ETL Framework User Guide 3.2

Contents

Introduction

ETL Framework is a standalone Extract Transform Load (ETL) engine. It includes executables for all major platforms and can be easily integrated into other applications. The framework is free and open source.

This document gives an overview, installation tips and general information about the product. It is mainly concentrated on installation, configuration and integration.

Glossary

Term	Definition
ETL	Extract Transform Load. ETL is a process which involves: • Extracting data from outside sources • Transforming it to fit operational needs (which can include quality levels) • Loading it into the end target database or data source
ETL Scenario	A program in the declarative XML-based language which describes extract, transform and load steps of the ETL process
Inner ETL Scenario	ETL scenario included in other ETL scenario
ETL Framework	The set of classes and interfaces coded in Java which implement feature rich ETL engine. Includes multiple Toolsverse and third-party jar files
ETL Engine	Same as ETL Framework
Standalone ETL tool	A standalone program which executes one or multiple ETL Scenarios
Embedding	A way to integrate ETL framework into customer's application using open API (application programming interface)
Source	The data set to extract. Can be populated by executing SQL query or reading file-based sources such as Excel worksheet, text and XML files, etc
Destination	The load target. Can be a database table or file-based data set such as Excel worksheet, text and XML files, etc
Data Set	The in-memory representation of the database table or file-based data source such as Excel worksheet, text and XML files, etc
Connection	Either database connection or connection to the file-based data set such as Excel worksheet, text and XML files, etc.
Connector	A pluggable add-on which reads and writes data in the particular format.
Streaming	A way to copy data from the source to destination using very small memory footprint. Basically only current row (record) is stored in the memory
Mapping	A way to map a field in the source to the field in the destination
Automatic mapping	Field in the source is mapped to the field in the destination by name
Scenario variable	Input parameter
Destination variable	Data set field or calculated variable

How it Works

Short version - ETL engine reads data sources, performs transformations and generates database-specific SQL code which is then executed within a transaction. If destination is not a SQL database, ETL engine uses pluggable connectors to write data in the designated format. The ETL scenarios are written in the XML-based language, but it is also possible to create them as Java objects.

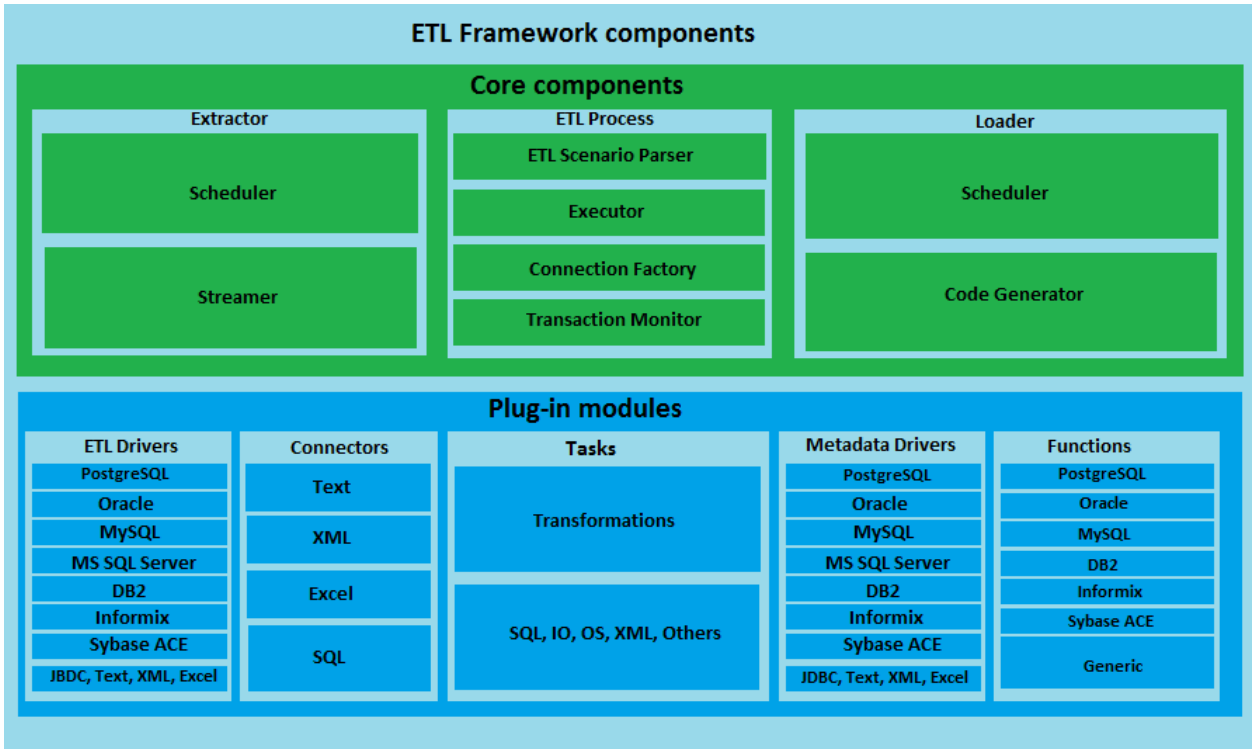


Figure 1: ETL Framework Components

The code generated by the ETL engine is specific to the database. For example for Oracle, it is PL/SQL code, for Microsoft – Transact-SQL, etc. The code is extremely efficient and supports a wide range of techniques from temporary tables and cursors to the native extract and bulk load. It's all done automatically, behind the scenes, by the translation layer and usually does not require any specific knowledge about the target database.

