# Oracle® Endeca Information Discovery Integrator

Integrator ETL Server Guide

Version 3.1.0 • October 2013

ORACLE

# Copyright and disclaimer

# Oracle Endeca Supplement to Clover Server

This supplement provides specific information about support and limitations when using the Clover Server as the Oracle Endeca Integrator Server.

## Supported Containers

Oracle Endeca Integrator Server is only supported in the following containers:

- Apache Tomcat
- Oracle WebLogic

While the Clover Server allows installation to other containers, installation into these containers is not supported for Oracle Endeca Integrator Server.

# CloverETL Server

## Reference Manual

CloverETL

# CloverETL Server: Reference Manual

This Reference Manual covers the Release 3.4.x of CloverETL Server.

Release 3.4



Javlin

www.cloveretl.com

www.javlininc.com

Feedback welcome:

If you have any comments or suggestions for this documentation, please send them by email to `docs@cloveretl.com`.

# Table of Contents

# Chapter 1. What is CloverETL Server?

The CloverETL Server is an enterprise runtime, monitoring, and automation platform for the CloverETL data integration suite. It provides the necessary tools to deploy, monitor, schedule, integrate, and automate data integration processes in large scale and complex projects.

CloverETL Server's HTTP and SOAP Web Services APIs provide additional automation control for integrating the CloverETL Server into existing application portfolios and processes.

The CloverETL Server is a Java application built to J2EE standards. We support a wide range of application servers including Apache Tomcat, Jetty, IBM Websphere, Sun Glassfish, JBoss, and Oracle Weblogic.

*Table 1.1. CloverETL Server and CloverETL Engine comparison*

| | CloverETL Server | CloverEngine as executable tool |
|---|---|---|
| possibilities of executing graphs | by calling http (or JMX, etc.) APIs (See details in Chapter 14, Simple HTTP API (p. 78).) | by executing external process or by calling java API |
| engine initialization | during server startup | init is called for each graph execution |
| thread and memory optimalization | threads recycling, graphs cache, etc. | not implemented |
| scheduling | scheduling by timetable, onetime trigger, logging included | external tools (i.e. Cron) can bes used |
| statistics | each graph execution has its own log file and result status is stored; each event triggered by the CS is logged | not implemented |
| monitoring | If graph fails, event listener will be notified. It may send email, execute shell command or execute another graph. See details in Chapter 7, Graph Event Listeners (p. 55)Additionally server implements various APIs (HTTP and JMX) which may be used for monitoring of server/graphs status. | JMX mBean can be used while graph is running |
| storage of graphs and related files | graphs are stored on server file system in so called sandboxes | |
| security and authorization support | CS supports users/groups management, so each sandbox may have its own access privileges set. All interfaces require authentication. See details in Chapter 3, Server Side Job files - Sandboxes (p. 25). | passwords entered by user may be encrypted |
| integration capabilities | CS provides APIs which can be called using common protocols like HTTP. See details in Chapter 14, Simple HTTP API (p. 78). | CloverEngine library can be used as embedded library in client's Java code or it may be executed as separated OS process for each graph. |
| development of graphs | CS supports team cooperation above one project (sandbox). CloverETL Designer will be integrated with CS in further versions. | |
| scalability | CS implements horisontal scalability of transformation requests as well as data scalability. See details in Chapter 24, Clustering (p. 117) In addition CloverEngine implements is vertical scalability natively. | Clover Engine implements vertival scalability |
| jobflow | CS implements various jobflow components. See details in the CloverETL manual. | Clover Engine itself has limited support of jobflow. |

# Chapter 2. Installation

The following sections describe two different installation types. The section titled "Evaluation Server" (p. 3) details the quickest and simplest installation – without configuration – and the section titled "Enterprise Server" (p. 4) includes details about further testing and production on your chosen app-container and database.

## Evaluation Server

The default installation of **CloverETL Server** does not need any extra **database server**. It uses the embedded Apache Derby DB. What is more, it does not need any subsequent **configuration**. CloverETL Server configures itself during the first startup. Database tables and some necessary records are automatically created on first startup with an empty database. In the **Sandboxes** section of the web GUI, you can then check that sandboxes and a few demo graphs are created there.

### Note

Default login credentials for CloverETL Server Console:

Username: **clover**

Password: **clover**

If you need to evaluate CloverETl Server features which need any configuration changes, e.g. sending emails, LDAP authentication, clustering, etc., or the CloverETL Server must be evaluated on another application container then Tomcat, please proceed with the common installation: the section called "Enterprise Server" (p. 4)

### Installation of Apache Tomcat

CloverETL Server requires Apache Tomcat version 6.0.x to run.

If you have Apache tomcat already installed, you can skip this section.

1. Download the ZIP with binary distribution from http://tomcat.apache.org/download-60.cgi. Tomcat may be installed as a service on Windows OS as well, however there may be some issues with access to file system, so it's not recommended aproach for evaluation.
2. After you download the zip file, unpack it.
3. Run Tomcat by `[tomcat_home]/bin/startup.sh` (or `[tomcat_home]/bin/startup.bat` on Windows OS).
4. Check whether Tomcat is running on URL: http://localhost:8080/. Apache Tomcat info page should appear.
5. Apache Tomcat is installed.

If in need of detailed installation instructions, go to: http://tomcat.apache.org/tomcat-6.0-doc/setup.html

### Installation of CloverETL Server

1. Check if you meet prerequisites:
   - Oracle JDK or JRE v. 1.6.x or higher
   - `JAVA_HOME` or `JRE_HOME` environment variable has to be set.
   - Apache Tomcat 6.0.x is installed. See Installation of Apache Tomcat (p. 3) for details.
2. Set memory limits and other switches. See section the section called "Memory Settings" (p. 23) for details.

   Create setenv file:

Unix-like systems: `[tomcat]/bin/setenv.sh`

```
export CATALINA_OPTS="$CATALINA_OPTS -XX:MaxPermSize=512m -Xms128m -Xmx2048m"
export CATALINA_OPTS="$CATALINA_OPTS -Dderby.system.home=$CATALINA_HOME/temp -server"
echo "Using CATALINA_OPTS: $CATALINA_OPTS"
```

Windows systems: `[tomcat]/bin/setenv.bat`

```
set "CATALINA_OPTS=%CATALINA_OPTS% -XX:MaxPermSize=512m -Xms128m -Xmx1024m"
set "CATALINA_OPTS=%CATALINA_OPTS% -Dderby.system.home=%CATALINA_HOME%/temp -server"
echo "Using CATALINA_OPTS: %CATALINA_OPTS%"
```

3. Download the web archive file (`clover.war`) containing CloverETL Server for Apache Tomcat and `clover-license.war` containing valid license.

4. Deploy both WAR files: `clover.war` and `clover-license.war` to `[tomcat_home]/webapps` directory.

   To avoid deployment problems, Tomcat should be down during the copying.

5. Run Tomcat by `[tomcat_home]/bin/startup.sh` (or `[tomcat_home]/bin/startup.bat` on Windows OS).

6. Check whether CloverETL Server is running on URLs:

   - Web-app root

     `http://[host]:[port]/[contextPath]`

     The default Tomcat port for the http connector is 8080 and the default `contextPath` for CloverETL Server is "clover", thus the default URL is:

     http://localhost:8080/clover/

   - Web GUI

     `http://[host]:[port]/[contextPath]/gui` http://localhost:8080/clover/gui

     Use default administrator credentials to access the web GUI: username - "clover", password - "clover".

7. CloverETL Server is now installed and prepared for basic evaluation. There are couple of sandboxes with various demo transformations installed.

# Enterprise Server

This section describes instalation of CloverETL Server on various app-containers in detail, also describes the ways how to configure the server. If you need just quickly evaluate CloverETl Server features which don't need any configuration, evaluation installation may be suitable: the section called “Evaluation Server” (p. 3)

CloverETL Server for enterprise environment is shipped as a *Web application archive* (WAR file). Thus standard methods for deploying a web application on you application server may be used. However each application server has specific behavior and features. Detailed information about their installation and configuration can be found in the chapters below.

List of suitable containers:

- Apache Tomcat (p. 5)

- Jetty (p. 8)
- IBM Websphere (p. 9)
- Glassfish / Sun Java System Application Server (p. 12)
- JBoss (p. 14)
- Oracle WebLogic Server (p. 17)

In case of problems during your installation see Possible issues during installation (p. 19).

## Apache Tomcat

## Installation of Apache Tomcat

CloverETL Server requires Apache Tomcat version 6.0.x to run.

If you have Apache tomcat already installed, you can skip this section.

1. Download the binary distribution from http://tomcat.apache.org/download-60.cgi.
2. After you download the zip file, unpack it.
3. Run Tomcat by `[tomcat_home]/bin/startup.sh` (or `[tomcat_home]/bin/startup.bat` on Windows OS).
4. Check whether Tomcat is running on URL: http://localhost:8080/. Apache Tomcat info page should appear.
5. Apache Tomcat is installed.

If in need of detailed installation instructions, go to: http://tomcat.apache.org/tomcat-6.0-doc/setup.html

## Installation of CloverETL Server

1. Download the web archive file (`clover.war`) containing CloverETL Server for Apache Tomcat.
2. Check if you meet prerequisites:
   - Oracle JDK or JRE v. 1.6.x or higher
   - `JAVA_HOME` or `JRE_HOME` environment variable has to be set.
   - Apache Tomcat 6.0.x is installed. CloverETL Server is developed and tested with the Apache Tomcat 6.0.x container (it may work unpredictably with other versions). See Installation of Apache Tomcat(p. 5) for details.
   - It is strongly recommended you change default limits for the `heap` and `perm gen` memory spaces.

     See section the section called “Memory Settings” (p. 23) for details.

     You can set the minimum and maximum memory heap size by adjusting the "Xms" and "Xmx" JVM parameters. You can set JVM parameters for Tomcat by setting the environment variable `JAVA_OPTS` in the `[TOMCAT_HOME]/bin/setenv.sh` file (if it does not exist, you may create it).

     Create setenv file:

     Unix-like systems: `[tomcat]/bin/setenv.sh`

```
export CATALINA_OPTS="$CATALINA_OPTS -XX:MaxPermSize=512m -Xms128m -Xmx1024m"
export CATALINA_OPTS="$CATALINA_OPTS -Dderby.system.home=$CATALINA_HOME/temp -server"
echo "Using CATALINA_OPTS: $CATALINA_OPTS"
```

Windows systems: `[tomcat]/bin/setenv.bat`

```
set "CATALINA_OPTS=%CATALINA_OPTS% -XX:MaxPermSize=512m -Xms128m -Xmx1024m"
set "CATALINA_OPTS=%CATALINA_OPTS% -Dderby.system.home=%CATALINA_HOME%/temp -server"
echo "Using CATALINA_OPTS: %CATALINA_OPTS%"
```

As visible in the settings above, there is also switch `-server`. For performance reasons, it is recommended to run the container in the "server" mode.

3. Copy `clover.war` (which is built for Tomcat) to `[tomcat_home]/webapps` directory.

   Please note, that copying is not an atomic operation. If Tomcat is running, mind duration of the copying process! Too long copying might cause failure during deployment as Tomcat tries to deploy an incomplete file. Instead, manipulate the file when the Tomcat is not running.

4. War file should be detected and deployed automatically without restarting Tomcat.

5. Check whether CloverETL Server is running on URLs:

   - Web-app root

     `http://[host]:[port]/[contextPath]`

     The default Tomcat port for the http connector is 8080 and the default `contextPath` for CloverETL Server is "clover", thus the default URL is:

     http://localhost:8080/clover/

   - Web GUI

     `http://[host]:[port]/[contextPath]/gui`

     The default Tomcat port for the http connector is 8080 and the default contextPath for CloverETL Server is "clover", thus the default URL is:

     http://localhost:8080/clover/gui

     Use default administrator credentials to access the web GUI: user name - "clover", password - "clover".

## Configuration of CloverETL Server on Apache Tomcat

Default installation (without any configuration) is recommended only for evaluation purposes. For production, at least DB data source and SMTP server configuration is recommended.

### Properties File on Specified Location

Example of such a file:

```
jdbc.driverClassName=...
jdbc.url=...
jdbc.username=...
jdbc.password=...
jdbc.dialect=...
license.file=/path/to/license.dat
```

Properties file is loaded from location specified by the system property or environment variable `clover_config_file` (`clover.config.file`).

On Apache Tomcat, you can set the system property in the `[TOMCAT_HOME]/bin/setenv.sh` file (if it does not exist, you may create it). Just add: `JAVA_OPTS="$JAVA_OPTS -Dclover_config_file=/path/to/cloverServer.properties"`.

## Installation of CloverETL Server License

To be able to execute graphs, CloverETL Server requires a valid license. You can install CloverETL Server without any license, but no graph will be executed.

There are two ways of installing license on Tomcat. A simpler way is a separate web application `clover-license.war`. However, in cluster environment, configuring the plain license file has to be done (common for all application containers).

### a) Separate License WAR

1. Download the web archive file `clover-license.war`.
2. Copy `clover-license.war` to the `[tomcat_home]/webapps` directory.
3. The war file should be detected and deployed automatically without restarting Tomcat.
4. Check whether the license web-app is running on:

   `http://[host]:[port]/clover-license/` (Note: contextPath `clover-license` is mandatory and cannot by changed)

### b) License.file Property

Alternatively, configure the server's "license.file" property. Set its value to full path to the `license.dat` file.

#### Note

CloverETL license can be **changed** any time by re-deploying `clover-license.war`. Afterwards, you have to let CloverETL Server know the license has changed.

- Go to **server web GUI →Configuration →CloverETL Info →License**
- Click **Reload license**.
- Alternatively, you can restart the CloverETL Server application.

*Warning:* Keep in mind that during the WAR file redeployment, directory `[tomcat_home]/webapps/[contextPath]` has to be deleted. If Tomcat is running, it should do it automatically. Still, we suggest you check it manually, otherwise changes will not take any effect.

## Apache Tomcat on IBM AS/400 (iSeries)

To run CloverETL Server on the iSeries platform, there are some additional settings:

1. Declare you are using Java 6.0 32-bit
2. Run java with parameter `-Djava.awt.headless=true`

To configure this you can modify/create a file `[tomcat_home]/bin/setenv.sh` which contains:

```
JAVA_HOME=/QOpenSys/QIBM/ProdData/JavaVM/jdk50/32bit
```

```
JAVA_OPTS="$JAVA_OPTS -Djava.awt.headless=true"
```

## Jetty

## Installation of CloverETL Server

1. Download the web archive file (`clover.war`) containing the CloverETL Server application which is built for Jetty.

2. Check if prerequisites are met:

   - Oracle JDK or JRE version 1.6.x or higher
   - Jetty 6.1.x - only this particular version is supported

     All jetty-6 releases are available from http://jetty.codehaus.org/jetty/. Jetty 7 is not supported (as of Jetty 7, there have been huge differences in distribution packages as it is hosted by the Eclipse Foundation.
   - Memory allocation settings

     It involves JVM parameters: -Xms -Xmx (heap memory) and -XX:MaxPermSize (classloaders memory limit). See section the section called “Memory Settings” (p. 23) for details.

     You can set the parameters by adding

```
JAVA_OPTIONS='$JAVA_OPTIONS -Xms128m -Xmx1024m -XX:MaxPermSize=256m'
```

     to `[JETTY_HOME]/bin/jetty.sh`

3. Copy `clover.war` to `[JETTY_HOME]/webapps`.

4. Create a context file `clover.xml` in `[JETTY_HOME]/contexts` and fill it with the following lines:

```
<?xml version="1.0"  encoding="ISO-8859-1"?>
<!DOCTYPE Configure PUBLIC "-//Jetty//Configure//EN" "http://www.eclipse.org/jetty/configure.dtd">
<Configure class="org.mortbay.jetty.webapp.WebAppContext">
 <Set name="contextPath">/clover</Set>
 <Set name="war"><SystemProperty name="jetty.home" default="."/>/webapps/clover.war</Set>
</Configure>
```

   `clover.xml` will be detected by Jetty and the application will be loaded automatically.

5. Run `[JETTY_HOME]/bin/jetty.sh start` (or `[JETTY_HOME]/bin/Jetty-Service.exe` on Windows OS).

Finally, you can check if the server is running e.g. on http://localhost:8080/test/.

## Configuration of CloverETL Server on Jetty

Default installation (without any configuration) is recommended only for evaluation purposes. For production, at least DB data source and SMTP server configuration is recommended.

There are more ways how to set config properties, yet the most common one is properties file in a specified location.

### Properties file in Specified Location

Example of such a file:

```
jdbc.driverClassName=...
jdbc.url=...
jdbc.username=...
jdbc.password=...
jdbc.dialect=...
license.file=/path/to/license.dat
```

The common properties file is loaded from a location which is specified by the environment/system property `clover_config_file` or `clover.config.file`. This is a recommended way of configuring Jetty.

On Jetty, you can set system property in the `[JETTY_HOME]/bin/jetty.sh` file. Add:

```
JAVA_OPTIONS="$JAVA_OPTIONS -Dclover_config_file=/path/to/cloverServer.properties"
```

## Installation of CloverETL Server license

In order to execute graphs, CloverETL Server requires a valid license file. Despite that, you can install CloverETL Server without a license, but no graph will be executed.

1. Get the `license.dat` file.
2. Set the CloverETL Server `license.file` parameter to the path to `license.dat`.
   - Add "license.file" property to the config properties file (as decribed in Configuration of CloverETL Server on Jetty (p. 8). Set its value to full path to the `license.dat` file.
   - Restart Jetty.

   There are more ways how to configure the license. See Chapter 18, Configuration (p. 97) for a description of all possibilities.

### Note

CloverETL license can be **changed** any time by replacing the `license.dat` file. Afterwards, you have to let CloverETL Server know the license has changed.

- Go to **server web GUI →Configuration →CloverETL Info →License**
- Click **Reload license**.
- Alternatively, you can restart the CloverETL Server application.

## IBM Websphere

## Installation of CloverETL Server

1. Get the web archive file (`clover.war`) with CloverETL Server application which is built for Websphere.
2. Check if you meet prerequisites:
   - IBM JDK or JRE version 1.6.x or higher

- IBM Websphere 7.0 (see http://www.ibm.com/developerworks/downloads/ws/was/)
- Memory allocation settings

  It involves JVM parameters: -Xms and -Xmx See section the section called "Memory Settings"(p. 23) for details.

  You can set heap size in IBM Websphere's **Integrated Solutions Console** (by default accessible at: `http://[host]:10003//ibm/console/`)

  - Go to **Servers →Application servers →[server1] (or another name of your server) →Process Management →Java Virtual Machine**
  - There is the **Maximum heap size** field. Default value is only 256 MB, which is not enough for ETL transformations.

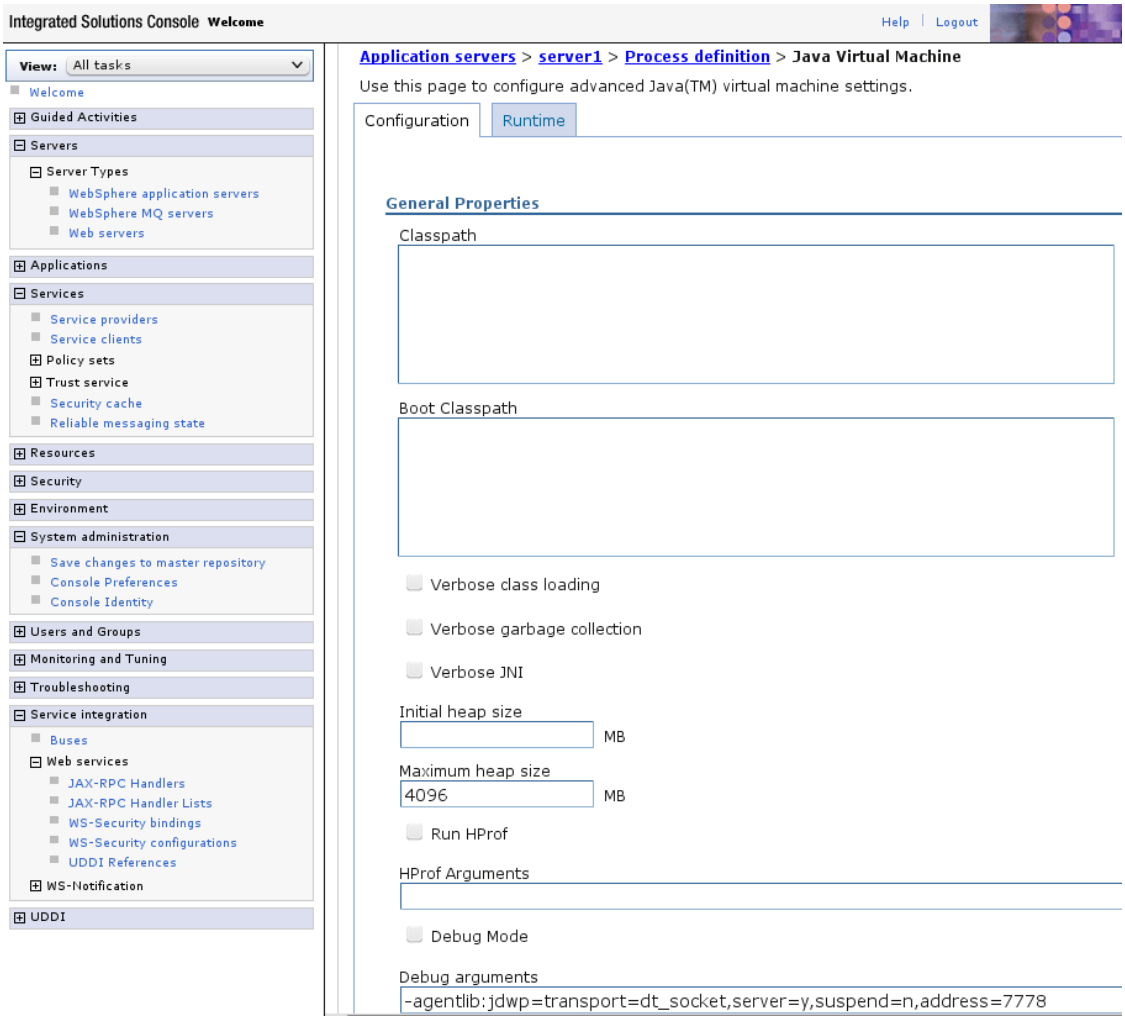



*Figure 2.1. Adjusting Maximum heap size limit*

  - On the same page, there is **Generic JVM arguments**. Add the perm space limit there, e.g. like this:

    ```
    -XX:MaxPermSize=512M
    ```

  - Restart the server to confirm these changes.

3. Deploy WAR file

- Go to **Integrated Solutions Console**

  ( http://localhost:9060/ibm/console/)

- Go to **Applications →New Application →New Enterprise Application**

4. Configure logging

   Websphere loggers do not use log4j by default. This may cause CloverETL Server logging to be ill-configured. As a result, some CloverETL Engine messages are missing in graph execution logs. Thus it is recommended to configure Websphere properly to use log4j.

   - Add a config file to the Websphere directory: `AppServer/profiles/AppSrv01/properties/commons-logging.properties`
   - Insert these lines into the file:

```
priority=1
org.apache.commons.logging.LogFactory=org.apache.commons.logging.impl.LogFactoryImpl
org.apache.commons.logging.Log=org.apache.commons.logging.impl.Log4JLogger
```

   - Copy jar files from the `clover.war/WEB-INF/lib` archive to the `AppServer/lib` directory. Copy all files like `commons-logging-*.jar` and `log4j-*.jar`.

5. Try if the server is running

   Provided you set `clover.war` as the application running with "clover" context path. Notice the port number has changed:

   http://localhost:9080/clover

### Note

Please note, that some CloverETL features using third party libraries don't work properly on IBM Websphere

- Hadoop is guaranteed to run only on Oracle Java 1.6+, but Hadoop developers do make an effort to remove any Oracle/Sun-specific code. See Hadoop Java Versions on Hadoop Wiki.
- OSGi framework (if configured and initialized) causes malfunction of WebServiceClient and EmailReader components. See Embedded OSGi framework section for details.

## Configuration of CloverETL Server on IBM Websphere

Default installation (without any configuration) is recommended only for evaluation purposes. For production, configuring at least the DB data source and SMTP server is recommended.

There are more ways how to set config properties. The most common one is properties file in a specified location.

### Properties File in Specified Location

Example of such a file:

```
jdbc.driverClassName=...
jdbc.url=...
jdbc.username=...
```

```
jdbc.password=...
jdbc.dialect=...
license.file=/path/to/license.dat
```

Set system property (or environment variable) `clover_config_file` pointing to the config properties file.

- go to **Integrated Solutions Console**

  (http://localhost:9060/ibm/console/)

- Go to **Servers →Application servers →[server-name] →Java and Process Management →Process Definition →Environment Entries**
- Create system property named `clover_config_file` whose value is full path to config file named e.g. `cloverServer.properties` on your file system.
- This change requires restarting IBM Websphere.

## Installation of CloverETL Server license

CloverETL Server requires a valid license for executing graphs. You can install CloverETL Server without a license, but no graph will be executed.

1. Get the `license.dat` file.
2. Set CloverETL Server's `license.file` parameter to path to the `license.dat` file.
   - Add "license.file" property to the config properties file as decribed in Configuration of CloverETL Server on IBM Websphere (p. 11). Value of the property has to be full path to the `license.dat` file.
   - Restart CloverETL Server.

   There are other ways how to do this. The most direct one is to set system property or environment variable `clover_license_file`. (See Chapter 18, Configuration (p. 97) for description of all possibilities).

### Note

Properly configured CloverETL license can be **changed** any time by replacing file `license.dat`. Then you have to let CloverETL Server know the license has changed.

- Go to **web GUI →Configuration →CloverETL Info →License**
- Click button **Reload license**.
- Alternatively, you can restart the CloverETL Server application.

## Glassfish / Sun Java System Application Server

## Installation of CloverETL Server

1. Get CloverETL Server web archive file (`clover.war`) which is built for Glassfish (Tomcat).
2. Check if you meet prerequisites
   - Oracle JDK or JRE version 1.6.x or higher
   - Glassfish (CloverETL Server is tested with V2.1)

- Memory allocation settings

  It involves JVM parameters: `-Xms -Xmx and -XX:MaxPermSize` See section the section called "Memory Settings" (p. 23) for details.

  You can set heap size and perm space in XML file `[glassfish]/domains/domain1/config/domain.xml` Add these sub-elements to <java-config>:

```
<jvm-options>-XX:MaxPermSize=512m</jvm-options>
<jvm-options>-Xmx2048m</jvm-options>
```

  These changes require restarting Glassfish.

3. Deploy WAR file

- Copy WAR file to the server filesystem. CloverETL Server is packed in a WAR file of 100 MB approx., so it cannot be uploaded directly from your local filesystem using the Admin Console.
- Fill in attributes **Application name** and **Context Root** with "clover". Fill in path to the WAR file on the server filesystem.
- Go to **Glassfish Admin Console**

  It is accessible at http://localhost:4848/ by default; default username/password is admin/adminadmin

- Go to **Applications** →**Web Applications** and click **Deploy**.
- Submit form

## Configuration of CloverETL Server on Glassfish

Default installation (without any configuration) is recommended only for evaluation purposes. For production, configuring at least the DB data source and SMTP server is recommended.

There are more ways how to set config properties. The most common one is properties file in a specified location.

### Properties File in Specified Location

Example of such a file:

```
jdbc.driverClassName=...
jdbc.url=...
jdbc.username=...
jdbc.password=...
jdbc.dialect=...
license.file=/path/to/license.dat
```

Set system property (or environment variable) `clover_config_file` pointing to the config properties file:

- Go to **Glassfish Admin Console**

  By default accessible at http://localhost:4848/ with username/password: admin/adminadmin

- Go to **Configuration** →**System Properties**
- Create system property named `clover_config_file` whose value is full path to a file on your file system named e.g. `cloverServer.properties`.

