined too. This tag is *optional*.

Child tags are: <variableColumn>, <userIDColumn> and <timeStampColumn>.

The <variableColumn> tag

<variableColumn> tag defines variables with parameter. This tag is optional. The Octopus parameter -v (defines variables) and variableColumn tag is required for using this option.

Table 5.20. Table of <variableColumn> tag attributes

attribute	description
name	Name of variable column. This value (value of the at-

	tribute name) will be replaced with value of the Loader class constructor parameter 'nameValue'.
targetTableName	Defines target table name. This attribute is **required**.
targetColumnName	Defines column name in target table. This attribute is **required**.
targetTableID	Defines logical target table. It is possible to have more logical target tables than physical target tables. This attribute can take values: 0, 1, 2, ... This attribute is required. This attribute defines more logical tables at the same physical table (uses the same table several times).
valueMode	Defines updating mode. Possible modes are: Overwrite, Update, SetIfCreated and SetNull. This attribute is required. ‘Overwrite mode’ - overwrites the value of the target column in appropriate row or insert the value in a new row; ‘Update’ mode - updates the value in empty column of the appropriate row; ‘SetIfCreated’ mode - the value is only written if a new row is inserted; ‘SetNull’ mode - value is only written if the new value is NULL.
defaultValue	This tag is optional. If we want to use 'If some value of table fields is null' metod of 'Data Cleaning' advanced features, we must set this tag to appropriate value.

After replacing values (in attribute 'name') OctopusLoader processes this tag as usual value columns.

```
e.g. of the <variableColumn> tag
```

```
<variableColumn name="-v_name" targetTableName="Person"
 targetColumnName="PersHobby" targetTableID="0" valueMode="Overwrite" defaultValue="defVal"/>
```

The <userIDColumn> tag

<userIDColumn> tag defines a table and column in the target database where the value of the Octopus argument "userID" is. This tag is *optional*.

Table 5.21. Table of <userIDColumn> tag attributes

attribute	description
targetTableName	Defines target table name. This attribute is **required**.
targetColumnName	Defines column name in target table. This attribute is **required**.
targetTableID	Defines logical target table. It is possible to have more logical target tables than physical target tables. This attribute can take values: 0, 1, 2, ... This attribute is required. This attribute defines more logical tables at the same physical table (uses the same table several times).
valueMode	Defines updating mode. Possible modes are: Key, Overwrite, Update, SetIfCreated and SetNull. This attribute is required. ‘Key’ mode - defines key column of the target table used for selection of row; ‘Overwrite’ mode - overwrites the value of the target column in appropriate row or inserts the value in a new row; ‘Update’ mode - updates the value in empty column of the appropriate row; ‘SetIfCreated’ mode - the value is written only if a new row is inserted; ‘SetNull’ mode - value is written only if

	the new value is NULL.
defaultValue	This tag is optional. If we want to use 'If some value of table fields is null' method of 'Data Cleaning' advanced features, we must set this tag to appropriate value.

OctopusLoader processes UserID variable column as a relation column (writes data into target table using relations criteria).

```
e.g. of the <userIDColumn> tag
```

```
<userIDColumn targetTableName="OMDSBENUTZER" targetColumnName="USERID"
 targetTableID="0" valueMode="Key" defaultValue="defUser"/>
```

The <timeStampColumn> tag

<timeStampColumn> tag defines target column in which current time value will be overwritten or updated. Whether it will be overwritten or updated depends on value mode. This tag is *optional*.

Table 5.22. Table of <timeStampColumn> tag attributes

attribute	description
targetTableName	Defines target table name. This attribute is **required**.
targetColumnName	Defines column name in target table. This attribute is **required**.
targetTableID	Defines logical target table. It is possible to have more logical target tables than actual target tables. This attribute can take values: 0, 1, 2, ... This attribute is required. This attribute defines more logical tables at the same physical table (uses the same table several times).
valueMode	Defines updating mode. Possible modes are: Overwrite, Update, SetIfCreated and SetNull. This attribute is required. ‘Overwrite’ mode - overwrites the value of the target column in appropriate row or inserts the value in a new row; ‘Update’ mode - updates the value in empty column of the appropriate row; ‘SetIfCreated’ mode - the value is written only if a new row is inserted; ‘SetNull’ mode - value is written only if the new value is NULL.

OctopusLoader processes timeStamp column as a relation column (writes data into target table using relations criteria).

```
e.g. of the <timeStampColumn> tag
```

```
<timeStampColumn targetTableName="Geschaeftsfallmenue"
 targetColumnName="ERZEUGTAMUM" targetTableID="0" valueMode="SetIfCreated"/>
```

The <relationColumns> tag

<relationColumns> tag defines relations between target tables. This tag is optional. Child tag is <relationColumn>.

The <relationColumn> tag

<relationColumn> tag defines a relationship between columns in target tables as well as relations between tables in target database. This tag may occur once or more times.

Table 5.23. Table of <relationColumn> tag attributes

attribute	description
relationSourceTableName	The name of the source table in relations. This table is in target database. This attribute is **required**.
relationSourceColumnName	The name of source column. This attribute is optional. Default value is oid.
relationSourceTableID	Defines logical source table. It is possible to have more logical source tables than actual source tables. This attribute can take values: 0, 1, 2, ... This attribute is required. This attribute defines more logical tables at the same physical table (uses the same table several times).
relationTargetTableName	The name of the target table in relations. This table is in target database. This attribute is **required**.
relationTargetColumnName	The name of target column. This attribute is optional. Default value is oid (when oidLogic = true for that table).
relationTargetTableID	Defines logical target table. It is possible to have more logical target tables then actual target tables. This attribute can take values: 0, 1, 2, ... This attribute is required. This attribute defines more logical tables at the same physical table (uses the same table several times).
relationMode	Defines updating mode. Possible modes are: Key, Overwrite, Update and SetNull. This attribute is required. ‘Key’ mode - defines key column of the target table used for selection of row; ‘Overwrite’ mode - overwrites the value of the target column in appropriate row or inserts the value in a new row; ‘Update’ mode - updates the value in empty column of the appropriate row; ‘SetNull’ mode - value is written only if the new value is NULL.
defaultValue	This tag is optional. If we want to use 'If some value of table fields is null' method of data cleaning advanced features, we must set this tag to appropriate value. Default value represents a SQL statement, which retrieves default value for relation column.

```
e.g. of the <relationColumn> tag

<relationColumn relationSourceTableName="PERSISTENTTREE"
 sourceColumnName="OID" relationSourceTableID="0"
 targetTableName="PERSISTENTTREE" targetColumnName="PERSISTENTTREETYPEOID"
 relationTargetTableID="0" valueMode="Overwrite" defaultValue="SELECT oid FROM PERSISTENTTREE WHERE
```

The <constantColumns> tag

<constantColumns> tag inserts constant values in the target columns. This tag is optional. Child tag is constantCol-<umn>.

The <constantColumn> tag

<constantColumn> tag inserts constant values in the target columns. This tag must occur at least once. It has five attributes: targetTableName, targetColumnName, targetTableID, valueMode and constantValue; and it hasn't any child tag.

Table 5.24. Table of <constantColumn> tag attributes

attribute	description
targetTableName	The name of the target table. This attribute is **required**.
targetColumnName	The name of target column. This attribute is optional. Default value is oid.
targetTableID	Defines logical target table. It is possible to have more logical target tables than actual target tables. This attribute can take values: 0, 1, 2, ... This attribute is required. This attribute defines more logical tables at the same physical table (uses the same table several times).
valueMode	Defines updating mode. Possible modes are: Key, Overwrite, Update, SetIfCreated and SetNull. This attribute is required. ‘Key’ mode - defines key column of the target table used for selection of row; ‘Overwrite’ mode - overwrites the value of the target column in appropriate row or inserts the value in a new row; ‘Update’ mode - updates the value in empty column of the appropriate row; ‘SetIfCreated’ mode - the value is written only if a new row is inserted; ‘SetNull’ mode - value is written only if the new value is NULL.
constantValue	Constant value. This attribute is optional. If a constantColumn has no constantValue attribute the value is NULL.
defaultValue	This tag is optional. If we want to use 'If some value of table fields is null' method of 'Data Cleaning' advanced features, we must set this tag to appropriate value.

```
e.g. of the <constantColumn> tag

<constantColumn targetTableName="OMDSBENUTZER"
 targetColumnName="GESCHLECHTOID" targetTableID="0" valueMode="Overwrite"
 constantValue="2"/>
```

The <counterColumns> tag

<counterColumns> tag inserts auto increment values in the target columns. This tag is optional. Subelements of this tag are <counterColumn>(inserting simple auto increment values in counterColumn) and subCounterColumn (inserting auto increment values in counterColumn for each different combination of key values defined by subCounterKeyColumns).

Table 5.25. Table of <counterColumns> tag attributes

attribute	description
counterTableName	The name of table of counters and its current values. This attribute is **required**.
counterNameColumn	Counter name column. Must be unique for all counters in one importDefinition. This attribute is **required**.
counterValueColumn	Column with current counter values for each counterColumn name. This attribute is **required**.

The <counterColumn> tag

<counterColumn> tag inserts auto increment values in the target columns. This element is optional. It has eight attributes: targetTableName, targetColumnName, targetTableID, valueMode, counterName, counterStartValue, counterIncrement and counterStartValueReset.

Table 5.26. Table of <counterColumn> tag attributes

attribute	description
counterName	Name of counter. It must be unique into one importDefinition job. This attribute is required.
counterStartValue	Start value of counter. This attribute is optional. Default value is 1.
counterIncrement	Increment of counter. This attribute is optional. Default value is 1.
counterStartValueReset	Possible values are **true** and **false**. If value is 'false' counter starts from value which it has read from counterTable. If value is 'true', counter start value is counterStartValue. This attribute is **required**.
valueMode	Defines updating mode. Possible mode is SetIfCreated (other modes not implemented yet). This attribute is **required**.
targetTableName	The name of the target table. This attribute is **required**.
targetColumnName	The name of target counter column. This attribute is **required**.
targetTableID	Defines logical target table. It is possible to have more logical target tables than actual target tables. This attribute can take values: 0, 1, 2, ... This attribute is required. This attribute defines more logical tables at the same physical table (uses the same table several times).

The <subCounterColumn> tag

<subCounterColumn> tag inserts auto increment values in counterColumn for each different combination of key values defined by subCounterKeyColumns (subelements of subCounterColumn). This element is optional. It has seven attributes: targetTableName, targetColumnName, targetTableID, valueMode, counterName, counterStartValue and counterIncrement.

Table 5.27. Table of <subCounterColumn> tag attributes

attribute	description
counterName	Name of counter. Must be unique in one importDefinition job. This attribute is **required**.
counterStartValue	Start value of counter. This attribute is optional. Default value is 1.
counterIncrement	Increment of counter. This attribute is optional. Default value is 1.
targetTableName	The name of the target table. This attribute is **required**.
targetColumnName	The name of target counter column. This attribute is **required**.

targetTableID	Defines logical target table. It is possible to have more logical target tables than actual target tables. This attribute can take values: 0, 1, 2, ... This attribute is required. This attribute defines more logical tables at the same physical table (uses the same table several times).
valueMode	Defines updating mode. Possible mode is SetIfCreated (other modes not implemented yet). This attribute is **required**.

The <subCounterKeyColumn> tag

<subCounterKeyColumn> tag defines key values for subCounterColumns. This element is optional. It has only one attribute: columnName. It occurs at least once.

```
e.g. of the <counterColumns> tag

      <counterColumns counterTableName="counterTable" counterNameColumn=Col01 counterValueColumn="va
         <counterColumn counterName="counter1" counterStartValue="100" counterIncrement="1" targetTa
            targetTableID="0" targetColumnName"ColCounter" valueMode="SetIfCreated" counterStartValu
         <subCounterColumn constantValue="myVal3" targetTableName="TargetTab1" targetTableID="0"
            targetColumnName="Col49" valueMode="Overwrite"/>
            <subCounterKeyColumn keyColumnName="Col1"/>
            <subCounterKeyColumn keyColumnName="ForeignKey1"/>
         </subCounterColumn>
      </counterColumns>
```

The <tables> tag

<tables> tag defines target tables. This tag is **required**.

The <table> tag

<table> tag defines target table, insert mode, table mode and oid logic. Target table is table in target database. This tag must occur at least once.

Table 5.28. Table of <table> tag attributes

attribute	description
tableName	The name of table in the target database. This attribute is **required**.
tableID	Defines logical target table. It is possible to have more logical target tables than actual target tables. This attribute can take values: 0, 1, 2, ... This attribute is required. This attribute defines more logical tables at the same physical table (uses the same table several times).
insert	Possible values are **true** or **false**. If value is true, data will be inserted otherwise data won't be inserted. This attribute is **required**.
tableMode	Possible values are **Cache** or **Query.** If value is *Cache*, all data will be read from the source table and put into memory and finally put into target database. If value is *Query*, data will be read and loaded raw by raw. This attribute is **required**.
oidLogic	Possible values are **true** or **false**. If value is true, target table has oid column and supports oid logic. This attribute is **required**.

autoMap	Possible values are **true** or **false**. If value is true, all source columns, which are not named in "importDefinition" job, will be mapped to appropriate columns in target table.
defaultMode	Defines mode for all columns that are auto mapped. Possible values are Overwrite, Update, SetIfCreated or SetNull.

```
e.g. of the <table> tag

<table tableName="PERSISTENTTREETYPE" tableID="0" insert="true"
 tableMode="Cache" oidLogic="true"/>
```

The important thing in loading process is to define Key columns in target tables. These columns are defined in valueColumn, variableColumn and relationColumn tags. All this columns are Key columns.

Working with TOS (Together Object Server) format tables

Tables used by TOS have global unique OID logic as well as DODS tables. TOS tables use OID column (VARCHAR type) as a primary key column. Also TOS creates Updt column (timestamp), LID, MKeys, VKeys, DKeys columns. OID column is unique for each table.

Differences between TOS and DODS oid logic are:

- type of OID column (decimal vs varchar)
- in TOS table there is no version column
- objectID table has two columns: one for name of OID counter, other for its value

In OctopusLoader you can load data into TOS table if objectIDNameColumnName and objectIDNameColumnValue attributes are set in loadJob element (default values). If these attributes are set in loadJob configuration file OctopusLoader automatically uses TOS tables format.

Attribute ‘objectIDAutoCreate’ defines whether OctopusLoader will auto create objectID table or not.

```
e.g. Setting Octopus attributes to work with TOS tables

<loaderJob logMode="normal" objectIDIncrement="1" objectIDTableName="objectid"
          objectIDColumnName="next" objectIDNameColumnName="Column1" objectIDNameColumnValue="test"/

In importDefinition job you can create variableColumn to fill Updt column (update time).

<variableColumns>
     <timeStampColumn targetTableName="TargetTable1" targetColumnName="Updt" targetTableID="0" value
</variableColumns>
```

Chapter 6. Transform data with Octopus

Transformations

Octopus loader gives you a way to transform data. You can define in import definitions way to transform data by defining new <transformations> tag like this one:

```
 <transformations>
   <transformation name="transformation"
          transformatorClassName="org.webdocwf.util.loader.TestTransformer"
          transformatorConfig="">
     <sourceColumns>
       <sourceColumn name="XMLNAME"/>
     </sourceColumns>
     <targetColumns>
       <targetColumn name="TRANSXMLNAME" tableName="BOOKS" tableID="0" valueMode="Overwrite"/>
     </targetColumns>
   </transformation>
 </transformations>
```

This tags are explained in chapter 5. OctopusLoader, section related to <transformations> tag.

Key part of transformations is implementation of Transformer interface. This is the class that is responsible for transforming data from one database to another (Note: Target database can be the same as source). Class that will be transformer used in Octopus must implement this interface. In attribute transformatorClassName, (<transformation transformatorClassName=".....) , full class name has to be specified, and this class must implement Transformer interface.

NOTE: Octopus will make new instance of class specified in transformatorClassName attribute for every transformation tag. It is possible to use same class for more than one transformation.