The ETL engine supports data streaming where "reading" and "writing" are combined in one operation. Basically it allows moving practically unlimited sets of data from the source to destination. It is also done automatically.

The high-level transformations such as de-duplication, pivoting, de-normalization, etc are all built in. There are also programmable transformations and validations.

The ETL engine supports multithreading at all levels: from extract to load to executing individual ETL scenarios.

Perhaps most important, the ETL engine is easily expandable. All core components (such as drivers, connectors, transformations, functions, code generators etc.) are dynamically loaded plug-in modules. It

is easy to add new or modify existing functionality. It is also easy to integrate it into your application by either embedding the ETL engine or running it in the client-server mode.

Please check out embedding ETL engine and configuring and running ETL in the client-server mode.

Features

	Details
Supported operating systems	• Windows, • OS X, • Linux/Unix
Supported databases	• Any JDBC and ODBC
Extended database support	• Oracle, • DB2, • MS SQL Server, • MySQL, • ProgreSQL, • Informix, • Sybase ASE
Supported data sources	• delimited text, • fixed length text, • Excel xls, • Excel xlsx, • XML, • XML with transformation • custom using pluggable connectors
ETL engine	• XML-based scenario language • Extract data from multiple sources and load into multiple destinations • All connectivity options are supported (jdbc, XML, XML transformation, text, Excel) • Stream unlimited data sets from the source to destination • All data types supported including CLOBs and BLOBs with automatic or manual conversion between source and destination databases (data sources) • Automatic and manual field's mapping • Extract and Load each data set in parallel with forks and joins • Inner scenarios with conditional and in-loop execution • Automatic table and index creation based on the source data set specification • Manual and automatic transactions management (commit intervals) • Per field functions in SQL and JavaScript • Support for automatic primary/foreign key generation with mapping to old primary/foreign key • Validation using JavaScript • Conditional sources and destinations • Conditional (IF-THEN-ELSE) execution • Automatic exception handling

	• Automatic Insert/Update/Delete/Merge • In-line SQL in scenarios • Pre/post/inline extract and load tasks • OS command execution • File based tasks (file system, ftp and sftp supported)
Transformations	• Regex transformation • XSL transformation • Transformation using JavaScript • Sorting • Transpose Matrix • Filtering • Remove Duplicates • Union • Join • Minus • Pivot • De-normalize
Oracle specific functionality	• Using sequences to generate primary keys • Full PLSQL support including anonymous SQL blocks, inner functions, procedures, named variables, etc. • Cursors as data sources • Extract using SQL*plus and load using SQL*loader (requires Oracle client) • Table copy using SQL*plus COPY command (requires Oracle client) • Support for MERGE, exception handling, date+time conversion, temporary tables
DB2 specific functionality	• Using sequences and auto-increment fields to generate primary keys • Full SQL PL support including functions, procedures, named variables, etc. • Cursors as data sources • Extract and load using SYSPROC.ADMIN_CMD • Support for MERGE, exception handling, date+time conversion, temporary tables
MS SQL Server specific functionality	• Using auto-increment fields to generate primary keys • Full Transact SQL support including functions, procedures, named variables, etc. • Cursors as data sources • Extract and load using BCP (requires MS SQL server client) • Support for exception handling, date+time conversion, temporary tables
MySQL specific functionality	• Using auto-increment fields to generate primary keys • Full MySql stored procedure language support including functions, procedures, named variables, etc. • Cursors as data sources • Extract using select INTO OUTFILE and Load using LOAD DATA • Support for exception handling, date+time conversion, temporary tables

PostreSQL specific functionality	• Using sequences and serial fields to generate primary keys • Full PL/pgSQL support including functions, named variables, etc. • Cursors as data sources • Extract and Load using COPY • Support for exception handling, date+time conversion, temporary tables
Informix specific functionality	• Using sequences and serial fields to generate primary keys • Full SPL support including functions, procedures, named variables, etc • Cursors as data sources • Extract and load using DBACCESS (requires Informix client) • Support for MERGE, exception handling, date+time conversion, temporary tables
Sybase ASE specific functionality	• Using auto-increment fields to generate primary keys • Full T-SQL support including functions, procedures, named variables, etc. • Cursors as data sources • Extract and load using BCP (requires Sybase Adaptive Server client) • Support for exception handling, date+time conversion, temporary tables

Examples of the ETL Scenarios

You can find examples of the ETL scenarios here.

Modes

ETL Framework supports 3 execution modes.