- This change requires restarting Glassfish.

## Installation of CloverETL Server License

CloverETL Server requires a valid license for executing graphs. You can install CloverETL Server without a license, but no graph will be executed.

License configuration is quite similar to WebSphere (p. 9).

1. Get the `license.dat` file.
2. Set CloverETL Server's `license.file` parameter to path to `license.dat`.
   - Add "license.file" property to the config properties file as decribed in Properties File in Specified Location (p. 13). Set its value to full path to the `license.dat` file.
   - Restart CloverETL Server.

   There are of course other ways how to do this. The most direct one is setting system property or environment variable `clover_license_file`. (See Chapter 18, Configuration (p. 97) for description of all possibilities).

### Note

Properly configured CloverETL license can be **changed** any time by replacing `license.dat`. Next, you need to let CloverETL Server know the license has changed.

- Go to **web GUI →Configuration →CloverETL Info →License**
- Click **Reload license**.
- Alternatively, you can restart CloverETL Server.

## JBoss

## Installation of CloverETL Server

1. Get CloverETL Server web archive file (`clover.war`) which is built for JBoss.
2. Check if you meet prerequisites
   - Oracle JDK or JRE version 1.6.x or higher
   - JBoss 6.0 or JBoss 5.1 - see http://www.jboss.org/jbossas/downloads
   - Memory settings for jboss java process. See section the section called "Memory Settings (p. 23)" for details.

     You can set the memory limits in `[jboss-home]/bin/run.conf` (`run.conf.bat` on Windows OS):

```
JAVA_OPTS="$JAVA_OPTS -XX:MaxPermSize=512m -Xms128m -Xmx2048m"
```

     On Windows, perform steps analogic to the ones above.
3. Create separated JBoss server configuration

This step is optional, you can use predefined server configuration, e.g. "default". However it may be useful to use specific JBoss server configuration, when it is necessary to run CloverETL:

- isolated from other JBoss applications
- with different set of services
- with different libraries on the classpath then other applications

See the JBoss manual for details about JBoss server configuration: JBoss Server Configurations Start the Server With Alternate Configuration

4. Configure DB data source

   Since JBoss does not work with embedded derby DB, a DB connection always has to be configured. We used MySQL in this case

   - Create datasource config file `[jboss-home]/server/[serverConfiguration]/deploy/mysql-ds.xml`

```
<datasources>
    <local-tx-datasource>
        <jndi-name>CloverETLServerDS</jndi-name>
        <connection-url>jdbc:mysql://localhost:3306/cloverServerDB</connection-url>
        <driver-class>com.mysql.jdbc.Driver</driver-class>
        <user-name>root</user-name>
        <password></password>
    </local-tx-datasource>
</datasources>
```

**Note**

Special characters in the XML file have to be typed in as XML entities. For instance, ampersand "&" as "&" etc.

   JNDI name has to be exactly "CloverETLServerDS". The thing to do here is to set DB connection parameters (`connection-url`, `driver-class`,`user-name` and `password`) to the created database. The database has to be empty before the first execution, the server creates its tables itself.

   JNDI data source is the only way of configuring CloverETL Server DB connection in JBoss.

   - Put JDBC driver for your DB to the app server classpath. We copied JDBC driver `mysql-connector-java-5.1.5-bin.jar` to `[jboss-home]/server/[serverConfiguration]/lib`

5. Configure CloverETL Server according to description in the next section (p. 16).

6. Deploy WAR file

   Copy `clover.war` to `[jboss-home]/server/[serverConfiguration]/deploy`

7. Start jboss via `[jboss-home]/bin/run.sh` (or `run.bat` on Windows OS) If you want to run JBoss with specific server configuration, it must be specified as parameter like this: `[jboss-home]/bin/run.sh -c [serverConfiguration]` If the serverConfiguration isn't specified, the "default" is used.

   It may take a couple of minutes until all applications are started.

8. Check JBoss response and CloverETL Server response

   - JBoss administration console is accessible at http://localhost:8080/ by default. Default username/password is admin/admin

- CloverETL Server is accessible at http://localhost:8080/clover by default.

9. If you like, you can move default and example sandboxes (created automatically in the `temp` directory) to a more suitable directory on your filesystem.

   - These sandboxes are created automatically during the first deployment and are located in the `web-app` directory, which is related to the specific deployment. If you redeployed the web application for a reason, the directory would be recreated. That is why it is better to move the sandboxes to a location which will not change.

## Configuration of CloverETL Server on JBoss

Default installation (without any configuration) is recommended only for evaluation purposes. For production, configurin at least tha DB data source and SMTP server is recommended.

There are more ways how to set config properties. The most common one is properties file in a specified location.

### Properties File in Specified Location

- Create `cloverServer.properties` in a suitable directory

```
datasource.type=JNDI
datasource.jndiName=java:/CloverETLServerDS
jdbc.dialect=org.hibernate.dialect.MySQLDialect
license.file=/home/clover/config/license.dat
```

  Do not change `datasource.type` and `datasource.jndiName` properties, but set a correct JDBC dialect according to your DB server (Chapter 18, Configuration (p. 97)). Also set path to your license file.

- Set system property (or environment property) `clover_config_file`.

  It should contain the full path to the `cloverServer.properties` file created in the previous step.

  The simplest way is seting java parameter in `[jboss-home]/bin/run.sh`, e.g.:

```
export JAVA_OPTS="$JAVA_OPTS -Dclover_config_file=/home/clover/config/cloverServer.properties"
```

  Please do not override other settings in the `JAVA_OPTS` property. i.e. memory settings described above.

  On Windows OS, edit `[jboss-home]/bin/run.conf.bat` and add this line to the section where options are passed to the JVM:

```
set JAVA_OPTS=%JAVA_OPTS% -Dclover_config_file=C:\JBoss6\cloverServer.properties
```

- This change requires restarting JBoss.

## Installation of CloverETL Server License

CloverETL Server requires a valid license for executing graphs. You can install CloverETL Server without a license, but no graph will be executed.

1. Get the `license.dat` file.

   If you only have `clover_license.war`, extract it as a common zip archive and you will find `license.dat` in the `WEB-INF` subdirectory

2. Fill CloverETL Server parameter `license.file` with path to `license.dat`.

   The best way how to configure license is setting the `license.file` property in the `cloverServer.properties` file as described in the previous section.

   There are other ways how to do this. (See Chapter 18, Configuration (p. 97) for description of all possibilities).

3. Changes in configuration require restarting the app-server.

### Note

CloverETL license can be **changed** any time by replacing file `license.dat`. Then you have to let CloverETL Server know the license is changed.

- Go to **web GUI →Configuration →CloverETL Info →License**
- Then click **Reload license**.
- Alternatively, you can restart CloverETL Server application.

## Oracle WebLogic Server

## Installation of CloverETL Server

1. Get CloverETL Server web archive file (`clover.war`) which is built for WebLogic.

2. Check if you meet prerequisites

   - Oracle JDK or JRE version 1.6.x or higher
   - WebLogic (CloverETL Server is tested with WebLogic Server 11g (10.3.6) and WebLogic Server 12c (12.1.1), see http://www.oracle.com/technetwork/middleware/ias/downloads/wls-main-097127.html)

     WebLogic has to be running and a domain has to be configured. You can check it by connecting to **Administration Console**: http://hostname:7001/console/ (7001 is the default port for HTTP). Username and password are specified during installation.

   - Memory allocation settings

     It involves JVM parameters: -Xms -Xmx and -XX:MaxPermSize

     See section the section called "Memory Settings (p. 23)" for details.

     You can set it i.e. by adding

```
export JAVA_OPTIONS='$JAVA_OPTIONS -Xms128m -Xmx2048m -XX:MaxPermSize=512m' to the start script
```

     This change requires restarting the domain.

3. Change HTTP Basic Authentication configuration

- When WebLogic finds "Authentication" header in an HTTP request, it tries to find a user in its own realm. His behavior has to be disabled so CloverETL could auhenticate users itself.
- Modify config file `[domainHome]/config/config.xml`. Add element: `<enforce-valid-basic-auth-credentials>false</enforce-valid-basic-auth-credentials>` into element `<security-configuration>` (just before the end tag).

4. Deploy WAR file (or application directory)
   - Get a clover.war suitable for your WebLogic version.
   - Deploy the clover.war using the **WebLogic Server Administration Console**. See the Oracle Fusion Middleware Administrator's Guide (http://docs.oracle.com/cd/E23943_01/core.1111/e10105/toc.htm) for details.
5. Configure license (and other configuration properties)
   - See separate section (p. 18) below
6. Check CloverETL Server URL
   - Web-app is started automatically after deploy, so you can check whether it is up and running.

     CloverETL Server is accessible at http://host:7001/clover by default. Port 7001 is the default WebLogic HTTP Connector port.

## Configuration of CloverETL Server on Weblogic

Default installation (without any configuration) is recommended only for evaluation purposes. For production, at least the DB data source and SMTP server configuration is recommended.

There are more ways how to set config properties. The most common one is properties file in a specified location.

### Properties File in Specified Location

Create `cloverServer.properties` in a suitable directory.

Config file should contain DB datasource config, SMTP connection config, etc.

Set system property (or environment variable) `clover_config_file` pointing to the config properties file

- Set JAVA_OPTIONS variable in the WebLogic domain start script `[domainHome]/startWebLogic.sh`

```
JAVA_OPTIONS="${JAVA_OPTIONS} -Dclover_config_file=/path/to/clover-config.properties
```

- This change requires restarting Weblogic.

## Installation of CloverETL Server License

CloverETL Server requires a valid license for executing graphs. You can install CloverETL Server without a license, but no graph will be executed.

1. Get the `license.dat` file.

   If you only have `clover_license.war`, extract it as a common zip archive and you will find `license.dat` file in `WEB-INF` subdirectory
2. Fill CloverETL Server parameter `license.file` with path to `license.dat` file

The best way how to configure license, is setting property `license.file` in the `cloverServer.properties` file as described in the previous section.

There are other ways how to do this. (See Chapter 18, Configuration (p. 97) for description of all possibilities).

3. Changes in configuration require restarting the app-server.

### Note

Properly configured CloverETL license can be **changed** any time by replacing file `license.dat`. Then you have to let CloverETL Server know the license has changed.

- Go to **web GUI →Configuration →CloverETL Info →License**
- Click **Reload license**.
- Or you can restart CloverETL Server application.

## Possible issues during installation

Since CloverETL Server is considered a universal JEE application running on various application servers, databases and jvm implementations, problems may occur during the installation. These can be solved by a proper configuration of the server environment. This section contains tips for the configuration.

## Memory issues on Derby

If your server suddenly starts consuming too much resources (CPU, memory) despite having been working well before, it might be beacuse of running the internal Derby DB. Typically, causes are incorrect/incomplete shutdown of Apache Tomcat and parallel (re)start of Apache Tomcat.

Solution: move to a standard (standalone) database.

How to fix this? Redeploy CloverETL Server:

1. Stop Apache Tomcat and verify there are no other instances running. If so, kill them.
2. Backup `config.properties` from `webapps/clover/WEB-INF` and `clover/WEB-INF/sandboxes` (if you have any data there).
3. Delete the `webapps/clover` directory.
4. Start Apache Tomcat server. It will automatically redeploy Clover Server.
5. Verify you can connect from Designer and from web.
6. Shutdown Apache Tomcat.
7. Restore `config.properties` and point it to your regular database.
8. Start Apache Tomcat.

## JAVA_HOME or JRE_HOME environment variables are not defined

If you are getting this error message during an attempt to start your application server (mostly Tomcat), perform the following actions.

**Linux:**

These two commands will help you set paths to the variables on the server.

- `[root@server /] export JAVA_HOME=/usr/local/java`
- `[root@server /] export JRE_HOME=/usr/local/jdk`

As a final step, restart the application server.

**Windows OS:**

Set `JAVA_HOME` to your JDK installation directory, e.g. `C:\Program Files\java\jdk1.6.0`. Optionally, set also `JRE_HOME` to the JRE base directory, e.g. `C:\Program Files\java\jre6`.

### Important

If you only have JRE installed, specify only `JRE_HOME`.

## Apache Tomcat Context Parameters don't have any effect

Tomcat may sometimes ignore some of context parameters. It may cause wierd CloverETL Server behaviour, since it looks like configured, but only partially. Some parameters are accepted, some are ignored. Usually it works fine, however it may occur in some environments. Such behaviour is consistent, so after restart it's the same. It's probably related to Tomcat issues: https://issues.apache.org/bugzilla/show_bug.cgi?id=47516 and https://issues.apache.org/bugzilla/show_bug.cgi?id=50700 To avoid this, please use properties file instead of context parameters to configure CloverETL Server.

## Tomcat log file catalina.out is missing on Windows

Tomcat start batch files for Windows aren't configured to create catalina.out file which contains standard output of the application. Catalinal.out may be vital when the Tomcat isn't started in console and any issue occurs. Or even when Tomcat is executed in the console, it may be closed automatically just after the error message appears in it.

Please follow these steps to enable catalina.out creation:

- Modify [tomcat_home]/bin/catalina.bat. Add parameter "/B" to lines where "_EXECJAVA" variable is set. There should be two these lines. So they will look like this:

  `set _EXECJAVA=start /B [the rest of the line]`
  Parameter /B causes, that "start" command doesn't open new console window, but runs the command it's own console window.

- Create new startup file. e.g. [tomcat_home]/bin/startupLog.bat, containing only one line:

  `catalina.bat start > ..\logs\catalina.out 2<&1`
  It executes Tomcat in the usual way, but standard output isn't put to the console, but to the catalina.out file.

Then use new startup file instead of [tomcat_home]/bin/startup.bat

## Timeouts waiting for JVM

If you get the Jetty application server successfully running but cannot start Clover Server, it might be because of the wrapper waiting for JVM too long (it is considered a low-memory issue). Examine `[JETTY_HOME]\logs\jetty-service.log` for a line like this:

```
Startup failed: Timed out waiting for signal from JVM.
```

If it is there, edit `[JETTY_HOME]\bin\jetty-service.conf` and add these lines:

```
wrapper.startup.timeout=60
wrapper.shutdown.timeout=60
```

If that does not help either, try setting 120 for both values. Default timeouts are 30 both.

## clover.war as default context on Websphere (Windows OS)

If you are deploying `clover.war` on the IBM Websphere server without context path specified, be sure to check whether it is the only application running in the context root. If you cannot start Clover Server on Websphere, check the log and look for a message like this:

```
com.ibm.ws.webcontainer.exception.WebAppNotLoadedException:
Failed to load webapp: Failed to load webapp: Context root /* is already bound.
Cannot start application CloverETL
```

If you can see it, then this is the case. Getting rid of the issue, the easiest way is to stop all other (sample) applications and leave only `clover.war` running on the server. That should guarantee the server will be available in the context root from now on (e.g. http://localhost:9080/).

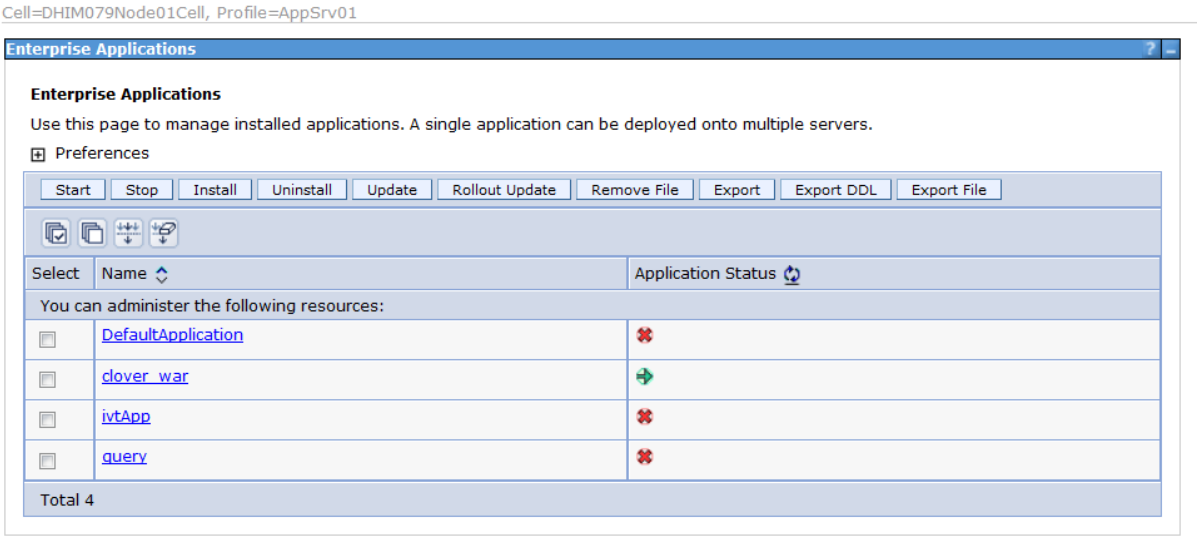



*Figure 2.2. Clover Server as the only running application on IBM Websphere*

## Tomcat 6.0 on Linux - Default DB

When using the internal (default) database on Linux, your Clover Server might fail on first start for no obvious reasons. Chances are that the `/var/lib/tomcat6/databases` directory was not created (because of access rights in parent folders).

Solution: Create the directory yourself and try restarting the server. This simple fix was successfully tested with Clover Server deployed as a `WAR` file via Tomcat web admin.

## Derby.system.home cannot be accessed

If the server cannot start and the following message is in the log:

```
java.sql.SQLException: Failed to start database 'databases/cloverserver'
```

then see the next exception for details. After that check settings of the `derby.system.home` system property. It may point to an unaccessible directory, or files may be locked by another process. We suggest you set a specific directory as the system property.

## Environment variables and more than one CloverETL Server instances running on single machine

If you are setting environment variables like `clover_license_file` or `clover_config_file` , remember you should not be running more than one CloverETL Server. Therefore if you ever needed to run more instances at once, use other ways of setting parameters (see Chapter 18, Configuration (p. 97) for description of all possibilities) The reason is the environment variable is shared by all applications in use causing them to share configurations and fail unexpectedly. Instead of environment variables you can use system properties (passed to the application container process using parameter with -D prefix: -Dclover_config_file).

## Special characters and slahes in path

When working with servers, you ought to stick to folder naming rules more than ever. Do not use any special characters in the server path, e.g. spaces, accents, diacritics are all not recommended. It's unfortunatelly common naming strategy on Windows systems. It can produce issues which are hard to find. If you are experiencing weird errors and cannot trace the source of them, why not install your application server in a safe destination like:

```
C:\JBoss6\
```

Similarly, use slashes but never backslahes in paths inside the `*.properties` files, e.g. when pointing to the Clover Server license file. If you incorrectly use backlash, it will be considered an escape character and the server may not work fine. This is an example of a correct path:

```
license.file=C:/CoverETL/Server/license.dat
```

## JAXB and early versions of JVM 1.6

CloverETL Server contains jaxb 2.1 libraries since version 1.3. This may cause conflicts on early versions of JVM 1.6 which contain jaxb 2.0. However JDK6 Update 4 release finally contains jaxb 2.1, thus update to this or newer version of JVM solves possible conflicts.

## File system permissions

Application server must be executed by OS user which has proper read/write permissions on file system. Problem may occur, if app-server is executed by root user for the first time, so log and other temp files are created by root user. When the same app-server is executed by another user, it will fail because it cannot write to root's files.

## JMS API and JMS third-party libraries

Missing JMS libraries do not cause fail of server startup, but it is issue of deployment on application server, thus it still suits to this chapter.

clover.war itself does not contain jms.jar, thus it has to be on application server's classpath. Most of the application servers have jms.jar by default, but i.e. tomcat does not. so if the JMS features are needed, the jms.jar has to be added explicitly.

If "JMS Task" feature is used, there must be third-party libraries on server's classpath as well. The same approach is recommended for JMS Reader/Writer components, even if these components allow to specify external libraries. It is due to common memory leak in these libraries which causes "OutOfMemoryError: PermGen space".

# Memory Settings

Current implementation of Java Virtual Machine allows only global configuration of memory for the JVM system process. Thus whole application server, together with WARs and EARs running on it, share one memory space.

Default JVM memory settings is to low for running application container with CloverETL Server. Some application servers, like IBM Websphere, increase JVM defaults themselves, however they still may be too low.

The best memory limits depend on many conditions, i.e. transformations which CloverETL should execute. Please note, that maximum limit isn't amount of permanently allocated memory, but limit which can't be exceeded. If the limit was exhaused, the OutOfMemoryError would be raised.

You can set the minimum and maximum memory heap size by adjusting the "Xms" and "Xmx" JVM parameters. There are more ways how to change the settings depending on the used application container.

If you have no idea about the memory required for the transformations, a maximum of 1-2 GB heap memory is recommended. This limit may be increased during transformations development when `OutOfMemoryError` occurs.

Memory space for loading classes (so called "PermGen space") is separated from heap memory, and can be set by the JVM parameter "-XX:MaxPermSize". By default, it is just 64 MB which is not enough for enterprise applications. Again, suitable memory limit depends on various criteria, but 512 MB should be enought in most cases. If the PermGen space maximum is too low, `OutOfMemoryError: PermGen space` may occur.

Please see the specific container section for details how to make the settings.

# Upgrading Server to Newer Version

## General Notes on Upgrade

- Upgrade of CloverETL Server requires a down time; plan maintenance window
- Succesful upgrade requires about 30 minutes; rollback requires 30 minutes
- Performing the below steps in development/testing environment first before moving onto production one

## Upgrade Prerequisites

- Having new CloverETL Server web application archive (`clover.war` appropriate for the application server used) & license files available
- Having release notes for particular CloverETL version available (and all version between current and intended version to be upgraded to)
- Having the graphs and jobs updated and tested with regards to Known Issues & Compatibility for particular CloverETL version.
- Having the CloverETL Server configuration properties file externalized from default location, see Config Sources and Their Priorities (p. 97)
- Standalone database schema where CloverETL Server stores configuration, see Examples of DB Connection Configuration (p. 98)

- Having a separate sandbox with test graph that can be run anytime to verify that CloverETL Server runs correctly and allows for running jobs

## Upgrade Instructions

1. Suspend all sandboxes, wait for running graphs to finish processing
2. Shutdown CloverETL Server application (or all servers, if they run in cluster mode)
3. Backup existing CloverETL database schema (if any changes to the database schema are necessary, the new server will automatically make them when you start it for the first time)
4. Backup existing CloverETL web application archive (`clover.war`) & license files (on all nodes)
5. Backup existing CloverETL sandboxes (on all nodes)
6. Re-deploy the CloverETL Server web application. Instructions how to do that are application server dependant - see Enterprise Server (p. 4) for installation details on all supported application servers. After you re-deploy, your new server will be configured based on the previous version's configuration.
7. Copy new license files in place (on all nodes). The license file is shipped as a text containing a unique set of characters. If you:
   - received the new license as a file (`*.dat`), then simply overwrite your old license file.
   - have been sent the licence text e.g inside an e-mail, then copy the license contents (i.e. all text between `Company` and `END LICENSE`) into a new file called `clover-license.dat`. Next, overwrite the old license file with the new one.
8. Start CloverETL Server application (on all nodes)
9. Review that contents of all tabs in CloverETL Server Console, especially scheduling and event listeners looks OK.
10. Update graphs to be compatible with the particular version of CloverETL Server (this should be prepared and tested in advance)
11. Resume the test sandbox and run a test graph to verify functionality
12. Resume all sandboxes

## Rollback Instructions

1. Shutdown CloverETL Server application
2. Restore CloverETL Server web application (`clover.war`) & license files (on all nodes)
3. Restore CloverETL Server database schema
4. Restore CloverETL sandboxes (on all nodes)
5. Start CloverETL Server application (on all nodes)
6. Resume the test sandbox and run a test graph to verify functionality
7. Resume all sandboxes

### Important

**Evaluation Version** - a mere upgrade of your license is not sufficient. When moving from evaluation to enterprise server, you should not use the default configuration and database. Instead, take some time to configure Clover Server so that it best fits your production environment.

# Chapter 3. Server Side Job files - Sandboxes

A sandbox is where you store all your project's transformation graph files, jobflows, data, and other resources. It's a server side analogy to a Designer project. The Server adds additional features to sandboxes, like user permissions management and global per-sandbox configuration options.

The Server and the Designer are integrated so that you are able to connect to a Server sandbox using a "Server Project" in your Designer workspace. Such a project works like a remote file system – all data is stored on the Server and accessed remotely. Nonetheless, you can do everything with Server Projects the same way as with local projects – copy and paste files, create, edit, and debug graphs, etcetera. See the CloverETL Designer manual for details on configuring a connection to the Server.

Technically, a sandbox is a dedicated directory on the Server host file system and its contents are managed by the Server. Advanced types of sandboxes, like "partitioned sandbox" have multiple locations to allow distributed parallel processing (more about that in Chapter 24, Clustering (p. 117)). A sandbox cannot contain another sandbox within – it's a single root path for a project.

We recommend putting all your sandboxes in a folder outside the CloverETL Server installation (e.g. /var/cloveretl/sandboxes). However, each sandbox can be located on the file system independently of the others if needed. The containing folder and all its contents must have read/write permission for the user under which the CloverETL Server/application server is running.