This is source code of Transformer Interface:

```
package org.webdocwf.util.loader;

import java.util.List;

public interface Transformer {

/**
 * Configure transformer
 */
public void configure(String s);

/**
 * Release resources if necessairly
 */
public void release();

/**
 * This method return List with transformed values for source column(s).
 */
public List transformValue(List valueToTransform);

}
```

Methods in Transformer interface:

configure() - with this method is possible to pass some external values(value from attribute transformatorConfig will be passed to this method) to class which implements this interface. After instanciate class specified in transformationClassName, Octopus will call this method with passed value specified in transformatorConfig attribute. This values (as shown in ExampleTransformer.java) can be used for configure transformator in different ways.

NOTE: This method will be called once, just after instanciate transformator class.

transformValues() - this method is key method for doing transformations. This method has as parameter List with values from source columns specified in <sourceColumns> tag. This list has values from ONE ROW in source database, for source coulmns. This means that if there is for example two source columns, than list will have size two, and values will be values from this two source columns, from one row in database. In this method this values can be transfored in any way. What is IMPORTANT is that List which is return value from this method, has size exactly as much as there is target columns specified in <targetColumns> tag. This mean that if in <targetColumns> there is three columns specified, than List which is return value from this method has to have size three.

NOTE: This method will be called once for every row for tables which are defined as target tables in transformations tag.

release() - this method can be used for explicit release of resources(connections to database, files, ...).

NOTE: This method wil be called once on the end of importDefinition in which this transformation is defined.

Octopus has example with transformations. In this example (called "Transformation Example"), is explained how to build your own example which will transform data in your database in the way that you are specified. First this is the TransformationExample.olj file for this example:

```
<?xml version="1.0" encoding="UTF-8" ?>
<loaderJob logMode="normal" objectIDIncrement="1" objectIDTableName="objectid" objectIDColumnName="n
  <jdbcDefaultParameters>
    <jdbcSourceParameters dbVendor = "csv">
      <jdbcSourceParameter name="JdbcDriver" value="org.relique.jdbc.csv.CsvDriver"/>
      <jdbcSourceParameter name="Connection.Url" value="jdbc:relique:csv:LoaderInput"/>
    </jdbcSourceParameters>
    <jdbcTargetParameters dbVendor="HypersonicSQL" driverName="hsql">
      <jdbcTargetParameter name="Password" value=""/>
      <jdbcTargetParameter name="Connection.Url" value="jdbc:hsqldb:LoaderOutput/TestDatabase/TestDB
      <jdbcTargetParameter name="JdbcDriver" value="org.hsqldb.jdbcDriver"/>
      <jdbcTargetParameter name="User" value="sa"/>
    </jdbcTargetParameters>
  </jdbcDefaultParameters>
  <sql name="CreateTables" onErrorContinue="false" commit="true">
    <sqlStmt>
      <include href="includes/TransformationCreateTables.sql" parse="text"/>
    </sqlStmt>
  </sql>
  <sql name="CreateOidDODS" onErrorContinue="false" commit="true">
    <sqlStmt>
      <include href="includes/CreateOidAdminData.sql" parse="text"/>
  </sqlStmt>
  </sql>
  <definitionInclude>
    <include href="TransformationDefinition.oli" parse="xml"/>
  </definitionInclude>
</loaderJob>
```

This example use CSV database as source. It use PERSON.csv (Note:PERSON.csv is placed in xam-Octopus-src\eples\ObjectLoader\LoaderInput) as source table and transform this values as it is specified in TransformationDefinition.oli:

```
<definitionInclude>
  <!--
  This definitions will load data from source table PERSON to target table PERSON
  and will do transformation of data while loading table.
  Transformation will make one column from two columns - FIRSTNAME and SECONDNAME will be concatenat
  one column NAME.
  This will be done in transformation called NameTransformer.
  -->
  <importDefinition name="PERSON" tableName="PERSON" logMode="normal" objectIDIncrement="1" >
    <valueColumns>
      <valueColumn sourceColumnName="ID" targetTableName="PERSON" targetColumnName="ID" targetTableI
    </valueColumns>
    <transformations>
      <transformation name="NameTransformer" transformatorClassName="ExampleTransformer" transformat
```

```
      <sourceColumns>
        <sourceColumn name="FIRSTNAME"/>
        <sourceColumn name="SECONDNAME"/>
      </sourceColumns>
      <targetColumns>
        <targetColumn name="NAME" tableName="PERSON" tableID="0" valueMode="Overwrite"/>
      </targetColumns>
    </transformation>
  </transformations>
  <tables>
    <table tableName="PERSON" tableID="0" insert="true" tableMode="Query" oidLogic="true"/>
  </tables>
</importDefinition>

<!--
This definition will load data into target table PERSONDATA.
Data will be loaded from source table PERSON to target table PERSONDATA.
Telephone number will be changed during this loading.
This will be done in transformation called TelephoneTransformer.
Also, street name will be splitted into two columns - STREETNAME will have
only name of street and STREETNUM will be number.
This will be done in transformation called StreetTransformer.
-->
<importDefinition name="PERSONDATA" tableName="PERSON" logMode="normal" objectIDIncrement="20">
  <valueColumns>
    <valueColumn sourceColumnName="CITY" targetTableName="PERSONDATA" targetColumnName="CITY" targ
    <valueColumn sourceColumnName="ID" targetTableName="PERSONDATA" targetColumnName="PERSONID" ta
  </valueColumns>
  <transformations>
    <transformation name="TelephoneTransformer" transformatorClassName="ExampleTransformer" transf
      <sourceColumns>
        <sourceColumn name="TELEPHONE"/>
      </sourceColumns>
      <targetColumns>
        <targetColumn name="TELEPHONE" tableName="PERSONDATA" tableID="0" valueMode="Overwrite"/>
      </targetColumns>
    </transformation>
    <transformation name="StreetTransformer" transformatorClassName="ExampleTransformer" transform
        <sourceColumns>
          <sourceColumn name="STREET"/>
        </sourceColumns>
      <targetColumns>
        <targetColumn name="STREETNAME" tableName="PERSONDATA" tableID="0" valueMode="Overwrite"/>
        <targetColumn name="STREETNUM" tableName="PERSONDATA" tableID="0" valueMode="Overwrite"/>
      </targetColumns>
    </transformation>
  </transformations>
  <tables>
    <table tableName="PERSONDATA" tableID="0" insert="true" tableMode="Query" oidLogic="true"/>
  </tables>
</importDefinition>

<!--
This definition will make relations between table PERSON and PERSONDATA,based on oid values.
PERSON.personDataOid=PERSONDATA.oid
-->
<importDefinition name="RELATIONS" tableName="PERSONDATA" logMode="normal" objectIDIncrement="1" >
  <jdbcParameters>
    <jdbcSourceParameters dbVendor="HypersonicSQL" driverName="hsql">
      <jdbcSourceParameter name="Password" value=""/>
      <jdbcSourceParameter name="Connection.Url" value="jdbc:hsqldb:LoaderOutput/TestDatabase/Test
      <jdbcSourceParameter name="JdbcDriver" value="org.hsqldb.jdbcDriver"/>
      <jdbcSourceParameter name="User" value="sa"/>
   </jdbcSourceParameters>
  </jdbcParameters>
  <valueColumns>
    <valueColumn sourceColumnName="OID" targetTableName="PERSONDATA" targetColumnName="OID" target
    <valueColumn sourceColumnName="PERSONID" targetTableName="PERSON" targetColumnName="ID" target
  </valueColumns>
  <relationColumns>
    <relationColumn relationSourceTableName="PERSONDATA" relationSourceTableID="1" relationTargetT
  </relationColumns>
```

```
    <tables>
      <table tableName="PERSON" tableID="0" insert="true" tableMode="Query" oidLogic="true"/>
      <table tableName="PERSONDATA" tableID="1" insert="true" tableMode="Query" oidLogic="true"/>
    </tables>
  </importDefinition>
</definitionInclude>
```

NOTE: If is used "selectStatement" instead of tableName in ImportDefinition, then columns with transformations must be at the end of statement. Also, they must be in same order in this select statement as they are in sourceColumns for transformation for this importDefinition.See example below:

```
<importDefinition name="PERSON" selectStatement="select ID,FIRSTNAME,SECONDNAME from PERSON" logMode
  <valueColumns>
    <valueColumn sourceColumnName="ID" targetTableName="PERSON" targetColumnName="ID" targetTableID=
  </valueColumns>
  <transformations>
    <transformation name="NameTransformer" transformatorClassName="ExampleTransformer" transformator
      <sourceColumns>
        <sourceColumn name="FIRSTNAME"/>
        <sourceColumn name="SECONDNAME"/>
      </sourceColumns>
      <targetColumns>
        <targetColumn name="NAME" tableName="PERSON" tableID="0" valueMode="Overwrite"/>
      </targetColumns>
  </transformation>
  </transformations>
  <tables>
    <table tableName="PERSON" tableID="0" insert="true" tableMode="Query" oidLogic="true"/>
  </tables>
</importDefinition>
```

Example transformer source code can be found in ExampleTransformer.java. (NOTE:This file is in Octopus-src\examples\classes.) This class is used to perform 3 different transformations. Which transformation will be performed depends on how ExampleTransformer is configured. For first transformation transformatorConfig has value 1, for second 2 and for third 3. Of course it was possible to write three different classes - for every transformation one class, but in this example is used approach that one class is used and it is configured for every transformation. :

NameTransformer - This transformation will during load of data, transfer data to one target column (NAME) from two source columns: FIRSTNAME, SECONDNAME. In target column name, data will be concatenated (FIRSTNAME + SECONDNAME).

TelephoneTransformer - This transformation will transfer data to one target column (TELEPHONE) from source column TELEPHONE. During of loading data, first digit in TELEPHONE will be replaced with "3".

StreetTransformer - This transformation will splitt data from one source column (STREET) into two target columns (STREETNAME and STREETNUM). In STREETNAME will be name of the street and in the STREETNUM, street number.

Source code from ExampleTransformer.java:

```
import java.util.List;
import java.util.Vector;

import org.webdocwf.util.loader.Transformer;

public class ExampleTransformer implements Transformer {

 List retValue = new Vector();
 int mode = 1;
 public List transformValue(List valueToTransform){
  retValue.clear();
//This part of code is for transformation called NameTransformer
//(see TransformationDefinition.oli above).Transformation will
//make one column from two columns - FIRSTNAME and SECONDNAME.
//This data will be concatenated to one column NAME.
```

```
  if( mode == 1 ) {
   //Transform names
   String firstname = "";
   String secondname = "";
   if(valueToTransform.get(0) != null)
    firstname = valueToTransform.get(0).toString();
   if(valueToTransform.get(1) != null)
    secondname = valueToTransform.get(1).toString();
   String name = firstname + " " + secondname;
   retValue.add(name);
//This part of code is for transformation called TelephoneTransformer.
//Data from column TELEPHONE will be changed during this loading and placed into TELEPHONE
//column in target database.It will replace first digit in telephone number with "3".

  } else if( mode == 2) {
   //Transform telephone
   String telephone = "";
   if(valueToTransform.get(0) != null) {
    telephone = valueToTransform.get(0).toString();
    if( telephone.length() != 0 ) {
     telephone = telephone.substring(1,telephone.length());
     telephone = "3"+telephone;
    }
    retValue.add(telephone);
   } else {
    retValue.add(null);
   }
//This part of code is for transformation called StreetTransformer
//(see TransformationDefinition.oli above)
//Column STREET will be splitted into two columns(STREETNAME and STREETNUM).STREETNAME will have
//only name of street and STREETNUM will be number.
  }else if( mode == 3) {
   //Transform street
   String street = "";
   String streetName = "";
   String streetNum = "";
   if(valueToTransform.get(0) != null) {
    street = valueToTransform.get(0).toString();
    int index = street.indexOf(',');
    streetName = street.substring(0, index).trim();
    streetNum = street.substring(index+1, street.length()).trim();
    retValue.add(streetName);
    retValue.add(streetNum);
   } else {
   retValue.add(null);
   retValue.add(null);
  }
  }
  return retValue;
}
//mode is attribute from TransformationDefinition.oli, and it is value from transformatorConfig attr
public void configure(String s) {
  this.mode = Integer.parseInt(s);
}

public void release() {
  //there is no resources to release
}

}
```

Predefined classes for transformations

Octopus has predefined classes which can be used to transform data during of loading process. These classes are in org.webdocwf.util.loader.transformation package.

ReplaceData

This class can be used for replacing data from defined "sourceColumn" with new data defined in property file. Value from source table is used as key to find new value in property file that will be inserted into target table.This transformation use one column as sourceColumn and one column as targetColumn. To use this class user must define transformation like this:

```
<importDefinition name="PERSON" tableName="PERSON" logMode="full" objectIDIncrement="1" >
  <valueColumns>
   <valueColumn sourceColumnName="ID" targetTableName="PERSON" targetColumnName="ID" targetTableID="
  </valueColumns>
  <transformations>
   <transformation name="NameTransformer" transformatorClassName="org.webdocwf.util.loader.transform
     transformatorConfig="c:/user/projects/test.properties">
    <sourceColumns>
     <sourceColumn name="NAME"/>
    </sourceColumns>
    <targetColumns>
     <targetColumn name="NAME" tableName="PERSON" tableID="0" valueMode="Overwrite"/>
    </targetColumns>
   </transformation>
 </transformations>
 <tables>
  <table tableName="PERSON" tableID="0" insert="true" tableMode="Query" oidLogic="true"/>
 </tables>
</importDefinition>
```

transformatorClassName - is parameter which point to class which is used for this transformation. This param user must not change.

transformatorConfig - is parameter which user must set. It is path to property file where are placed parameters for this class.

Parameters in property file:

replaceAll - if this parameter is set to true then all values from column "NAME" in source table "PERSON" will be replaced with value defined for parameter **newPropValue**. If replaceAll=false or not defined then will be replaced only defined values with its specific replacements (like user1 with john)

newPropValue - value for this key is value that will be used for replacing data

```
#parameters
replaceAll=true
newPropValue=newname

#if replaceAll is false or is not defined then replace only user1 with john and user2 wih mike
#(oldValue - value from source table which will be replaced)
#(newValue - value which will be inserted in target table)

#oldValue=newValue
user1 = john
user2 = mike
```

ConcatenateData

This class can be used to cancatenate data from "sourceColumn" with value defined in property file. Value from source table is used as key to find new value in property file that will be concatenated to value from source table.This transformation use one column as sourceColumn and one column as targetColumn.To use this class user must define transformation like this:

```
<importDefinition name="PERSON" tableName="PERSON" logMode="full" objectIDIncrement="1" >
  <valueColumns>
   <valueColumn sourceColumnName="ID" targetTableName="PERSON" targetColumnName="ID" targetTableID="
  </valueColumns>
  <transformations>
   <transformation name="NameTransformer" transformatorClassName="org.webdocwf.util.loader.transform
      transformatorConfig="c:/users/projects/test.properties">
```

```
    <sourceColumns>
     <sourceColumn name="NAME"/>
    </sourceColumns>
    <targetColumns>
     <targetColumn name="NAME" tableName="PERSON" tableID="0" valueMode="Overwrite"/>
    </targetColumns>
   </transformation>
  </transformations>
 <tables>
  <table tableName="PERSON" tableID="0" insert="true" tableMode="Query" oidLogic="true"/>
 </tables>
</importDefinition>
```

transformatorClassName - is parameter which point to class which is used for this transformation. This param user must not change.

transformatorConfig - is parameter which user must set. It is path to property file where are placed parameters for this class.