Standalone executable

There are a standalone executables for all major operation systems. Program runs in the command line mode. User can configure connections to use and ETL scenarios to run using XML file.

Embedded

The ETL framework is integrated directly into application using open API.

Client-server

ETL process is executed remotely using Toolsverse SOA framework.

Compatibility

ETL Framework is tested in Windows XP and above (including Windows 8) 32 and 64 bit, OS X Leopard and above (including Mountain Lion). It is expected to work in all major versions of the UNIX and Linux.

ETL Framework requires Java 6 and above. It is tested in Java 7.

Installation

To run ETL Framework you need a Java runtime. If you are using Windows, you can download a zip archive that includes Java, or let the application automatically check for Java. If Java VM is not found on your computer or you have an older version of Java, application will display a warning message. You can manually install Java for Windows, Linux, Unix and OS X by clicking on this link: http://www.java.com/en/.

Standalone executable

1. Download archive file for the particular platform. Use downloads which **do not have bin-only** suffix.
2. Extract it anywhere in the file system. Example after extracting: c:/etl. On the OS X it is recommended to extract it to the **applications** folder (or extract anywhere and then copy to **applications** folder).
3. Find executable in the APP_HOME and create a shortcut/link if needed. The executables are: etl.**exe** file on Windows, etl.**app** on OS X and etl.**sh** on Linux/Unix. APP_HOME is a root folder where application is installed. For example: c:/etl.
4. Use executable/link to executable to run application.
5. Alternatively you can use executable etl.jar.

Embedded

1. Use this instruction to embed ETL engine into your application.

Client-server

1. Use this instruction to run ETL in the client-server mode.

Redistribution

ETL framework can be embedded and redistributed. Please check out file etl_framewrok_redist.txt located in the app_home/doc folder.

Installing JDBC driver

ETL framework requires JDBC drivers to work. To install a new JDBC driver follow these steps:

1. Download JDBC driver from the vendor's website
2. Create a folder under APP_HOME/jdbc. For example APP_HOME/jdbc/sqlanywhere
3. Copy all downloaded files into this folder

The following JDBC drivers are included in ETL Framework:

JDBC Driver	Location
IBM DB2	APP_HOME/jdbc/db2
Informix	APP_HOME/jdbc/informix
MS SQL Server	APP_HOME/jdbc/mssql
MySQL	APP_HOME/jdbc/mysql
Oracle	APP_HOME/jdbc/oracle
PostgreSQL	APP_HOME/jdbc/postgres
Sybase Adaptive Server	APP_HOME/jdbc/sybase
Derby (Java DB)	APP_HOME/lib

Configuration

ETL Framework is ready to use right after installation and typically does not require any additional configuration steps. However, if you absolutely have to you can change startup Java system properties (for example minimum and maximum memory limits for jvm).

To change startup Java system properties:

1. Open APP_HOME/etlappstart.properties file in your favorite text editor.
2. Modify the line containing **app.vm.options**. The default min/max memory limits for JVM are app.vm.options=-Xms100m -Xmx1000m
3. Save the file

You can also change logging properties such as log level, etc by modifying file log4.properties located under APP_HOME/config.

ETL Configuration File (etl_config.xml)

ETL configuration file tells ETL framework what connections to use and what ETL scenarios to run. It is located under APP_HOME/config.

For example: c:/etl/config/etl_config.xml.

File contains the following elements:

Properties

This section includes Java system properties which can be used by ETL Framework.

Example:

```
<properties>
   <log.step>777</log.step>
   <cache>com.toolsverse.cache.MemoryCache</cache>
   <oracle.oraclehome>c:/oracle</oracle.oraclehome>
</properties>
```

List of default properties:

Property	Description	Default value	Example
log.step	Log step is a how many rows of the data set to skip until log the event. For example if "log step" property set to 10 while iterating through the rows the ETL process will log every 10 rows. **Note:** to log this kind of events the log level must be set to **INFO** in the log4j.properties located under APP_HOME/config	0 (never)	`<log.step>777</log.step>`
cache	Defines the class which is used by ETL framework to cache values such as primary and foreign keys. **Note:** It must implement com.toolsverse.cache. Cache interface	com.toolsverse.cache.SynchMemoryCache	`<cache>com.toolsverse.cache.MemoryCache</cache>`

connection.factory	Defines the class which is used to create connections used by ETL framework. **Note:** It must implement com.toolsverse.etl.core.connection.EtlConnectionFactory interface	com.toolsverse.etl.core.connection.EtlConnectionFactoryImpl	`<connection.factory>abc</connection.factory>`
oracle.oraclehome	The Oracle client home		`<oracle.oraclehome>c:/oracle</oracle.oraclehome>`
db2.db2home	The DB2 client home		`<db2.db2home>c:/db2</db2.db2home>`
mssql.mssqlhome	The MS SQL Server client home		`<mssql.mssqlhome>c:/sqlserver</mssql.mssqlhome>`
sybase.sybasehome	The Sybase client home		`<sybase.sybasehome>c:/ase</sybase.sybasehome>`
mysql.mysqlhome	The MySQL client home		`<mysql.mysqlhome>c:/mysql</mysql.mysqlhome>`
postgres.postgreshome	The PostgreSQL client home		`<postgres.postgreshome>c:/postgres</postgres.postgreshome>`
informix.informixhome	The Informix client home		`<informix.home>c:/informix</informix.home>`

Connections

In this section you describe SQL and non-sql connections (for example Excel, XML or text files). There can be as many connections as you want, not necessary the once you are going to use when running particular ETL scenario. Just keep them here; you might need them next time.