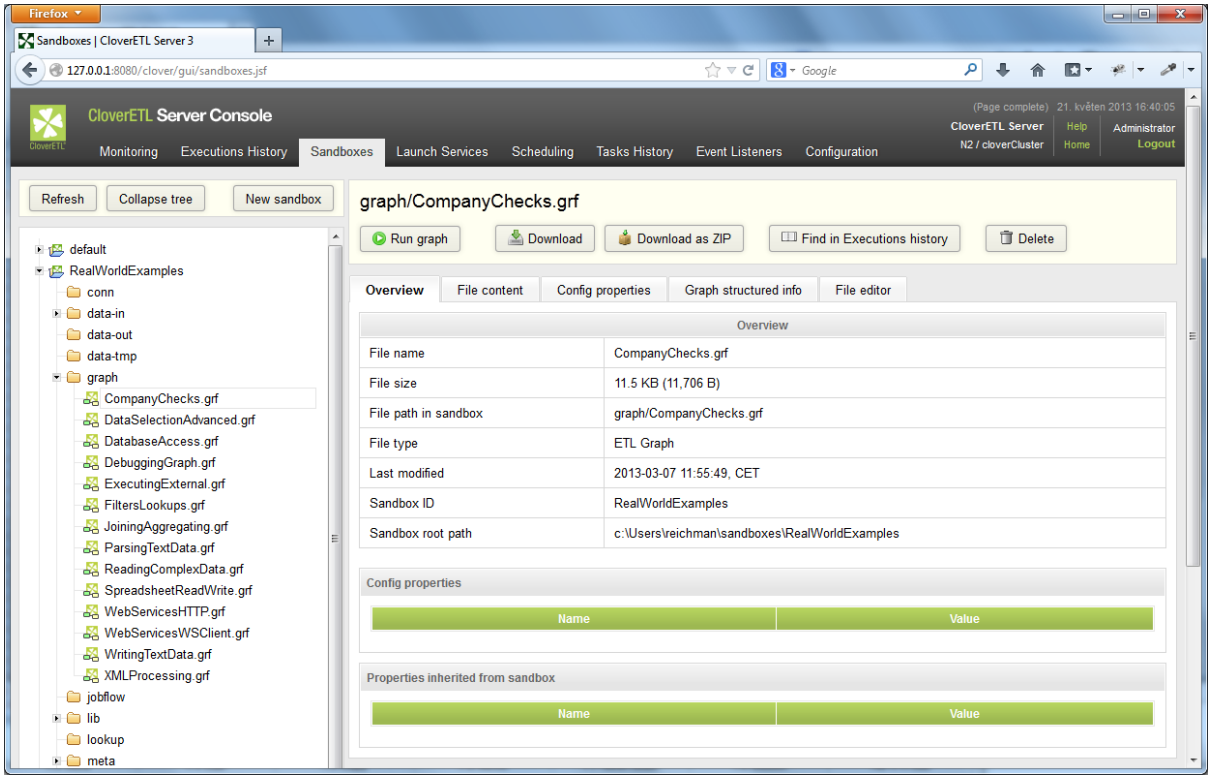



*Figure 3.1. Sandboxes Section in CloverETL Server Web GUI*

Each sandbox is defined by following attributes:

*Table 3.1. Sandbox attributes*

| | |
|---|---|
| Sandbox ID | Unique "name" of the sandbox. It is used in server APIs to identify sandbox. It must meet common rules for identifiers. It is specified by user in during sandbox creation and it can be modified later. *Note: modifying is not recommended, because it may be already used by some CS APIs clients.* |
| Sandbox | Sandbox name used just for display. It is specified by user in during sandbox creation and it can be modified later. |
| Sandbox root path | Absolute server side file system path to sandbox root. It is specified by user during sandbox creation and it can be modified later. This attribute is used only in standalone mode. See Chapter 24, Clustering (p. 117) for details about cluster mode. |
| Owner | It is set automatically during sandbox creation. It may be modified later. |

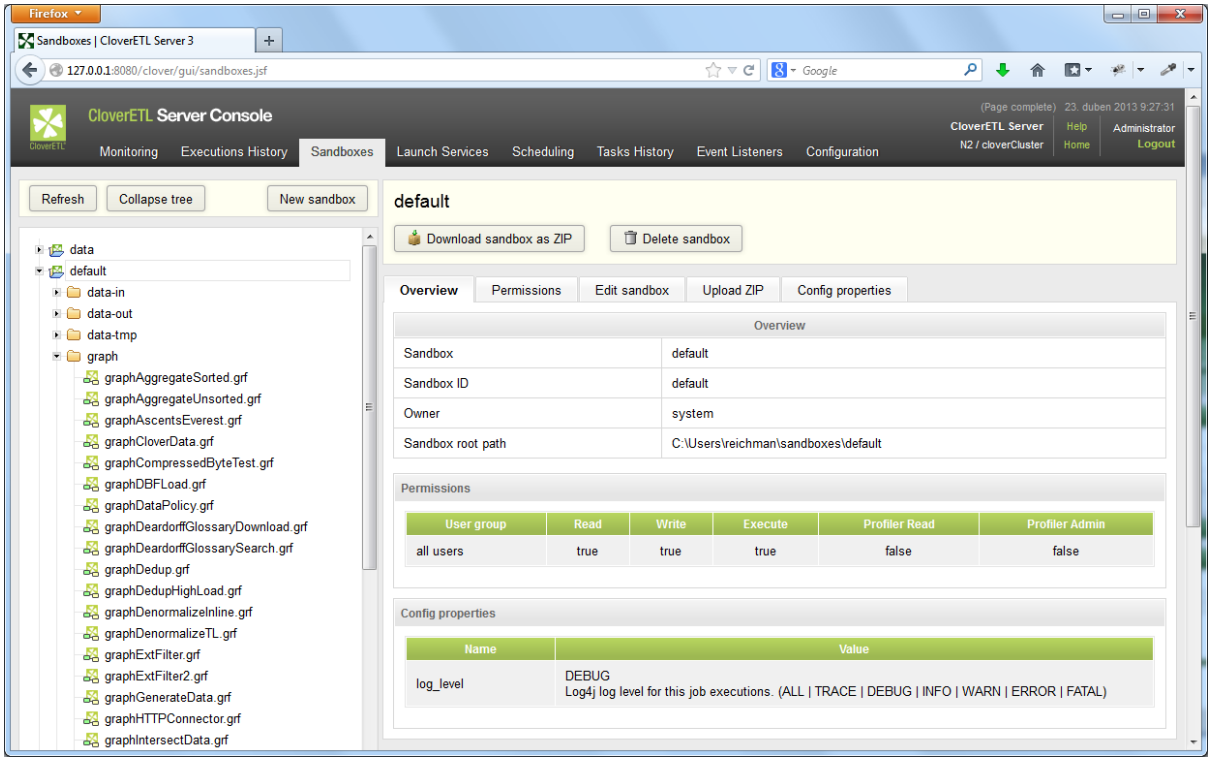



*Figure 3.2. Sandbox Detail in CloverETL Server Web GUI*

# Referencing files from the ETL graph or Jobflow

In some components you can specify file URL attribute as a reference to some resource on the file system. Also external metadata, lookup or DB connection definition is specified as reference to some file on the filesystem. With CloverETL Server there are more ways how to specify this relation.

- Relative path

  All relative paths in your graphs are considered as relative paths to the root of the same sandbox which contains job file (ETL graph or Jobflow).

- sandbox:// URLs

  Sandbox URL allows user to reference the resource from different sandboxes with standalone CloverETL Server or the cluster. In cluster environment, CloverETL Server transparently manages remote streaming if the resource is accessible only on some specific cluster node.

  See Using a Sandbox Resource as a Component Data Source (p. 122) for details about the sandbox URLs.

# Sandbox Content Security and Permissions

Each sandbox has its owner which is set during sandbox creation. This user has unlimited privileges to this sandbox as well as administrators. Another users may have access according to sandbox settings.

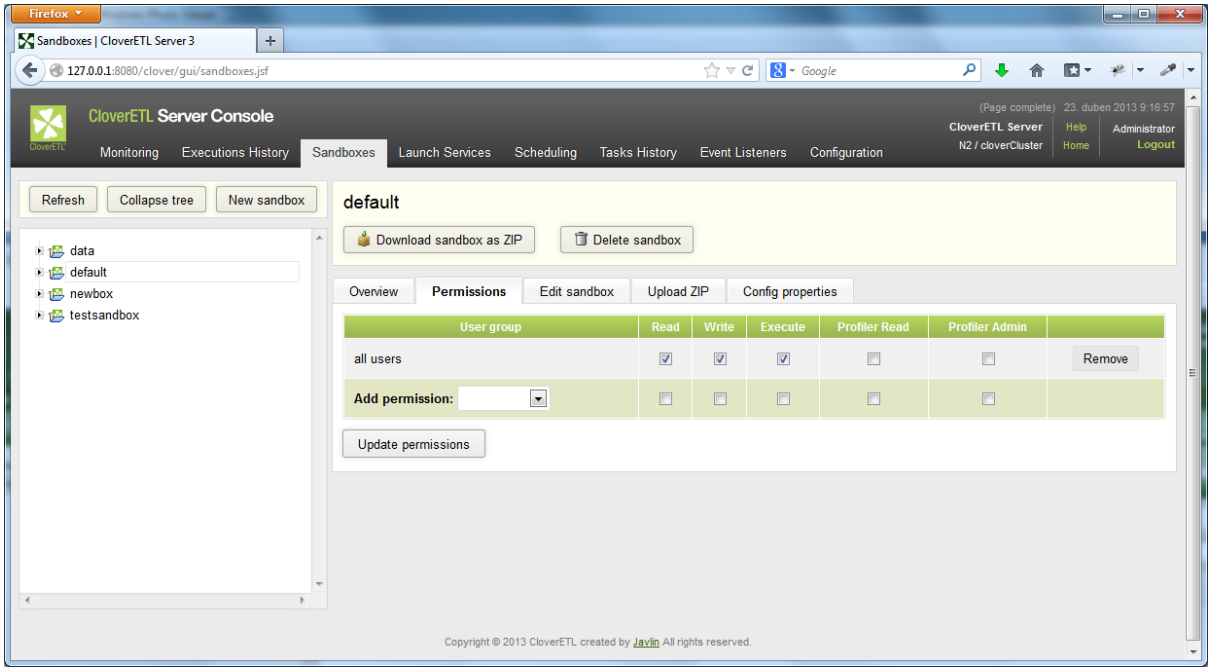



*Figure 3.3. Sandbox Permissions in CloverETL Server Web GUI*

Permissions to specific sandbox are modifiable in **Permissions** tab in sandbox detail. In this tab, selected user groups may be allowed to perform particular operations.

There are 3 types of operations:

*Table 3.2. Sandbox permissions*

| | |
|---|---|
| Read | Users can see this sandbox in their sandboxes list. |
| Write | Users can modify files in the sandbox through CS APIs. |
| Execute | Users can execute jobs in this sandbox. *Note: jobs executed by "graph event listener" and similar features is actually executed by the same user as job which is source of event. See details in "graph event listener". Job executed by schedule trigger is actually executed by the schedule owner. See details in Chapter 6, Scheduling (p. 45) If the job needs any files from the sandbox (e.g. metadata), user also must have read permission, otherwise the execution fails.* |
| Profiler Read | User can view results of profiler jobs executed from the sandbox. |
| Profiler Admin | User can administer results of profiler jobs executed from the sandbox. |

Please note that, these permissions modify access to the content of specific sandboxes. In additions, it's possible to configure permissions to perform operations with sandbox configuration. e.g. create sandbox, edit sandbox, delete sandbox, etc. Please see Chapter 5, Users and Groups (p. 37) for details.

# Sandbox Content

Sandbox should contain jobflows, graphs, metadata, external connection and all related files. Files especially graph or jobflow files are identified by relative path from sandbox root. Thus you need two values to identify specific job file: sandbox and path in sandbox. Path to the Jobflow or ETL graph is often referred as "Job file".

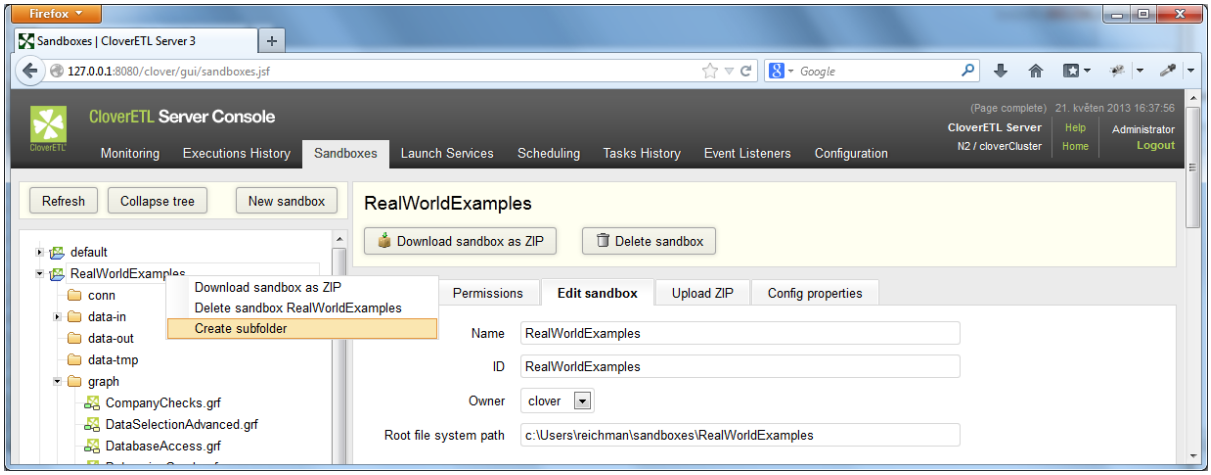



*Figure 3.4. Web GUI - section "Sandboxes" - context menu on sandbox*

Although web GUI section **sandboxes** isn't file-manager, it offers some useful features for sandbox management.

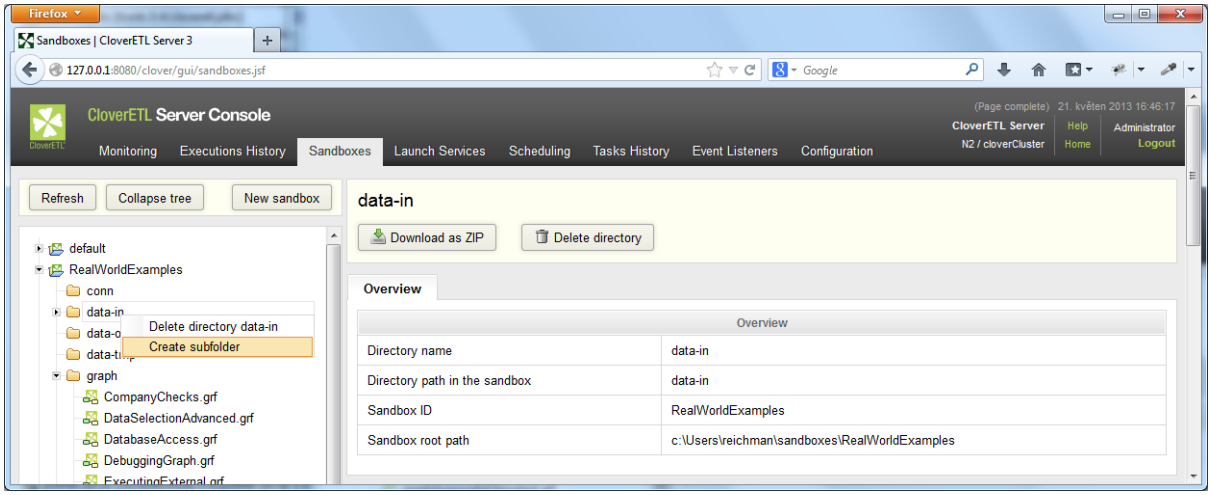



*Figure 3.5. Web GUI - section "Sandboxes" - context menu on folder*

## Download sandbox as ZIP

Select sandbox in left panel, then web GUI displays button "Download sandbox as ZIP" in the tool bar on the right side.

Created ZIP contains all readable sandbox files in the same hierarchy as on file system. You can use this ZIP file for upload files to the same sandbox, or another sandbox on different server instance.

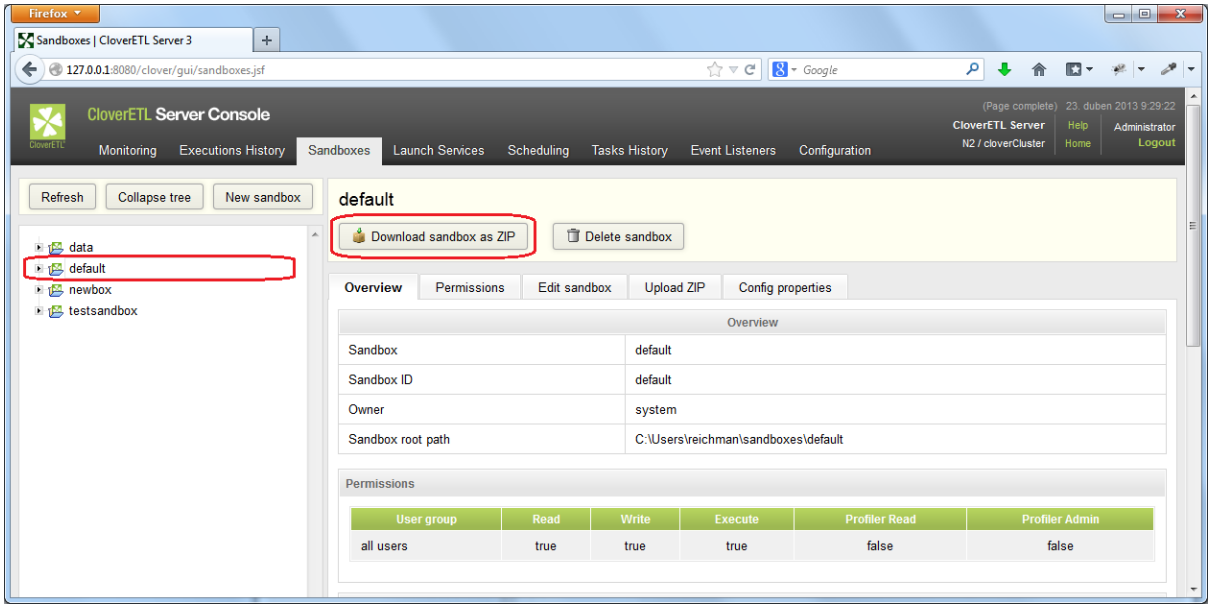



*Figure 3.6. Web GUI - download sandbox as ZIP*

## Upload ZIP to sandbox

Select sandbox in left panel. You must have write permission to the selected sandbox. Then select tab "Upload ZIP" in the right panel. Upload of ZIP is parametrized by couple of switches, which are described below. Open common file chooser dialog by button "+ Upload ZIP". When you choose ZIP file, it is immediately uploaded to the server and result message is displayed. Each row of the result message contains description of one single file upload. Depending on selected options, file may be skipped, updated, created or deleted.

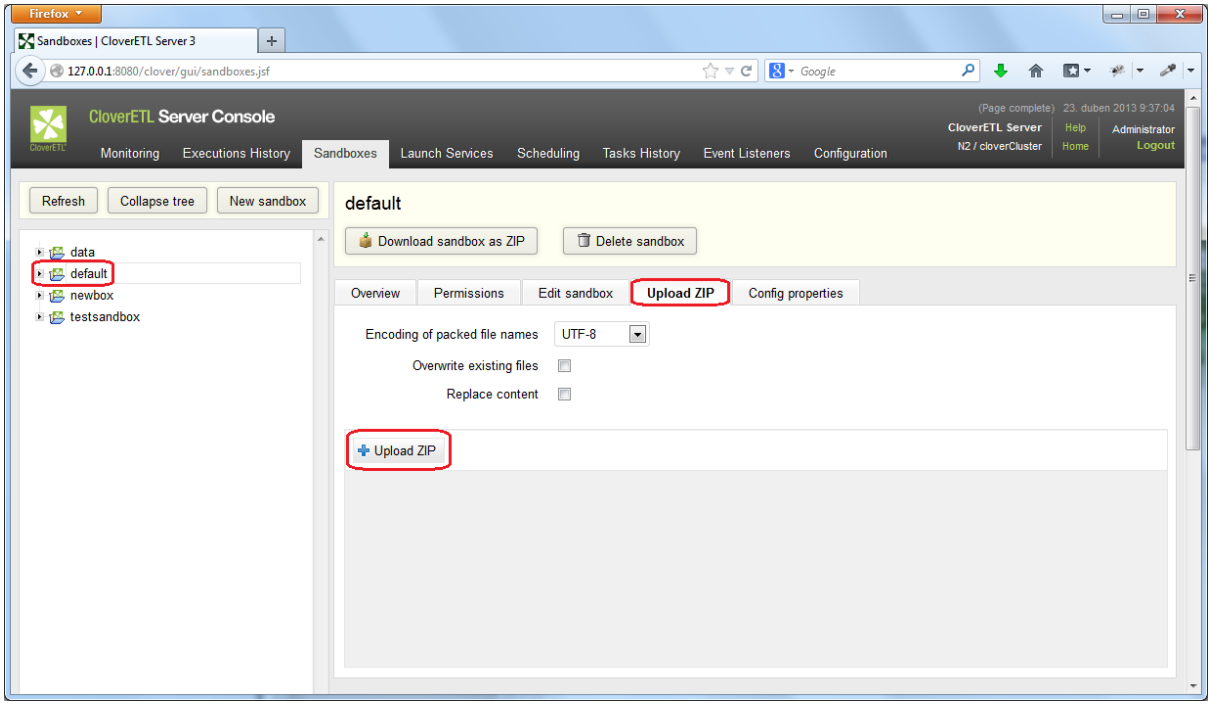



*Figure 3.7. Web GUI - upload ZIP to sandbox*

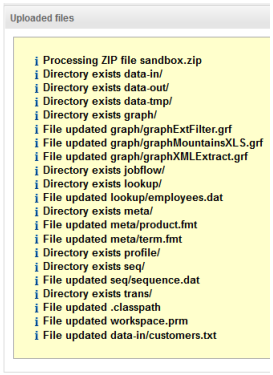



*Figure 3.8. Web GUI - upload ZIP results*

*Table 3.3. ZIP upload parameters*

| Label | Description |
|---|---|
| Encoding of packed file names | File names which contain special characters (non ASCII) are encoded. By this select box, you choose right encoding, so filenames are decoded properly. |
| Overwrite existing files | If this switch is checked, existing file is overwriten by new one, if both of them are stored in the same path in the sandbox and both of them have the same name. |
| Replace content | If this option is enabled, all files which are missing in uploaded ZIP file, but they exist in destination sandbox, will be deleted. This option might cause loose of data, so user must have special permission "May delete files, which are missing in uploaded ZIP" to enable it. |

## Download file in ZIP

Select file in left panel, then web GUI displays button "Download file as ZIP" in the tool bar on the right side.

Created ZIP contains just selected file. This feature is useful for large files (i.e. input or output file) which cannot be displayed directly in web GUI. So user can download it.

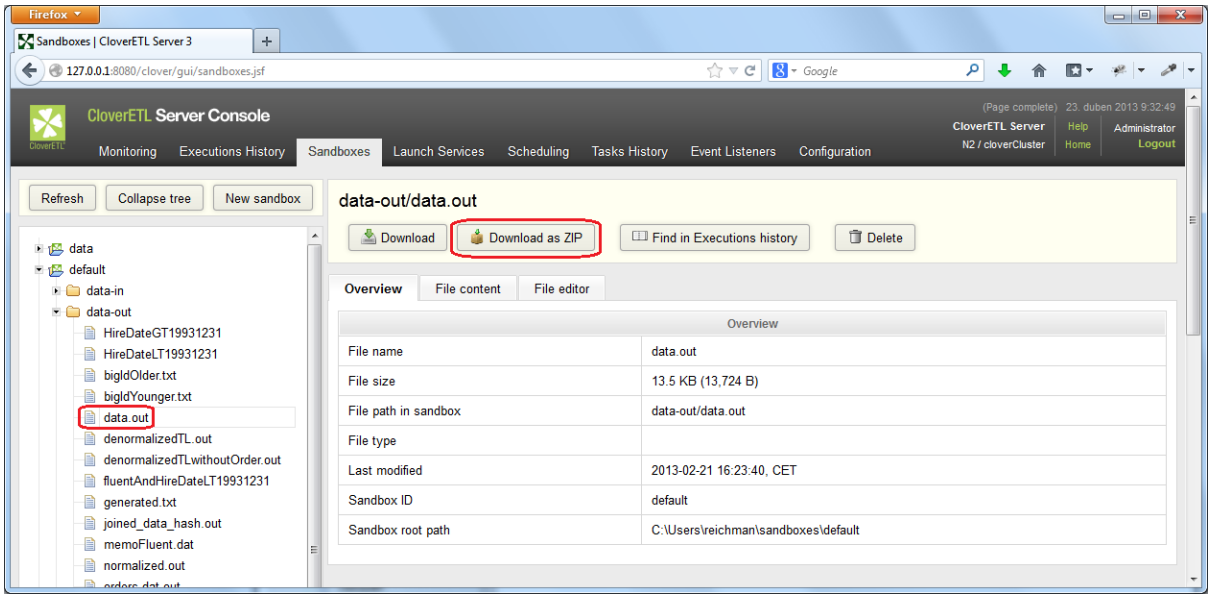



*Figure 3.9. Web GUI - download file as ZIP*

## Download file HTTP API

It is possible to download/view sandbox file accessing "download servlet" by simple HTTP GET request:

```
http://[host]:[port]/[Clover Context]/downloadFile?[Parameters]
```

Server requires BASIC HTTP Authentication. Thus with linux command line HTTP client "wget" it would look like this:

```
wget --user=clover --password=clover
    http://localhost:8080/clover/downloadFile?sandbox=default\&file=data-out/data.dat
```

Please note, that ampersand character is escaped by back-slash. Otherwise it would be interpreted as command-line system operator, which forks processes.

URL Parameters

- sandbox - Sandbox code. Mandatory parameter.
- file - Path to the file relative from sandbox root. Mandatory parameter.
- zip - If set to "true", file is returned as ZIP and response content type is "application/x-zip-compressed". By default it is false, so response is content of the file.

# Job config properties

Each ETL graph or Jobflow may have set of config properties, which are applied during the execution. Properties are editable in web GUI section "sandboxes". Select job file and go to tab "Config properties".

The same config properties are editable even for each sandbox. Values specified for sandbox are applied for each job in the sandbox, but with lower priority then config properties specified for the job.

If neither sandbox or job have config properties specified, defaults from main server configuration are applied. Global config properties related to Job config properties have prefix "execution.". E.g. server property "executor.classpath" is default for Job config property "classpath". (See Chapter 18, Configuration (p. 97) for details)

In addition, it is possible to specify additional job parameters, which can be used as placeholders in job XML. Please keep in mind, that these placeholders are resolved during loading and parsing of XML file, thus such job couldn't be pooled.

*Table 3.4. Job config parameters*

| | | |
|---|---|---|
| max_graph_instance_age | 0 | Time interval in ms which specifies how long may transformation instance last in server's cache. 0 means that transformation is initialized and released for each execution. Transformation cannot be stored in the pool and reused in some cases (transformation uses placeholders using dynamically specified parameters) |
| classpath | | List of paths or jar files which contain external classes used in the job file (transformations, generators, JMS processors). All specified resources will be added to runtime classpath of the transformation job. All Clover Engine libraries and libraries on application-server's classpath are automatically on the classpath. Separator is specified by Engine property "DEFAULT_PATH_SEPARATOR_REGEX". Directory path must always end with slash character "/", otherwise ClassLoader doesn't recognize it's a directory. Server always automatically adds "trans" subdirectory of job's sandbox, so It doesn't have to be added explicitly. |
| compile_classpath | | List of paths or jar files which contain external classes used in the job file (transformations, generators, JMS processors) and related libraries for their compilation. Please note, that libraries on application-server's classpath aren't included automatically. Separator is specified by Engine property "DEFAULT_PATH_SEPARATOR_REGEX". Directory path must always end with slash character "/", otherwise ClassLoader doesn't recognize it's a directory. Server always automatically adds "SANDBOX_ROOT/trans/" directory and all JARs in "SANDBOX_ROOT/lib/" directory, so they don't have to be added explicitly. |
| skip_check_config | default value is taken from engine property | Switch which specifies whether check config must be performed before transformation execution. |
| password | | Password for decoding of encoded DB connection passwords. |
| verbose_mode | true | If true, more descriptive logs of job runs are generated. |
| use_jmx | true | If true, job executor registers jmx mBean of running transformation. |
| debug_mode | false | If true, edges with enabled debug store data into files in debug directory. See property "graph.debug_path".<br><br>Without explicit setting, running of a graph from Designer with server integration would set the debug_mode to true. On the other hand, running of a graph from the server console sets the debug_mode to false. |
| executor.use_local_context_url | If true, the context URL of a running job will be a local "file:" URL. Otherwise, a "sandbox:" URL will be used. | false |
| executor.jobflow_token_tracking | If false, token tracking in jobflow executions will be disabled. | true |

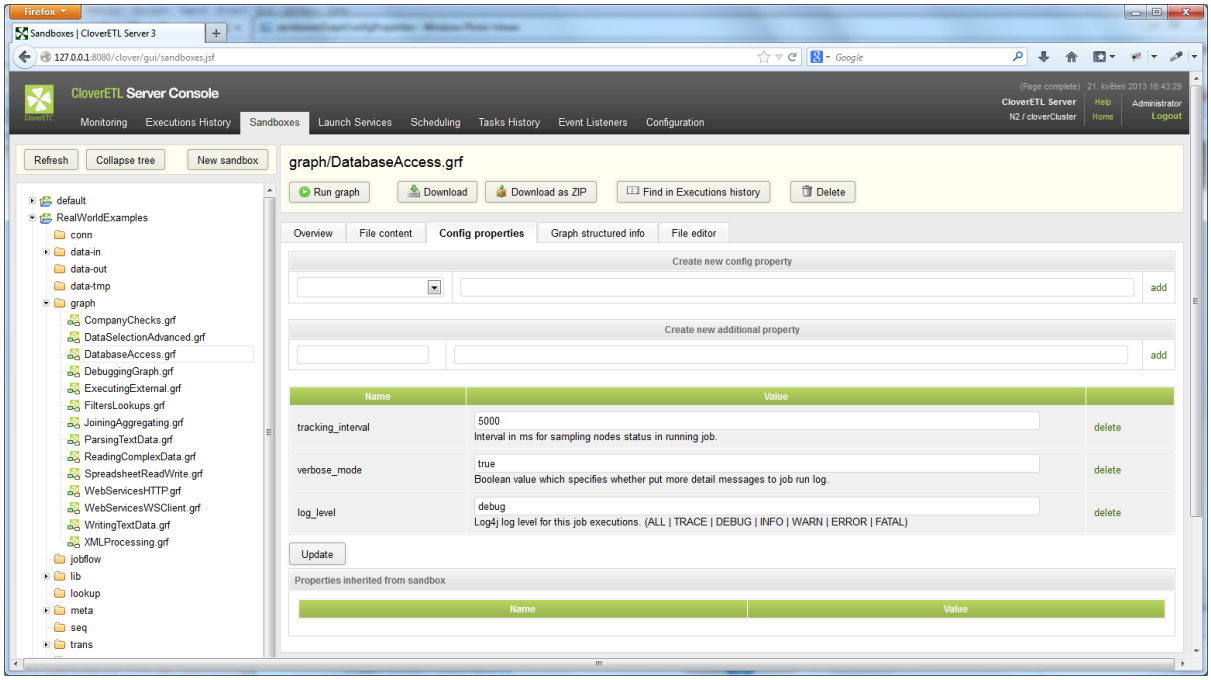



*Figure 3.10. Job config properties*

# Chapter 4. Viewing Job Runs - Executions History

Executions History shows the history of all jobs that the Server has executed – transformation graphs, jobflows, and Data Profiler jobs. You can use it to find out why a job failed, see the parameters that were used for a specific run, and much more.