Parameters in property file:

prefix - if value is true then new value (exampel: **john**) will be concatenated on oldValue (value from source column "NAME". Example: **user1**) as prefix. Finally transformed value will be: **johnuser1 . If value is false or not set then no transformation will be done.**

postfix - if value is true then new value (exampel: **john**) will be concatenated on oldValue (value from source column "NAME". Example: **user1**) as postfix. Finally transformed value will be: **user1john . If value is false or not set then no transformation will be done.**

insertBlank - if value is set to true then class will insert one blank field between oldValue (value from source column "NAME". Example: **user1**) and it prefix (or postfix). Example: **john user1 . If value is false or not set then no transformation will be done.**

If postfix and prefix are set on true result will be: **johnuser1john**

If postfix, prefix and insertBlank is set to true then result will be: **john user1 john**

```
#parameters
prefix = true
postfix = true
insertBlank = true

#oldValue=valueForConcatenate
user1 = john
user2 = mike
```

If prefix, postfix or insertBlank are set to false or not set then

CurrencyConverter

This class can be used for currency conversion. Value from source table is used as key to find number which will be used to multiply value from source table. This transformation use one column as sourceColumn and one column as targetColumn. To use this class user must define transformation like this:

```
<importDefinition name="PERSON" tableName="PERSON" logMode="full" objectIDIncrement="1" >
 <valueColumns>
  <valueColumn sourceColumnName="ID" targetTableName="PERSON" targetColumnName="ID" targetTableID="
 </valueColumns>
 <transformations>
  <transformation name="MoneyTransformer" transformatorClassName="org.webdocwf.util.loader.transfor
       transformatorConfig="c:/users/zeljko/projects/test_de.properties">
   <sourceColumns>
    <sourceColumn name="money"/>
   </sourceColumns>
   <targetColumns>
```

```
     <targetColumn name="money" tableName="PERSON" tableID="0" valueMode="Overwrite"/>
    </targetColumns>
   </transformation>
  </transformations>
 <tables>
  <table tableName="PERSON" tableID="0" insert="true" tableMode="Query" oidLogic="true"/>
 </tables>
</importDefinition>
```

transformatorClassName - is parameter which point to class which is used for this transformation. This param user must not change.

transformatorConfig - is parameter which user must set. It is path to property file where are placed parameters for this class.

Parameters in property file:

```
#parameters
multiplyValue = 2.4561
multiplyAll = true
scale = 6
#oldValue=multiplyValue
2000.00=2
3000.00=3
1800.90=3.789
5000.00=4.5
10000.00=5.11
```

multiplyValue - if set to true it will multiply oldValue (value from source column "money". Example:15000.00) with this value

multiplyAll - if this parameter is set to true all values from sourceColumn will be multiplied for value of "multiply-Value". If this parameter is set to false or not set then will be multiplied values that defined in section below section #parameters in property file.Example:if value in source column is 1800.90 then multiply it with 3.789.

scale - If is set class will use it to set scale for transformed value.By default this parameter is set to 2. Scale represents number of decimal places. If scale is to small and there is multiplyValue like this 2.4561 there can be problem with ArithmeticException because transformed value can have more decimal values then defined value for scale.This parameter must be integer value.

NOTE:This class use input values from source table as BigDecimal type and transformed values are also in this format (BigDecimal).

DateFormat

This class can be used for transformation of date format. This transformation use one column as sourceColumn and one column as targetColumn. To use this class user must define transformation like this:

```
<importDefinition name="PERSON" tableName="PERSON" logMode="full" objectIDIncrement="1" >
  <valueColumns>
   <valueColumn sourceColumnName="ID" targetTableName="PERSON" targetColumnName="ID" targetTableID="
  </valueColumns>
  <transformations>
   <transformation name="DateTransformer" transformatorClassName="org.webdocwf.util.loader.transform
        transformatorConfig="c:/users/zeljko/projects/test_de.properties">
    <sourceColumns>
     <sourceColumn name="STARTDATE"/>
    </sourceColumns>
    <targetColumns>
     <targetColumn name="STARTDATE" tableName="PERSON" tableID="0" valueMode="Overwrite"/>
    </targetColumns>
   </transformation>
  </transformations>
 <tables>
  <table tableName="PERSON" tableID="0" insert="true" tableMode="Query" oidLogic="true"/>
 </tables>
```

```
</importDefinition>
```

transformatorClassName - is param which point to class which is used for this transformation. This param user must not change.

transformatorConfig - is parameter which user must set. It is path to property file where are placed parameters for this class.

Parameters in property file:

dateFormat - date format which will be used to transform date from source table to adequate date format for target table, if not set, no transformations will be done. Example of usage:

```
<!-- DateFormat -->
G  Era designator  Text  AD
y  Year  Year  1996; 96
M  Month in year  Month  July; Jul; 07
w  Week in year  Number  27
W  Week in month  Number  2
D  Day in year  Number  189
d  Day in month  Number  10
F  Day of week in month  Number  2
E  Day in week  Text  Tuesday; Tue
a  Am/pm marker  Text  PM
H  Hour in day (0-23)  Number  0
k  Hour in day (1-24)  Number  24
K  Hour in am/pm (0-11)  Number  0
h  Hour in am/pm (1-12)  Number  12
m  Minute in hour  Number  30
s  Second in minute  Number  55
S  Millisecond  Number  978
z  Time zone  General time zone  Pacific Standard Time; PST; GMT-08:00
Z  Time zone  RFC 822 time zone  -0800

Example:
"yyyy-MM-dd HH:mm:ss.SSS" example "2004-05-24 12:12:12.111"
"yyyy.MM.dd G 'at' HH:mm:ss z"  example "2001.07.04 AD at 12:08:56 PDT"
"EEE, MMM d, ''yy" example "Wed, Jul 4, '01"
"yyyyy.MMMMM.dd GGG hh:mm aaa"  example "02001.July.04 AD 12:08 PM"

dateFormat = yyyy-MM-dd
```

NOTE: When this class is used for transformation, user must define adequate date format for target database.

Transformations using JavaScript

Octopus gives you a way to transform data using JavaScript. Using JavaScript user can easily transform data without any new java classes. This JavaScript code can be placed into *.olj (oli) file or included from external file.See example below:

```
<definitionInclude>

<!--This definition will load data into target table PERSONDATA.
Data will be loaded from source table PERSON.
This example will transform CITY and TELEPHONE during data load.
It will change all appearances of CITY named 'London' to 'Amsterdam'.
Also will change TELEPHONE where city is 'London', replacing
first three digits with '055'. See file script.js for details.-->

<importDefinition name="TransformCity" tableName="PERSON" logMode="normal" objectIDIncrement="1" >
 <valueColumns>
  <valueColumn sourceColumnName="FIRSTNAME" targetTableName="PERSONDATA" targetColumnName="FIRSTNAME
  <valueColumn sourceColumnName="SECONDNAME" targetTableName="PERSONDATA" targetColumnName="SECONDNA
  <valueColumn sourceColumnName="STREET" targetTableName="PERSONDATA" targetColumnName="STREET" targ
 </valueColumns>
 <transformations>
  <transformation name="CityAndTelephoneTransformer" transformatorClassName="" transformatorConfig="
   <sourceColumns>
```

```
    <sourceColumn name="CITY"/>
    <sourceColumn name="TELEPHONE"/>
   </sourceColumns>
   <targetColumns>
    <targetColumn name="CITY" tableName="PERSONDATA" tableID="0" valueMode="Overwrite"/>
    <targetColumn name="TELEPHONE" tableName="PERSONDATA" tableID="0" valueMode="Overwrite"/>
   </targetColumns>
   <javaScript>
    <include href="includes/script.js" parse="text"/>
   </javaScript>
  </transformation>
 </transformations>
 <tables>
  <table tableName="PERSONDATA" tableID="0" insert="true" tableMode="Query" oidLogic="true"/>
 </tables>
</importDefinition>

<!--This definition will load data into target table PERSONDATA.
Data will be loaded from source table PERSON.
This example will transform STREET during data load.
It will add "Ser" as prefix on every
STREET where CITY is 'Paris'.-->

<importDefinition name="TransformStreet" tableName="PERSON" logMode="normal" objectIDIncrement="1" >
 <valueColumns>
  <valueColumn sourceColumnName="FIRSTNAME" targetTableName="PERSONDATA" targetColumnName="FIRSTNAME
  <valueColumn sourceColumnName="SECONDNAME" targetTableName="PERSONDATA" targetColumnName="SECONDNA
 </valueColumns>
 <transformations>
  <transformation name="StreetTransformer" transformatorClassName="" transformatorConfig="1">
   <sourceColumns>
    <sourceColumn name="STREET"/>
     <sourceColumn name="CITY"/>
    </sourceColumns>
   <targetColumns>
    <targetColumn name="STREET" tableName="PERSONDATA" tableID="0" valueMode="Overwrite"/>
   </targetColumns>
   <javaScript>
    retVal = new Array();
    if(CITY == 'Paris'){
     STREET = "Ser "+STREET;
     retVal[0]=STREET;
    }else{
     retVal[0]=STREET;
    }
    retVal;
   </javaScript>
  </transformation>
 </transformations>
 <tables>
  <table tableName="PERSONDATA" tableID="0" insert="true" tableMode="Query" oidLogic="true"/>
 </tables>
</importDefinition>
</definitionInclude>
```

NOTE:As source table for this example is used file PERSON.csv placed in OCTOPUS_HOME\LoaderInput\ folder.

In next listing is script.js file which is used in transformation named "CityAndTelephoneTransformer". This file contains javaScript code which will change all apperance of CITY named 'London' to 'Amsterdam'. Also will change TELEPHONE where CITY is 'London', replacing first three digits with '055'.

```
/**
This example will change all apperance of CITY named 'London' to 'Amsterdam'. Also will
change TELEPHONE where CITY is 'London', replacing first three digits with '055'.
**/
retVal = new Array();
 if(CITY == 'London'){
  retVal[0]='Amsterdam';
 }else{
  retVal[0]=CITY;
```

```
 }
 if (CITY == 'London'){
  if (TELEPHONE != null){
   pos = TELEPHONE.indexOf("/");
   TELEPHONE = TELEPHONE.substring(pos,TELEPHONE.length);
   retVal[1] = "055" + TELEPHONE;
  }else{
   retVal[1]= null;
  }
 }else{
 retVal[1]=TELEPHONE;
 }
retVal;
```

NOTE: Source column name must be the same as variable name in javaScript code. Also variable name (in JavaScript code)and column name (in sourceColumn tag) are "key sensitive".

Chapter 7. ‘Backup’ and ‘Restore’ of database

Introduction

EnhydraOctopus application can be used for ‘backup’ and ‘restore’ any database of your choice.

Database ‘backup’

If you start *‘backup’* process, **OctopusGenerator** will generate SQL statements for all named database vendors and **OctopusLoader** will copy all data from source to target database.

Note

This is necessary because we don't want to loose any META DATA of source database.

Table 7.1. Table of database backup attributes

Property	Name	Description
- o	generatorOutput	It represents the directory, where the OctopusLoader places created files. If this argument does not exist, Octopus places created files into current directory.
- sdb	sourceDatabase	It represents the part of Connection property and defines database name and its URL for source database. This argument is **required**.
- sdn	sourceDriverName	It represents driver name of defined source database. This parameter is optional. If this argument doesn't exist, application takes first driver name, which is placed in conf file, which belongs to source database.
- su	sourceUser	Defines user name of the source database. If the source database requires user name, you must enter corresponding user name.
-sp	sourcePassword	Defines user password for the source database. If the source database requires user password, you must enter corresponding user password.
- st	sourceType	Defines the type of the source database. This argument is **required**.
- tdb	targetDatabase	Represents the part of Connection property and defines database name and its URL for target database. This argument is **required**.
- tdn	targetDriverName	Represents driver name of defined target database. This parameter is op-

		tional. If this argument doesn't exist, application takes first driver name, which is placed in conf file, which belongs to target database.
- tu	targetUser	Defines user name of the target database. If the target database requires user name, you must enter corresponding user name.
- tp	targetPassword	Defines user password for the target database. If the target database requires user password, you must enter corresponding user password.
- tt	targetType	Defines the type of the target database. This argument is **required**.
-it	includeTables	Defines the list of tables whic you want to include in 'backup' process
-cjs	confJarStructure	Defines the path to jar file in which conf file are placed. This parameter should be used only if you put conf files in separate jar file.

You can 'backup' database using :

- backup scripts,
- ant task, or
- backup API.

Backup scripts are created and added into EnhydraOctopus application and can be found in ‘bin’ directory of binary distribution.

```
e.g. of starting backup scripts

backup.cmd -o Backup -sdb "localhost:1433/Test" -su sa -st MSQL -tdb "localhost/Test" -tt MySQL
```

Backup using Ant task example is listed below.

```
e.g. of Ant task

...

<taskdef name="BackupTask" classname="org.webdocwf.util.loader.task.BackupTask">
    <classpath>
            <pathelement location="../lib/OctopusGenerator.jar"/>
            <pathelement location="../lib/Octopus.jar"/>
    </classpath>
</taskdef>

<target name="backup">
    <BackupTask
           generatorOutput ="Backup"
           sourceDatabase="../ObjectLoader/LoaderInput/Database/TestDB"
           sourceUser ="sa"
           sourcePassword =""
           sourceType="HypersonicSQL"
           sourceDriverName="hsql"
           targetDatabase ="TestDatabase;create=true"
           targetUser =""
```

```
            targetPassword =""
            targetType="Csv"
            targetDriverName="csv"
            >
    </BackupTask>
</target>

...
```

Backup API is way to start 'backup' proces from any java application by it's public constructor.

Databases ‘restore’

‘Restore’ database’s attributes are the same as ‘Backup’ database attributes.

Since ‘backup’ process create all needed SQL statements in folder {generatorOutput}/SqlForAllVendors, and ‘restore’ process use the same SQL statements (because they contain META DATA), generatorOutput parameter in restore scripts must be {generatorOutput}/SqlForAllVendors/type_of_target_database.

You can 'restore' databse using:

- restore scripts,
- ant task, or
- restore API.

‘Restore’ scripts are created and added into OctopusLoader application and can be found in bin directory of binary distribution.

```
e.g. of starting ‘restore’ scripts

restore.cmd -o Backup/SQlForAllVendors/MSQL -sdb Backup/database -st Csv -tdb localhost:1433/Test11
```

Restore using Ant task example is listed below.

```
e.g. of restor Ant task
...
<taskdef name="RestoreTask" classname="org.webdocwf.util.loader.task.RestoreTask">
    <classpath>
            <pathelement location="../lib/OctopusGenerator.jar"/>
            <pathelement location="../lib/Octopus.jar"/>
    </classpath>
</taskdef>

<target name="restore">
    <RestoreTask
            generatorOutput ="Backup/SQLForAllVendors/HypersonicSQL"
            sourceDatabase="../../TestDatabase"
            sourceUser =""
            sourcePassword =""
            sourceType="Csv"
            sourceDriverName="csv"
            targetDatabase ="../../../Restore/TestDatabase"
            targetUser ="sa"
            targetPassword =""
            targetType="HypersonicSQL"
            targetDriverName="hsql"
            >
    </RestoreTask>
</target>
...
```

Restore API is way to start restore proces from any java application by it's public constructor.

Chapter 8. Configuration files

Introduction

There are two types of files among configuration files for EnhydraOctopus. The first type is OctopusDBVendors.xml. In this file information about all supported database vendors is stored, as well as path to corresponding configuration files (which is the second type of EnhydraOctopus configuration files). Here are JDBC data types too, mapped in appropriate java classes. The second type of configuration files stores information about oid and version columns, and all necessary data which describes drivers for supported database and tags which map all JDBC data types to specific database vendors data type.

OctopusDBVendors.xml file

OctopusGenerator reads the conf.xml and OctopusDBVendors.xml files which are placed in jar file, by default. If you want to use your own conf file, which is placed in conf/ directory of EnhydraOctopus, you must set the OCTOPUS_HOME parameter, as Java property (using -D option while starting Java application.(e.g. java -DOCTOPUS_HOME=c:\Octopus\)).

In OctopusDBVendors.xml file, applications search for the tags <Database>, which represents all databases supported by EnhydraOctopus.

e.g.

```
    <Database>
        <Vendor name="Standard">C:/Octopus/conf/StandardConf.xml</Vendor>
        <Vendor name="InstantDB">C:/Octopus/conf/InstantDBConf.xml</Vendor>
        <Vendor name="Informix">C:/Octopus/conf/InformixConf.xml</Vendor>
        <Vendor name="MSQL">C:/Octopus/conf/MSQLConf.xml</Vendor>
        <Vendor name="HypersonicSQL">C:/Octopus/conf/HypersonicSQLConf.xml</Vendor>
        <Vendor name="DB2">C:/Octopus/conf/DB2Conf.xml</Vendor>
        <Vendor name="QED">C:/Octopus/conf/QEDConf.xml</Vendor>
        <Vendor name="MySQL">C:/Octopus/conf/MySQLConf.xml</Vendor>
        <Vendor name="Oracle">C:/Octopus/conf/OracleConf.xml</Vendor>
        <Vendor name="Csv">C:/Octopus/conf/CsvConf.xml</Vendor>
        <Vendor name="Xml">C:/Octopus/conf/XmlConf.xml</Vendor>
        <Vendor name="Access">C:/Octopus/conf/AccessConf.xml</Vendor>
        <Vendor name="Excel">C:/Octopus/conf/ExcelConf.xml</Vendor>
        <Vendor name="PostgreSQL">C:/Octopus/conf/PostgreSQLConf.xml</Vendor>
        <Vendor name="Mckoi">C:/Octopus/conf/MckoiConf.xml</Vendor>
        <Vendor name="Octetstring">C:/Octopus/conf/OctetstringConf.xml</Vendor>
        <Vendor name="CJDBC">C:/Octopus/conf/CJDBCConf.xml</Vendor>
        <Vendor name="Sybase">C:/Octopus/conf/SybaseConf.xml</Vendor>
        <Vendor name="Paradox">C:/Octopus/conf/ParadoxConf.xml</Vendor>
    </Database>
```

Afterwards, depending on the type of the database, which is equal to value of attribute ‘name’ of <Vendor> tag, application reads the place (path) where corresponding conf file is placed.

```
e.g If type of the database is MSQL than, conf file is placed in C:/Octopus/conf/MSQLConf.xml direct
```

When corresponding conf file is found, application reads from it all required data. That applies for source and target database.