Example:

```
<connections>
      <connection alias="test javadb">
         <driver>org.apache.derby.jdbc.EmbeddedDriver</driver>
         <url>jdbc:derby:./data-test/javadb</url>
      </connection>

      <connection alias="test oracle">
         <driver>oracle.jdbc.driver.OracleDriver </driver>
         <url>jdbc:oracle:thin:@localhost:1521:orcl1</url>
         <userid>user</userid>
         <password>password</password>
      </connection>
</connections>
```

Attributes of the connection node

Name	Description	Attribute or node	Example
alias	The alias name	attribute	`alias="test javadb"`
driver	JDBC driver class name	node	`<driver>org.apache.derby.jdbc.EmbeddedDriver</driver>` **Note:** Can be empty for non SQL connections such as Excel, text, XML
url	The URL	node	`<url>jdbc:derby:./data-test/javadb</url>`
connector	The connector class name	node	Possible values: • empty or com.toolsverse.etl.connector.sql.SqlConnector for database connections • com.toolsverse.etl.connector.excel.ExcelConnector for Excel (*.xls) connections • com.toolsverse.etl.connector.excel.ExcelXlsxConnector for Excel (*.xlsx) connections • com.toolsverse.etl.connector.xml.XmlConnector for XML and XML with Transformation connections • com.toolsverse.etl.connector.text.TextConnector for Delimited and Fixed Length Text connections

userid	The user name for JDBC connection	node	<userid>user</userid>
password	The password for JDBC connection	node	<password>password</password>
params	The connection properties. Use ';' as a delimiter	node	<params>SERVER=ol_svr_custom;DB=etl</params>
sql	The SQL which will be executed when connection established	node	<sql>insert into test (abc) values ('123')</sql>

Database Connection

Create database connection if you need to extract or load data into database which supports SQL. Database connection uses JDBC or ODBC driver. D**river** and **url** attributes are required when configuring database connection. Other attributes such as **userid**, **password**, **params**, and **sql** are optional. C**onnector** attribute can be either omitted or set to com.toolsverse.etl.connector.sql.SqlConnector.

Example of the Database connection:

```
<connection alias="test oracle">
   <driver>oracle.jdbc.driver.OracleDriver </driver>
   <url>jdbc:oracle:thin:@localhost:1521:orcl1</url>
   <userid>user</userid>
   <password>password</password>
</connection>
```

If url points to the file or folder in the file system system variables can be used as a part of the url.

Excel connection using ODBC

The Excel Connection using ODBC is a flavor of the database connection.

Example of the Excel connection using ODBC:

```
<connection alias="test excel">
      <driver>sun.jdbc.odbc.JdbcOdbcDriver</driver>
      <url>jdbc:odbc:Driver={Microsoft Excel Driver
      (*.xls)};DBQ={app.data}/test.xls</url>
</connection>
```

If url points to the file or folder in the file system system variables can be used as a part of the url.

Excel (.xls) Connection*

Create an Excel (*.xls) connection if you need to extract or load data into Excel (1997-2003) spreadsheet.

Required attributes:

Node	Value
Connector	com.toolsverse.etl.connector.excel.ExcelConnector
url	Example: {app. data}/test.xls

Possible values of the **params** attribute:

Property	Description	Example	Default
sheetname	The name of worksheet in the spreadsheet	sheetname=Employee	None
date	Date format	date=MMddyyyy	MM/dd/yyyy
datetime	Date+time format	datetime=MM/dd/yyyy HH:mm	MM/dd/yyyy HH:mm
time	Time format	time=HH:mm	HH:mm

You can combine them together using semicolon:

```
<params>sheetname=Employee;date=MMddyy;datetime=MMddyyyy;time=HH:mm</params>
```

Example of the Excel (*.xls) connection:

```
<connection alias="test excel">
   <url>{app.data}/test.xls</url>
   <connector> com.toolsverse.etl.connector.excel.ExcelConnector</connector>
   <params>sheetname=Employee;date=MMddyy;datetime=MMddyyyy;time=HH:mm</params>
</connection>
```

If url points to the file or folder in the file system system variables can be used as a part of the url.

Excel (.xlsx) Connection*

Create an Excel (*.xlsx) connection if you need to extract or load data into Excel (2007 and above) spreadsheet.