The table shows basic information about the job: Run ID, Job file, current status, and time of execution, as well as some useful links. You will also find additional details after clicking on the job name in the list – details such as associated log files, parameter values, tracking, and more.

Please note that some jobs might not appear in the Executions History list. These are jobs that have disabled persistency for increased performance – e.g. some Launch Services disable storing the run information in order to increase service responsiveness.

## Filtering and ordering

Use the Filter panel to filter the view. By default, only parent tasks are shown (Show executions children) – e.g. master nodes in the Cluster and their workers are hidden by default.

Use the up and down arrows in the table header to sort the list. By default, the latest job is listed first.

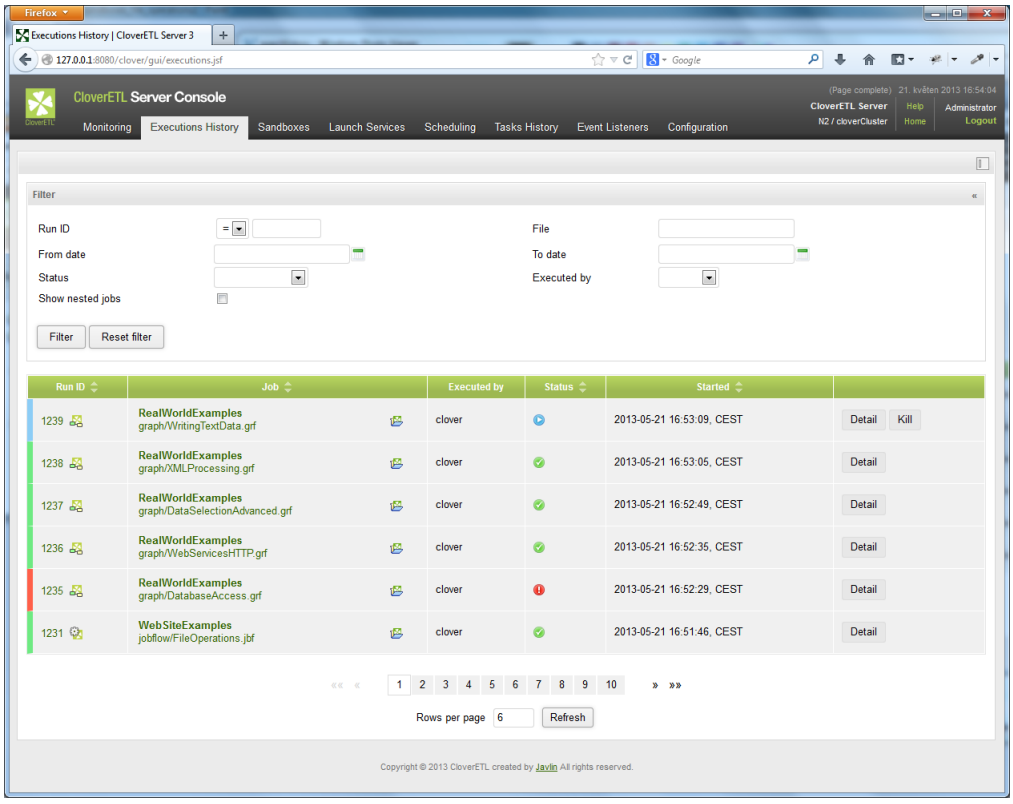



*Figure 4.1. Executions History - executions table*

When some job execution is selected in the table, the detail info is shown on the right side.

*Table 4.1. Persistent run record attributes*

| Attribute | Description |
|---|---|
| Run ID | Unique number identifying the run of the job. Server APIs usually return this number as simple response of execution request. It's useful as parameter of subsequent calls for specification of the job execution. |
| Execution type | Type of job as recognized by the server. STANDALONE for ETL graph, JOBFLOW for Jobflow, PROFILER_JOB for profiler, MASTER for main record of partitioned execution in cluster, PARTITIONED_WORKER for worker record of partitioned execution in cluster |
| Parent run ID | Run ID of the parent job. Typically Jobflow which executed this job, or master execution which encapsulate this worker execution. |
| Root run ID | Run ID of the root parent job. Job execution which wasn't executed by another parent job. |
| Execution group | Jobflow components may group sub-jobs using this attribute. See description of Jobflow componentes for details. |
| Nested jobs | Indication that this job execution has or has not any child execution. |
| Node | In cluster mode shows ID of the cluster node which this execution was running on. |
| Executed by | User which executed the job. Either directly using some API/GUI or indirectly using the scheduling or event listeners. |
| Sandbox | Sandbox containing job file. For jobs which are sent together with execution request, so the job file doesn't exist on the server site, it's set to "default" sandbox. |
| Job file | Path to job file, relative to the sandbox root. For jobs which are sent together with execution request, so the job file doesn't exist on the server site, it's set to generated string. |
| Job version | Revision of the job file. It's string generated by CloverETL Designer and stored in the job file. |
| Status | Status of the job execution. READY - waiting for execution start, RUNNING - processing job, FINISHED OK - job finished without any error, ABORTED - job was aborted directly using some API/GUI or by parent Jobflow, ERROR - job failed, N/A (not awailable) - server process died suddenly, so it couldn't properly abort the jobs, so after sertart the jobs with unknown status are set as N/A |
| Started | Server date-time (and timezone) of the execution start. |
| Finished | Server date-time (and timezone) of the execution finish. |
| Duration | Execution duration |
| Error in component ID | If the job failed due the error in a component, this field contains ID of the component. |
| Error in component type | If the job failed due the error in a component, this field contains type of the component. |
| Error message | If the job failed, this field contains error description. |
| Exception | If the job failed, this field contains error stack trace. |
| Input parameters | List of input parameters passed to the job. Job file can't be cached, since the parameters are applied during loading from the job file. Job file isn't cached by default. |
| Input dictionary | List of dictionary elements passed to the job. Dictionary is used independently on job file caching. |
| Output dictionary | List of dictionary elements at the moment the job ends. |

For jobs which have some children executions, e.g. partitioned or jobflows also executions hierarchy tree is shown.

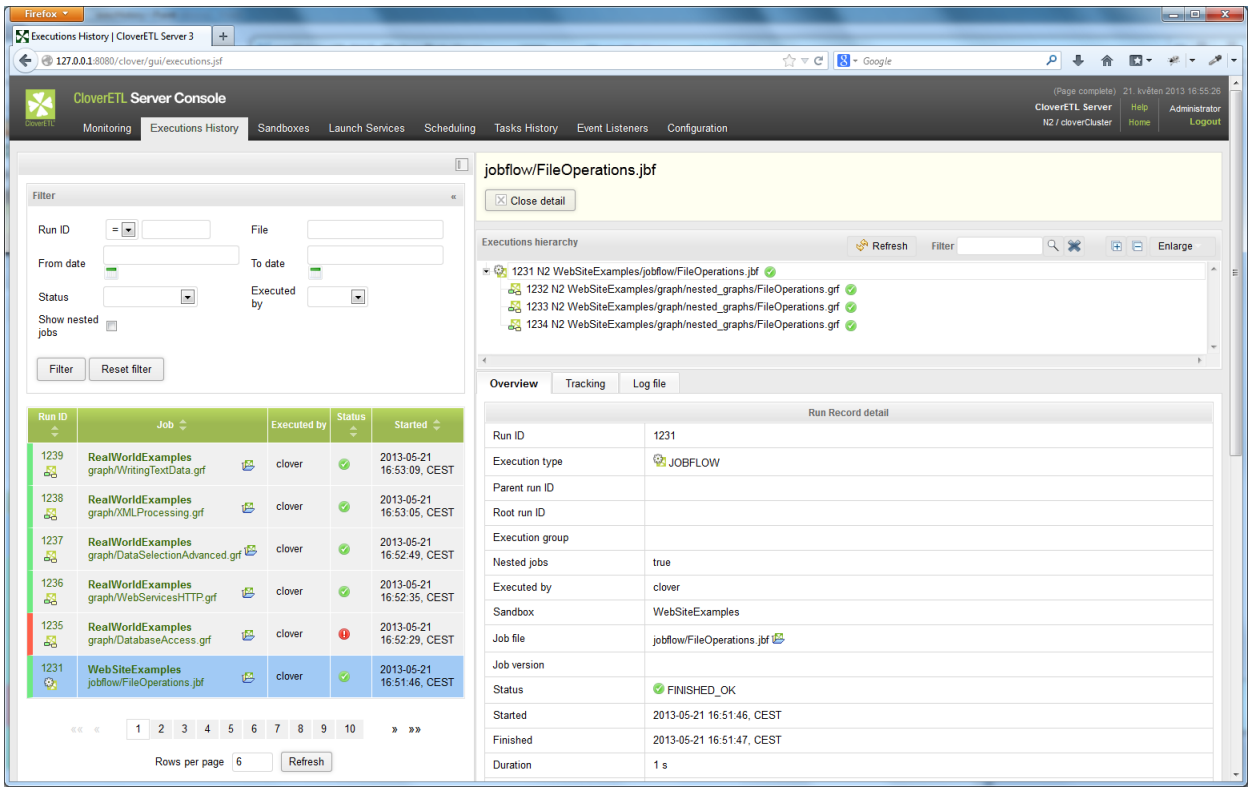



*Figure 4.2. Executions History - overall perspective*

Since the detail panel and expecially job logs may be wide, it may be usefull to hide table on the left, so the detail panel spreads. Click on the minimize icon on the top of the list panel to hide panel. Then to show list panel agan, click to the "Expand panel" icon on the left.

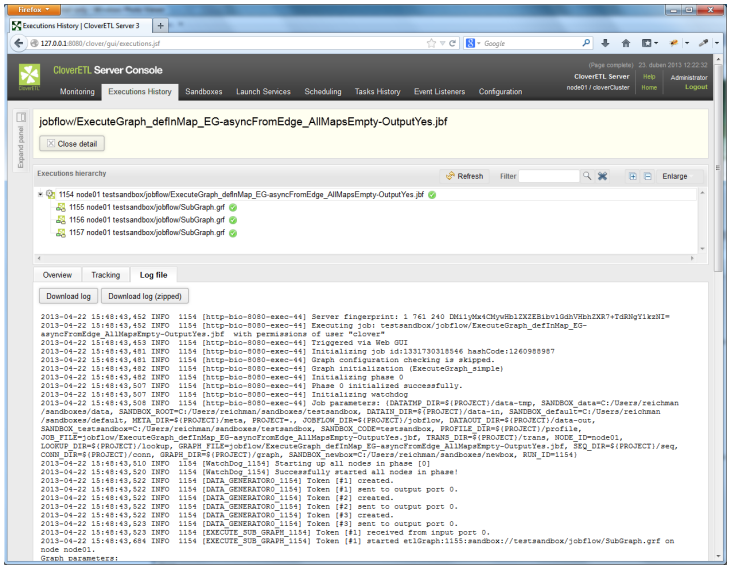



*Figure 4.3. Executions Hierarchy with docked list of jobs*

Executions hierarchy may be rather complex, so it's possible to filter the content of the tree by fulltext filter. However when the filter is used, the selected executions aren't hierarchically structured.

# Chapter 5. Users and Groups

The CloverETL Server has a built-in security module that manages users and groups. User groups control access permissions to sandboxes and operations the users can do on the Server, including authenticated calls to Server API functions. A single user can belong to multiple groups.

LDAP or Active Directory can be configured with the Server to authenticate users and assign their effective groups (and permissions) from a global directory.

You can manage users and user groups in **Configuration/Users and Groups**. Please note that you need a “List users” (“List groups” respectively) permission for that.

The security module can be globally switched off (see Chapter 18, Configuration (p. 97)).

## LDAP authentication

Since 3.2 it's possible to configure CloverETL Server to use LDAP server for users authentication. So the credentials of users registered in LDAP may be used for authentication to any CloverETL Server interface (API or GUI).

However authorization (access levels to sandboxes content and privileges for operations) is still handled by Clover security module. Each user, event though logged-in using LDAP authentication, must have his own "user" record (with related groups) in CloverETL security module. So there must be the user with the same username and domain set to "LDAP". If no such user record exists, it's automatically created according to CloverETL configuration.

What does the CloverETL do to authenticate a LDAP user?

1. User specifies the LDAP credentials i.e. in login form to the web GUI
2. CloverETL Server connects to the LDAP and checks whether the user exists (it uses specified search to lookup in LDAP)
3. If the user exists in LDAP, CloverETL Server performs authentication
4. If succeeded, CloverETL Server searches for LDAP user's groups.
5. CloverETL Server checks whether the user is assigned in LDAP groups which are allowed to login to Clover.
6. Clover user record is created/updated according to current LDAP values.
7. Clover user is assigned to the Clover groups according to his current assignation to the LDAP groups.
8. User is logged-in

### Note

Switching domains:

- If a user was **created as LDAP** and then switched to clover domain, you have to **set a password** for him in **Change password tab**.
- If a user was **created as clover** and then switched to LDAP domain, he has a password in clover domain, but it is overriden by the LDAP password. After switching back to clover domain, the **original password is re-used**. It can be reset in the **Change password** tab if needed (e.g. forgotten).

### Configuration

By default CloverETL Server allows only its own internal mechanism for authentication. To enable authentication with LDAP, set config property "security.authentication.allowed_domains" properly. It's list of user domains which are used for authentication.

Currently there are 2 authentication mechanism implemented: "LDAP" and "clover" ("clover" is identifier of CloverETL internal authentication and may be changed by security.default_domain property, but only for white-labelling purposes). To enable LDAP authentication, set value to "LDAP" (only LDAP) or "clover,LDAP". Users from both domain may login. It's recommended to allow both mechanisms together, until the LDAP is properly configured. So the admin user can still login to web GUI although the LDAP connection isn't properly configured.

## Basic LDAP connection properties

```
# Implementation of context factory
security.ldap.ctx_factory=com.sun.jndi.ldap.LdapCtxFactory
# timeout for all queries sent to LDAP server
security.ldap.timeout=5000
# limit for number of records returned from LDAP
security.ldap.records_limit=50

# URL of LDAP server
security.ldap.url=ldap://hostname:port
# Some generic UserDN which allows lookup for the user and groups.
security.ldap.userDN=
# Password for the user specified above
security.ldap.password=
```

## Configuration of user lookup

Specified values work for this specific LDAP tree:

- dc=company,dc=com
  - ou=groups
    - cn=admins (objectClass=groupOfNames,member=(uid=smith,dc=company,dc=com),member=(uid=jones,dc=company,dc=com))
    - cn=developers (objectClass=groupOfNames,member=(uid=smith,dc=company,dc=com))
    - cn=consultants (objectClass=groupOfNames,member=(uid=jones,dc=company,dc=com))
  - ou=people
    - uid=smith (fn=John,sn=Smith,mail=smith@company.com)
    - uid=jones (fn=Bob,sn=Jones,mail=jones@company.com)

Following properties are necessary for lookup for the LDAP user by his username. (step [2] in the login process above)

```
# Base specifies the node of LDAP tree where the search starts
security.ldap.user_search.base=dc=company,dc=eu
# Filter expression for searching the user by his username.
# Please note, that this search query must return just one record.
# Placeholder ${username} will be replaced by username specified by the logging user.
security.ldap.user_search.filter=(uid=${username})
# Scope specifies type of search in "base". There are three possible values: SUBTREE | ONELEVEL | OBJECT
# http://download.oracle.com/javase/6/docs/api/javax/naming/directory/SearchControls.html
security.ldap.user_search.scope=SUBTREE
```

Following properties are names of attributes from the search defined above. They are used for getting basic info about the LDAP user in case the user record has to be created/updated by Clover security module: (step [6] in the login process above)

```
security.ldap.user_search.attribute.firstname=fn
security.ldap.user_search.attribute.lastname=sn
security.ldap.user_search.attribute.email=mail
# This property is related to the following step "searching for groups".
# Groups may be obtained from specified user's attribute, or found by filter (see next paragraph)
# Please leave this property empty if the user doesn't have such attribute.
security.ldap.user_search.attribute.groups=memberOf
```

In the following step, clover tries to find groups which the user is assigned to. (step [4] in the login process above). There are two ways how to get list of groups which the user is assigned to. The user-groups relation is specified on the "user" side. The user record has some attribute with list of groups. It's "memberOf" attribute usually. Or the relation is specified on the "group" side. The group record has attribute with list of assigned users. It's "member" attribute usually.

In case the relation is specifien on users side, please specify property:

```
security.ldap.user_search.attribute.groups=memberOf
```

Leave it empty otherwise.

In case the relation is specifien on groups side, please specify properties for searching:

```
security.ldap.groups_search.base=dc=company,dc=com
# Placeholder ${userDN} will be replaced by user DN found by the search above
# If the filter is empty, searching will be skipped.
security.ldap.groups_search.filter=(&(objectClass=groupOfNames)(member=${userDN}))
security.ldap.groups_search.scope=SUBTREE
```

Otherwise, please leave property security.ldap.groups_search.filter empty, so the search will be skipped.

Clover user record will be assigned to the clover groups according to the LDAP groups found by the search (or the attribute). (Groups check is performed during each login)

```
# Value of the following attribute will be used for lookup for the Clover group by its code.
# So the user will be assigned to the Clover group with the same "code"
security.ldap.groups_search.attribute.group_code=cn
```

It's also possible to specify LDAP groups which are able to login to Clover. (step [5] in the login process above)

```
# Semicolon separated list of LDAP group DNs (distinguished names).
# LDAP user must be assigned to one or more of these groups, otherwise new clover user can't be created.
# Special value "_ANY_" disables this check and basically any LDAP user may login.
# If the LDAP group DNs are configured, also security.ldap.groups_search.* properties must be configured.
# value could be e.g. "cn=developers,dc=company,dc=com;cn=admins,dc=company,dc=com"
security.ldap.allowed_ldap_groups=_ANY_
```

## Web GUI section Users

This section is intended to users management. It offers features in dependence of user's permissions. i.e. User may enter this section, but cannot modify anything. Or user may modify, but cannot create new users.

All possible features of users section:

- *create new user*
- *modify basic data*
- *change password*
- *disable/enable user*
- *assign user to groups* - Assignment to groups gives user proper permissions

*Table 5.1. After default installation above empty DB, there are two users created*

| User name | Description |
|---|---|
| clover | Clover user has admin permissions, thus default password "clover" should be changed after installation. |
| system | System user is used by application instead of common user, when no other user can be used. i.e. when security is globally switched off. This user cannot be removed and it is impossible to login as this user. |

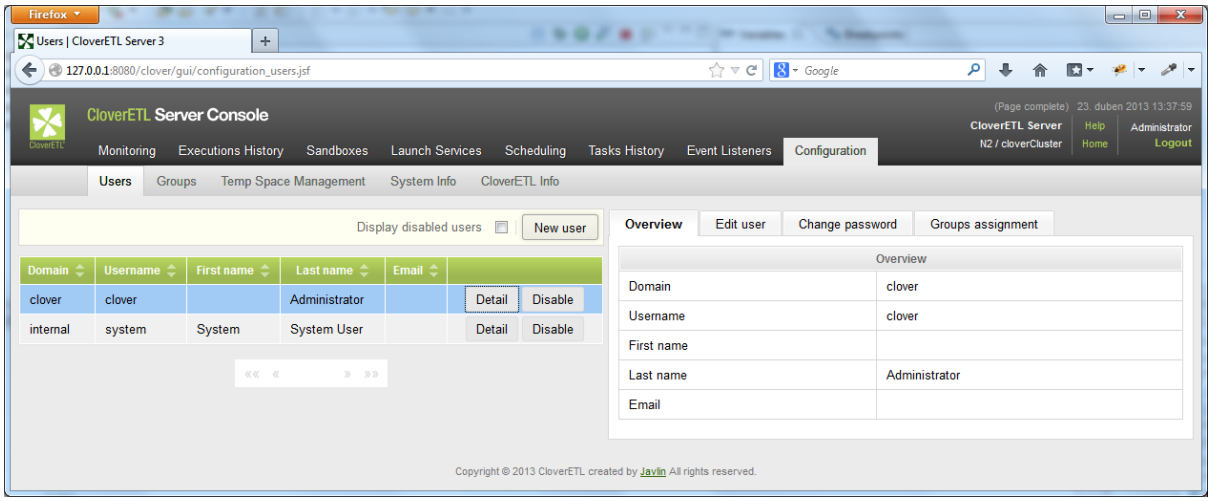



*Figure 5.1. Web GUI - section "Users" under "Configuration"*

*Table 5.2. User attributes*

| Attribute | Description |
|---|---|
| Domain | Domain which is the origin of the user. There are only two possible values currently: "clover" or "ldap". |
| Username | Common user identifier. Must be unique, cannot contain spaces or special characters, just letters and numbers. |
| Password | Case sensitive password. If user looses his password, the new one must be set. Password is stored in encrypted form for security reasons, so it cannot be retrieved from database and must be changed by the user who has proper permission for such operation. |
| First name | |
| Last name | |
| Email | Email which may be used by CloverETL administrator or by CloverETL server for automatic notifications. See Task - Send Email (p. 57) for details. |

## Edit user record

User with permission "Create user" or "Edit user" can use this form to set basic user parameters.

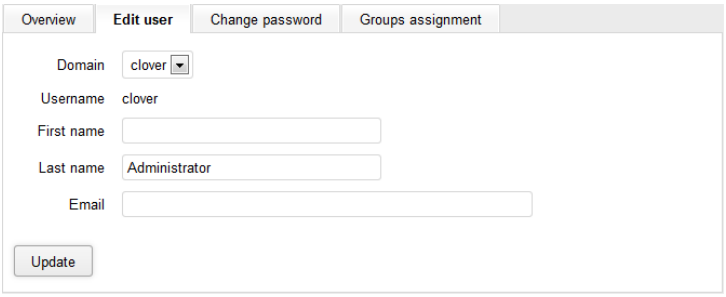



*Figure 5.2. Web GUI - edit user*

## Change users Password

If user looses his password, the new one must be set. So user with permission "Change passwords" can use this form to do it.

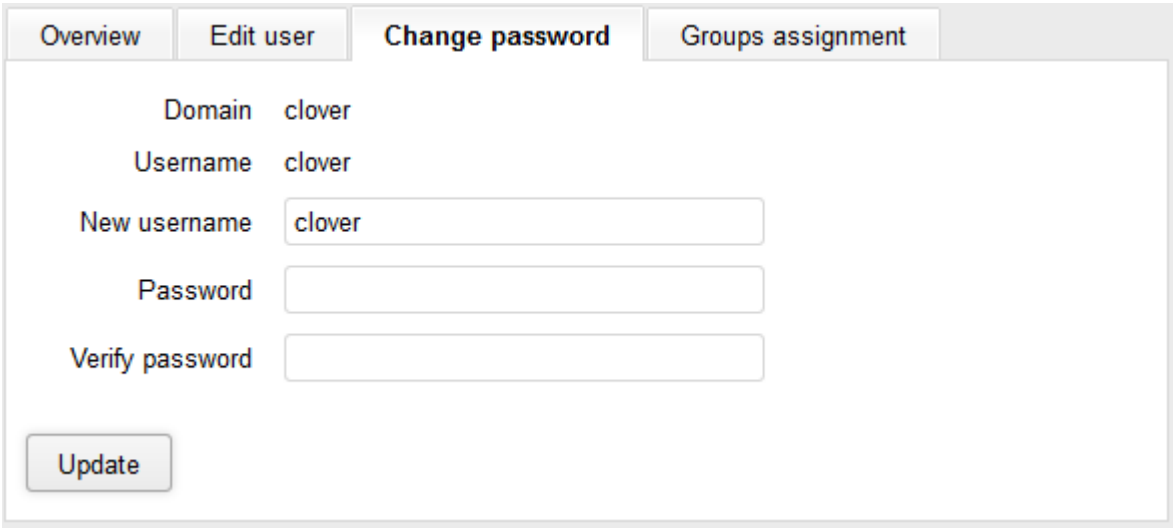



*Figure 5.3. Web GUI - change password*

## Group assignment

Assignment to groups gives user proper permissions. Only logged user with permission "Groups assignment" can access this form and specify groups which the user is assigned in. See Web GUI section Groups (p. 42) for details about permissions.

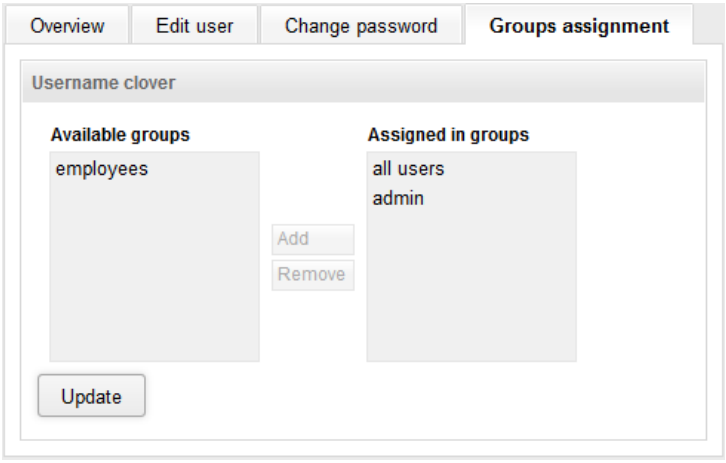



*Figure 5.4. Web GUI - groups assignment*

## Disabling / enabling users

Since user record has various relations to the logs and history records, it can't be deleted. So it's disabled instead. It basically means, that the record doesn't display in the list and the user can't login.