Mapping JDBC data types into JAVA classes is used for creating DODS doml file.

e.g.

```
    <JAVAType>
        <BIT>java.lang.Boolean</BIT>
        <BIGINT>java.lang.Long</BIGINT>
        <BINARY>java.lang.Byte[]</BINARY>
        <BLOB>java.sql.BLob</BLOB>
```

```
        <BOOLEAN>java.lang.Boolean</BOOLEAN>
        <CHAR>java.lang.String</CHAR>
        <CLOB>java.sql.Clob</CLOB>
        <DATE>java.sql.Date</DATE>
        <DECIMAL>java.math.BigDecimal</DECIMAL>
        <DOUBLE>java.lang.Double</DOUBLE>
        <FLOAT>java.lang.Double</FLOAT>
        <INTEGER>java.lang.Integer</INTEGER>
        <LONGVARBINARY>java.lang.Byte[]</LONGVARBINARY>
        <LONGVARCHAR>java.lang.String</LONGVARCHAR>
        <NUMERIC>java.math.BigDecimal</NUMERIC>
        <REAL>java.lang.Float</REAL>
        <SMALLINT>java.lang.Short</SMALLINT>
        <TIME>java.sql.Time</TIME>
        <TIMESTAMP>java.sql.Timestamp</TIMESTAMP>
        <TINYINT>java.lang.Byte</TINYINT>
        <VARBINARY>java.lang.Byte[]</VARBINARY>
        <VARCHAR>java.lang.String</VARCHAR>
    </JAVAType>
```

Database Conf.xml file

In [dabase_type]Conf.xml file all the data required for working with database are placed.

- *The Database type used to stores ObjectId's and Version's.*

e.g

```
    <OidDbType>DECIMAL (19,0)</OidDbType>
    <VersionDbType>BIGINT</VersionDbType>
```

Those tags represent Oid and Versions types, for that database (e.g. MSQL).

The OidDbColumnName and VersionDbColumnName are used for creating SQL CREATE TABLE statements in the CreateTables.sql files.

e.g

```
    <OidDbColumnName>oid</OidDbColumnName>
    <VersionDbColumnName>version</VersionDbColumnName>
```

- *ExcludeTables tag - defines system tables which you want to exclude from reading*

```
<ExcludeTables>dtproperties</ExcludeTables>
```

- *DateFormat tag - defines date format for this db vendor.*

```
<DateFormat>yyyy-MM-dd HH:mm:ss.SSS</DateFormat>
```

- *The listed tags below are used for storing parameters which describe used drivers and connections.*

1. Driver name -Driver name
2. ClassName - Driver class
3. Connection - Connection for the database

4. RequiredUser = "true/false" - if database requires user and password for connecting
5. FirstColumnResult = "0/1" - first column in result set
6. EnableJumpInResult = "true/false" - if jdbc driver supports absolute(int) and relative(int) methods - jump in result set
7. AfterLastRow = "true/false" - if jdbc driver supports moving pointer after the last row in result set
8. EnableOrderBy = "true/false" - if jdbc driver supports "order by" sql statements in sql commands.
9. AlterTablePrimaryKey = "true/false" - defines if the databases support ALTER TABLE sql statements for creating primary keys.
10. MetaData = "true/false" - defines if the database is supporting meta data methods
11. RowCountEnabled = "true/false" - defines if the database support count(*) method
12. SetFetchSizeEnabled = "true/false" - defines if database support setFetchSize() method
13. SetCursorNameEnabled="true/false" - defines if database support setCursorName() method
14. SetEmptyStringAsNull = "true/false" - defines the possibility to convert an empty string to 'null' object
15. ReadingOrderRelevant ="true/false" - defines if order of reading data from source database is relevant or not
16. FileSystemDatabase="true/false" - defines if this driver read/write on file system
17. MaxConstraintLength="-1" - defines the maximum length of all constraints in used database. In this restrictions doesn't exists, parameter has value '-1'.
18. GetColumnsSupported="true/false" - defines if this driver suport Connection.getMetaData().getColumns() method
19. SetMaxRowsSupported - defines if this driver support java.sql.Statement.setMaxRows().Used for target database because of better performance.

Note

As default for MySql database, in configuration file for this database, value for RequiredUser parameter is set to "true". This must be done because specific structure of MySql database. If you want to change this specification in MySql database, you must edit “user” table in default (system) “mysql” database.

```
e.g (for QED type of database)

    <Driver name="quadcap">
        <ClassName value="com.quadcap.jdbc.JdbcDriver"/>
        <Connection value="jdbc:qed:"/>
        <RequiredUser value="false"/>
        <FirstColumnResult value="1"/>
        <EnableJumpInResult value="false"/>
        <AfterLastRow value="true"/>
        <EnableOrderBy value="true"/>
        <AlterTablePrimaryKey value="true"/>
        <MetaData value="true"/>
        <RowCountEnabled value="true"/>
        <SetFetchSizeEnabled value="true"/>
        <SetCursorNameEnabled value="true"/>
        <SetEmptyStringAsNull value="false"/>
        <ReadingOrderRelevant value="false"/>
        <FileSystemDatabase value="true"/>
```

```
        <MaxConstraintLength value="-1"/>
        <GetColumnsSupported value="false"/>
        <SetMaxRowsSupported value="false"/>
    </Driver>
```

- *Mapping SQL specific types to JDBC data types which is universal for all databases is used for synchronization of source type of data with JDBC type of data.*

```
e.g.
<SQLType>
        <BIGINT isDecimal="false" isDate="false" isBinary="false" isNumber="true"
                hasSize="false" javaType="java.lang.Long">BIGINT</BIGINT>
        <DATETIME isDecimal="false" isDate="true" isBinary="false" isNumber="false"
                hasSize="false" javaType="java.sql.Timestamp">TIMESTAMP</DATETIME>
        <MONEY isDecimal="false" isDate="false" isBinary="false" isNumber="true"
                hasSize="false" javaType="none">NONE</MONEY>
        <SMALLDATETIME isDecimal="false" isDate="true" isBinary="false" isNumber="false"
                hasSize="false" javaType="java.sql.Timestamp">TIMESTAMP</SMALLDATETIME>
        <TINYINT isDecimal="false" isDate="false" isBinary="false" isNumber="false"
                hasSize="false" javaType="java.lang.Byte">TINYINT</TINYINT>
        <BINARY isDecimal="false" isDate="false" isBinary="true" isNumber="false"
                hasSize="false" javaType="byte[]">BINARY</BINARY>
        <DECIMAL isDecimal="true" isDate="false" isBinary="false" isNumber="true"
                hasSize="true" javaType="java.math.BigDecimal">DECIMAL</DECIMAL>
        <NCHAR isDecimal="false" isDate="false" isBinary="false" isNumber="false"
                hasSize="true" javaType="java.lang.String">VARCHAR</NCHAR>
        <SMALLINT isDecimal="false" isDate="false" isBinary="false" isNumber="true"
                hasSize="false" javaType="java.lang.Short">SMALLINT</SMALLINT>
        <VARBINARY isDecimal="false" isDate="false" isBinary="true" isNumber="false"
                hasSize="true" javaType="byte[]">VARBINARY</VARBINARY>
        <BIT isDecimal="false" isDate="false" isBinary="false" isNumber="false"
                hasSize="false" javaType="java.lang.Boolean">BIT</BIT>
        <FLOAT isDecimal="false" isDate="false" isBinary="false" isNumber="true"
                hasSize="true" javaType="java.lang.Float">FLOAT</FLOAT>
        <NTEXT isDecimal="false" isDate="false" isBinary="false" isNumber="false"
                hasSize="false" javaType="byte[]">BINARY</NTEXT>
        <SMALLMONEY isDecimal="false" isDate="false" isBinary="false" isNumber="true"
                hasSize="false" javaType="none">NONE</SMALLMONEY>
        <VARCHAR isDecimal="false" isDate="false" isBinary="false" isNumber="false"
                hasSize="true" javaType="java.lang.String">VARCHAR</VARCHAR>
        <CHAR isDecimal="false" isDate="false" isBinary="false" isNumber="false"
                hasSize="true" javaType="java.lang.String">CHAR</CHAR>
        <IMAGE isDecimal="false" isDate="false" isBinary="true" isNumber="false"
                hasSize="false" javaType="byte[]">LONGVARBINARY</IMAGE>
        <NVARCHAR isDecimal="false" isDate="false" isBinary="false" isNumber="false"
                hasSize="false" javaType="java.lang.String">LONGVARCHAR</NVARCHAR>
        <NUMERIC isDecimal="false" isDate="false" isBinary="false" isNumber="true"
                hasSize="true" javaType="java.math.BigDecimal">NUMERIC</NUMERIC>
        <TEXT isDecimal="false" isDate="false" isBinary="false" isNumber="false"
                hasSize="false" javaType="java.lang.String">LONGVARCHAR</TEXT>
        <UNIQUEIDENTIFIER isDecimal="false" isDate="false" isBinary="false" isNumber="false"
                hasSize="true" javaType="none">NONE</UNIQUEIDENTIFIER>
        <CURSOR isDecimal="false" isDate="false" isBinary="false" isNumber="false"
                hasSize="true" javaType="none">NONE</CURSOR>
        <INT isDecimal="false" isDate="false" isBinary="false" isNumber="true"
                hasSize="false" javaType="java.lang.Integer">INTEGER</INT>
        <REAL isDecimal="false" isDate="false" isBinary="false" isNumber="true"
                hasSize="false" javaType="java.lang.Float">REAL</REAL>
        <TIMESTAMP isDecimal="false" isDate="true" isBinary="false" isNumber="false"
                hasSize="false" javaType="java.sql.Timestamp">TIMESTAMP</TIMESTAMP>
    </SQLType>
```

javaType attribute is used to determinate which setXXX prepared statement method will be used for handling specific SQL data type.

```
e.g.
     <BIGINT javaType="java.lang.Long">BIGINT</BIGINT>
```

```
for handling BIGING data type, Octopus will use setLong() method of
prepared statement object
```

Octopus support next javaType attribute:

- java.math.BigDecimal
- java.lang.Double
- java.lang.Float
- java.lang.Integer
- java.lang.Long
- java.lang.Short
- java.lang.String
- java.sql.Date
- java.sql.Time
- java.sql.Timestamp
- java.lang.Boolean
- java.lang.Byte
- byte[]

Note

For all 'binary' types, Octopus expected byte[] attribute (javaType='byte[]')

isBinary attribute is used to determine is this type binary or not (isBinaryObject=true/false).Types which are binary:blob, clob, image, varbinary, longvarbinary, binary, bytea.

isNumber attribute is used to determine if type is number or not (isNumber=true/false). Types which are number:decimal, int, numeric, real, short, long, float, double, bigint, money, smallmoney, smallint, number.

hasSize attribute is used to determine if type has defined size or not(hasSize=true/false). This attribute is used by Octopus Generator. If attribute hasSize=false OctopusGenerator will generate sql scripts without size for this type. Types for which we dont have to define size are:int, integer, datetime, smalldatetime, money, smallmoney, bigint, smallint, timestamp, tinyint, ntext, bit, int2, int4, int8, bytea, text, real, date, time, tiny, long, double, identity, image, longvarchar, long varchar, nvarchar, ntext, binary, blob, graphic, longraw, long raw, byte.

isDecimal attribute is used to determine if type is decimal or not. Types which are decimal:bigdecimal, decimal, number.

isDate attribute is used to determine if this type is date. Types which are date: time, date, year, timestamp(**NOTE**:for Microsoft SQL server, timestamp isn't date format), datetime.

isWithNattribute is used to determine if this type need prefix N when using unicode characters. If this parameter is set to 'true', than octopus will add prefix 'N' before value of column during update of rows.

- *Mapping JDBC types to specific SQL types is used for synchronization of universal types of data with target types of data.*

e.g

```
    <JDBCType>
        <BIT>BIT</BIT>
        <BIGINT>BIGINT</BIGINT>
        <BINARY>BINARY</BINARY>
        <BLOB>BLOB</BLOB>
        <BOOLEAN>BOOLEAN</BOOLEAN>
        <CHAR>CHAR</CHAR>
        <CLOB>CLOB</CLOB>
        <DATE>DATE</DATE>
        <DECIMAL>DECIMAL</DECIMAL>
        <DOUBLE>DOUBLEPRECISION</DOUBLE>
        <FLOAT>FLOAT</FLOAT>
        <INTEGER>INTEGER</INTEGER>
        <LONGVARBINARY>VARBINARY</LONGVARBINARY>
        <LONGVARCHAR>NONE</LONGVARCHAR>
        <NUMERIC>NUMERIC</NUMERIC>
        <REAL>REAL</REAL>
        <SMALLINT>SMALLINT</SMALLINT>
        <TIME>TIME</TIME>
        <TIMESTAMP>TIMESTAMP</TIMESTAMP>
        <TINYINT>SMALLINT</TINYINT>
        <VARBINARY>VARBINARY</VARBINARY>
        <VARCHAR>VARCHAR</VARCHAR>
    </JDBCType>
```

Chapter 9. JDBC Drivers

Introduction

In this chapter we described drivers which are distributed with Enhydra Octopus application (freetds, xml, csv), and drivers which are the most frequently used for MSSQL server.

Using FreeTDS overview

This is a sneak peek of free type 4 JDBC drivers for SQLServer and Sybase ASE. Currently it has only been tested mostly on SQLServer 7.0 and SQLServer 2000.

Connecting to a Database

The name of the driver class, for using in the Class.forName() property, is com.internetcds.jdbc.tds.Driver and it accepts any of the following forms of URLs:

```
jdbc:freetds:sqlserver://servername/database
jdbc:freetds:sqlserver://servername:port/database
jdbc:freetds:sybase://servername/database
jdbc:freetds:sybase://servername:port/database
jdbc:freetds://servername/database
jdbc:freetds://servername:port/database
```

Note

These last two forms are obsolete and should not be used.

The driver can also recognize several properties in the URL. The properties include:

Table 9.1. Connection URL properties for FreeTDS driver

CHARSET	Character set to use. Defaults to iso_1.
user	user name that will be used to connect to the database.
password	password that will be used to connect to the database.
APPNAME	Application name to tell to the database. Defaults to "jdbclib".
PROGNAME	Program name to tell to the database. Defaults to "java_app".
TDS	TDS protocol version to use. Can be "4.2", "5.0", or "7.0". Defaults to "7.0".

`e.g.`

```
"jdbc:freetds:sqlserver://servername/database;CHARSET=cp950;TDS=7.0"
```

The class com.internetcds.jdbc.tds.SybaseDriver is no longer needed and has been retained only for backward compatibility. It should not be used for new code.

Right now unless you specify sybase it defaults to port 1433 for SQLServer and port 7100 for Sybase. Generally you will need to specify the port number for Sybase, but not for SQLServer.

JDBC 2.0

The driver can be compiled for JDBC version 2.0.

Using Microsoft JDBC Driver overview

Connecting to a Database

The complete connection URL format used with the driver manager and driver class are:

```
com.microsoft.jdbc.sqlserver.SQLServerDriver
jdbc:microsoft:sqlserver://hostname:port[;property=value...]
```

```
e.g.
```

```
jdbc:microsoft:sqlserver://servername;DatabaseName="YourDatabase"
jdbc:microsoft:sqlserver://servername:port;DatabaseName="YourDatabase"
```

If you don't specify database name, or the database name is wrong, driver connects to the master database.