Required attributes:

Node	Value
connector	com.toolsverse.etl.connector.excel.ExcelXlsxConnector
url	Example: {app. data}/test.xls

Possible values of the **params** attribute:

Property	Description	Example	Default
sheetname	The name of worksheet in the spreadsheet	sheetname=Employee	None
date	Date format	date=MMddyyyy	MM/dd/yyyy
datetime	Date+time format	datetime=MM/dd/yyyy HH:mm	MM/dd/yyyy HH:mm
time	Time format	time=HH:mm	HH:mm

You can combine them together using semicolon:

```
<params>sheetname=Employee;date=MMddyy;datetime=MMddyyyy;time=HH:mm</params>
```

Example of the Excel (*.xlsx) connection:

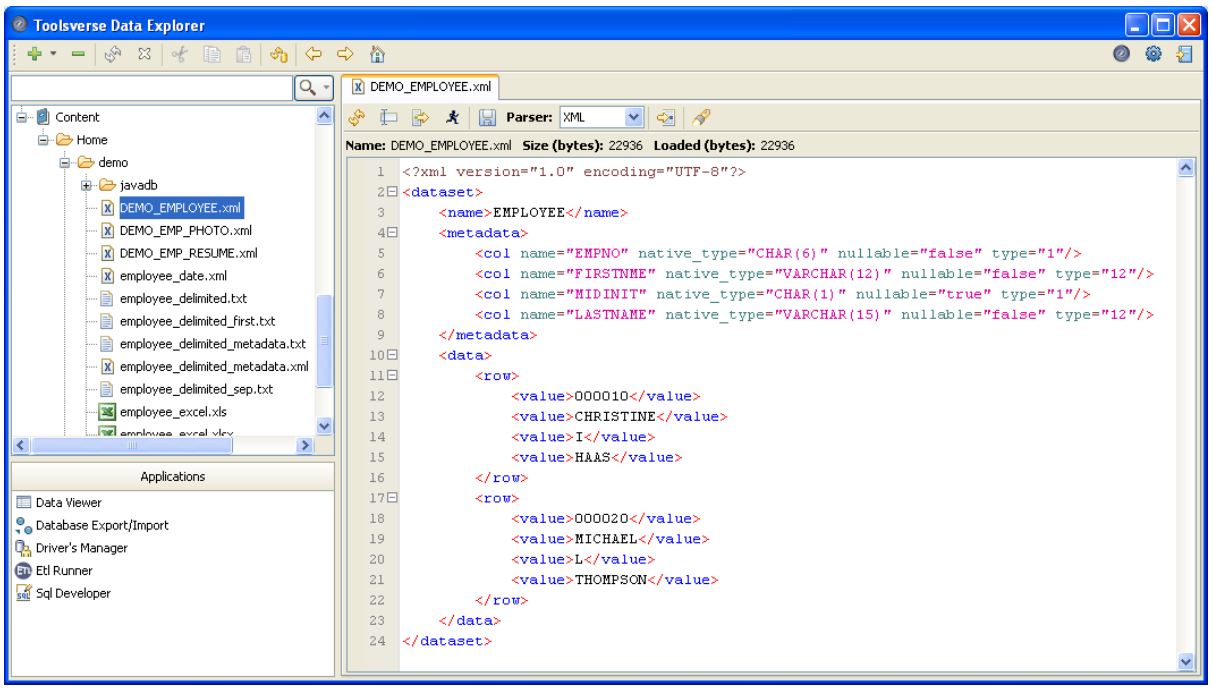
```
<connection alias="test excel xlsx">
	<url>{app.data}/test.xlsx</url>
	<connector>
	com.toolsverse.etl.connector.excel.ExcelXlsxConnector</connector>
	<params>sheetname=Employee;date=MMddyy;datetime=MMddyyyy;time=HH:mm</params>
</connection>
```

If url points to the file or folder in the file system system variables can be used as a part of the url.

XML Connection

The XML connection provides an access to the file in the internal ETL Framework XML format called XML dataset. It used by to serialize/de-serialize data. You can find schema for the XML dataset format in the APP_HOME/data/schema/xmldataset.xsd.

Example of the file in the XML dataset format:



Required attributes:

Node	Value
connector	com.toolsverse.etl.connector.xml.XmlConnector
url	Example: {app. data}/test.xml

Possible values of the **params** attribute:

Property	Description	Example	Default
date	Date format	date=MMddyyyy	MM/dd/yyyy
datetime	Date+time format	datetime=MM/dd/yyyy HH:mm	MM/dd/yyyy HH:mm
time	Time format	time=HH:mm	HH:mm

You can combine them together using semicolon:

```
<params>date=MMddyyyy;datetime=MMddyyyy;time=HH:mm</params>
```

Example of the XML connection:

```
<connection alias="test xml">
   <url>{app.data}/test.xml</url>
   <connector> com.toolsverse.etl.connector.xml.XmlConnector</connector>
   <params>date=MMddyy;datetime=MMddyyyy;time=HH:mm</params>
</connection>
```

If url points to the file or folder in the file system system variables can be used as a part of the url.