However disabled user may be enabled again. Please note, that disabled user is removed from its groups, so groups should be assigned properly after re-enabling.

# Web GUI section Groups

Group is abstract set of users, which gives assigned users some permissions. So it is not necessary to specify permission for each single user.

There are independent levels of permissions implemented in CloverETL Server

- *permissions to Read/Write/eXecute in sandboxes* - sandbox owner can specify different permissions for different groups. See Sandbox Content Security and Permissions (p. 27) for details.
- *permissions to perform some operation* - user with operation permission "Permission assignment" may assign specific permission to existing groups.
- *permissions to launch specific service* - see Chapter 17, Launch Services (p. 91) for details.

*Table 5.3. Default groups created during installation*

| Group name | Description |
|---|---|
| admins | This group has operation permission "all" assigned, which means, that it has unlimited permission. Default user "clover" is assigned to this group, which makes him administrator. |
| all users | Every single CloverETL user is assigned to this group by default. It is possible to remove user from this group, but it is not a recommended approach. This group is useful for some permissions to sandbox or some operation, which you would like to make accessible for all users without exceptions. |

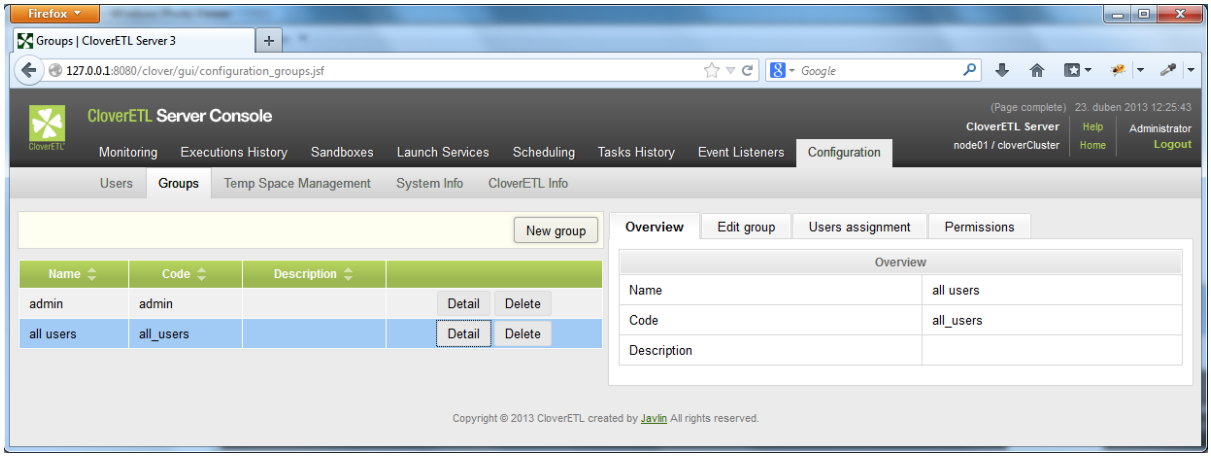



*Figure 5.5. Web GUI - section "Groups"*

## Users Assignment

Relation between users and groups is N:M. Thus in the same way, how groups are assignable to users, users are assignable to groups.

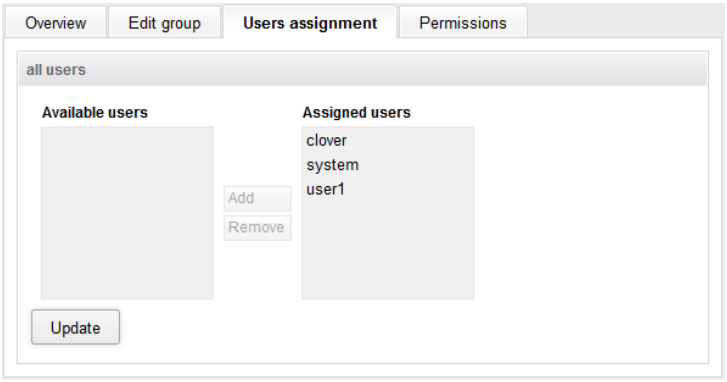



*Figure 5.6. Web GUI - users assignment*

## Groups permissions

Groups permissions are structured as tree, where permissions are inherited from root to leafs. Thus if some permission (tree node) is enabled (blue dot), all permissions in sub tree are automatically enabled (white dot). Permissions with red cross are disabled.

Thus for "admin" group just "all" permission is assigned, every single permission in sub tree is assigned automatically.

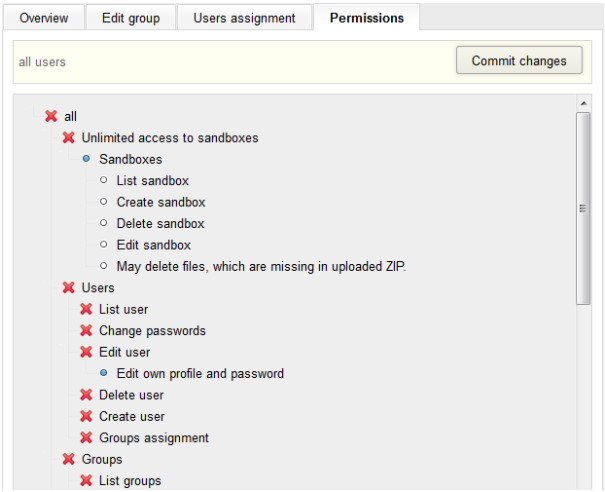



*Figure 5.7. Tree of permissions*

# Chapter 6. Scheduling

The scheduling module allows you to create a time schedule for operations you need to trigger in a repetitive or timely manner.

Similar to “cron” from Unix systems, each schedule represents a separate time schedule definition and a task to perform.

In the Cluster, you can explicitly specify which node should execute the scheduled task using the Node ID parameter. However, if not set, the node will be selected automatically from all available nodes (but always just one).

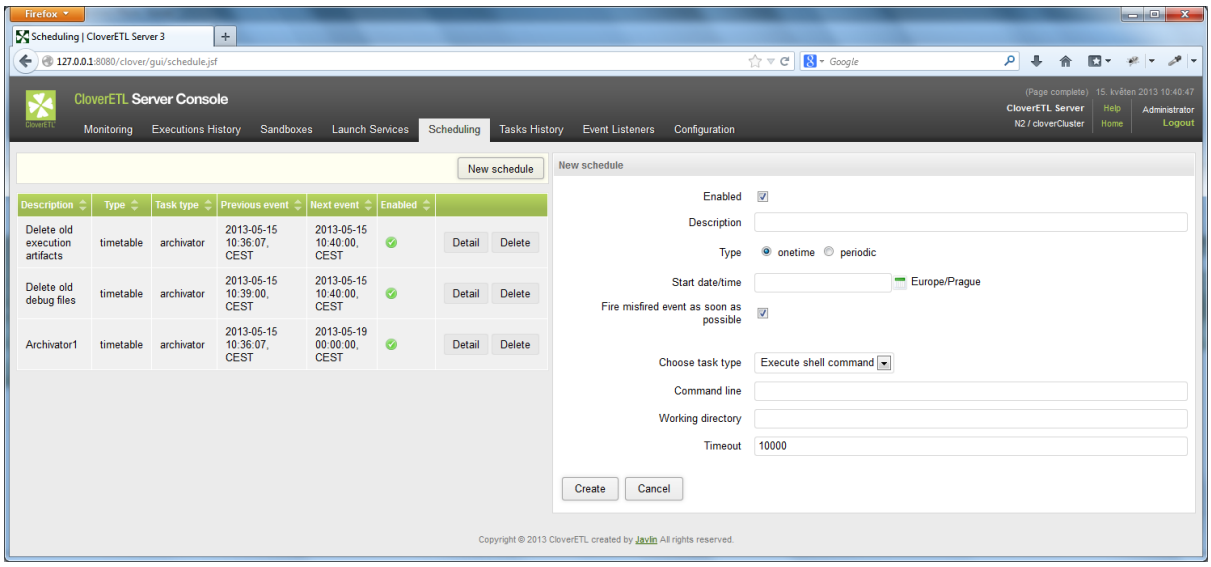



*Figure 6.1. Web GUI - section "Scheduling" - create new*

## Timetable Setting

This section should describe how to specify WHEN schedule should be triggered. Please keep in mind, that exact trigger times are not guaranteed. There may be couple of seconds delay. Schedule itself can be specified in different ways.

- Onetime Schedule (p. 45)
- Periodical schedule by Interval (p. 47)
- Periodical schedule by timetable (Cron Expression) (p. 47)

### Onetime Schedule

It is obvious, that this schedule is triggered just once.

*Table 6.1. Onetime schedule attributes*

| | |
|---|---|
| Type | "onetime" |
| Start date/time | Date and time, specified with minutes precision. |
| Fire misfired ASAP switch | If checked and trigger time is missed because of any reason (i.e. server restart), it will be triggered immediately, when it is possible. Otherwise it is ignored and it will be triggered at next scheduled time. |

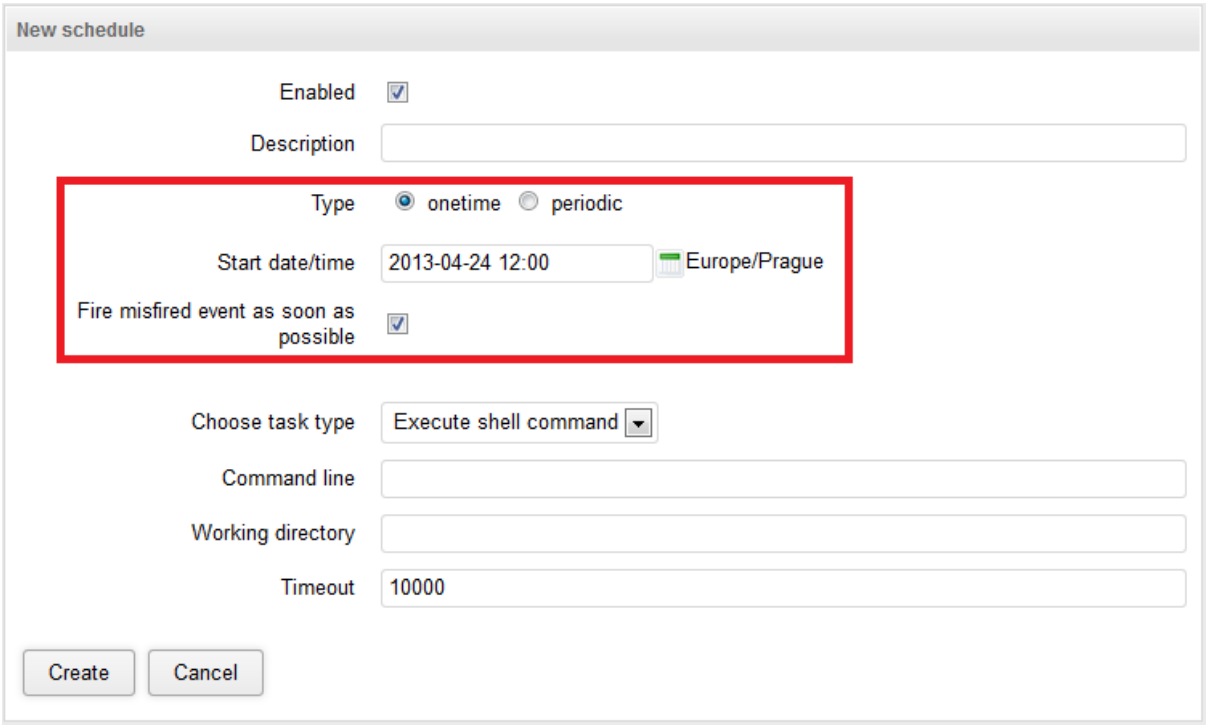



*Figure 6.2. Web GUI - onetime schedule form*

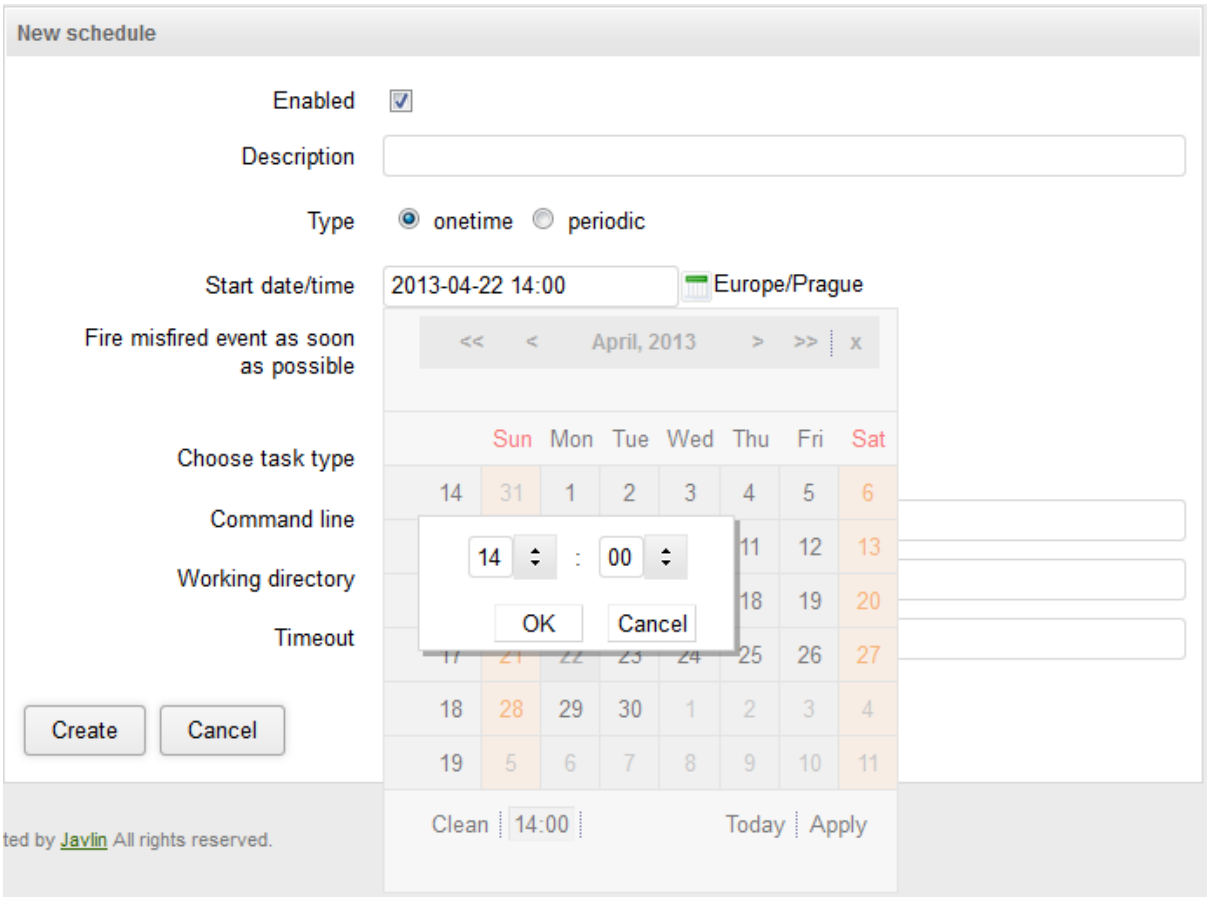



*Figure 6.3. Web GUI - schedule form - calendar*

## Periodical schedule by Interval

This type of schedule is the simplest periodical type. Trigger times are specified by these attributes:

*Table 6.2. Periodical schedule attributes*

| | |
|---|---|
| Type | "periodic" |
| Periodicity | "interval" |
| Not active before date/time | Date and time, specified with minutes precision. |
| Not active after date/time | Date and time, specified with minutes precision. |
| Interval (minutes) | Specifies interval between two trigger times. Next task is triggered even if previous task is still running. |
| Fire misfired ASAP switch | If checked and trigger time is missed because of any reason (i.e. server restart), it will be triggered immediately, when it is possible. Otherwise it is ignored and it will be triggered at next scheduled time. |

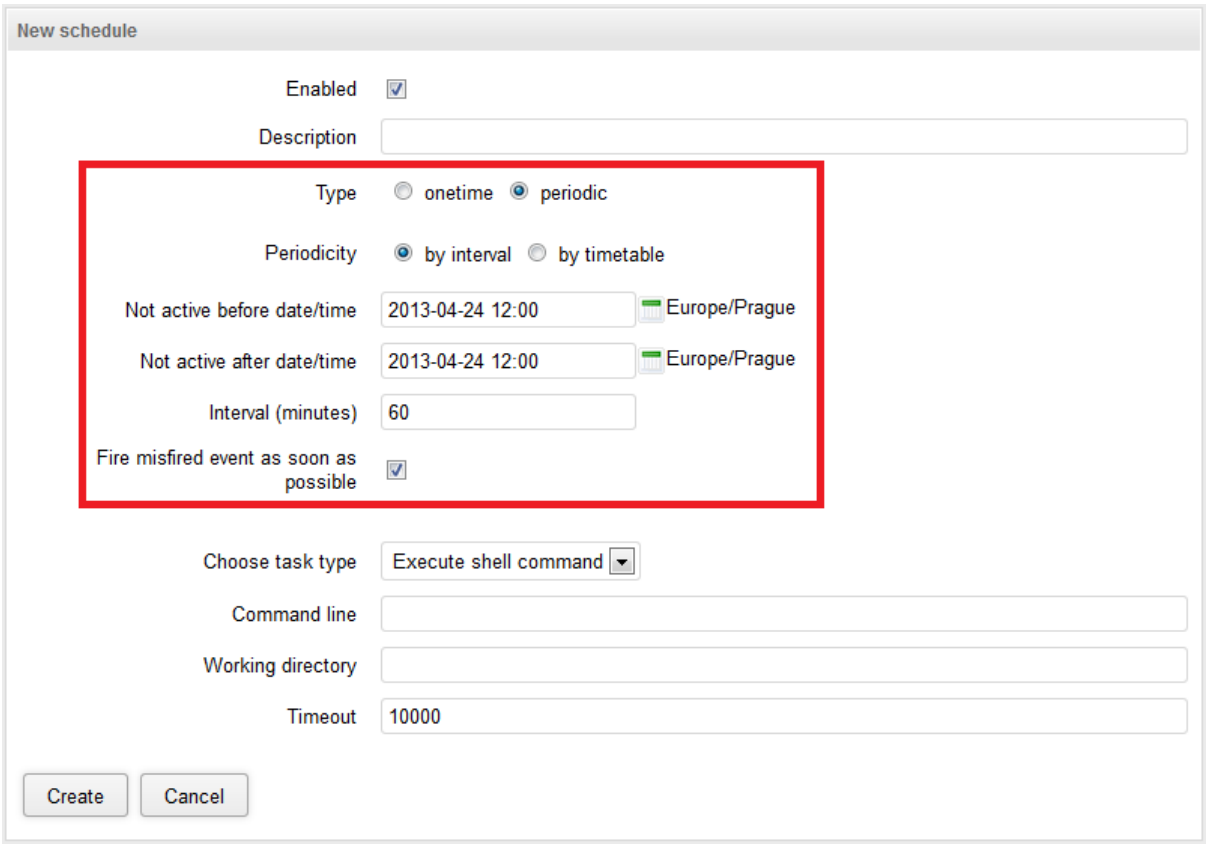



*Figure 6.4. Web GUI - periodical schedule form*

## Periodical schedule by timetable (Cron Expression)

Timetable is specified by powerful (but a little bit tricky) cron expression.

*Table 6.3. Cron periodical schedule attributes*

| | |
|---|---|
| Type | "periodic" |
| Periodicity | "interval" |
| Not active before date/time | Date and time, specified with minutes precision. |
| Not active after date/time | Date and time, specified with minutes precision. |
| Cron expression | Cron is powerful tool, which uses its own format for scheduling. This format is well known among UNIX administrators. i.e. "0 0/2 4-23 * * ?" means "every 2 minutes between 4:00am and 11:59pm". |
| Fire misfired ASAP switch | If checked and trigger time is missed because of any reason (i.e. server restart), it will be triggered immediately when it is possible. Otherwise it is ignored and it will be triggered at next scheduled time. |

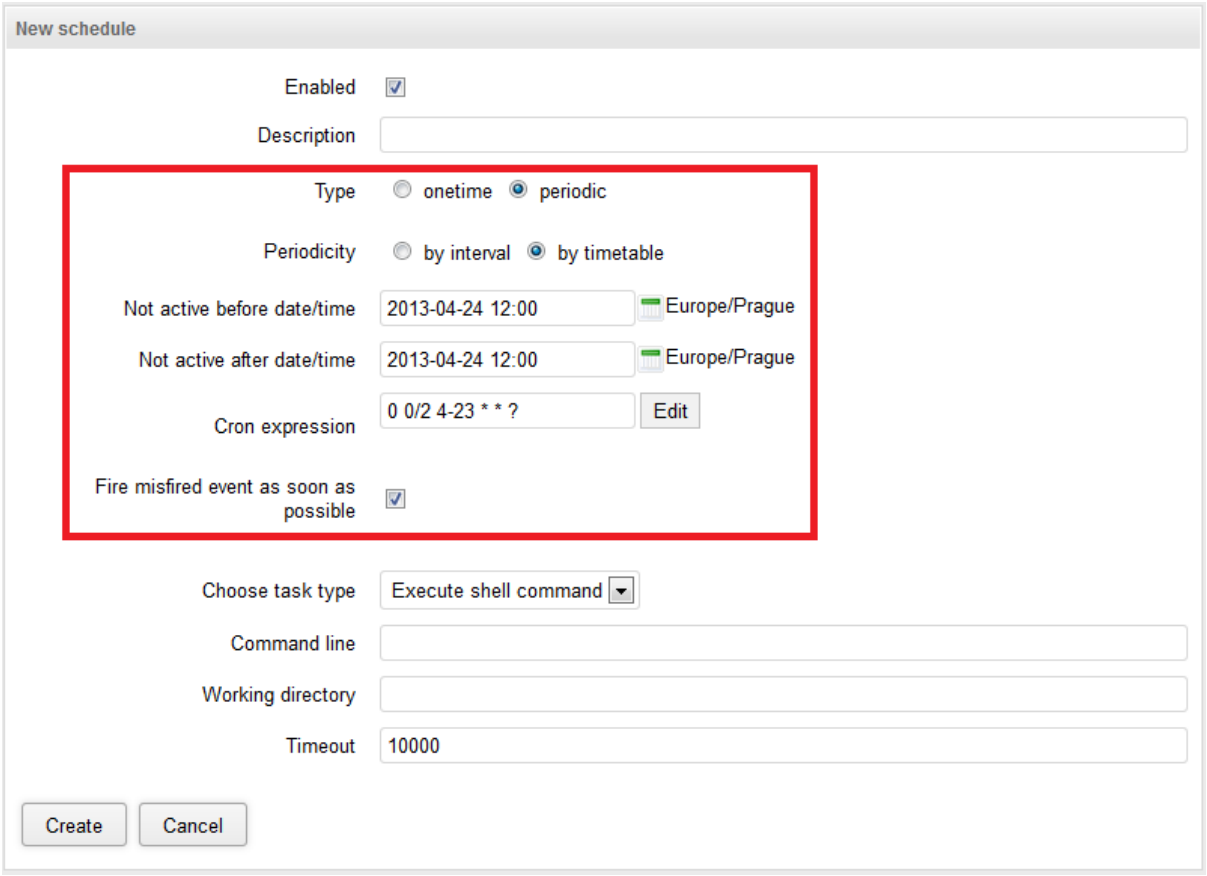



*Figure 6.5. Cron periodical schedule form*

# Tasks

Task basically specifies WHAT to do at trigger time. There are several tasks implemented for schedule and for graph event listener as follows:

- Task - Execution of Graph (p. 49)
- Task - Execution of Jobflow (p. 50)
- Task - Abort job (p. 50)
- Task - Execution of Shell Command (p. 51)

- Task - Send Email (p. 52)
- Task - Execute Groovy Code (p. 52)
- Task - Archivator (p. 53)

## Task - Execution of Graph

Please note that behaviour of this task type is almost the same as Task - Execution of Jobflow (p. 50)

*Table 6.4. Attributes of "Graph execution" task*

| | |
|---|---|
| Task type | "Start a graph" |
| Sandbox | This select box contains sandboxes which are readable for logger user. Select sandbox which contains graph to execute. |
| Graph | This select box is filled by all graphs files accessible in selected sandbox. |
| Parameters | Key-value pairs which are passed to the executed job as parameters. Besides, if this task is triggered by job (graph or jobflow) event, you can specify source job parameters, which shall be passed from the source job to executed job. i.e. event source has these parameters: paramName2 with value "val2", paramName3 with value "val3", paramName5 with value "val5". Task has "Parameters" attribute set like this:<br><br>`paramName1=paramValue1`<br>`paramName2=`<br>`paramName3`<br>`paramName4`<br><br>So executed job gets these parameters and values: paramName1 with value "paramValue1" (specified explicitly in the task configuration) paramName2 with value "" (empty string specified explicitly in the task configuration overrides event source parameters), paramName3 with value "val3" (value is taken from event source). These parameters aren't passed: paramName4 isn't passed, since it does not have any value in event source. paramName5 isn't passed, since it is not specified among the parameters to pass in the task.<br><br>Event parameters like "EVENT_RUN_RESULT", "EVENT_RUN_ID" etc. are passed to the executed job without limitations. |

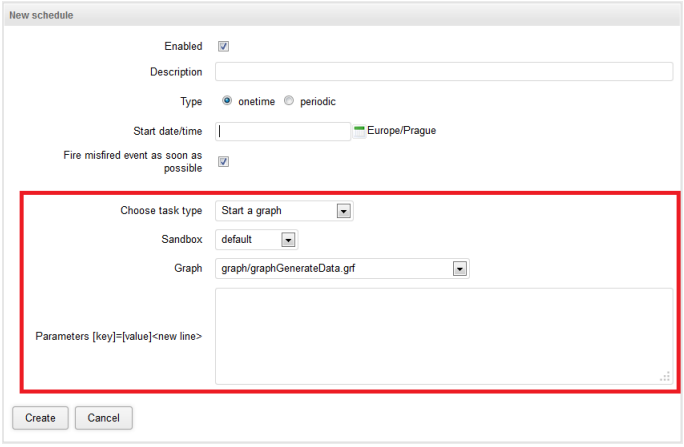



*Figure 6.6. Web GUI - Graph execution task*

## Task - Execution of Jobflow

Please note that behaviour of this task type is almost the same as Task - Execution of Graph (p. 49)

*Table 6.5. Attributes of "Jobflow execution" task*

| | |
|---|---|
| Task type | "Start a jobflow" |
| Sandbox | This select box contains sandboxes which are readable for logger user. Select sandbox which contains jobflow to execute. |
| Jobflow | This select box is filled by all jobflow files accessible in selected sandbox. |
| Parameters | Key-value pairs which are passed to the executed job as parameters. Besides, if this task is triggered by job (graph or jobflow) event, you can specify source job parameters, which shall be passed from the source job to executed job. i.e. event source has these parameters: paramName2 with value "val2", paramName3 with value "val3", paramName5 with value "val5". Task has "Parameters" attribute set like this:<br><br>`paramName1=paramValue1`<br>`paramName2=`<br>`paramName3`<br>`paramName4`<br><br>So executed job gets these parameters and values: paramName1 with value "paramValue1" (specified explicitly in the task configuration) paramName2 with value "" (empty string specified explicitly in the task configuration overrides event source parameters), paramName3 with value "val3" (value is taken from event source). These parameters aren't passed: paramName4 isn't passed, since it does not have any value in event source. paramName5 isn't passed, since it is not specified among the parameters to pass in the task.<br><br>Event parameters like "EVENT_RUN_RESULT", "EVENT_RUN_ID" etc. are passed to the executed job without limitations. |

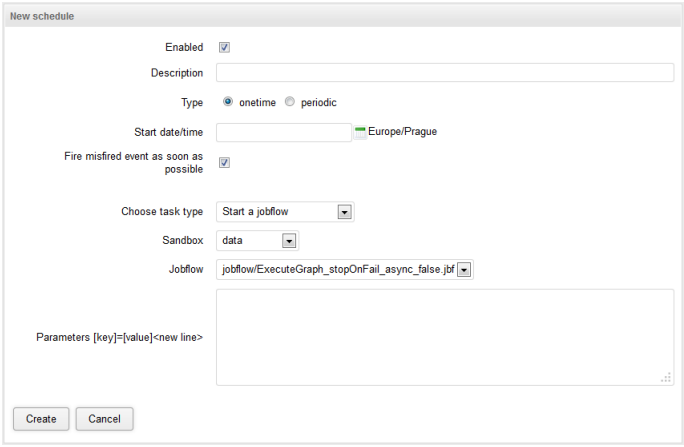



*Figure 6.7. Web GUI - Jobflow execution task*

## Task - Abort job

This task, when activated kills/aborts specified job (ETL graph or jobflow), if it is currently running.