The driver can also recognize several properties in the URL. The properties include:

Table 9.2. Connection property table

host name	is the TCP/IP address or TCP/IP host name of the server to which you are connecting. NOTE: Entrusted applets cannot open a socket to a machine other than the originating host.
port	is the number of the TCP/IP port.
property=value	specifies connection properties. See "Connection String Properties" for a list of connection properties and their values.

The following example shows a typical connection URL:

```
jdbc:microsoft:sqlserver://server1:1433;user=test;password=secret
```

Connection String Properties

You can use the following connection properties with the JDBC driver manager or SQL Server 2000 Driver for JDBC data sources.

This table below lists all JDBC connection properties supported by the SQL Server driver, and describes each property. The properties have the form:

```
property=value
```

Note

All connection string property names are case-insensitive. For example, PortNumber is the same as port number.

Table 9.3. SQL Server Connection String Properties

DatabaseName OPTIONAL	The name of the SQL Server database to which you want to connect.

HostProcess OPTIONAL	The process ID of the application connecting to SQL Server 2000. The supplied value appears in the "hostprocess" column of the sysprocesses table.
NetAddress OPTIONAL	The MAC address of the network interface card of the application connecting to SQL Server 2000. The supplied value appears in the "net_address" column of the sysprocesses table.
Password	The case-insensitive password used to connect to your SQL Server database.
PortNumber OPTIONAL	The TCP port (use for DataSource connections only). The default is 1433.
ProgramName OPTIONAL	The name of the application connecting to SQL Server 2000. The supplied value appears in the "program_name" column of the sysprocesses table.
SelectMethod	SelectMethod={cursor | direct}. Determines whether database cursors are used for Select statements. Performance and behaviour of the driver are affected by the SelectMethod setting. Direct-The direct method sends the complete result set in one request to the driver. It is useful for queries that only produce a small amount of data that you fetch completely. You should avoid using direct when executing queries that produce a large amount of data, as the result set is cached completely on the client and constrains memory. In this mode, each statement requires its own connection to the database. This is accomplished by "cloning" connections. Cloned connections use the same connection properties as the original connection; however, because transactions must occur on a single connection, auto commit mode is required. Due to this, JTA is not supported in direct mode. In addition, some operations, such as updating an insensitive result set, are not supported in direct mode because the driver must create a second statement internally. Exceptions generated due to the creation of cloned statements usually return an error message similar to "Cannot start a cloned connection while in manual transaction mode." Cursor-When the SelectMethod is set to cursor, a server-side cursor is generated. The rows are fetched from the server in blocks. The JDBC Statement method setFetchSize can be used to control the number of rows that are fetched per request. The cursor method is useful for queries that produce a large amount of data, data that is too large to cache on the client. Performance tests show that the value of setFetchSize has a serious impact on performance when SelectMethod is set to cursor. There is no simple rule for determining the value that you should use. You should experiment with different setFetchSize values to find out which value gives the best performance for your application. The default is direct. Octopus supports cursor method. If you don't set SelectMethod to cursor Octopus automatically change this property.
SendStringParameters AsUnicode	SendStringParametersAsUnicode={true | false}. Determines whether string parameters are sent to the SQL Server database in Unicode or in the default character encoding of the database. True means that string parame-

	ters are sent to SQL Server in Unicode. False means that they are sent in the default encoding, which can improve performance because the server does not need to convert Unicode characters to the default encoding. You should, however, use default encoding only if the parameter string data that you specify is consistent with the default encoding of the database. The default is true.
ServerName	DataSource
User	The case-insensitive user name used to connect to your SQL Server database.

Using JTurbo driver overview

This is a JDBC driver for SQLServer 7.0 and SQLServer 2000.

Connecting to a Database

The name of the driver class for using in the Class.forName() is com.newatlanta.jturbo.driver.Driver and it accepts any of the following forms of URLs:

```
jdbc:JTurbo:sqlserver://servername:port/database
jdbc:JTurbo:sqlserver://servername:port/database/parameter=value
```

The driver can also recognize several properties in the URL. The properties include:

Table 9.4. Connection property table

user	user name that will be used to connect to the database.
password	password used to connect to the database.
sql70	To support SQL Server 7. Can be "true" or "false". Default is "false". If use sql70=true parameter, you have to uncomment setings for JTurbo driver (sql70=true parameter) in OctopusDBvendors.xml file.

```
e.g.
"jdbc:JTurbo:sqlserver://servername/database/sql70=true"
```

Using JDBC-LDAP Bridge overview

The JDBC-LDAP Bridge is a JDBC driver that implements JDBC operations by translating them into LDAPv3 compliant operations. To LDAP servers, it appears as a normal LDAP client application program. The bridge allows JDBC connection to any directory supporting LDAPv3. The Bridge is implemented as the com.octetstring.jdbcldap Java package

Connecting to a Database

The Bridge is used by opening a JDBC connection using a URL with the jdbcldap sub protocol.

```
Class.forName("com.octetstring.jdbcLdap.sql.JdbcLdapDriver")
```

When loaded, the JdbcLdapDriver creates an instance of itself and registers this with the JDBC driver manager.

The next step is to establish a connection using the JndiLdapConnection manager.

```
e.g.
String ldapConnectString ="jdbc:ldap://localhost:389/dc=yourcompany,dc=com?SEARCH_SCOPE:=subTreeScop

java.sql.Connection con;
con = DriverManager.getConnection(ldapConnectString,"cn=Admin","manager")
```

The ldapConnectString is based on the standard LDAP URL format(RFC 2255 [http://www.ietf.org/rfc/rfc2255.txt?number=2255]). And it has the syntax:

```
jdbc:ldap://host[:port]/basedn[?property1:=value1 &property2:=value2&propertyn:=valuen]
```

The last two parameters of the getConnection method, are the authentication credientials. In this case, a valid user distinguished name (e.g. cn=Phil Hunt,ou=People,o=OctetString,c=US) and a password. To connect to the directory anonymously, leave these parameters as empty strings ("").

Valid property qualifiers are:

- **cache_statements**: set to true to cache SQL statements
- **secure**: set to true for TLS/SSL connections over LDP
- **concat_atts**: set to true if attributes with multiple values should be concatenated into a single value surrounded by []. E.g. [val1][val2][val2]

Note

Detailed information about this driver you can found at JDBC-LDAP Bridge [http://www.octetstring.com/products/jdbcldapdriver/index.html] page

Using C - JDBC driver overview

C-JDBC is a database cluster middleware that allows any Java™ application (standalone application, servlet or EJB™ container, ...) to transparently access a cluster of databases through JDBC™. You do not have to modify client applications, application servers or database server software. You just have to ensure that all database accesses are performed through JDBC.

C-JDBC is a free, open source ObjectWeb Consortium [http://www.objectweb.org/]'s project. It is licensed under the GNU Lesser General Public License [http://www.gnu.org/copyleft/lesser.html] (LGPL).

Connecting to a Database

The C-JDBC driver can be loaded as any standard JDBC driver from the client program using:

```
Class.forName("org.objectweb.cjdbc.driver.Driver")
```

The JDBC URL expected for the use with C-JDBC is the following:

```
 jdbc:cjdbc://host1:port1,host2:port2/database
```

host is the machine name (or IP address) where the C-JDBC controller is running, port is the port where the controller is bound on this host.

At least one host must be specified but a list of comma separated hosts can be specified. If several hosts are given, one is picked up randomly from the list. If the currently selected controller fails, another one is automatically picked up from the list.

Default port number is 25322 (C-JDBC on the phone !) if omitted. Those two examples are equivalent:

```
e.g.

DriverManager.getConnection("jdbc:cjdbc://localhost:/tpcw");
DriverManager.getConnection("jdbc:cjdbc://localhost:25322/tpcw");
```

Note

Detailed information about this driver you can found at C-JDBC [http://c-jdbc.objectweb.org/] home page

Using the csv driver overview

Detailed information about csv driver can be find on CSV home [..\extras\Csvjdbc_readme.html] page.

Using the xml driver overview

Detailed information about csv driver can be find on XML home [..\extras\XMLjdbc_readme.html] page.

Chapter 10. Java Web Start

Introduction

Java Web Start is a helper application that gets associated with a Web browser.

When a user clicks on a link that points to a special launch file (JNLP file), it causes the browser to launch Java Web Start, which then automatically downloads, caches, and runs the given Java Technology-based application.

The entire process is typically completed without requiring any user interaction, except for the initial single click.

Setting up the Web site

Java Web Start leverages existing Internet technology, such as the HTTP protocol and Web servers, so existing infrastructure for deploying HTML-based contents can be reused to deploy Java Technology-based applications using Java Web Start.

In order to deploy your application to client machines, you must make sure that all files containing your application are accessible through a Web server. This typically amounts to copying one or more JAR files, along with a JNLP file, into the Web server's directories. The set-up required for enabling the Web site to support Java Web Start is very similar to deploying HTML-based contents. The only caveat is that a new MIME type needs to be configured for the Web server.

Configure the Web server to use the Java Web Start MIME type

Configure the Web server so that all files with the .jnlp file extension are set to the application/x-java-jnlp-file MIME type.

Each Web server has a specific way in which to add MIME types. For example, for the Apache Web server you must add the following line to the .mime.types configuration file:

```
application/x-java-jnlp-file JNLP
```

Create a JNLP file for the application

The format used in this release corresponds to the format specified in the Java Network Launching Protocol and API (JNLP) Specification, v1.0. This document describes the most commonly used elements of the JNLP file.

The JNLP file is an XML document. The following shows a complete example of a JNLP file:

e.g.

```
<definitionInclude>
<?xml version="1.0" encoding="utf-8"?>
<!-- JNLP File for OctopusGenerator Demo Application -->
<jnlp
  spec="1.0+"
  codebase="http://javaweb.eng.com/jaws/apps"
  href="generator.jnlp">
  <information>
    <title>OctopusGenerator Demo Application</title>
    <vendor>Together, Inc.</vendor>
    <homepage href="docs/help.html"/>
    <description>OctopusGenerator Demo Application</description>
    <description kind="short">A demo of the capabilities of the Swing Graphical User Interface.</des
    <icon href="images/generator2.jpg"/>
    <icon kind="splash" href="images/splash.gif"/>
    <offline-allowed/>
  </information>
  <security>
```

```
      <all-permissions/>
  </security>
  <resources>
    <j2se version="1.4+"/>
    <jar href="lib/OctopusGenerator.jar"/>
    <jar href="lib/OctopusGenerator.jar"/>
    <jar href="lib/xmlutil.jar"/>
    <jar href="lib/xerces.jar"/>
  </resources>
  <application-desc main-class="org/webdocwf/util/loader/wizard/WizardFrame"/>
</jnlp>
```

The JNLP Element

- *spec attribute*: This attribute must be 1.0 or higher to work with this release. The default value is "1.0+". Thus, it can typically be omitted,
- *codebase attribute*: All relative URLs specified in href attributes in the JNLP file is using this URL as a base,
- *href attribute*: This is a URL pointing to the location of the JNLP file itself. Java Web Start requires this attribute to be set in order for the application to be included in the Application Manager.

The Information Element

- *title element*: The name of the application.
- *vendor element*: The name of the vendor of the application.
- *homepage element*: Contains a single attribute, href, which is a URL locating the home page for the Application.
- *description element*: A short statement about the application. Description elements are optional.
- *icon element*: Contains an HTTP URL to an image file in either GIF or JPEG format.
- *offline-allowed element*: The optional offline-allowed element indicates if the application can be launched offline.

The Security Element

Each application is, by default, run in a restricted execution environment, similar to the Applet sandbox. The security element can be used to request unrestricted access.

If the all-permissions element is specified, the application will have full access to the client machine and local network. If an application requested full access, then all JAR files must be signed. The user will be prompted to accept the certificate the first time the application is launched.

User also needs to adjust Securuty Policy on his computer (ProgramFiles\Java\current_version_of_java\lib\security\java.policy).

```
e.g.
grant {
       permission java.security.AllPermission "", "";
      }
```

The Resources Element

The resources element is used to specify all the resources, such as Java class files, native libraries, and system prop-

erties that are part of the application.

The j2se element specifies what Java 2 SE Runtime Environment (JRE) versions an application is supported on, as well as standard parameters to the Java Virtual Machine.

```
e.g.
<j2se version="1.4+"/>
```

A jar element specifies a JAR file that is part of the application's classpath.

```
e.g.
<jar href="lib/OctopusGenerator.jar"/>
```

Signing JAR Files with a Test Certificate

All application resources must be retrieved from the JAR files specified in the resources section of the JNLP file, and all jars must be signed.

All information about Test Certificate can be found on: WebStart [http://java.sun.com/products/javawebstart/1.2/docs/developersguide.html#jnlp"]

The Application-Desc Element

The application element indicates that the JNLP file is launching an application (as opposed to an Applet). The application element has an optional attribute, main-class, which can be used to specify the name of the application's main class.

```
e.g.
<application-desc main-class="org/webdocwf/util/loader/wizard/WizardFrame">
</application-desc>
```

All information about WebStart can be found on: WebStart [http://java.sun.com/products/javawebstart/1.2/docs/developersguide.html#jnlp"]

Make the application accessible on the Web server

Ensure your application's JAR files and the JNLP file are accessible at the URLs listed in the JNLP file.

Create a link from the Web page to the JNLP file

The link to the JNLP file is a standard HTML link.

```
e.g.
<a href="generator.jnlp">Launch My OctopusGenerator Application</a>
```

Chapter 11. Examples

Introduction

In this chapter, we describe all examples used in Enhydra Octopus application.

These examples are divided in two groups:

- Octopus Generator examples, and
- Octopus Loader examples.

Octopus Generator Examples

These examples show you how to start end use OctopusGenerator application.

1. **ExampleGenerator** (cmd or sh) - In this example we use JDBC database (HypersonicSQL) as source data. This database, named 'TestDB', is placed in ObjectLoader/LoaderInput/Database directory. This example, will generate XML, SQL and Doml files as output files:

    - LoadJob.olj file will be placed in GeneratorOutput directory,
    - ImportDefinition.oli file will be placed in GeneratorOutput/xml directory,
    - SQL files (CreateIndex.sql, CreateIntegrity.sql, DropTables.sql, CreatePrimary.sql and CreateTables.sql) will be placed in GeneratorOutput/sql directory , and
    - Doml file will be placed in GeneratorOutput/Doml directory.

2. **ExampleGeneratorDoml** (cmd or sh) - Differences between this example and OctopusGenerator example is, that this example use DODS Doml file as source data, and this example doesn't generate doml file as one of the output files.
3. **Generator_using_ant** (cmd or sh) - You can start OctopusGenerator using Ant, also. Ant uses build.xml file in the same directory to start OctopusGenerator. In build.xml file are specified all needed parameters for starting OctopusGenerator application. If you use the driver, which isn't supported in LoaderGenerator, you must install it first, and add them in to class path which are specified in build.xml file.
4. **OctopusGenerator** (cmd or sh) - If you want to start OctopusGenerator application using GUI (graphic user interface), you just need to start this script (batch or shell file).

Octopus Loader Examples

These examples show you how to start end use OctopusLoader application.

How to create input XML File?

In this chapter we will help you to create your own input XML file.

Simple XML File without Relation Columns

Main tag of the XML file is <loaderJob> tag, so we need to define it first.

attributes:

```
<loaderJob logMode="full" objectIDIncrement="1" commitCount="1">
```

First child of <loaderJob> tag, which we use in this example, is <jdbcDefaultParameters>.It defines JDBC default source and target parameters (connections and drivers).

```
<jdbcDefaultParameters>
    <jdbcSourceParameters>
        <jdbcSourceParameter name="JdbcDriver" value="org.myDriver.JDBCDriver" />
        <jdbcSourceParameter name="Connection.Url" value = "jdbc:myDriver:MySourceDatabase" />
    </jdbcSourceParameters>
    <jdbcTargetParameters>
        <jdbcTargetParameter name="JdbcDriver" value="org.myDriver2.JDBCDriver"/>
        <jdbcTargetParameter name="Connection.Url" value =  "jdbc:myDriver2:MySourceDatabase2" />
        <jdbcTargetParameter name="User" value="sa"/>
        <jdbcTargetParameter name="Password" value=""/>
    </jdbcTargetParameters>
</jdbcDefaultParameters>
```

Note

Detailed information about this tag, you can find in 'Octopus Loader application' chapter.