XML Connection with Transformation

If you need to extract and load data into the file in the XML format different from the XML dataset you can use XML connection with Transformation. You must have a XSL transformation style sheet. Please see examples of the XML dataset to WebRowSet and WebRowSet to XML dataset style sheets in APP_HOME/data/schema folder. The WebRowSet is an XML document representation of a JDBC result set which was introduced by Sun in JDK 1.5.

Required attributes:

Node	Value
connector	com.toolsverse.etl.connector.xml.XmlConnector
url	Example: {app. data}/test.xml
params	Example: xsl={app.root.data}/schema/webrowset2dataset.xsl

Possible values of the **params** attribute:

Property	Description	Example	Default
Xsl	Name of the xlst scenario file	xsl={app.root.data}/schema/webrowset2dataset.xsl	None
xslfrom	Name of the xlst scenario file used to transform **from** other XML format to XML dataset	xslfrom={app.root.data}/schema/dataset2webrowset.xsl	None
xslto	Name of the xlst scenario file used to transform to other XML format from XML dataset	xslfrom={app.root.data}/schema/webrowset2dataset.xsl	None
date	Date format	date=MMddyyyy	MM/dd/yyyy
datetime	Date+time format	datetime=MM/dd/yyyy HH:mm	MM/dd/yyyy HH:mm
time	Time format	time=HH:mm	HH:mm

You can combine them together using semicolon:

```
<params>xsl={app.root.data}/schema/webrowset2dataset.xsl;date=MMddyy;datetime=MMddyyyy;time=HH:mm</params>
```

Example of the XML connection with Transformation:

```
<connection alias="test xml with transformation">
	<url>{app.data}/test.xml</url>
	<connector>com.toolsverse.etl.connector.xml.XmlConnector</connector>
	<params>xsl={app.root.data}/schema/webrowset2dataset.xsl;date=MMddyy;datetime=
	MMddyyyy;time=HH:mm
	</params>
</connection>
```

If url points to the file or folder in the file system system variables can be used as a part of the url.

Delimited Text File Connection

You can create a connection to the delimited text file using wide range or properties.

Required attributes:

Node	**Value**
connector	com.toolsverse.etl.connector.text.TextConnector
url	Example: {app. data}/test.txt

Possible values of the **params** attribute:

Property	**Description**	**Example**	**Default**
delimiter	The field delimiter	delimiter=';'	'|'
firstrow	Use first row for data	firstrow=false	True
metadata	Store metadata in XML dataset format	metadata=false	False
charseparator	The character used to enclose string values into	charseparator=';'	Nothing
lineseparator	The separator between lines	lineseparator=w. Possible values: • s – os default • w – windows • u - unix	S
date	Date format	date=MMddyyyy	MM/dd/yyyy
datetime	Date+time format	datetime=MM/dd/yyyy HH:mm	MM/dd/yyyy HH:mm
time	Time format	time=HH:mm	HH:mm

You can combine them together using semicolon:

```
<params>delimiter=';';charseparator='"';firstrow=false</params>
```

Example of the Delimited Text File Connection:

```
<connection alias="test delimited text">
	<url>{app.data}/test.txt</url>
	<connector>com.toolsverse.etl.connector.text.TextConnector</connector>
	<params>delimiter=';';charseparator='"';firstrow=false</params>
</connection>
```

If url points to the file or folder in the file system system variables can be used as a part of the url.

Fixed Length Text File Connection

Fixed length text file connection uses the same connector as Delimited text file connection. The **fields** attribute is what differentiates it from the Delimited text file connection. The **fields** attribute must include a length of the each field in the data set. The numbers must be delimited by the value of the **delimiter** attribute. Example: delimiter=';';fields='6;12;15;8' defines a file with a 4 fields with a length 6, 12,15 and 8 respectfully.

Required attributes:

Node	Value
connector	com.toolsverse.etl.connector.text.TextConnector
url	Example: {app. data}/test.txt
props	Example: delimiter=';';fields='6;12;15;8'

Possible values of the **params** attribute:

Property	Description	Example	Default
delimiter	The field delimiter	delimiter=';'	'|'
fields	The length of the each field in the data set	fields='6;12;15;8'	none
firstrow	Use first row for data	firstrow=false	true
metadata	Store metadata in XML dataset format	metadata=false	false
charseparator	The character used to enclose string values into	charseparator=';'	nothing
lineseparator	The separator between lines	lineseparator=w. Possible values: • s – os default • w – windows • u - unix	s
date	Date format	date=MMddyyyy	system defined
datetime	Date+time format	datetime=MM/dd/yyyy HH:mm	system defined
time	Time format	time=HH:mm	system defined

You can combine them together using semicolon:

```
<params>delimiter=';';firstrow=false;fields='6;12;15;8</params>
```

Example of the Fixed Length Text File Connection:

```
<connection alias="test delimited text">
      <url>{app.data}/test.txt</url>
      <connector>com.toolsverse.etl.connector.text.TextConnector</connector>
      <params>delimiter=';';firstrow=false;fields='6;12;15;8</params>
</connection>
```

If url points to the file or folder in the file system system variables can be used as a part of the url.