*Table 6.6. Attributes of "Abort job" task*

| | |
|---|---|
| Task type | "Abort job" |
| Kill source of event | If this switch is on, task will kill job which is source of the event, which activated this task. Attributes sandbox and job are ignored. |
| Sandbox | Select sandbox which contains job to kill. This attribute works only when "Kill source of event" switch is off. |
| Job | This select box is filled by all jobs accessible in selected sandbox. All instances of selected job, whose are currently running will be killed. This attribute works only when "Kill source of event" switch is off. |

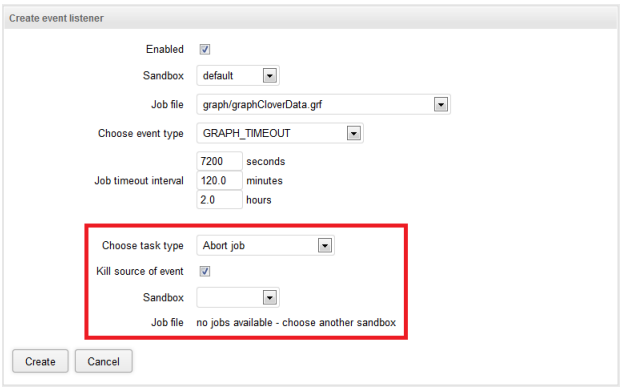



*Figure 6.8. Web GUI - "Abort job"*

## Task - Execution of Shell Command

*Table 6.7. Attributes of "Execute shell command" task*

| | |
|---|---|
| Task type | "Execute shell command" |
| Command line | Command line for execution of external process. |
| Working directory | Working directory for process. If not set, working directory of application server process is used. |
| Timeout | Timeout in milliseconds. After period of time specified by this number, external process is terminated and all results are logged. |

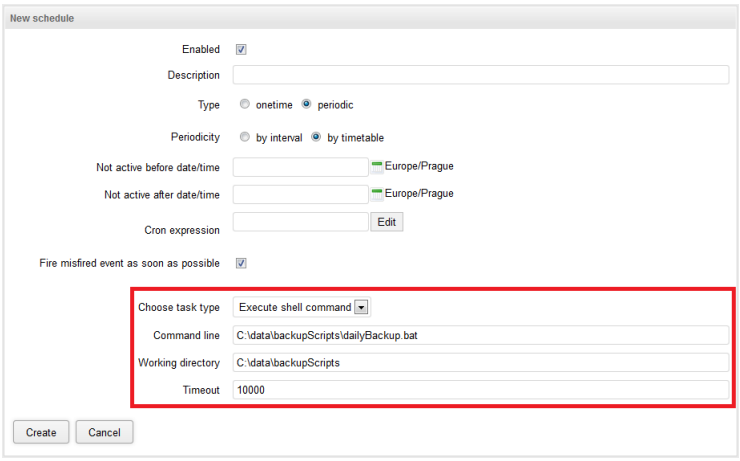



*Figure 6.9. Web GUI - shell command*

## Task - Send Email

This task is very useful, but for now only as response for graph events. This feature is very powerful for monitoring. (see Chapter 7, Graph Event Listeners (p. 55) for description of this task type).

*Note: It seems useless to send emails periodically, but it may send current server status or daily summary. These features will be implemented in further versions.*

## Task - Execute Groovy Code

This type of task allows execute code written in script language Groovy. It is possible to use some variables. Only parameter of this task is source code of written in Groovy.

CloverETL Server contains Groovy version 2.0.0

*Table 6.8. List of variables available in Groovy code*

| variable | class | description | availability |
|---|---|---|---|
| event | com.cloveretl.server.events.AbstractServerEvent | | every time |
| task | com.cloveretl.server.persistent.Task | | every time |
| now | java.util.Date | current time | every time |
| parameters | java.util.Properties | Properties of task | every time |
| user | com.cloveretl.server.persistent.User | Same as event.getUser() | every time |
| run | com.cloveretl.server.persistent.RunRecord | | When the event is instance of GraphServerEvent |
| tracking | com.cloveretl.server.persistent.TrackingGraph | Same as run.getTrackingGraph() | When the event is instance of GraphServerEvent |
| sandbox | com.cloveretl.server.persistent.Sandbox | Same as run.getSandbox() | When the event is instance of GraphServerEvent |
| schedule | com.cloveretl.server.persistent.Schedule | Same as ((ScheduleServerEvent)event).getSchedule() | When the event is instance of ScheduleServerEvent |
| servletContext | javax.servlet.ServletContext | | every time |
| cloverConfiguration | com.cloveretl.server.spring.CloverConfiguration | Configuration values for CloverETL Server | every time |
| serverFacade | com.cloveretl.server.facade.api.ServerFacade | Reference to the facade interface. Useful for calling CloverETL Server core.<br><br>WAR file contains JavaDoc of facade API and it is accessible on URL: http://host:port/clover/javadoc/index.html | every time |
| sessionToken | String | Valid session token of the user who owns the event. It is useful for authorisation to the facade interface. | every time |

Variables run, tracking and sandbox are available only if event is instance of GraphServerEvent class. Variable schedule is available only for ScheduleServerEvent as event variable class.

### Example of use Groovy script

This example shows script which writes text file describing finished graph. It shows use of 'run' variable.

```
import com.cloveretl.server.persistent.RunRecord;
String dir = "/tmp/";
RunRecord rr = (RunRecord)run;

String fileName = "report"+rr.getId()+"_finished.txt";

FileWriter fw = new FileWriter(new File(dir+fileName));
fw.write("Run ID       :"+rr.getId()+"\n");
fw.write("Graph ID     :"+rr.getGraphId()+"\n");
fw.write("Sandbox      :"+rr.getSandbox().getName()+"\n");
fw.write("\n");
fw.write("Start time   :"+rr.getStartTime()+"\n");
fw.write("Stop time    :"+rr.getStopTime()+"\n");
fw.write("Duration     :"+rr.getDurationString()+"\n");
fw.write("Final status :"+rr.getFinalStatus()+"\n");
fw.close();
```

## Task - Archivator

As name suggests, this task can archive (or delete) obsolete records from DB.

*Table 6.9. Attributes of "Archivator" task*

| | |
|---|---|
| Task type | "Archivator" |
| Older then | Time period (in minutes) - it specifies which records are evaluated as obsolete. Records older then the specified interval are stored in archives. |
| Archivator type | There are two possible values: "archive" or "delete". Delete removes records without any possibility of UNDO operation. Archive removes records from DB, but creates ZIP package with CSV files containing deleted data. |
| Output path for archives | This attribute makes sense only for "archive" type. |
| Include executions history | |
| Run record with status | If status is selected, only run records with specified status will be archived. It is useful e.g. If you want to delete records for successfully finished jobs, but you want to keep failed jobs for further investigation. |
| Include temp files | If checked, archivator removes all graph temporary files older then given timestamp defined in "Older than" attribute. The temporary files are files with graph debug data, dictionary files and files created by graph components. |
| Temp files with record status | If status is selected, only temp files related to the run records with selected status will be archived. It is useful e.g. If you want to delete files for successfully finished jobs, but you want to keep failed jobs for further investigation. |
| Include tasks history | If checked, archivator will include run records. Log files of graph runs are included as well. |
| Task types | If this task type is selected, only logs for selected task type are archived. |
| Task result mask | Mask applied to task log result attribute. Only records whose result meets this mask are archived. Specify string without any wildcards. Each task log which contains specified string in the "result" attribute will be deleted/archived. Case sensitivity depends on database collation. |
| Include profiler runs | If checked, archivator will include profiler job results. |

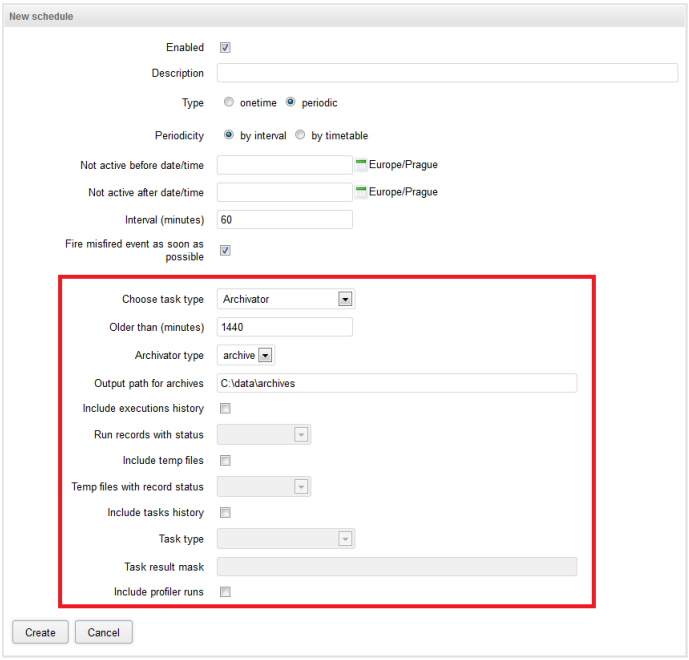



*Figure 6.10. Web GUI - archive records*

# Chapter 7. Graph Event Listeners

Graph Event Listeners allow you to define a task that the Server will execute as a reaction to the success or failure of executing a specific job (a transformation graph).

Each listener is bound to a specific graph and is evaluated every time the graph is executed (no matter whether manually, scheduled, via an API call, etc.).

You can use listeners to chain multiple jobs (creating a success listener that starts the next job in a row). However, we recommend using Jobflows to automate complex processes because of its better development and monitoring capabilities.

Graph Event Listeners are similar to Jobflow Event Listeners (Chapter 8, Jobflow Event Listeners (p. 63)) – for CloverETL Server both are simply “jobs”.

In the Cluster, the event and the associated task are executed on the same node the job was executed on. If the graph is distributed, the task will be executed on the master worker node. However, you can override where the task will be executed by explicitly specifying a Node ID in the task definition.

## Graph Events

Each event carries properties of graph, which is source of event. If there is a event listener specified, task may use these properties. i.e. next graphs in the chain may use "EVENT_FILE_NAME" placeholder which activated first graph in the chain. Graph properties, which are set specifically for each graph run (i.e. RUN_ID), are overridden by last graph.

For now, there are these types of graph events:



### graph started

Event of this type is created, when ETL graph execution successfully started.

### graph phase finished

Event of this type is created, everytime when graph phase is finished and all its nodes are finished with status `FINISHED_OK`.

### graph finished OK

Event of this type is created, when all phases and nodes of graph are finished with status `FINISHED_OK`.

### graph error

Event of this type is created, when graph cannot be executed from any reason, or when any node of graph fails.

### graph aborted

Event of this type is created, when graph is explicitly aborted.

### graph timeout

Event of this type is created, when graph runs longer then specified interval. Thus you have to specify "Job timeout interval" attribute for each listener of graph timeout event. You can specify this interval in seconds or in minutes or in hours.

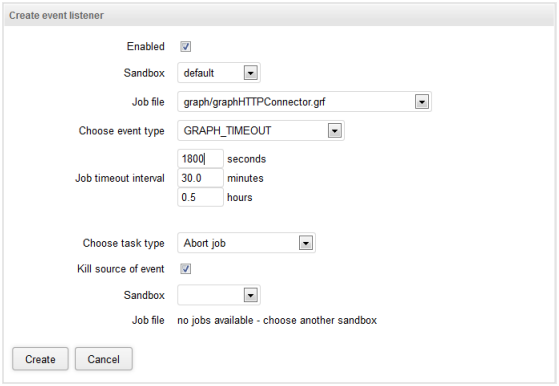



*Figure 7.1. Web GUI - graph timeout event*

### graph status unknown

Event of this type is created, when the server, during the startup, detects run records with undefined status in the executions history. Undefined status means, that server has been killed during graph run. Server automatically changes state of graph to "Not Available" and sends 'graph status unknown' event. Please note, that this works just for executions, which have persistent record in executions history. It is possible to execute transformation without persistent record in executions history, typically for better performance of fast running transformations (i.e. using Launch Services).

## Listener

User may create listener for specified event type and graph (or all graphs in sandbox). Listener is actually connection between graph event and task, where graph event specifies WHEN and task specifies WHAT to do.

So progress is like this:

- event is created
- listeners for this event are notified
- each listener performs related task

## Tasks

Task types "execute shell command", "execute graph" and "archivator" are described in section "scheduling" see this section for details about these task types. There is one more task type, which is useful especially with graph event listeners, thus it is described here. It is task type "send email".

*Note: You can use task of any type for both scheduling and graph event listener. Description of task types is divided into two sections just to show the most obvious use cases.*



## Task - Send Email

This type of task is useful for notifications about result of graph execution. I.e. you can create listener with this task type to be notified about each failure in specified sandbox or failure of particular graph.

*Table 7.1. Attributes of "Send email" task*

| | |
|---|---|
| Task type | "email" |
| Email pattern | This select box is available only when user is creating new record. It contains all predefined email patterns. If user chooses any of them, all fields below are automatically filled by values from pattern. |
| To | Recipient's email address. It is possible to specify more addresses separated by comma. It is also possible to use placeholders. *See* Placeholders *(p. 58) for details.* |
| Cc | Cc stands for 'carbon copy'. Copy of email will be delivered to these addresses. It is possible to specify more addresses separated by comma. It is also possible to use placeholders. *See* Placeholders *(p. 58) for details.* |
| BCc | Bcc: stands for 'Blind carbon copy'. It is the same as Cc, but the others recipients aren't aware, that these recipients get copy of email. |
| Reply-to (Sender) | Email address of sender. It must be valid address according to SMTP server. It is also possible to use placeholders. *See* Placeholders *(p. 58) for details.* |
| Subject | Email subject. It is also possible to use placeholders. *See* Placeholders *(p. 58) for details.* |
| Text | Body of email in plain text. Email is created as multipart, so HTML body should have a precedence. Plain text body is only for email clients which do not display HTML. It is also possible to use placeholders. *See* Placeholders *(p. 58) for details.* |
| HTML | Body of email in HTML. Email is created as multipart, so HTML body should have a precedence. Plain text body is only for email clients which do not display HTML. It is also possible to use placeholders. *See* Placeholders *(p. 58) for details.* |
| Log file as attachment | If this switch is checked, email will have an attachment with packed log file of related graph execution. |

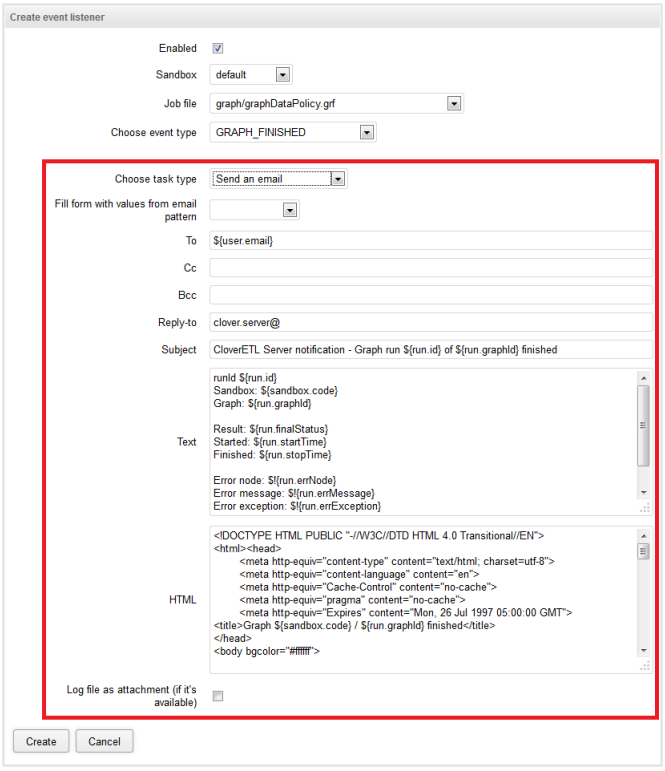



*Figure 7.2. Web GUI - send email*

*Note: Do not forget to configure connection to SMTP server (See Chapter 18, Configuration (p. 97) for details).*

## Placeholders

Place holder may be used in some fields of tasks. They are especially useful for email tasks, where you can generate content of email according to context variables.

*Note: In most cases, you can avoid this by using email patterns (See Email task for details)*

These fields are preprocessed by Apache Velocity templating engine. See Velocity project URL for syntax description http://velocity.apache.org/

There are several context variables, which you can use in place holders and even for creating loops and conditions.

- *event*
- *now*
- *user*
- *run*
- *sandbox*

Some of them may be empty in dependence of occasion which field is processed in. I.e. If task is processed because of graph event, then *run* and *sandbox* variables contain related data, otherwise they are empty,

*Table 7.2. Placeholders useful in email templates*

| Variable name | Contains |
|---|---|
| now | Current date-time |
| user | User, who caused this event. It may be owner of schedule, or someone who executed graph. Contains sub-properties, which are accessible using dot notation (i.e. ${user.email}) email, username, firstName, lastName, groups (list of values) |
| run | Data structure describing one single graph execution. Contains sub-properties, which are accessible using dot notation (i.e. ${run.graphId}) graphId, finalStatus, startTime, stopTime, errNode, errMessage, errException, logLocation |
| tracking | Data structure describing status of components in graph execution. Contains sub-properties, which are accessible using Velocity syntax for loops and conditions.<br>`#if (${tracking})`<br>`<table border="1" cellpadding="2" cellspacing="0">`<br>`#foreach ($phase in $tracking.trackingPhases)`<br>`<tr><td>phase: ${phase.phaseNumber}</td>`<br>`<td>${phase.execTime} ms</td>`<br>`<td></td><td></td><td></td></tr>`<br>`#foreach ($node in $phase.trackingNodes)`<br>`<tr><td>${node.nodeName}</td>`<br>`<td>${node.result}</td>`<br>`<td></td><td></td><td></td></tr>`<br>`#foreach ($port in $node.trackingPorts)`<br>`<tr><td></td><td></td>`<br>`<td>${port.portType}:${port.index}</td>`<br>`<td>${port.totalBytes} B</td>`<br>`<td>${port.totalRows} rows</td></tr>`<br>`#end`<br>`#end`<br>`#end`<br>`</table>`<br>`#end`<br>`}` |
| sandbox | Data structure describing sandbox containing executed graph. Contains sub-properties, which are accessible using dot notation (i.e. ${sandbox.name}) name, code, rootPath |
| schedule | Data structure describing schedule which triggered this task. Contains sub-properties, which are accessible using dot notation (i.e. ${schedule.description}) description, startTime, endTime, lastEvent, nextEvent, fireMisfired |

## Task - JMS Message

This type of task is useful for notifications about result of graph execution. I.e. you can create graph event listener with this task type to be notified about each failure in specified sandbox or failure of particular graph.

JMS messaging requires JMS API (jms.jar) and third-party libraries. All these libraries must be available on application server classpath. Some application servers contain these libraries by default, some do not, thus the libraries must be added explicitly.

*Table 7.3. Attributes of JMS message task*

| | |
|---|---|
| Task type | "JMS message" |
| Initial context class name | Full class name of javax.naming.InitialContext implementation. Each JMS provider has own implementation. i.e. for Apache MQ it is "org.apache.activemq.jndi.ActiveMQInitialContextFactory". If it is empty, server uses default initial context |
| Connection factory JNDI name | JNDI name of connection factory. Depends on JMS provider. |
| Destination JNDI name | JNDI name of message queue/topic on the server |
| Username | Username for connection to JMS message broker |
| Password | Password for connection to JMS message broker |
| URL | URL of JMS message broker |
| JMS pattern | This select box is available only when user is creating new record. It contains all predefined JMS message patterns. If user chooses any of them, text field below is automatically filled by value from pattern. |
| Text | Body of JMS message. It is also possible to use placeholders. *See* Placeholders *(p. 58) of send email task for details.* |

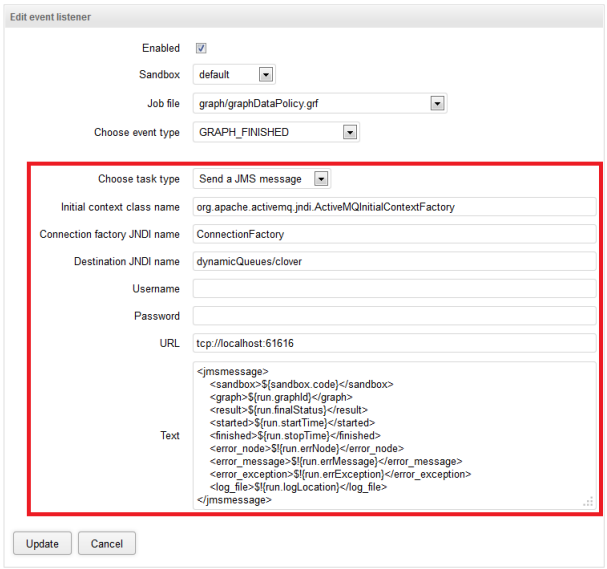



*Figure 7.3. Web GUI - Task JMS message editor*

# Use cases

Possible use cases are the following:

- Execute graphs in chain (p. 60)
- Email notification about graph failure (p. 61)
- Email notification about graph success (p. 62)
- Backup of data processed by graph (p. 62)

## Execute graphs in chain

Let's say, that we have to execute graph B, only if another graph A finished without any error. So there is some kind of relation between these graphs. We can achieve this behaviour by creating graph event listener. We create

listener for event `graph finished OK` of graph A and choose task type `execute graph` with graph B specified for execution. And that is it. If we create another listener for graph B with task `execute graph` with graph C specified, it will work as chain of graphs.

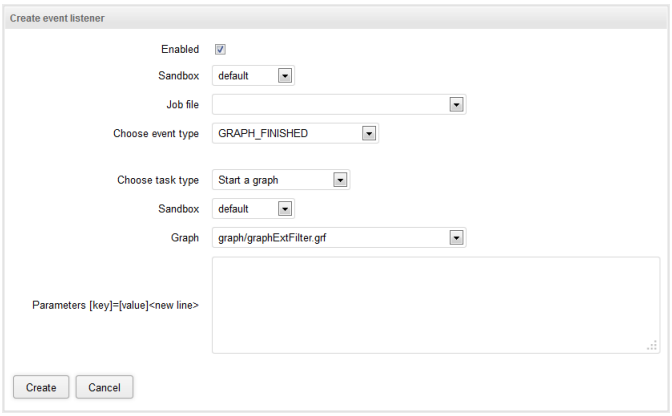



*Figure 7.4. Event source graph isn't specified, thus listener works for all graphs in specified sandbox*

## Email notification about graph failure

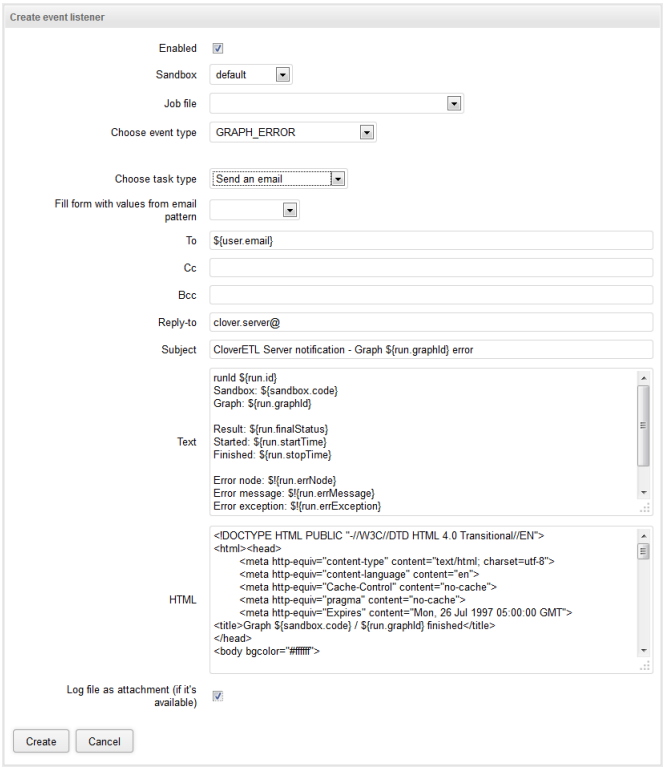



*Figure 7.5. Web GUI - email notification about graph failure*

## Email notification about graph success

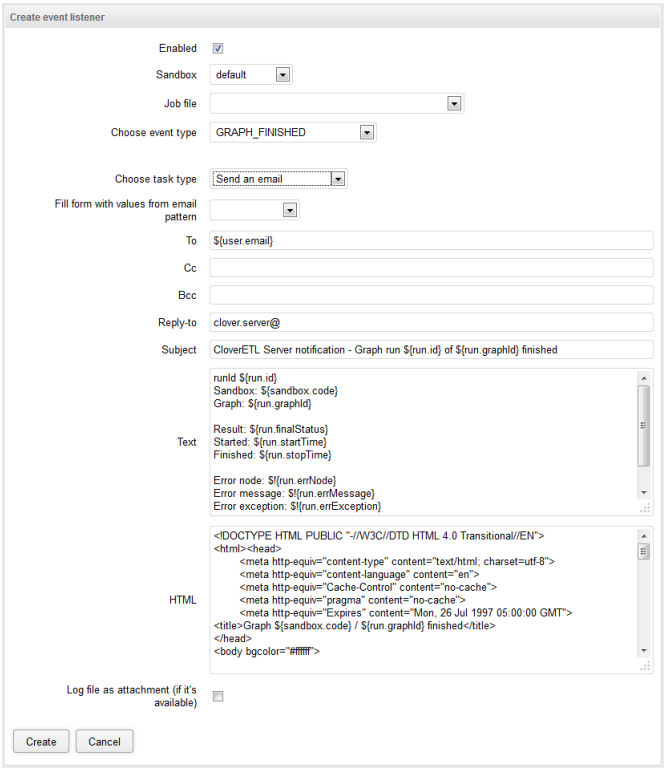



*Figure 7.6. Web GUI - email notification about graph success*

## Backup of data processed by graph

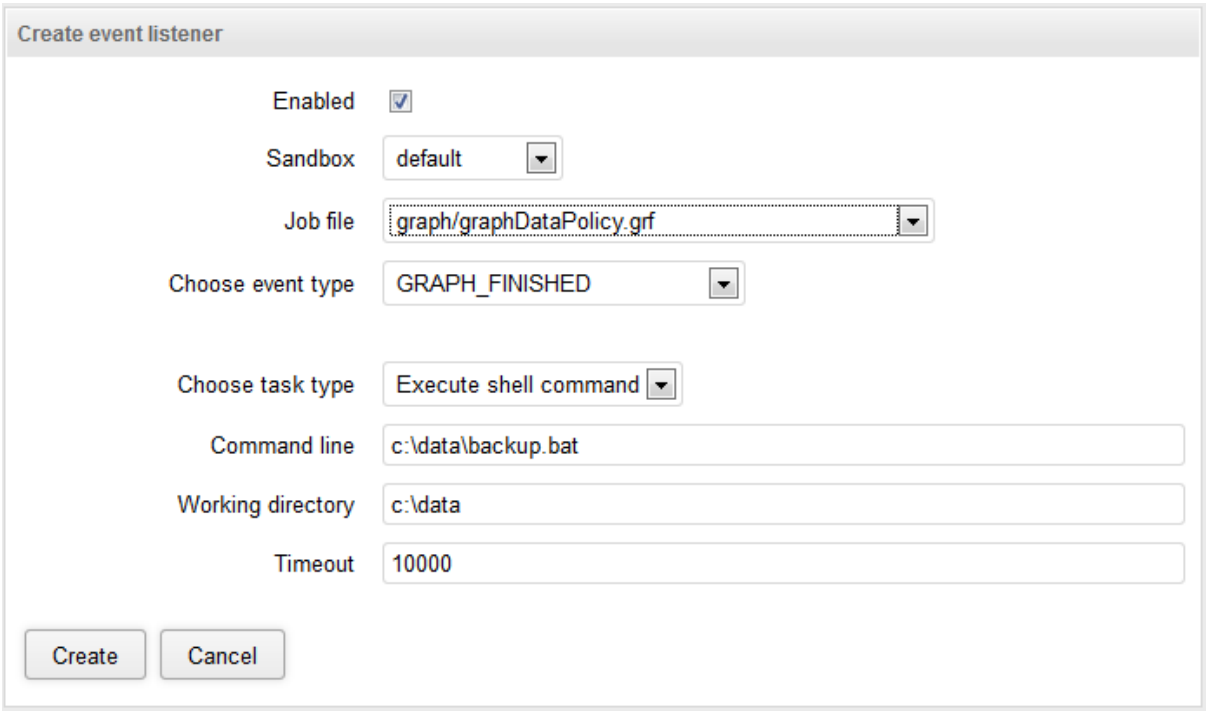



*Figure 7.7. Web GUI - backup of data processed by graph*

# Chapter 8. Jobflow Event Listeners

Jobflow Event Listeners allow you to define a task that the Server will execute as a reaction to the success or failure of executing a specific job (a jobflow).