Next child tag of the <loaderJob> tag is <variables> tag. It defines logical variables which are going to be used in <constantColumns> tag, SQL-statements, source-data and JDBC parameters.

```
<variables prefix="!" suffix="!" override="false"
     replaceInConstants="true" replaceInSQL="false" replaceInData="true" replaceInJDBC="true">
    <variable name="var1" value="myVal1" prefix="$" suffix="$" replaceInConstants="false"
     replaceInSQL="true" replaceInData="false" replaceInJDBC="false"/>
    <variable name="var2" value="myVal2"/>
</variables>
```

Note

Detailed information about this tag, you can find in 'Octopus Loader application' chapter.

Next child tag of the <loaderJob> tag is <sql> tag. We need to define its 'name', 'logMode', if application should continue on error, and if it will be committed.

```
<sql name="DropDatabase" onErrorContinue="true" commit="true" logMode="full">
```

Child tag of <sql> tag is <jdbcTargetParameters> tag. It defines JDBC target parameters.

```
<jdbcTargetParameters>
    <jdbcTargetParameter name="JdbcDriver" value = "org.myDriver2.JDBCDriver" />
    <jdbcTargetParameter name="Connection.Url" value = " jdbc:myDriver2:MySourceDatabase3"/>
    <jdbcTargetParameter name="User" value="sa"/>
    <jdbcTargetParameter name="Password" value=""/>
</jdbcTargetParameters>
```

Next child tag of <sql> tag is <sqlStmt> tag. It defines sql statement.

```
    <sqlStmt>
        DROP DATABASE MyDatabase
    </sqlStmt>
</sql>
```

Note

Detailed information about this tag, you can find in 'Octopus Loader application' chapter.

Next child tag, of the <loaderJob> tag, which we use is <sql> tag too. We need it, to create MyDatabase database.

```
<sql name="CreateDatabase" onErrorContinue="false" commit="true" logMode="full">
    <jdbcTargetParameters>
        <jdbcTargetParameter name="JdbcDriver" value="com.newatlanta.jturbo.driver.Driver"/>
        <jdbcTargetParameter name="Connection.Url" value="jdbc:JTurbo://localhost:1433/master"/>
        <jdbcTargetParameter name="User" value="sa"/>
        <jdbcTargetParameter name="Password" value=""/>
    </jdbcTargetParameters>
    <sqlStmt>
CREATE DATABASE MyDatabase
    </sqlStmt>
  </sql>
```

Note

Detailed information about this tag, you can find in 'Octopus Loader application' chapter.

Next child tag of the <loaderJob> tag, which we use is <importDefinition> tag. In this tag, we define source table name and name of import definition job (in this case name is 'FIRSTOMDSBENUTZER').

```
  <importDefinition name="FIRSTOMDSBENUTZER" tableName="FIRSTOMDSBENUTZER">
```

If the JDBC source and target parameters are not defined, default values will be used.

Child of <importDefinition> tag, which we use, is <valueColumns> tag. It defines which source column will be load into which target column (table).

```
    <valueColumns>
        <valueColumn sourceColumnName="VORNAME" targetTableName="OMDSBENUTZER"
         targetColumnName="VORNAME" targetTableID="0" valueMode="Overwrite"/>
        <valueColumn sourceColumnName="NACHNAME" targetTableName="OMDSBENUTZER"
         targetColumnName="NACHNAME" targetTableID="0" valueMode="Overwrite"/>
        <valueColumn sourceColumnName="TITEL" targetTableName="OMDSBENUTZER"
         targetColumnName="TITEL" targetTableID="0" valueMode="Overwrite"/>
        <valueColumn sourceColumnName="EMAILADRESSE" targetTableName="OMDSBENUTZER"
         targetColumnName="EMAILADRESSE" targetTableID="0" valueMode="Overwrite"/>
        <valueColumn sourceColumnName="TELEFONNUMMER" targetTableName="OMDSBENUTZER"
         targetColumnName="TELEFONNUMMER" targetTableID="0" valueMode="Overwrite"/>
    </valueColumns>
```

Next tag which we used is <variableColumns> tag. This tag defines columns which has the time stamp and user ID.

```
    <variableColumns>
        <timeStampColumn targetTableName="OMDSBENUTZER" targetColumnName="ERZEUGTAMUM"
         targetTableID="0" valueMode="SetIfCreated"/>
        <timeStampColumn targetTableName="OMDSBENUTZER" targetColumnName="GEAENDERTAMUM"
         targetTableID="0" valueMode="Overwrite"/>
        <userIDColumn targetTableName="OMDSBENUTZER" targetColumnName="USERID"
         targetTableID="0" valueMode="Key"/>
    </variableColumns>
```

Next tag which we use is <constantColumns> tag. This tag defines constant columns, which has the constant values and defines the value.

```
<constantColumns>
        <constantColumn targetTableName="OMDSBENUTZER" targetColumnName="GESCHLECHTOID"
         targetTableID="0" valueMode="Overwrite" constantValue="2"/>
        <constantColumn targetTableName="OMDSBENUTZER" targetColumnName="ERZEUGTVONOID"
         targetTableID="0" valueMode="SetIfCreated" constantValue="1"/>
        <constantColumn targetTableName="OMDSBENUTZER" targetColumnName="GEAENDERTVONOID"
         targetTableID="0" valueMode="Overwrite" constantValue="1"/>
    </constantColumns>
```

If you want, you can add <counterColumns> tag for auto increment columns. <counterColumn> tag defines simple auto increment columns with different counterName for all columns in one importDefinition job. If you wish to con-

tinue using the same counter values in next importDefinition jog, just give the same name to the counterColumn in next importDefinition job.

SubCounterColumn inserts auto increment values in counterColumn depending on values in keyColumns. For each different value of key columns (or combination of keyColumns - in case of more key columns in one subCounter-Column element) Octopus inserts auto increment values beginning from counterStartValue.

```
    <counterColumns counterTableName="counterTable" counterNameColumn="counteName" counterValueColum
         <counterColumn counterName="counter1" counterStartValue="100" counterIncrement="1" targetTa
            targetTableID="0" targetColumnName"NUMMER" valueMode="SetIfCreated" counterStartValueRes
         <subCounterColumn constantValue="myVal3" targetTableName="OMDSBENUTZER" targetTableID="0"
            targetColumnName="SUBNUMMER" valueMode="Overwrite"/>
            <subCounterKeyColumn keyColumnName="GESCHLECHTOID"/>
         </subCounterColumn>
    </counterColumns>
```

Next tag which we use in this example is <table> tag. It defines target tables:

- 'name'- defines the name of the target table,
- 'insert' logic - if it is true, data will be inserted, otherwise, the table is going to be used only as part of a relations,
- 'tableMode' - defines Query or Cache mode,
- oidLogic - if it is true, target table has oid column, otherwise it doesn't.

```
    <tables>
        <table tableName="OMDSBENUTZER" tableID="0" insert="true" tableMode="Query"
         oidLogic="true"/>
    </tables>
```

At the end of XML file, we just close open tags: <importDefinition> and <loaderJob> .

```
</importDefinition>
</loaderJob>
```

Note

Detailed information about this tag (<ImportDefinition>), you can find in 'Octopus Loader application' chapter.

This is the end of simple XML without relation columns.

The most complicated tag is relation tag, so it will be discussed more detailed in the next chapter.

Relation Colulmns

If we want to create relations we have to notice a few parameters:

- how many target tables, which establish relations, are in loading process,
- does the values in relation columns changing during the loading process, and
- are there any 'Key' column, which are not 'oid' columns, in both tables connected with relations.

Generally speaking in relations we make connections between target tables. Relation tag has source table name, source table column name, target table name and target table column name. Only thing to do is to set this attributes to appropriate values and relation is done.

Problem is how Octopus will know which row from source table to connect with appropriate row from target table?

Example 11.1. Example of simple relations tag

The simplest problem is: There is only one source table, which is spread on two target tables. XML will have one <importDefinition> tag.

There will be two <table> tags and <valueColumn> tags will be as many as target columns in both target tables are.

There will be at least two 'key' valueMode, for each target table at least one.

There will be one relation tag where source and target tables will be tables from the target database. Source and target columns will be appropriate columns in these tables.

Example 11.2. Example of complex relations tag

Next example is complicated one with different kind of relations. This example is continuing from the example in section "Simple XML File without Relation Columns".

This <importDefinition> tag has two relation tags. These relations are between two target tables:

- the first one is between source table OMDSBENUTZER and target table OMDSSTAMMGESCHLECH (value from oid column (where USERID is sort column) from source table is set into ERZEUGTVONOID column in target table), and
- the second one is between source table OMDSBENUTZER and target table OMDSSTAMMGESCHLECH (value from oid column (where USERID is sort column) from source table is set into GEAENDERTVONOID column in target table).

Tabe:	*OMDSBENUTZER (source)*	*OMDSSTAMMGESCHLECH (target)*
Columns:	*OID (relation1-source, relation2-source)*	*ERZEUGTVONOID* (relation1-target)
		GEAENDERTVONOID (relation2-target)

```
<importDefinition name="OMDSSTAMMGESCHLECHT" tableName="OMDSSTAMMGESCHLECHT"
   logMode="normal" objectIDIncrement="1" objectIDTableName="oidAdminData"
   objectIDColumnName="myNext">
    <valueColumns>
        <valueColumn sourceColumnName="KEYVALUE"
         targetTableName="OMDSSTAMMGESCHLECHT" targetColumnName="KEYVALUE"
         targetTableID="0" valueMode="Key"/>
        <valueColumn sourceColumnName="KURZBEZ"
         targetTableName="OMDSSTAMMGESCHLECHT" targetColumnName="KURZBEZ"
         targetTableID="0" valueMode="Overwrite"/>
        <valueColumn sourceColumnName="LANGBEZ"
         targetTableName="OMDSSTAMMGESCHLECHT" targetColumnName="LANGBEZ"
         targetTableID="0" valueMode="Overwrite"/>
    </valueColumns>
    <variableColumns>
        <timeStampColumn targetTableName="OMDSSTAMMGESCHLECHT"
         targetColumnName="ERZEUGTAMUM" targetTableID="0" valueMode="SetIfCreated"/>
        <timeStampColumn targetTableName="OMDSSTAMMGESCHLECHT"
         targetColumnName="GEAENDERTAMUM" targetTableID="0" valueMode="Overwrite"/>
```

```
        <userIDColumn targetTableName="OMDSBENUTZER" targetColumnName="USERID"
         targetTableID="0" valueMode="Key"/>
        <userIDColumn targetTableName="OMDSBENUTZER" targetColumnName="USERID"
         targetTableID="1" valueMode="Key"/>
    </variableColumns>
    <relationColumns>
        <relationColumn relationSourceTableName="OMDSBENUTZER"
         relationSourceTableID="0" relationTargetTableName="OMDSSTAMMGESCHLECHT"
         relationTargetColumnName="ERZEUGTVONOID" relationTargetTableID="0"
         relationMode="SetIfCreated"/>
        <relationColumn relationSourceTableName="OMDSBENUTZER"
         relationSourceTableID="1" relationTargetTableName="OMDSSTAMMGESCHLECHT"
         relationTargetColumnName="GEAENDERTVONOID" relationTargetTableID="0"
         relationMode="Overwrite"/>
    </relationColumns>
    <tables>
        <table tableName="OMDSBENUTZER" tableID="0" insert="false" tableMode="Cache"
         oidLogic="true"/>
        <table tableName="OMDSBENUTZER" tableID="1" insert="false" tableMode="Cache"
         oidLogic="true"/>
        <table tableName="OMDSSTAMMGESCHLECHT" tableID="0" insert="true"
         tableMode="Query" oidLogic="true"/>
    </tables>
  </importDefinition>
```

Next <importDefinition> tag has three relation tags, difference between the one above is that the relationMode is ‘Key’:

- the first relation tag means that the sort columns in target table PERSISTENTTREETYPE are: KEYVALUE and OID,
- In second relation tag, sort columns in target table PERSISTENTTREETYPE are: KEYVALUE and OID,
- and in third relation tag, sort column in target table PERSISTENTTREE is KEYVALUE.

Table:	*PERSISTENTTREETYPE (relation1-source, relation2-source)*	*PERSISTENTTREE (relation1-target, relation2-target, relation3-source, target)*
Columns:	*OID (relation1-source, relation1-sort, relation2-source, relation2-sort)*	*OID (relation3-source, relation3-sort)*
	KEYVALUE (relation1-sort, relation2-sort)	*PARENTTREEOID (relation3-target)*
		PERSISTENTTREETYPEOID (relation1-target, relation2-target)
		KEYVALUE (relation3-sort)

```
<importDefinition name="PARENTTREE" tableName="PERSISTENTTREE" logMode="normal">
    <valueColumns>
        <valueColumn sourceColumnName="KEYVALUE" targetTableName="PERSISTENTTREE"
         targetColumnName="KEYVALUE" targetTableID="0" valueMode="Key"/>
        <valueColumn sourceColumnName="TYPEKEY" targetTableName="PERSISTENTTREETYPE"
         targetColumnName="KEYVALUE" targetTableID="1" valueMode="Key"/>
        <valueColumn sourceColumnName="PARENTKEY" targetTableName="PERSISTENTTREE"
         targetColumnName="KEYVALUE" targetTableID="2" valueMode="Key"/>
        <valueColumn sourceColumnName="PARENTTYPEKEY"
         targetTableName="PERSISTENTTREETYPE" targetColumnName="KEYVALUE"
         targetTableID="3" valueMode="Key"/>
    </valueColumns>
    <relationColumns>
        <relationColumn relationSourceTableName="PERSISTENTTREETYPE"
         relationSourceTableID="3" relationTargetTableName="PERSISTENTTREE"
```

```
         relationTargetColumnName="PERSISTENTTREETYPEOID" relationTargetTableID="2"
         relationMode="Key"/>
        <relationColumn relationSourceTableName="PERSISTENTTREETYPE"
         relationSourceTableID="1" relationTargetTableName="PERSISTENTTREE"
         relationTargetColumnName="PERSISTENTTREETYPEOID" relationTargetTableID="0"
         relationMode="Key"/>
        <relationColumn relationSourceTableName="PERSISTENTTREE"
         relationSourceTableID="2" relationTargetTableName="PERSISTENTTREE"
         relationTargetColumnName="PARENTTREEOID" relationTargetTableID="0"
         relationMode="Overwrite"/>
    </relationColumns>
    <tables>
        <table tableName="PERSISTENTTREETYPE" tableID="1" insert="false"
         tableMode="Cache" oidLogic="true"/>
        <table tableName="PERSISTENTTREE" tableID="0" insert="false"
         tableMode="Cache" oidLogic="true"/>
        <table tableName="PERSISTENTTREETYPE" tableID="3" insert="false"
         tableMode="Cache" oidLogic="true"/>
        <table tableName="PERSISTENTTREE" tableID="2" insert="false"
         tableMode="Cache" oidLogic="true"/>
    </tables>
  </importDefinition>
```

Next two <importDefinitions> tags are example of deep relations. In this cases, first must be done first level of relation than the second one.

- First <importDefinition> tag has one relation. Source table is GENERICENUMTYPEDEFINITION source column is OID target table is GENERICENUMVALUEDEFINITION target column name is ENUMTYPEDEFINITIONOID and sort column is KEYVALUE.
- Second <importDefinition> tag has 4 relations.