Active connections

In this section you describe source and destination connections which are going to be used when running particular ETL scenario. There can be multiple source and destination connections. Connection can have a name which must be referenced from the ETL scenario. The default name for the source connection is **source** and for destination is **dest**.

Example of the single source and destination connections with default names:

```
<active.connections>
  <sourses>
     <source alias="test javadb" />
  </sourses>
  <destination alias="test oracle"/>
</active.connections>
```

Example of the multiple source and destination connections with user's defined names:

```
<active.connections>
  <sourses>
     <source alias="test excel" name="excel" />
     <source alias="test javadb" name="javadb" />
  </sourses>
  <destinations>
     <destination alias="test oracle" name="oracle" />
     <destination alias="test xyz" name="xyzcon" />
  </destinations>
</active.connections>
```

Attributes

Attribute	Description	Example
alias	The name of the alias. Must be the same as in the **connections** section	`<source alias="test javadb" />`
name	The name of the connection. Name must be referenced from the ETL scenario	`<destination alias="test xyz" name="xyzcon" />`

Execute (scenarios)

This section contains ETL scenarios which should be executed. You can execute multiple scenarios one by one or in parallel. If one of the scenarios fails the rest will be terminated as well. Each scenario can be executed using different action.

Example (single scenario) :

```
<execute>
   <scenario name="test.xml" action="extract_load" />
</execute>
```

Example (multiple scenarios, different actions) :

```
<execute>
   <scenario name="test1.xml" action="extract" />
   <scenario name="test2.xml" action="load" />
</execute>
```

Example (multiple scenarios, executed in parallel) :

```
<execute>
   <scenario name="test1.xml" action="extract_load" parallel="true"/>
   <scenario name="test2.xml" action=" extract_load" parallel="true" />
</execute>
```

Attributes

Attribute	Description	Example
name	Scenario file name. **Note:** If scenario file name does not have a folder it is expected to be in the APP_HOME/data/scenario folder.	`name="test1.xml"`
action	The ETL action. Possible actions: • extract – only extract • load – only load • extract_ load – extract and load	`action="extract"`
parallel	If set to true the scenario will be executed in the separate thread. Makes sense when there is more than one scenario to execute	`parallel="true"`

Example of the etl_config.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<config>
   <properties>
      <log.step>1000</log.step>
   </properties>

   <connections>
      <connection alias="test excel">
         <url>{app.data}/test.xls</url>
         <connector> com.toolsverse.etl.connector.excel.ExcelConnector</connector>
         <params>sheetname=Employee;date=MMddyy;datetime=MMddyyyy;time=HH:mm</params>
      </connection>

      <connection alias="test javadb">
         <driver>org.apache.derby.jdbc.EmbeddedDriver</driver>
         <url>jdbc:derby:{app.data}/javadb</url>
      </connection>

     <connection alias="test oracle">
         <driver>oracle.jdbc.driver.OracleDriver </driver>
         <url>jdbc:oracle:thin:@localhost:1521:orcl1</url>
         <userid>user</userid>
         <password>password</password>
     </connection>
   </connections>