Each listener is bound to a specific jobflow and is evaluated every time the jobflow is executed (no matter whether manually, through another jobflow, scheduled, via an API call, etc.).

Jobflow Event Listeners work very similarly to Graph Event Listeners (the section called “Tasks” (p. 56)) in many ways, since ETL Graphs and Jobflows are both "jobs" from the point of view of the CloverETL Server.

In the Cluster, the event and the associated task are executed on the same node the job was executed on. If the jobflow is distributed, the task will be executed on the master worker node. However, you can override where the task will be executed by explicitly specifying a Node ID in the task definition.

## Jobflow Events

Each event carries properties of the event source job. If there is a event listener specified, task may use these properties. e.g. next job in the chain may use "EVENT_FILE_NAME" placeholder which activated first job in the chain. Job properties, which are set specifically for each run (e.g. RUN_ID), are overridden by last job.

There are these types of jobflow events:



### jobflow started

Event of this type is created, when jobflow execution successfully started.

### jobflow phase finished

Event of this type is created, everytime when jobflow phase is finished and all its nodes are finished with status `FINISHED_OK`.

### jobflow finished OK

Event of this type is created, when all phases and nodes of jobflow are finished with status `FINISHED_OK`.

### jobflow error

Event of this type is created, when jobflow cannot be executed from any reason, or when any node of the jobflow fails.

### jobflow aborted

Event of this type is created, when jobflow is explicitly aborted.

### jobflow timeout

Event of this type is created, when jobflow runs longer then specified interval. Thus you have to specify "Job timeout interval" attribute for each listener of jobflow timeout event. You can specify this interval in seconds or in minutes or in hours.

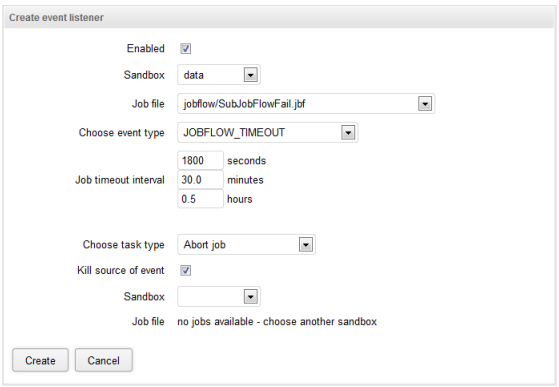



*Figure 8.1. Web GUI - jobflow timeout event*

### jobflow status unknown

Event of this type is created, when the server, during the startup, detects run records with undefined status in the executions history. Undefined status means, that server has been killed during jobflow run. Server automatically changes state of jobflow to "Not Available" and sends 'jobflow status unknown' event. Please note, that this works just for executions, which have persistent record in executions history. It is possible to execute transformation without persistent record in executions history, typically for better performance of fast running transformations (e.g. using Launch Services).

## Listener

User may create listener for specified event type and jobflow (or all jobflows in sandbox). Listener is actually connection between jobflow event and task, where jobflow event specifies WHEN and task specifies WHAT to do.

So progress is like this:

- event is created
- listeners for this event are notified
- each listener performs related task

## Tasks

Task specifies operation which should be performed as the reaction to the triggered event.

Task types are described in the section called “Tasks” (p. 48) and the section called “Tasks” (p. 56)

*Note: You can use task of any type for jobflow event listener. Description of task types is divided into two sections just to show the most obvious use cases.*

# Chapter 9. JMS messages listeners

JMS Message Listeners allow you to listen for incoming JMS messages. You specify the source of the messages (JMS Topic or JMS Queue) and a task that will be executed for each incoming message.

JMS messaging requires a JMS API (jms.jar) and specific third-party libraries. Every one of these libraries must be available on an application server classpath. Some application servers contain these libraries by default; however, some do not. In such a case, libraries must be added explicitly before starting the CloverETL Server.

JMS is a complex topic that goes beyond the scope of this document. For more detailed information about JMS, refer to the Oracle website: http://docs.oracle.com/javaee/6/tutorial/doc/bncdq.html

Note that the JMS implementation is dependent on the application server that the CloverETL Server is running in.

In Cluster, you can either explicitly specify which node will listen to JMS or not. If unspecified, all nodes will register as listeners. In the case of JMS Topic, all nodes will get the message and will trigger the task (multiple instances) or, in the case of JMS Queue, a random node will consume the message and will run the task (just one instance).

*Table 9.1. Attributes of JMS message task*

| Attribute | Description |
|---|---|
| Node ID to handle the event | This attribute makes sense only in cluster environment. It is node ID where the listener should be initialized. If it is not set, listener is initialized on all nodes in the cluster. |
| Initial context class name | Full class name of javax.naming.InitialContext implementation. Each JMS provider has own implementation. i.e. for Apache MQ it is "org.apache.activemq.jndi.ActiveMQInitialContextFactory". If it is empty, server uses default initial context. Specified class must be on web-app classpath or application-server classpath. It is usually included in one library with JMS API implementation for each specific JMS broker provider. |
| Connection factory JNDI name | JNDI name of connection factory. Depends on JMS provider. |
| Destination JNDI name | JNDI name of message queue/topic on the server |
| Username | Username for connection to JMS message broker |
| Password | Password for connection to JMS message broker |
| URL | URL of JMS message broker |
| Durable subscriber (only for Topics) | If it is false, message consumer is connected to the broker as "non-durable", so it receives only messages which are sent while the connection is active. Other messages are lost. If it is true, consumer is subscribed as "durable" so it receives even messages which are sent while the connection is inactive. The broker stores such messages until they can be delivered or until the expiration is reached. This switch makes sense only for Topics destinations, because Queue destinations always store messages until they can be delivered or the expiration is reached. Please note, that consumer is inactive i.e. during server restart and during short moment when user updates the "JMS message listener" ant it must be re-initialized. So during these intervals the message in the Topic may get lost if the consumer does not have durable subscription.<br><br>If the subscription is durable, client must have "ClientId" specified. This attribute can be set in different ways in dependence of JMS provider. I.e. for ActiveMQ, it is set as URL parameter tcp://localhost:1244?jms.clientID=TestClientID |
| Message selector | This "query string" can be used as specification of conditions for filtering incoming messages. Syntax is well described on Java EE API web site: http://java.sun.com/j2ee/1.4/docs/api/javax/jms/Message.html It has different behaviour depending on type of consumer (queue/topic) Queue: If a its a queue the messages that are filtered out remain on the queue. Topic: Messages filtered out by a Topic subscriber's message selector will never be delivered to the subscriber. From the subscriber's perspective, they do not exist. |
| Groovy code | Groovy code may be used for additional message processing and/or for refusing message. Both features are described below. |

## Optional Groovy code

Groovy code may be used for additional message processing or for refusing a message.

- **Additional message processing** Groovy code may modify/add/remove values stored in the containers "properties" and "data".

- **Refuse/acknowledge the message** If the Groovy code returns Boolean.FALSE, the message is refused. Otherwise, the message is acknowledged. A refused message may be redelivered, however the JMS broker should configure a limit for redelivering messages. If the groovy code throws an exception, it’s considered a

coding error and the JMS message is NOT refused because of it. So, if the message refusal is to be directed by some exception, it must be handled in groovy.

*Table 9.2. Variables accessible in groovy code*

| type | key | description |
|---|---|---|
| javax.jms.Message | msg | instance of JMS message |
| java.util.Properties | properties | See below for details. Contains values (String or converted to String) read from message and it is passed to the task which may use them somehow. I.e. task "execute graph" passes these parameters to the executed graph. |
| java.util.Map<String, Object> | data | See below for details. Contains values (Object, Stream, ..) read or proxied from the message instance and it is passed to the task which may use them somehow. I.e. task "execute graph" passes it to the executed graph as "dictionary entries". |
| javax.servlet.ServletContext | servletContext | instance of ServletContext |
| javax.jms.Message | msg | instance of JMS message |
| com.cloveretl.server.api.ServerFacade | serverFacade | instance of serverFacade usable for calling CloverETL Server core features. |
| String | sessionToken | sessionToken, needed for calling serverFacade methods |

## Message data available for further processing

A JMS message is processed and the data it contains is stored in two data structures: Properties and Data.

*Table 9.3. Properties Elements*

| key | description |
|---|---|
| JMS_PROP_[property key] | For each message property is created one entry, where "key" is made of prefix "JMS_PROP_" and property key. |
| JMS_MAP_[map entry key] | If the message is instance of MapMessage, for each map entry is created one entry, where "key" is made of prefix "JMS_MAP_" and map entry key. Values are converted to String. |
| JMS_TEXT | If the message is instanceof TextMessage, this property contains content of the message. |
| JMS_MSG_CLASS | Class name of message implementation |
| JMS_MSG_CORRELATIONID | Correlation ID is either provider-specific message ID or application-specific String value |
| JMS_MSG_DESTINATION | The JMSDestination header field contains the destination to which the message is being sent. |
| JMS_MSG_MESSAGEID | A JMSMessageID is a String value that should function as a unique key for identifying messages in a historical repository. The exact scope of uniqueness is provider-defined. It should at least cover all messages for a specific installation of a provider, where an installation is some connected set of message routers. |
| JMS_MSG_REPLYTO | Destination to which a reply to this message should be sent. |
| JMS_MSG_TYPE | Message type identifier supplied by the client when the message was sent. |
| JMS_MSG_DELIVERYMODE | The DeliveryMode value specified for this message. |
| JMS_MSG_EXPIRATION | The time the message expires, which is the sum of the time-to-live value specified by the client and the GMT at the time of the send. |
| JMS_MSG_PRIORITY | The JMS API defines ten levels of priority value, with 0 as the lowest priority and 9 as the highest. In addition, clients should consider priorities 0-4 as gradations of normal priority and priorities 5-9 as gradations of expedited priority. |
| JMS_MSG_REDELIVERED | true" if this message is being redelivered. |
| JMS_MSG_TIMESTAMP | The time a message was handed off to a provider to be sent. It is not the time the message was actually transmitted, because the actual send may occur later due to transactions or other client-side queueing of messages. |

Note that all values in the “Properties” structure are stored as String type – however they are numbers or text.

For backwards compatibility, all listed properties can also be accessed using lower-case keys; it is, however, a deprecated approach.

*Table 9.4. "data" elements*

| key | description |
|---|---|
| JMS_MSG | instance of javax.jms.Message |
| JMS_DATA_STREAM | Instance of java.io.InputStream. Accessible only for TextMessage, BytesMessage, StreamMessage, ObjectMessage(only if payload object is instance of String). Strings are encoded in UTF-8. |
| JMS_DATA_TEXT | Instance of String. Only for TextMessage and ObjectMessage, where payload object is instance of String. |
| JMS_DATA_OBJECT | Instance of java.lang.Object - message payload. Only for ObjectMessage. |

The “Data” container is passed to a task that can use it, depending on its implementation. For example, the task "execute graph" passes it to the executed graph as “dictionary entries.”

In the Cluster, you can specify an explicit node that will execute the task. However, if the “data” payload is not serializable and the receiving and executing node differ, an error will be thrown as the Cluster cannot pass the “data” to the executing node.

Inside a graph or a jobflow, data passed as dictionary entries can be used in some component attributes. For example, a File URL attribute would look like: "dict:JMS_DATA_STREAM:discrete" for reading the data directly from the incoming JMS message using a proxy stream.

For backwards compatibility, all listed dictionary entries can also be accessed using lower-case keys; it is, however, a deprecated approach.

# Chapter 10. Universal event listeners

Since 2.10

Universal Event Listeners allow you to write a piece of Groovy code that controls when an event is triggered, subsequently executing a predefined task. The Groovy code is periodically executed and when it returns TRUE, the task is executed.

*Table 10.1. Attributes of Universal message task*

| Attribute | Description |
|---|---|
| Node ID to handle the event | (only in the Cluster). Specifies the node ID where the listener should be initialized. If not set, the listener is initialized on all nodes in the Cluster. |
| Interval of check in seconds | Periodicity of Groovy code execution. |
| Groovy code | Groovy code that evaluates either to TRUE (execute the task) or FALSE (no action). See below for more details. |

## Groovy code

A piece of Groovy is repeatedly executed and evaluated; based on the result, the event is either triggered and the task executed or no action is taken.

For example, you can continually check for essential data sources before starting a graph. Or, you can do complex checks of a running graph and, for example, decide to kill it if necessary. You can even call the CloverETL Server core functions using the ServerFacade interface, see Javadoc: http://host:port/clover/javadoc/index.html

## Evaluation Critieria

If the Groovy code returns Boolean.TRUE, the event is triggered and the associated task is executed. Otherwise, nothing happens.

If the Groovy code throws an exception, it’s considered a coding error and the event is NOT triggered. Thus, exceptions should be properly handled in the Groovy code.

*Table 10.2. Variables accessible in groovy code*

| type | key | description |
|---|---|---|
| java.util.Properties | properties | Empty container which may be filled by String-String key-value pairs in your Groovy code. It is passed to the task which may use them somehow. I.e. task "execute graph" passes these parameters to the executed graph. |
| java.util.Map<String, Object> | data | Empty container which may be filled by String-Object key-value pairs in your Groovy code. It is passed to the task which may use them somehow according to its implementation. I.e. task "execute graph" passes it to the executed graph as "dictionary entries". Please note that it is not serializable, thus if the task is relying on it, it can be processed properly only on the same cluster node. |
| javax.servlet.ServletContext | servletContext | instance of ServletContext |
| com.cloveretl.server.api.ServerFacade | serverFacade | instance of serverFacade usable for calling CloverETL Server core features. |
| String | sessionToken | sessionToken, needed for calling serverFacade methods |

# Chapter 11. Manual task execution

Since 3.1

Manual task execution allows you to invoke a task directly with an immediate effect, without defining and triggering an event.

There are a number of task types that are usually associated with a triggering event, such as a file listener or a graph/jobflow listener. You can execute any of these tasks manually.

Additionally, you can specify task parameters to simulate a source event that would normally trigger the task. The following is a figure displaying how a “file event” could be simulated. The parameters for various event sources are listed in the section "Graph parameters".

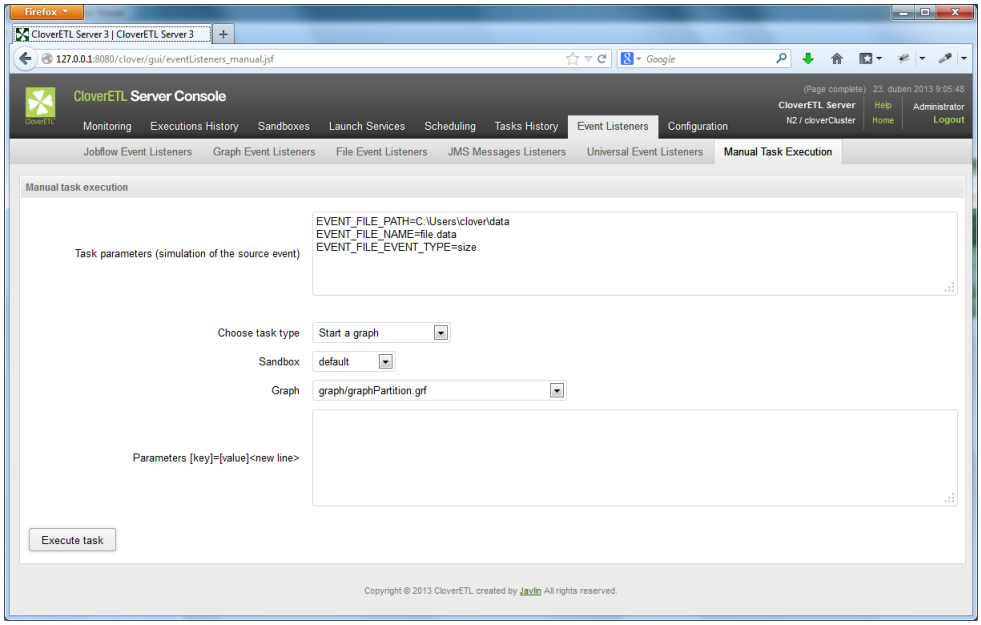



*Figure 11.1. Web GUI - "Manual task execution" section*

# Chapter 12. File event listeners

Since 1.3

File Event Listeners allow you to monitor changes on a specific file system path – for example, new files appearing in a folder – and react to such an event with a predefined task.

You can either specify an exact path or use a wildcard, then set a checking interval in seconds, and finally, define a task to process the event.

There is a global minimal check interval that you can change if necessary in the configuration ("clover.event.fileCheckMinInterval" property).

In the Cluster, each event listener has a "Node ID" attribute that specifies which Cluster node will perform the checks on its local file system. In standalone environment, the "Node ID" attribute is ignored.

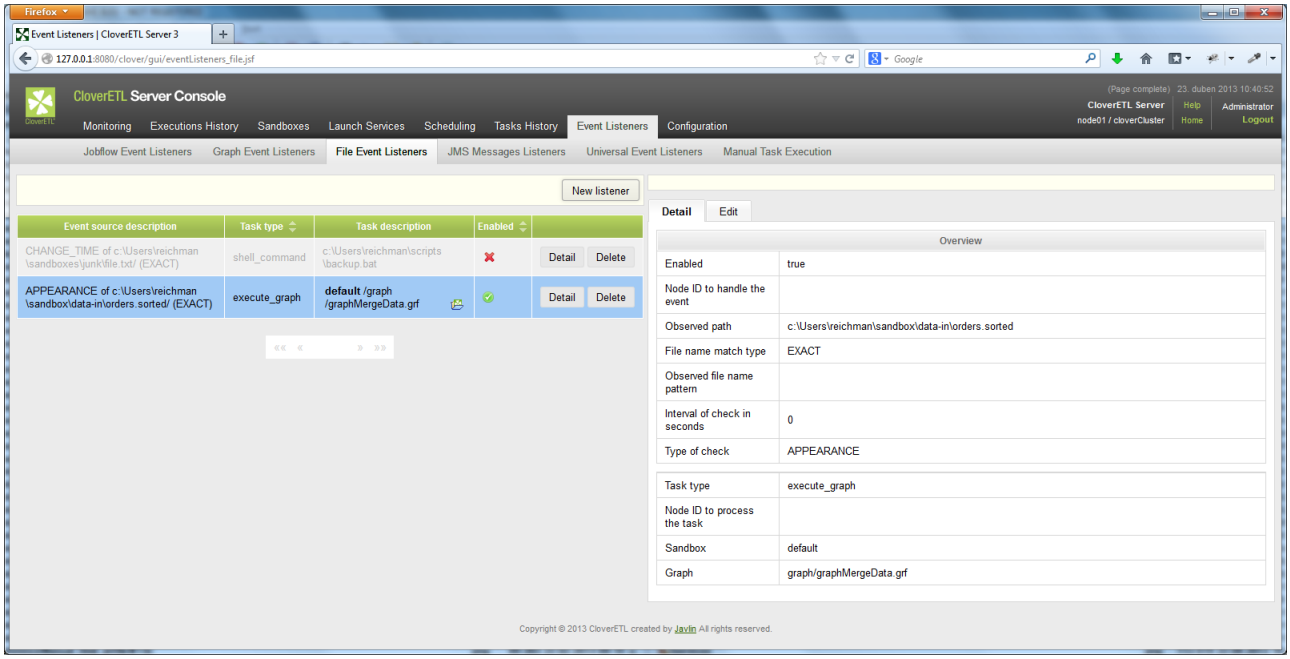



*Figure 12.1. Web GUI - "File event listeners" section*

## Observed file

Observed file is specified by directory path and file name pattern.

User may specify just one exact file name or file name pattern for observing more matching files in specified directory. If there are more changed files matching the pattern, separated event is triggered for each of these files.

There are three ways how to specify file name pattern of observed file(s)

- Exact match (p. 73)
- Wildcards (p. 74)
- Regullar expression (p. 74)

### Exact match

You specify exact name of the observed file.

### Wildcards

You can use wildcards common in most operating systems (*, ?, etc.)

- * - Matches zero or more instances of any character
- ? - Matches one instance of any character
- [...] - Matches any of characters enclosed by the brackets
- \ - Escape character

Examples

- *.csv - Matches all CSV files
- input_*.csv - Matches i.e. input_001.csv, input_9.csv
- input_???.csv - Matches i.e. input_001.csv, but does not match input_9.csv

### Regullar expression

Examples

- (.*?)\.(jpg|jpeg|png|gif)$ - Matches image files

### Notes

- It is strongly recommended to use absolute paths. It is possible to use relative path, but working directory depends on application server.
- Use forward slashes as file separators, even on MS Windows OS. Backslashes might be evaluated as escape sequences.

## File Events

For each listener you have to specify event type, which you are interested in.

There are four types of file events:

- file APPEARANCE (p. 74)
- file DISAPPEARANCE (p. 75)
- file SIZE (p. 75)
- file CHANGE_TIME (p. 75)

### file APPEARANCE

Event of this type occurs, when the observed file is created or copied from another location between two checks. Please keep in mind, that event of this type occurs immediately when new file is detected, regardless it is complete or not. Thus task which may need complete file is executed when file is still incomplete. Recommended approach is to save file to the different location and when it is complete, move/rename to observed location where CloverETL Server may detect it. File moving/renaming should be atomic operation.

Event of this type does not occur when the file has been updated (change of timestamp or size) between two checks. Appearance means that the file didn't exist during previous check and it exists now, during current check.

### file DISAPPEARANCE

Event of this type occurs, when observed file is deleted or moved to another location between two checks.

### file SIZE

Event of this type occurs when the size of the observed file has changed between two checks. Event of this type is never produced when file is created or removed. File must exist during both checks.

### file CHANGE_TIME

Event of this type occurs, when change time of observed file has changed between two checks. Event of this type is never produced when file is created or removed. File must exist during both checks.

## Check interval, Task and Use cases

- User may specify minimal time interval between two checks. It is specified in seconds.
- Each listener defines task, which will be processed as the reaction for file event. All task types and theirs attributes are described in section Scheduling and GraphEventListeners
- • Graph Execution, when file with input data is accessible
  - Graph Execution, when file with input data is updated
  - Graph Execution, when file with generated data is removed and must be recreated

### How to use source of event during task processing

File, which caused event (considered as source of event) may be used during task processing. i.e. reader/writer components in graph transformations may refer to this file by special placeholders: `${EVENT_FILE_PATH}` - path to directory which contains event source `${EVENT_FILE_NAME}` - name of event source.

Please note that previous versions used lower-case placeholders. Since version 3.3, placeholders are upper-case, however lower-case still work for backward compatibility.

i.e. if event source is: `/home/clover/data/customers.csv`, placeholders will contain: EVENT_FILE_PATH - `/home/clover/data`, EVENT_FILE_NAME - `customers.csv`

For "graph execution" task this works only if the graph is not pooled. Thus "keep in pool interval" must be set to 0 (default value).

# Chapter 13. WebDAV

Since 3.1

WebDAV API allows you to access and manage sandbox content using a standard WebDAV specification.

Specifically, it allows for:

- Browsing a directory structure
- Editing files
- Removing files/folders
- Renaming files/folders
- Creating files/folders
- Copying files
- Moving files

The WebDAV interface is accessible from the URL: "http://[host]:[port]/clover/webdav".

Note: Although common browsers will open this URL, most of them are not rich WebDAV clients. Thus, you will only see a list of items, but you cannot browse the directory structure.

## WebDAV clients

There are many WebDAV clients for various operating systems, some OS support WebDAV natively.

### Linux like OS

Great WebDAV client working on linux systems is Konqueror. Please use different protocol in the URL: webdav://[host]:[port]/clover/webdav

### MS windows

Last distributions of MS Windows (Win XP and later) have native support for WebDAV. Unfortunatelly, it is more or less unreliable, so it is recommended to use some free or commercial WebDAV client.

- The best WebDAV client we've tested is BitKinex: http://www.bitkinex.com/webdavclient
- Another option is to use Total Commander (http://www.ghisler.com/index.htm) with WebDAV plugin: http://www.ghisler.com/plugins.htm#filesys

### Mac OS

Mac OS supports WebDAV natively and in this case it should be without any problems. You can use "finder" application, select "Connect to the server ..." menu item and use URL with HTTP protocol: "http://[host]:[port]/clover/webdav".

## WebDAV authentication/authorization

CloverETL Server WebDAV API uses the HTTP Basic Authentication by default. However it may be reconfigured to use HTTP Digest Authentication. Please see the Configuration section for details.

Digest Authentication may be useful, since some WebDAV clients can't work with HTTP Basic Authentication, only with Digest Authentication.

HTTP Digest Authentication is feature added to the version 3.1. If you upgraded your older CloverETL Server distribution, users created before the upgrade cannot use the HTTP Digest Authentication until they reset their passwords. So when they reset their passwords (or the admin does it for them), they can use Digest Authentication as well as new users.

# Chapter 14. Simple HTTP API

The Simple HTTP API is a basic Server automation tool that lets you control the Server from external applications using simple HTTP calls.

All operations are accessible using the HTTP GET method and return plain text. Thus, both “request” and “response” can be conveniently sent and parsed using very simple tools (wget, grep, etc.).