Table:	*GENERICENUMTYPEDEFINITION (relation1-source, relation2-source)*	*GENERICENUMVALUEDEFINITION (relation1-target, relation2-target, relation3-source, relation4-source)*	*GENERICSTATETRANSITION (relation3-target, relation4-target)*
Columns:	*OID (relation1-source, relation1-sort, relation2-source, relation2-sort)* *KEYVALUE (relation1-sort, relation2-sort)*	*OID (relation3-source, relation3-sort, relation4-source, relation4-sort)* *ENUMTYPEDEFINITIONOID (relation1-target, relation2-target)* *KEYVALUE (relation3-sort, relation4-sort)*	*ENUMVALUEDEFINITIONFROMOID (relation3-target, relation4-target)*

```
<importDefinition name="GENERICENUMVALUEDEFINITION"
   tableName="GENERICENUMVALUEDEFINITION" logMode="normal" objectIDIncrement="1"
   objectIDTableName="oidAdminData" objectIDColumnName="myNext">
    <valueColumns>
        <valueColumn sourceColumnName="ENUMTYPEDEFINITIONKEY"
         targetTableName="GENERICENUMTYPEDEFINITION" targetColumnName="KEYVALUE"
         targetTableID="0" valueMode="Key"/>
        <valueColumn sourceColumnName="BEGINSTATE"
         targetTableName="GENERICENUMVALUEDEFINITION" targetColumnName="BEGINSTATE"
         targetTableID="0" valueMode="Overwrite"/>
        <valueColumn sourceColumnName="ENDSTATE"
         targetTableName="GENERICENUMVALUEDEFINITION" targetColumnName="ENDSTATE"
```

```
        targetTableID="0" valueMode="Overwrite"/>
      <valueColumn sourceColumnName="LONGNAME"
        targetTableName="GENERICENUMVALUEDEFINITION" targetColumnName="LONGNAME"
        targetTableID="0" valueMode="Overwrite"/>
      <valueColumn sourceColumnName="SHORTNAME"
        targetTableName="GENERICENUMVALUEDEFINITION" targetColumnName="SHORTNAME"
        targetTableID="0" valueMode="Overwrite"/>
      <valueColumn sourceColumnName="KEYVALUE"
        targetTableName="GENERICENUMVALUEDEFINITION" targetColumnName="KEYVALUE"
        targetTableID="0" valueMode="Key"/>
      <valueColumn sourceColumnName="XMLNAME"
        targetTableName="GENERICENUMVALUEDEFINITION" targetColumnName="XMLNAME"
        targetTableID="0" valueMode="Overwrite"/>
  </valueColumns>
  <relationColumns>
      <relationColumn relationSourceTableName="GENERICENUMTYPEDEFINITION"
        relationSourceTableID="0"
        relationTargetTableName="GENERICENUMVALUEDEFINITION"
        relationTargetColumnName="ENUMTYPEDEFINITIONOID" relationTargetTableID="0"
        relationMode="Key"/>
  </relationColumns>
  <tables>
      <table tableName="GENERICENUMTYPEDEFINITION" tableID="0" insert="false"
        tableMode="Cache" oidLogic="true"/>
      <table tableName="GENERICENUMVALUEDEFINITION" tableID="0" insert="true"
        tableMode="Cache" oidLogic="true"/>
  </tables>
</importDefinition>

<importDefinition name="GENERICSTATETRANSITION" tableName="GENERICSTATETRANSITION"
 logMode="normal" objectIDIncrement="1" objectIDTableName="oidAdminData"
 objectIDColumnName="myNext">
  <valueColumns>
      <valueColumn sourceColumnName="ENUMTYPEDEFINITIONKEY"
        targetTableName="GENERICENUMTYPEDEFINITION" targetColumnName="KEYVALUE"
        targetTableID="0" valueMode="Key"/>
      <valueColumn sourceColumnName="ENUMTYPEDEFINITIONKEY"
        targetTableName="GENERICENUMTYPEDEFINITION" targetColumnName="KEYVALUE"
        targetTableID="1" valueMode="Key"/>
      <valueColumn sourceColumnName="ENUMVALUEDEFINITIONFROMKEY"
        targetTableName="GENERICENUMVALUEDEFINITION" targetColumnName="KEYVALUE"
        targetTableID="0" valueMode="Key"/>
      <valueColumn sourceColumnName="ENUMVALUEDEFINITIONTOKEY"
        targetTableName="GENERICENUMVALUEDEFINITION" targetColumnName="KEYVALUE"
        targetTableID="1" valueMode="Key"/>
  </valueColumns>
  <relationColumns>
      <relationColumn relationSourceTableName="GENERICENUMTYPEDEFINITION"
        relationSourceTableID="0"
        relationTargetTableName="GENERICENUMVALUEDEFINITION"
        relationTargetColumnName="ENUMTYPEDEFINITIONOID" relationTargetTableID="0"
        relationMode="Key"/>
      <relationColumn relationSourceTableName="GENERICENUMTYPEDEFINITION"
        relationSourceTableID="1"
        relationTargetTableName="GENERICENUMVALUEDEFINITION"
        relationTargetColumnName="ENUMTYPEDEFINITIONOID" relationTargetTableID="1"
        relationMode="Key"/>
      <relationColumn relationSourceTableName="GENERICENUMVALUEDEFINITION"
        relationSourceTableID="0" relationTargetTableName="GENERICSTATETRANSITION"
        relationTargetColumnName="ENUMVALUEDEFINITIONFROMOID"
        relationTargetTableID="0" relationMode="Key"/>
      <relationColumn relationSourceTableName="GENERICENUMVALUEDEFINITION"
        relationSourceTableID="1" relationTargetTableName="GENERICSTATETRANSITION"
        relationTargetColumnName="ENUMVALUEDEFINITIONTOOID"
        relationTargetTableID="0" relationMode="Key"/>
  </relationColumns>
  <tables>
      <table tableName="GENERICENUMTYPEDEFINITION" tableID="0" insert="false"
        tableMode="Cache" oidLogic="true"/>
      <table tableName="GENERICENUMTYPEDEFINITION" tableID="1" insert="false"
        tableMode="Cache" oidLogic="true"/>
      <table tableName="GENERICENUMVALUEDEFINITION" tableID="0" insert="false"
```

```
         tableMode="Cache" oidLogic="true"/>
        <table tableName="GENERICENUMVALUEDEFINITION" tableID="1" insert="false"
         tableMode="Cache" oidLogic="true"/>
        <table tableName="GENERICSTATETRANSITION" tableID="0" insert="true"
         tableMode="Cache" oidLogic="true"/>
    </tables>
</importDefinition>
```

Examples

1. **LoadeExample1** (cmd or sh) - This example shows using relations in <importDefinition> tags.

2. **LoadeExample2** (cmd or sh) - This example shows how to use variables in configuration xml files to substitute values in source tables.

3. **LoadeExample3** (cmd or sh) - This example shows using SQL statements instead of source tables.

4. **LoadeExample4** (cmd or sh) - This example shows using *csv* jdbc driver for loading data into csv target tables.

5. **LoadXMLExample** (cmd or sh) - This example shows using XmlDriver,JDBC driver which use XML file as source data. Example loads data from csv files (same files as in LoadExample1.cmd) into XML file, which is placed in ObjectLoader\LoaderOutput\testdatabase.xml. Data in XML file are placed in form used by XmlDriver. See readme.html (OCTOPUS_HOME\modules\XMLjdbc\readme.html) for details about XmlDriver.

6. **DataCleaningExample** (cmd or sh) - This example shows using advanced features of Octopus Loader application, named "*Data Cleaning*". For more information see HowToStartExamples.txt file in OCTOPUS_HOME/output/tdt-{version}/examples directory.

7. **LoadAutoMapExample** (cmd or sh) - This example shows using advance features "autoMap" in Octopus application. In this example, we just map few columns in InportDefinition file, and OctopusLoader is mapped all other columns.

8. **CsvSplitFilesExample** (cmd or sh) - This example demonstrates how you can limit maximum size of files, and this is new feature of csv driver.

9. **LoadBinaryExample** (cmd or sh) - This example shows copying Blob object form one database to another, and transformation of String object into Blob object.

10. **Start sample Octopus application as a ant task**

    - To start example of OctopusLoader application, LoadExample1, using ant task please run start_using_ant (cmd or sh) script. Ant uses build.xml file in the same directory to execute Octopus.

11. **Start sample Octopus applications as an extension of JUnit test case**

    - There are three classes to demonstrate how to use Octopus as a test case (for manipulating with databases). To start these examples just vstart LoadTest with one of the following arguments:

      -LoaderTest

      -LoaderTest2

      -LoaderTest3

      For running these examples, you need to set junit.jar into you classpath. (JUnit version 3.8 is placed in lib

directory of Octopus).

12. **Start sample Octopus applications that works with TOS server**

    - To start example TOS-OctopusLoader application please run TOSExample (cmd or sh) script.

      Octopus loads data using TOS syntax (see documentation).

'Backup' and 'Restore'

These examples show you how you can backup or restore database of your choice.

- **BackupExample** (cmd or sh) - This example shows using backup scripts. HypersonicSQL database is backuped, in Csv database.
- **BackupExampleAPI** (cmd or sh) - This example shows using backup API. HypersonicSQL database is backuped, in Csv database.
- **Backup_using_ant** (cmd or sh) - This example shows using ant task for backup. HypersonicSQL database is backuped, in Csv database.
- **RestoreExample** (cmd or sh) - This example shows using restore scripts. Csv database is restored, in HypersonicSQL database.
- **RestoreExampleAPI** (cmd or sh) - This example shows using restore API. Csv database is restored, in HypersonicSQL database.
- **Restore_using_ant** (cmd or sh) - This example shows using ant task for restore. Csv database is restored, in HypersonicSQL database.

Example of LDAP Server

This example shows you haw you can use Octopus application to transfer data from source database (LDAP server) to target database (MSQL server).

First, you must create LoaderJob.olj file, which should look like this:

```
e.g.
<?xml version="1.0" encoding="UTF-8" ?>
<loaderJob logMode="normal" objectIDIncrement="1" objectIDTableName="objectid" objectIDColumnName="n
    <jdbcDefaultParameters>
        <jdbcSourceParameters dbVendor="Octetstring" driverName="jdbc-ldap">
        <jdbcSourceParameter name="JdbcDriver" value="com.octetstring.jdbcLdap.sql.JdbcLdapDriver"/>
        <jdbcSourceParameter name="Connection.Url" value="jdbc:ldap://localhost/?SEARCH_SCOPE:=subTr
        <jdbcSourceParameter name="User" value="cn=Manager,dc=proZone,dc=local"/>
        <jdbcSourceParameter name="Password" value="secret"/>
        </jdbcSourceParameters>
        <jdbcTargetParameters dbVendor="MSQL" driverName="jTurbo">
        <jdbcTargetParameter name="JdbcDriver" value="com.newatlanta.jturbo.driver.Driver"/>
        <jdbcTargetParameter name="Connection.Url" value="jdbc:JTurbo://localhost:1433/Test"/>
        <jdbcTargetParameter name="User" value="sa"/>
        <jdbcTargetParameter name="Password" value=""/>
        </jdbcTargetParameters>
    </jdbcDefaultParameters>
```

```
<sql name="DropTables" onErrorContinue="true" commit="true" logMode="normal">
    <sqlStmt>
        <include href="ObjectLoader/sql/DropTables.sql" parse="text"/>
    </sqlStmt>
</sql>

<sql name="CreateTables" onErrorContinue="false" commit="true" logMode="normal">
    <sqlStmt>
        <include href="ObjectLoader/sql/CreateTables.sql" parse="text"/>
    </sqlStmt>
</sql>

<definitionInclude>
    <include href="ObjectLoader/xml/ImportDefinition.oli" parse="xml"/>
</definitionInclude>
</loaderJob>
```

Then, you must create appropriate sql and xml files, in appropriate directories.

Xml file (ImportDefinition.oli) should look like this:

```
e.g.
<definitionInclude>
    <importDefinition logMode="full" name="Test Jdbc Ldap" selectStatement="SELECT * FROM ou=katedra
        <valueColumns>
            <valueColumn sourceColumnName="sn" targetColumnName="prezime" targetTableID="0" targetTa
            <valueColumn sourceColumnName="objectClass_0" targetColumnName="objectClass_0" targetTab
            <valueColumn sourceColumnName="objectClass_1" targetColumnName="objectClass_1" targetTab
            <valueColumn sourceColumnName="objectClass_2" targetColumnName="objectClass_2" targetTab
            <valueColumn sourceColumnName="cn" targetColumnName="ime" targetTableID="0" targetTableN
            <valueColumn sourceColumnName="DN" targetColumnName="DN" targetTableID="0" targetTableNa
        </valueColumns>
        <tables>
            <table insert="true" oidLogic="false" tableID="0" tableMode="Query" tableName="Katedra"/
        </tables>
    </importDefinition>
</definitionInclude>
```

And Sql file (CreateTables.sql) should look like this:

```
e.g.
create table Katedra
(
ime varchar(100),
prezime varchar(100),
objectClass_0 varchar(100),
objectClass_1 varchar(100),
objectClass_2 varchar(100),
DN nvarchar(254)
);
```

When you are finished with creating files, you simply run Octopus application.

Example of Using C-JDBC driver

This example shows you haw you can use Octopus application to transfer data from source database (MSQL server) to target database (MSQL server), using C-JDBC driver.

First, you must create LoaderJob.olj file, which should look like this:

```
e.g.
<?xml version="1.0" encoding="UTF-8" ?>
<loaderJob logMode="normal" objectIDIncrement="1" objectIDTableName="objectid" objectIDColumnName="n
    <jdbcDefaultParameters>
        <jdbcSourceParameters dbVendor="MSQL" driverName="jTurbo">
        <jdbcSourceParameter name="JdbcDriver" value="com.newatlanta.jturbo.driver.Driver"/>
        <jdbcSourceParameter name="Connection.Url" value="jdbc:JTurbo://localhost:1433/Test"/>
        <jdbcSourceParameter name="User" value="sa"/>
```

```
        <jdbcSourceParameter name="Password" value=""/>
        </jdbcSourceParameters>
        <jdbcTargetParameters dbVendor="CJDBC" driverName="cjdbc">
        <jdbcTargetParameter name="JdbcDriver" value="org.objectweb.cjdbc.driver.Driver"/>
        <jdbcTargetParameter name="Connection.Url" value="jdbc:cjdbc://localhost:/cjdbc"/>
        <jdbcTargetParameter name="User" value="sa"/>
        <jdbcTargetParameter name="Password" value="sa"/>
        </jdbcTargetParameters>
    </jdbcDefaultParameters>

    <sql name="DropTable" onErrorContinue="true" commit="true" logMode="normal">
        <sqlStmt>
            <include href="ObjectLoader/sql/DropTables.sql" parse="text"/>
        </sqlStmt>
    </sql>

    <sql name="CreateTables" onErrorContinue="false" commit="true" logMode="normal">
        <sqlStmt>
            <include href="ObjectLoader/sql/CreateTables.sql" parse="text"/>
        </sqlStmt>
    </sql>


    <definitionInclude>
        <include href="ObjectLoader/xml/ImportDefinition.oli" parse="xml"/>
    </definitionInclude>

</loaderJob>
```

Then, you must create appropriate sql and xml files, in appropriate directories.