   <active.connections>
      <sourses>
         <source alias="test excel" name="excel" />
         <source alias="test javadb" name="derby" />
      </sourses>
      <destination alias="test oracle"/>
   </active.connections>
   <execute>
       <scenario name="test.xml" action="extract_load" />
   </execute>
</config>
```

In this example **test.xml** ETL scenario located under the {app.data}/scenario folder will be executed using **extract_load** action. Connections **excel** and **derby** which linked to the aliases **test excel** and **test javadb** will be used as a **source** connections. Connection linked to the alias **test oracle** will be used as a **destination** connection. ETL framework is set to log every 1000 extracted or loaded records.

Embedding ETL Engine

To embed ETL framework into your application you will need the following files:

Folder	Files	Vendor	Require
lib	toolsverse-core.jar toolsverse-license.jar toolsverse-etl-common.jar toolsverse-etl-core.jar toolsverse-io.jar toolsverse-service.jar toolsverse-storage.jar toolsverse-mvc.jar toolsverse-updater.jar	Toolsverse	Yes
lib	commons-beanutils-1.8.3.jar commons-collections-3.2.1.jar commons-pool-1.5.4.jar commons-dbcp-1.4.jar commons-logging-1.1.1.jar commons-math3-3.0.jar commons-net-2.0.jar derby.jar cglib-nodep-2.2.jar dom4j-1.6.1.jar jsch-0.1.43.jar log4j-1.2.16.jar poi-3.8-20120326.jar poi-ooxml-3.8-20120326.jar poi-ooxml-schemas-3.8-20120326.jar saxon9he.jar socks.jar xercesImpl.jar xmlbeans-2.3.0.jar rsyntaxtextarea.jar httpclient-4.1.2.jar httpclient-cache-4.1.2.jar httpcore-4.1.2.jar httpmime-4.1.2.jar	third-party	Yes
plugin	toolsverse-etl-db2.jar toolsverse-etl-informix.jar toolsverse-etl-mysql.jar toolsverse-etl-oracle.jar toolsverse-etl-postgres.jar toolsverse-etl-sqlserver.jar	Toolsverse	No

Download ETL Framework Eclipse project and check out examples under **`src/com/toolsverse/etl/demo/embedded`**.

All Toolsverse and third-party files are included into Eclipse project. Source code and javadoc for the Toolsverse open-source components also included. You can also get latest versions of the third-party libraries from the vendor web sites.

Please check out online javadoc available in the Toolsverse.com website.

Configuring and running ETL in the client-server mode

To run ETL engine in the client-server mode you will need the following files:

Folder	Files	Vendor	Require
lib	toolsverse-core.jar toolsverse-license.jar toolsverse-etl-common.jar toolsverse-etl-core.jar toolsverse-io.jar toolsverse-service.jar toolsverse-storage.jar toolsverse-mvc.jar toolsverse-updater.jar	Toolsverse	Yes
lib	commons-beanutils-1.8.3.jar commons-collections-3.2.1.jar commons-dbcp-1.4.jar commons-logging-1.1.1.jar dom4j-1.6.1.jar log4j-1.2.16.jar xercesImpl.jar httpclient-4.1.2.jar httpclient-cache-4.1.2.jar httpcore-4.1.2.jar httpmime-4.1.2.jar	third-party	Yes

1. Download and install server version of the Data Explorer ETL edition(require license to run after 20 days evaluation period)
2. Download ETL Framework Eclipse project and check out examples under **`src/com/toolsverse/etl/demo/soa.`**

All Toolsverse and third-party files are included into Eclipse project. Source code and javadoc for the Toolsverse open-source components also included. You can also get latest versions of the third-party libraries from the vendor web sites.

Please check out online javadoc available in the Toolsverse.com website.

How To

Create ETL scenario

You can use ETL Integrated Development Environment included in the Data Explorer ETL edition to create, manage and run ETL scenarios. Or use any text editor of your choice.

You can find examples of the ETL scenarios here.

Execute ETL scenario using standalone executable

1. Open APP_HOME/config/etl_config.xml file in your favorite text editor.
2. Add connections for the particular ETL scenario
3. Specify connections to use and scenarios to run
4. Save
5. Run ETL executable. For example c:\etl\etl.exe on Windows
6. When it is finished check the etl.log file located under APP_HOME/logs

Example of the etl_config.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<config>
   <connections>
      <connection alias="test javadb">
         <driver>org.apache.derby.jdbc.EmbeddedDriver</driver>
         <url>jdbc:derby:{app.root.data}/demo/javadb</url>
      </connection>
      <connection alias="test oracle">
         <driver>oracle.jdbc.driver.OracleDriver </driver>
         <url>jdbc:oracle:thin:@localhost:1521:orcl1</url>
         <userid>user</userid>
         <password>password</password>
         <params/>
      </connection>
   </connections>

   <active.connections>
      <sourses>
         <source alias="test javadb" />
      </sourses>
      <destination alias="test oracle"/>
   </active.connections>
   <execute>
       <scenario name="move.xml" action="extract_load" />
   </execute>
</config>
```

In this example **move.xml** ETL scenario located under the {app.data}/scenario folder will be executed using **extract_load** action. Alias **test javadb** will be used for the source connection and alias **test oracle** for the destination.

Check for updates and download updates

To check for updates run etl executable with –v argument.

Example: c:\etl\etl.exe -v

If update is found it can be automatically downloaded and installed. To download and install update run etl executable with –u argument.

Example: c:\etl\etl.exe -u

Get a list of drivers and connectors

ETL framework uses pluggable drivers and connectors. Some of them might require additional license. To get a list of available drivers and connectors run etl executable with –c argument.

Example: c:\etl\etl.exe -c

Output:

Toolsverse Etl Framework 3.2-42934. Use -? for help.

Drivers:
- Generic File
- Generic Jdbc
- Generic ODBC
- Excel ODBC
- QED
- DB2
- Informix
- MySQL
- Oracle
- PostgreSQL
- MS Access ODBC
- MS SQL Server
- Sybase SQL Server

Connectors:
- Excel (*.xls)
- Excel (*.xlsx)
- SQL
- Text
- XML

Use named connections

1. Create named connection(s) in the `active.connections` section of the etl_config.xml file
2. Reference connection from the ETL scenario

Change log level to INFO

1. Open file APP_HOME/config/log4j.properties in your favorite text editor
2. Add INFO to the line containing log4j.rootLogger. Example:
   `log4j.rootLogger=ERROR,INFO,CONSOLE,FILE`
3. Save

Appendix 1. System Variables and Folders

ETL Framework uses APP_HOME/DATA folder by default to store all sort of files, from ETL scenarios to data files.

ETL scenarios are stored in the APP_HOME/DATA/scenario.

System variables can be used when defining URLs for the connections, file names etc.

Variable	Definition	Example
{app.home}	The root folder where application is installed	{app.home}/doc
{app.data}	The DATA folder: {app.home}/data	{app.data }/errors
{app.root.data}	The root DATA folder.	{app.root.data}/scenario