Xml file (ImportDefinition.oli) should look like this:

```
e.g.
<definitionInclude>
    <importDefinition logMode="normal" name="Activities" tableName="Activities">
        <valueColumns>
            <valueColumn sourceColumnName="oid" targetColumnName="ObjectId" targetTableID="0" target
            <valueColumn sourceColumnName="Id" targetColumnName="Id" targetTableID="0" targetTableNa
            <valueColumn sourceColumnName="ActivityDefinitionId" targetColumnName="ActivityDefinitio
            <valueColumn sourceColumnName="Process" targetColumnName="Process" targetTableID="0" tar
            <valueColumn sourceColumnName="IsAccepted" targetColumnName="IsAccepted" targetTableID="
            <valueColumn sourceColumnName="State" targetColumnName="State" targetTableID="0" targetT
            <valueColumn sourceColumnName="BlockActivity" targetColumnName="BlockActivity" targetTab
            <valueColumn sourceColumnName="Priority" targetColumnName="Priority" targetTableID="0" t
            <valueColumn sourceColumnName="Name" targetColumnName="Name" targetTableID="0" targetTab
            <valueColumn sourceColumnName="Description" targetColumnName="Description" targetTableID
        </valueColumns>
        <constantColumns>
            <constantColumn constantValue="0" targetColumnName="ObjectVersion" targetTableID="0" tar
        </constantColumns>
        <tables>
            <table insert="true" oidLogic="false" tableID="0" tableMode="Query" tableName="Activitie
        </tables>
    </importDefinition>

</definitionInclude>
```

And Sql file (CreateTables.sql) should look like this:

```
e.g.
create table Activities
(
    ObjectId DECIMAL (19,0) NOT NULL,
    ObjectVersion INTEGER ,
    Id VARCHAR (254) NOT NULL ,
    ActivityDefinitionId VARCHAR (254) NOT NULL ,
    Process DECIMAL (19,0) NOT NULL ,
    IsAccepted BOOL NOT NULL ,
    State DECIMAL (19,0) NOT NULL ,
    BlockActivity DECIMAL (19,0),
```

```
    Priority INTEGER ,
    Name VARCHAR (254) ,
    Description VARCHAR (254)
);
```

After that you must create configuration file for c-jdbc driver (ExampleCjdbc.xml), which should look like this:

```
e.g.
<?xml version="1.0" encoding="UTF8"?>
<!DOCTYPE C-JDBC PUBLIC "-//ObjectWeb//DTD C-JDBC 1.0a4//EN" "http://c-jdbc.objectweb.org/dtds/c-jdb

<C-JDBC>
  <VirtualDatabase name="cjdbc">
    <AuthenticationManager>
      <AdminLogin aLogin="admin" aPassword="admin"/>
      <VirtualLogin vLogin="sa" vPassword="sa">
        <RealLogin backendName="myDb1" rLogin="sa" rPassword=""/>
        <RealLogin backendName="myDb2" rLogin="sa" rPassword=""/>
      </VirtualLogin>
    </AuthenticationManager>
    <DatabaseBackend name="myDb1" driver="org.postgresql.Driver" url="jdbc:postgresql://grunf/OdbcTe
      <ConnectionManager vLogin="sa">
        <FailFastPoolConnectionManager poolSize="40"/>
      </ConnectionManager>
    </DatabaseBackend>
    <DatabaseBackend name="myDb2" driver="org.postgresql.Driver" url="jdbc:postgresql://grunf/mydb">
      <ConnectionManager vLogin="sa">
        <FailFastPoolConnectionManager poolSize="40"/>
      </ConnectionManager>
    </DatabaseBackend>

    <RequestManager>
      <RequestScheduler>
         <RAIDb-1Scheduler level="pessimisticTransaction"/>
      </RequestScheduler>
      <LoadBalancer>
         <RAIDb-1>
        <RAIDb-1-RoundRobin/>
         </RAIDb-1>
      </LoadBalancer>
    </RequestManager>

  </VirtualDatabase>

</C-JDBC>
```

And than, in C_JDBC_HOME\config directory you must edit controller.xml file. In this file you must set path to configuration file for c-jdbc driver (ExampleCjdbc.xml).

```
e.g.

<C-JDBC-CONTROLLER>
<Controller port="25322">
<JmxSettings enabled="false"/>
    <VirtualDatabase virtualName="cjdbc" configFile="c:\Users\Octopus\ObjectLoader\ExampleCjdbc.xml"
</Controller>
</C-JDBC-CONTROLLER>
```

When you are finished with creating files, you simply run controller.but (C_JDBC_HOME\bin) and Octoups application.

Chapter 12. Tools in Octopus

P6spy

P6Spy is an open source framework for applications that intercept and optionally modify database statements. The P6Spy distribution includes the following modules:

1. P6Log

P6Log intercepts and logs the database statements of any application that uses JDBC. This application is particularly useful for developers to monitor the SQL statements produced by EJB servers, enabling the developer to write code that achieves maximum efficiency on the server. The P6Log module is enabled by default. Disable or enable the P6Log module by editing the **spy.properties** (this file is placed in OCTOPUS_HOME/examples) configuration file. If the module is commented out, it is not loaded, and the functionality is not available. If the module is not commented out, the functionality is available.

The applicable portion of the spy.properties file follows:

```
#################################################################
    # MODULES #
    # #
    # Modules provide the P6Spy functionality. If a module, such #
    # as module_log is commented out, that functionality will not #
    # be available. If it is not commented out (if it is active), #
    # the functionality will be active. #
    # #
    # Values set in Modules cannot be reloaded using the #
    # reloadproperties variable. Once they are loaded, they remain #
    # in memory until the application is restarted. #
    # #
    #################################################################
module.log=com.p6spy.engine.logging.P6LogSpyDriver
#module.outage=com.p6spy.engine.outage.P6OutageSpyDriver
```

The following are P6Log-specific properties:

executionthreshold - This feature only logs queries that take longer than a specified threshold to execute.

2. P6Outage

P6Outage detects long-running statements that may be indicative of a database outage problem and will log any statement that surpasses the configurable time boundary during its execution. P6Outage was designed to minimize any logging performance penalty by logging only long running statements.The P6Outage module is disabled by default. Disable or enable the P6Outage module by editing the spy.properties configuration file. If the module is commented out, it is not loaded, and the functionality is not available. If the module is not commented out, the functionality is available. The applicable portion of the spy.properties file follows:

```
#################################################################
    # MODULES #
    # #
    # Modules provide the P6Spy functionality. If a module, such #
    # as module_log is commented out, that functionality will not #
    # be available. If it is not commented out (if it is active), #
    # the functionality will be active. #
    # #
    # Values set in Modules cannot be reloaded using the #
    # reloadproperties variable. Once they are loaded, they remain #
    # in memory until the application is restarted. #
    # #
    #################################################################
#module.log=com.p6spy.engine.logging.P6LogSpyDriver
module.outage=com.p6spy.engine.outage.P6OutageSpyDriver
```

The following are P6Outage-specific properties:

outagedetection - *t*his feature detects long-running statements that may be indicative of a database outage problem. When enabled, it logs any statement that surpasses the configurable time boundary during its execution. No other statements are logged except the long-running statements.

outagedetectioninterval - The interval property is the boundary time set in seconds. For example, if set to 2, any statement requiring at least 2 seconds is logged. The same statement will continue to be logged for as long as it executes. So, if the interval is set to 2 and a query takes 11 seconds, it is logged 5 times (at the 2, 4, 6, 8, 10-second intervals).

Example LoadExampleP6spy.cmd (LoadExampleP6spy.sh fo Linux) use P6spy. This is the spy.properties for this example:

```
#################################################################
# P6Spy Options File                                            #
# See documentation for detailed instructions                   #
#################################################################

#################################################################
# MODULES                                                       #
#                                                               #
# Modules provide the P6Spy functionality.  If a module, such   #
# as module_log is commented out, that functionality will not   #
# be available.  If it is not commented out (if it is active),  #
# the functionality will be active.                             #
#                                                               #
# Values set in Modules cannot be reloaded using the            #
# reloadproperties variable.  Once they are loaded, they remain #
# in memory until the application is restarted.                 #
#                                                               #
#################################################################

module.log=com.p6spy.engine.logging.P6LogFactory
#module.outage=com.p6spy.engine.outage.P6OutageFactory

#################################################################
# REALDRIVER(s)                                                 #
#                                                               #
# In your application server configuration file you replace the #
# "real driver" name with com.p6spy.engine.P6SpyDriver. This is #
# where you put the name of your real driver P6Spy can find and #
# register your real driver to do the database work.            #
#                                                               #
# If your application uses several drivers specify them in      #
# realdriver2, realdriver3.  See the documentation for more     #
# details.                                                      #
#                                                               #
# Values set in REALDRIVER(s) cannot be reloaded using the      #
# reloadproperties variable.  Once they are loaded, they remain #
# in memory until the application is restarted.                 #
#                                                               #
#################################################################

# oracle driver
# realdriver=oracle.jdbc.driver.OracleDriver

# mysql Connector/J driver
# realdriver=com.mysql.jdbc.Driver

# informix driver
# realdriver=com.informix.jdbc.IfxDriver

# ibm db2 driver
# realdriver=COM.ibm.db2.jdbc.net.DB2Driver

# the mysql open source driver
# realdriver=org.gjt.mm.mysql.Driver
```

```
#specifies another driver to use
realdriver=org.hsqldb.jdbcDriver

#specifies a third driver to use
realdriver3=


#the DriverManager class sequentially tries every driver that is
#registered to find the right driver.  In some instances, it's possible to
#load up the realdriver before the p6spy driver, in which case your connections
#will not get wrapped as the realdriver will "steal" the connection before
#p6spy sees it.  Set the following property to "true" to cause p6spy to
#explicitily deregister the realdrivers
deregisterdrivers=false

#################################################################
# P6LOG SPECIFIC PROPERTIES                                     #
#################################################################
# no properties currently available

#################################################################
# EXECUTION THRESHOLD PROPERTIES                                #
#################################################################
# This feature applies to the standard logging of P6Spy.       #
# While the standard logging logs out every statement          #
# regardless of its execution time, this feature puts a time   #
# condition on that logging.  Only statements that have taken  #
# longer than the time specified (in milliseconds) will be     #
# logged.  This way it is possible to see only statements that #
# have exceeded some high water mark.                          #
# This time is reloadable.                                     #
#
# executionthreshold=integer time (milliseconds)
#
executionthreshold=

#################################################################
# P6OUTAGE SPECIFIC PROPERTIES                                  #
#################################################################
# Outage Detection
#
# This feature detects long-running statements that may be indicative of
# a database outage problem. If this feature is turned on, it will log any
# statement that surpasses the configurable time boundary during its execution.
# When this feature is enabled, no other statements are logged except the long
# running statements. The interval property is the boundary time set in seconds.
# For example, if this is set to 2, then any statement requiring at least 2
# seconds will be logged. Note that the same statement will continue to be logged
# for as long as it executes. So if the interval is set to 2, and the query takes
# 11 seconds, it will be logged 5 times (at the 2, 4, 6, 8, 10 second intervals).
#
# outagedetection=true|false
# outagedetectioninterval=integer time (seconds)
#
outagedetection=false
outagedetectioninterval=

#################################################################
# COMMON PROPERTIES                                             #
#################################################################

# filter what is logged
filter=false

# comma separated list of tables to include when filtering
include     =
# comma separated list of tables to exclude when filtering
exclude     =

# sql expression to evaluate if using regex filtering
sqlexpression =
```

```
# turn on tracing
autoflush   = true

# sets the date format using Java's SimpleDateFormat routine
dateformat=

#list of categories to explicitly include
includecategories=

#list of categories to exclude: error, info, batch, debug, statement,
#commit, rollback and result are valid values
excludecategories=info,debug,result,batch


#allows you to use a regex engine or your own matching engine to determine
#which statements to log
#
#stringmatcher=com.p6spy.engine.common.GnuRegexMatcher
#stringmatcher=com.p6spy.engine.common.JakartaRegexMatcher
stringmatcher=

# prints a stack trace for every statement logged
stacktrace=false
# if stacktrace=true, specifies the stack trace to print
stacktraceclass=

# determines if property file should be reloaded
reloadproperties=false
# determines how often should be reloaded in seconds
reloadpropertiesinterval=60

#if=true then url must be prefixed with p6spy:
useprefix=false

#specifies the appender to use for logging
#appender=com.p6spy.engine.logging.appender.Log4jLogger
#appender=com.p6spy.engine.logging.appender.StdoutLogger
appender=com.p6spy.engine.logging.appender.FileLogger

# name of logfile to use, note Windows users should make sure to use forward slashes in their pathna
logfile=spy.log

# append to  the p6spy log file.  if this is set to false the
# log file is truncated every time.  (file logger only)
append=true

#The following are for log4j logging only
log4j.appender.STDOUT=org.apache.log4j.ConsoleAppender
log4j.appender.STDOUT.layout=org.apache.log4j.PatternLayout
log4j.appender.STDOUT.layout.ConversionPattern=p6spy - %m%n

#log4j.appender.CHAINSAW_CLIENT=org.apache.log4j.net.SocketAppender
#log4j.appender.CHAINSAW_CLIENT.RemoteHost=localhost
#log4j.appender.CHAINSAW_CLIENT.Port=4445
#log4j.appender.CHAINSAW_CLIENT.LocationInfo=true

log4j.logger.p6spy=INFO,STDOUT


#################################################################
# DataSource replacement                                        #
#                                                               #
# Replace the real DataSource class in your application server  #
# configuration with the name com.p6spy.engine.spy.P6DataSource,#
# then add the JNDI name and class name of the real             #
# DataSource here                                               #
#                                                               #
# Values set in this item cannot be reloaded using the          #
# reloadproperties variable.  Once it is loaded, it remains     #
# in memory until the application is restarted.                 #
```

```
#                                                                   #
#####################################################################
#realdatasource=/RealMySqlDS
#realdatasourceclass=com.mysql.jdbc.jdbc2.optional.MysqlDataSource

#####################################################################
# DataSource properties                                             #
#                                                                   #
# If you are using the DataSource support to intercept calls       #
# to a DataSource that requires properties for proper setup,        #
# define those properties here. Use name value pairs, separate      #
# the name and value with a semicolon, and separate the             #
# pairs with commas.                                                #
#                                                                   #
# The example shown here is for mysql                               #
#                                                                   #
#####################################################################
#realdatasourceproperties=port;3306,serverName;ibmhost,databaseName;mydb


#####################################################################
# JNDI DataSource lookup                                            #
#                                                                   #
# If you are using the DataSource support outside of an app         #
# server, you will probably need to define the JNDI Context         #
# environment.                                                      #
#                                                                   #
# If the P6Spy code will be executing inside an app server then     #
# do not use these properties, and the DataSource lookup will       #
# use the naming context defined by the app server.                 #
#                                                                   #
# The two standard elements of the naming environment are           #
# jndicontextfactory and jndicontextproviderurl. If you need        #
# additional elements, use the jndicontextcustom property.          #
# You can define multiple properties in jndicontextcustom,          #
# in name value pairs. Separate the name and value with a           #
# semicolon, and separate the pairs with commas.                    #
#                                                                   #
# The example shown here is for a standalone program running on     #
# a machine that is also running JBoss, so the JDNI context         #
# is configured for JBoss (3.0.4).                                  #
#                                                                   #
#####################################################################
#jndicontextfactory=org.jnp.interfaces.NamingContextFactory
#jndicontextproviderurl=localhost:1099
#jndicontextcustom=java.naming.factory.url.pkgs;org.jboss.nameing:org.jnp.interfaces

#jndicontextfactory=com.ibm.websphere.naming.WsnInitialContextFactory
#jndicontextproviderurl=iiop://localhost:900
```

Note: realdriver=org.hsqldb.jdbcDriver - is used in this example for target database

This is the real driver. In LoaderExampleP6spy.olj is fake driver :com.p6spy.engine.spy.P6SpyDriver.

This is the LoaderExampleP6spy.olj for this example:

```
<loaderJob logMode="normal" objectIDIncrement="1" objectIDTableName="objectid" objectIDColumnName="n
<jdbcDefaultParameters>
  <jdbcSourceParameters dbVendor = "csv">
    <jdbcSourceParameter name="JdbcDriver" value="org.relique.jdbc.csv.CsvDriver"/>
    <jdbcSourceParameter name="Connection.Url" value="jdbc:relique:csv:LoaderInput"/>
  </jdbcSourceParameters>
  <jdbcTargetParameters dbVendor="HypersonicSQL" driverName="hsql">
    <jdbcTargetParameter name="Password" value=""/>
    <jdbcTargetParameter name="Connection.Url" value="jdbc:hsqldb:LoaderOutput/TestDatabase/TestDB"/
    <jdbcTargetParameter name="JdbcDriver" value="com.p6spy.engine.spy.P6SpyDriver"/>
    <jdbcTargetParameter name="User" value="sa"/>
   </jdbcTargetParameters>
 </jdbcDefaultParameters>
 <sql name="CreateTables" onErrorContinue="false" commit="true">
  <sqlStmt>
```

```
   <include href="includes/CreateTables.sql" parse="text"/>
  </sqlStmt>
 </sql>
 <sql name="CreateOidDODS" onErrorContinue="false" commit="true">
  <sqlStmt>
   <include href="includes/CreateOidAdminData.sql" parse="text"/>
  </sqlStmt>
 </sql>
 <definitionInclude>
  <include href="BooksDefinition.oli" parse="xml"/>
 </definitionInclude>
 <sql name="CreateIndizes" logMode="normal" onErrorContinue="false" commit="true">
  <sqlStmt>
   <include href="includes/CreateIndex.sql" parse="text"/>
  </sqlStmt>
 </sql>
 <sql name="CreateForeign" logMode="normal" onErrorContinue="false" commit="true">
  <sqlStmt>
   <include href="includes/CreateIntegrity.sql" parse="text"/>
  </sqlStmt>
 </sql>
</loaderJob>
```

In **spy.log** will be all activities on target database. All activities will be traced by P6spy.

Documentation for p6spy [../extras/tools/p6spy/index.htm].

More information about P6spy available on www.p6spy.com [http://www.p6spy.com]