If global security is “on” (on by default), Basic HTTP authentication is used. Authenticated operations will require valid user credentials with corresponding permissions.

Note that the ETL graph-related operations "graph_run", "graph_status" and "graph_kill" also work for jobflows and Data Profiler jobs.

The generic pattern for a request URL:

```
http://[domain]:[port]/[context]/[servlet]/[operation]?[param1]=[value1]&[param2]=[value2]...
```

For a wget client, you can use following command line:

```
wget --user=$USER --password=$PASS -O ./$OUTPUT_FILE $REQUEST_URL
```



## Operation help

**parameters**

no

**returns**

list of possible operations and parameters with its descriptions

**example**

```
http://localhost:8080/clover/request_processor/help
```

# Operation graph_run

Call this operation to start execution of the specified job. Operation is called graph_run for backward compatibility, however it may execute ETL graph, jobflow or profiler job.

**parameters**

*Table 14.1. Parameters of graph_run*

| parameter name | mandatory | default | description |
|---|---|---|---|
| graphID | yes | - | File path to the job file, relative to the sandbox root. |
| sandbox | yes | - | Text ID of sandbox |
| additional job parameters | no | | Any URL parameter with "param_" prefix is passed to executed job and may be used in transformation XML as placeholder, but without "param_" prefix. e.g. "param_FILE_NAME" specified in URL may be used in the XML as ${FILE_NAME}. These parameters are resolved only during loading of XML, so it cannot be pooled. |
| nodeID | no | - | In cluster mode it's ID of node which should execute the job. However it's not final. If the graph is distributed, or the node is disconnected, the graph may be executed on some another node. |
| verbose | no | MESSAGE | MESSAGE \| FULL - how verbose should possible error message be. |

**returns**

run ID: incremental number, which identifies each execution request

**example**

```
http://localhost:8080/clover/request_processor/graph_run?graphID=graph/graphDBExecute.grf&sandbox=mva
```

# Operation graph_status

Call this operation to obtain status of specified job execution. Operation is called graph_status for backward compatibility, however it may return status of ETL graph or jobflow.

**parameters**

*Table 14.2. Parameters of graph_status*

| parameter name | mandatory | default | description |
|---|---|---|---|
| runID | yes | - | Id of each graph execution |
| returnType | no | STATUS | STATUS \| STATUS_TEXT \| DESCRIPTION \| DESCRIPTION_XML |
| waitForStatus | no | - | Status code which we want to wait for. If it is specified, this operation will wait until graph is in required status. |
| waitTimeout | no | 0 | If waitForStatus is specified, it will wait only specified amount of milliseconds. Default 0 means forever, but it depends on application server configuration. When the specified timeout expires and graph run still isn't in required status, server returns code 408 (Request Timeout). 408 code may be also returned by application server if its HTTP request timeout expires before. |
| verbose | no | MESSAGE | MESSAGE \| FULL - how verbose should possible error message be. |

**returns**

Status of specified graph. It may be number code, text code or complex description in dependence of optional parameter returnType. Description is returned as plain text with pipe as separator, or as XML. Schema describing XML format of the XML response is accessible on CloverETL Server URL: http://[host]:[port]/clover/schemas/executions.xsd In dependence on waitForStatus parameter may return result immediately or wait for specified status.

**example**

```
http://localhost:8080/clover/request_processor/graph_status ->
            -> ?runID=123456&returnType=DESCRIPTION&waitForStatus=FINISHED&waitTimeout=60000
```

# Operation graph_kill

Call this operation to abort/kill job execution. Operation is called graph_kill for backward compatibility, however it may abort/kill ETL graph, jobflow or profiler job.

**parameters**

*Table 14.3. Parameters of graph_kill*

| parameter name | mandatory | default | description |
|---|---|---|---|
| runID | yes | - | Id of each graph execution |
| returnType | no | STATUS | STATUS \| STATUS_TEXT \| DESCRIPTION |
| verbose | no | MESSAGE | MESSAGE \| FULL - how verbose should possible error message be. |

**returns**

Status of specified graph after attempt to kill it. It may be number code, text code or complex description in dependence of optional parameter.

**example**

```
http://localhost:8080/clover/request_processor/graph_kill?runID=123456&returnType=DESCRIPTION
```

## Operation server_jobs

**parameters**

no

**returns**

List of runIDs currently running jobs.

**example**

```
http://localhost:8080/clover/request_processor/server_jobs
```

## Operation sandbox_list

**parameters**

no

**returns**

List of all sandbox text IDs. In next versions will return only accessible ones.

**example**

```
http://localhost:8080/clover/request_processor/sandbox_list
```

## Operation sandbox_content

**parameters**

*Table 14.4. Parameters of sandbox_content*

| parameter name | mandatory | default | description |
|---|---|---|---|
| sandbox | yes | - | text ID of sandbox |
| verbose | no | MESSAGE | MESSAGE \| FULL - how verbose should possible error message be. |

**returns**

List of all elements in specified sandbox. Each element may be specified as file path relative to sandbox root.

**example**

```
http://localhost:8080/clover/request_processor/sandbox_content?sandbox=mva
```

## Operation executions_history

**parameters**

*Table 14.5. Parameters of executions_history*

| parameter name | mandatory | default | description |
|---|---|---|---|
| sandbox | yes | - | text ID of sandbox |
| from | no | | Lower datetime limit of start of execution. Operation will return only records after(and equal) this datetime. Format: "yyyy-MM-dd HH:mm" (must be URL encoded). |
| to | no | | Upper datetime limit of start of execution. Operation will return only records before(and equal) this datetime. Format: "yyyy-MM-dd HH:mm" (must be URL encoded). |
| stopFrom | no | | Lower datetime limit of stop of execution. Operation will return only records after(and equal) this datetime. Format: "yyyy-MM-dd HH:mm" (must be URL encoded). |
| stopTo | no | | Upper datetime limit of stop of execution. Operation will return only records before(and equal) this datetime. Format: "yyyy-MM-dd HH:mm" (must be URL encoded). |
| status | no | | Current execution status. Operation will return only records with specified STATUS. Meaningful values are RUNNING \| ABORTED \| FINISHED_OK \| ERROR |
| sandbox | no | | Sandbox code. Operation will return only records for graphs from specified sandbox. |
| graphId | no | | Text Id, which is unique in specified sandbox. File path relative to sandbox root |
| orderBy | no | | Attribute for list ordering. Possible values: id \| graphId \| finalStatus \| startTime \| stopTime. There is no ordering by default. |
| orderDescend | no | true | Switch which specifies ascending or descending ordering. If it is true (which is default), ordering is descending. |
| returnType | no | IDs | Possible values are: IDs \| DESCRIPTION \| DESCRIPTION_XML |
| index | no | 0 | Index of the first returned records in whole record set. (starting from |
| records | no | infinite | Max amount of returned records. |
| verbose | no | MESSAGE | MESSAGE \| FULL - how verbose should possible error message be. |

**returns**

List of executions according to filter criteria.

For `returnType==IDs` returns simple list of runIDs (with new line delimiter).

For `returnType==DESCRIPTION` returns complex response which describes current status of selected executions, their phases, nodes and ports.

```
execution|[runID]|[status]|[username]|[sandbox]|[graphID]|[startedDatetime]|[finishedDatetime]|[clusterNode]|[grap
phase|[index]|[execTimeInMilis]
node|[nodeID]|[status]|[totalCpuTime]|[totalUserTime]|[cpuUsage]|[peakCpuUsage]|[userUsage]|[peakUserUsage]
port|[portType]|[index]|[avgBytes]|[avgRows]|[peakBytes]|[peakRows]|[totalBytes]|[totalRows]
```

**example of request**

```
http://localhost:8080/clover/request_processor/executions_history ->
```

```
              -> ?from=&to=2008-09-16+16%3A40&status=&sandbox=def&graphID=&index=&records=&returnType=DESCRIPTION
```

**example of DESCRIPTION (plain text) response**

```
execution|13108|FINISHED_OK|clover|def|test.grf|2008-09-16 11:11:19|2008-09-16 11:11:58|nodeA|2.4
phase|0|38733
node|DATA_GENERATOR1|FINISHED_OK|0|0|0.0|0.0|0.0|0.0
port|Output|0|0|0|0|0|130|10
node|TRASH0|FINISHED_OK|0|0|0.0|0.0|0.0|0.0
port|Input|0|0|0|5|0|130|10
node|SPEED_LIMITER0|FINISHED_OK|0|0|0.0|0.0|0.0|0.0
port|Input|0|0|0|0|0|130|10
port|Output|0|0|0|5|0|130|10
execution|13107|ABORTED|clover|def|test.grf|2008-09-16 11:11:19|2008-09-16 11:11:30
phase|0|11133
node|DATA_GENERATOR1|FINISHED_OK|0|0|0.0|0.0|0.0|0.0
port|Output|0|0|0|0|0|130|10
node|TRASH0|RUNNING|0|0|0.0|0.0|0.0|0.0
port|Input|0|5|0|5|0|52|4
node|SPEED_LIMITER0|RUNNING|0|0|0.0|0.0|0.0|0.0
port|Input|0|0|0|0|0|130|10
port|Output|0|5|0|5|0|52|4
```

For `returnType==DESCRIPTION_XML` returns complex data structure describing one or more selected executions in XML format. Schema describing XML format of the XML response is accessible on CloverETL Server URL: http://[host]:[port]/clover/schemas/executions.xsd

# Operation suspend

Suspends server or sandbox(if specified). Suspension means, that no graphs may me executed on suspended server/sandbox.

**parameters**

*Table 14.6. Parameters of suspend*

| parameter name | mandatory | default | description |
|---|---|---|---|
| sandbox | no | - | Text ID of sandbox to suspend. If not specified, it suspends whole server. |
| atonce | no | | If this param is set to true, running graphs from suspended server(or just from sandbox) are aborted. Otherwise it can run until it is finished in common way. |

**returns**

Result message

# Operation resume

**parameters**

*Table 14.7. Parameters of resume*

| parameter name | mandatory | default | description |
|---|---|---|---|
| sandbox | no | - | Text Id of sandbox to resume. If not specified, server will be resumed. |
| verbose | no | MESSAGE | MESSAGE \| FULL - how verbose should possible error message be. |

**returns**

Result message

## Operation sandbox_create

This operation creates specified sandbox. If it's sandbox of "partitioned" or "local" type, create also locations by "sandbox_add_location" operation.

**parameters**

*Table 14.8. Parameters of sandbox create*

| parameter name | mandatory | default | description |
|---|---|---|---|
| sandbox | yes | - | Text Id of sandbox to be created. |
| path | no | - | Path to the sandbox root if server is running in standalone mode. |
| type | no | shared | Sandbox type: shared \| partitioned \| local. For standalone server may be left empty, since the default "shared" is used. |
| createDirs | no | true | Switch whether to create directory structure of the sandbox (only for standalone server or "shared" sandboxes in cluster environment). |
| verbose | no | MESSAGE | MESSAGE \| FULL - how verbose should possible error message be. |

**returns**

Result message

## Operation sandbox_add_location

This operation adds location to specified sandbox. Only useable for sandboxes of type partitioned or local.

**parameters**

*Table 14.9. Parameters of sandbox add location*

| parameter name | mandatory | default | description |
|---|---|---|---|
| sandbox | yes | - | Sandbox which we want to add location to. |
| nodeId | yes | - | Location attribute - node which has direct access to the location. |
| path | yes | - | Location attribute - path to the location root on the specified node. |
| location | no | - | Location attribute - location storage ID. If it's not specified, new one will be generated. |
| verbose | no | MESSAGE | MESSAGE \| FULL - how verbose should possible error message be. |

**returns**

Result message

## Operation sandbox_remove_location

This operation removes location from specified sandbox. Only sandboxes of type partitioned or local can have locations asociated.

**parameters**

*Table 14.10. Parameters of sandbox add location*

| parameter name | mandatory | default | description |
|---|---|---|---|
| sandbox | yes | - | Removes specified location from its sandbox. |
| location | yes | - | Location storage ID. If the specified location isn't attached to the specified sandbox, sandbox won't be changed. |
| verbose | no | MESSAGE | MESSAGE \| FULL - how verbose should possible error message be. |

**returns**

Result message

## Cluster status

This operation displays cluster's nodes list.

**parameters**

no

**returns**

Cluster's nodes list.

# Chapter 15. JMX mBean

The CloverETL Server JMX mBean is an API that you can use for monitoring the internal status of the Server.

MBean is registered with the name:

```
com.cloveretl.server.api.jmx:name=cloverServerJmxMBean
```

.

## JMX configuration

Application's JMX MBeans aren't accessible outside of JVM by default. It needs some changes in application server configuration to make them accessible.

This section describes how to configure JMX Connector for development and testing. Thus authentication may be disabled. For production deployment authentication should be enabled. Please refer further documentation to see how to achieve this. i.e. http://java.sun.com/j2se/1.5.0/docs/guide/management/agent.html#auth

Configurations and possible problems:

- How to configure JMX on Apache Tomcat (p. 86)
- How to configure JMX on Glassfish (p. 87)
- Websphere 7 (p. 88)
- Possible problems (p. 89)

### How to configure JMX on Apache Tomcat

Tomcat's JVM must be executed with these self-explanatory parameters:

1. `-Dcom.sun.management.jmxremote=true`
2. `-Dcom.sun.management.jmxremote.port=8686`
3. `-Dcom.sun.management.jmxremote.ssl=false`
4. `-Dcom.sun.management.jmxremote.authenticate=false`
5. `-Djava.rmi.server.hostname=your.server.domain (necessary only for remote JMX connections)`

On UNIX like OS set environment variable CATALINA_OPTS i.e. like this:

```
export CATALINA_OPTS="-Dcom.sun.management.jmxremote=true
                      -Dcom.sun.management.jmxremote.port=8686
                      -Dcom.sun.management.jmxremote.ssl=false
                      -Dcom.sun.management.jmxremote.authenticate=false
                      -Djava.rmi.server.hostname=your.server.domain.com"
```

File TOMCAT_HOME/bin/setenv.sh (if it does not exist, you may create it) or TOMCAT_HOME/bin/catalina.sh

On Windows it might be tricky, that each parameter must be set separately:

```
set CATALINA_OPTS=-Dcom.sun.management.jmxremote=true
```

```
set CATALINA_OPTS=%CATALINA_OPTS% -Dcom.sun.management.jmxremote.port=8686
set CATALINA_OPTS=%CATALINA_OPTS% -Dcom.sun.management.jmxremote.authenticate=false
set CATALINA_OPTS=%CATALINA_OPTS% -Dcom.sun.management.jmxremote.ssl=false
set CATALINA_OPTS=%CATALINA_OPTS% -Djava.rmi.server.hostname=your.server.domain
```

File TOMCAT_HOME/bin/setenv.bat (if it does not exist, you may create it) or TOMCAT_HOME/bin/catalina.bat

With these values, you can use URL

```
service:jmx:rmi:///jndi/rmi://localhost:8686/jmxrmi
```

for connection to JMX server of JVM. No user/password is needed

## How to configure JMX on Glassfish

Go to Glasfish admin console (by default accessible on http://localhost:4848 with admin/adminadmin as user/password)

Go to section "Configuration" > "Admin Service" > "system" and set attributes like this:

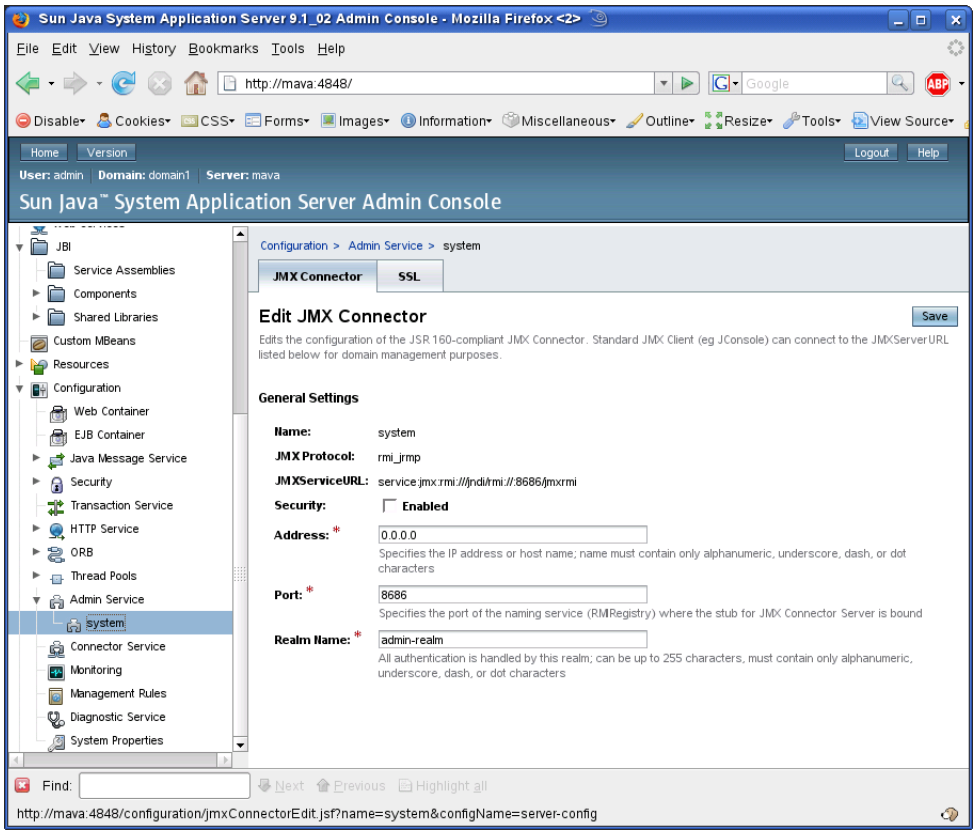



*Figure 15.1. Glassfish JMX connector*

With these values, you can use URL

```
service:jmx:rmi:///jndi/rmi://localhost:8686/jmxrmi
```

for connection to JMX server of JVM.

Use admin/adminadmin as user/password. (admin/adminadmin are default glassfish values)

## How to configure JMX on Websphere

Websphere does not require any special configuration, but the clover MBean is registered with the name, that depends on application server configuration:

```
com.cloveretl.server.api.jmx:cell=[cellName],name=cloverServerJmxMBean,node=[nodeName],
            process=[instanceName]
```

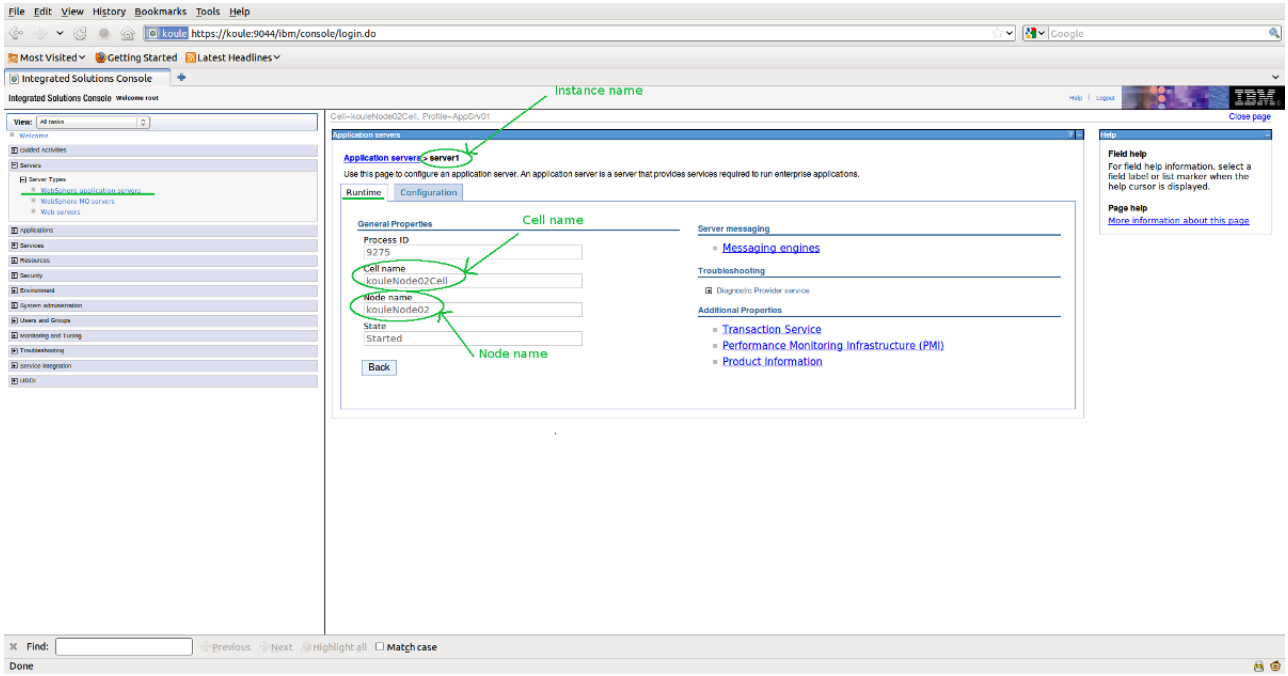



*Figure 15.2. Websphere configuration*

### Websphere 7

URL for connecting to JMX server is:

```
service:jmx:iiop://[host]:[port]/jndi/JMXConnector
```

where *host* is the host name you are connectiong to and *port* is RMI port number. If you have a default Websphere installation, the JNDI port number will likely be 2809, 2810, ... depending on how many servers there are installed on one system and the specific one you want to connect to. To be sure, when starting Websphere, check the logs, as it will dump a line like

```
0000000a RMIConnectorC A   ADMC0026I: The RMI Connector is available at port 2810
```

## How to configure JMX on Websphere7

Websphere does not require any special configuration, but the clover MBean is registered with the name, that depends on application server configuration:

```
com.cloveretl.server.api.jmx:cell=[cellName],name=cloverServerJmxMBean,node=[nodeName],
            process=[instanceName]
```

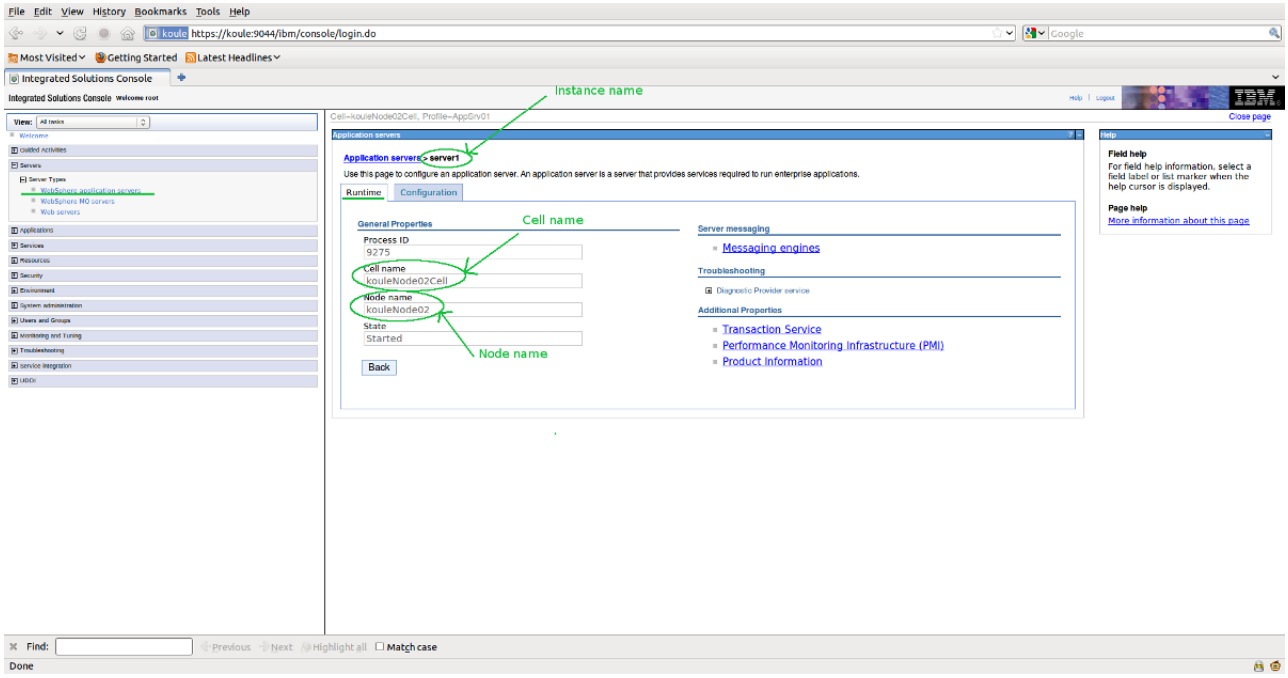



*Figure 15.3. Websphere7 configuration*

URL for connecting to JMX server is:

```
service:jmx:iiop://[host]:[port]/jndi/JMXConnector
```

where *host* is the host name you are connectiong to and *port* is RMI port number. If you have a default Websphere installation, the JNDI port number will likely be 2809, 2810, ... depending on how many servers there are installed on one system and the specific one you want to connect to. To be sure, when starting Websphere, check the logs, as it will dump a line like

```
0000000a RMIConnectorC A   ADMC0026I: The RMI Connector is available at port 2810
```

You will also need to set on the classpath following jar files from Websphere home directory:

/runtimes/com.ibm.ws.admin.client_7.0.0.jar
/runtimes/com.ibm.ws.ejb.thinclient_7.0.0.jar
/runtimes/com.ibm.ws.orb_7.0.0.jar

### Possible problems

- Default JMX mBean server uses RMI as a transport protocol. Sometimes RMI cannot connect remotely when one of peers uses Java version 1.6. Solution is quite easy, just set these two system properties: `-Djava.rmi.server.hostname=[hostname or IP address] -Djava.net.preferIPv4Stack=true`

## Operations

For details about operations please see the JavaDoc of the MBean interface:

JMX API MBean JavaDoc is accessible in the running CloverETL Server instance on URL: http://[host]:[port]/[contextPath]/javadoc-jmx/index.html

# Chapter 16. SOAP WebService API

The CloverETL Server SOAP Web Service is an advanced API that provides an automation alternative to the Simple HTTP API. While most of the HTTP API operations are available in the SOAP interface too (though not all of them), the SOAP API provides additional operations for manipulating sandboxes, monitoring, etc.

The SOAP API service is accessible on the following URL:

```
http://[host]:[port]/clover/webservice
```

The SOAP API service descriptor is accessible on URL:

```
http://[host]:[port]/clover/webservice?wsdl
```

Protocol HTTP can be changed to secured HTTPS based on the web server configuration.

## SOAP WS Client

Exposed service is implemented with the most common binding style "document/literal", which is widely supported by libraries in various programming languages.

To create client for this API, only WSDL document (see the URL above) is needed together with some development tools according to your programming language and development environments.

JavaDoc of WebService interface with all related classes is accessible in the running CloverETL Server instance on URL http://[host]:[port]/[contextPath]/javadoc-ws/index.html

If the web server has HTTPS connector configured, also the client must meet the security requirements according to web server configuration. i.e. client trust + key stores configured properly

## SOAP WS API authentication/authorization

Since exposed service is stateless, authentication "sessionToken" has to be passed as parameter to each operation. Client can obtain authentication sessionToken by calling "login" operation.

# Chapter 17. Launch Services

Launch Services allow you to publish a transformation graph or a jobflow as a Web Service. With Launch Services, CloverETL transformations can be exposed to provide request-response based data interface (e.g. searches, complicated lookups, etc.) for other application or directly to users.

## Launch Services Overview

The architecture of a Launch Service is relatively simple, following the basic design of multi-tiered applications utilizing a web browser.

For example, you can build a user-friendly form that the user fills in and sends to the CloverETL Server for processing.

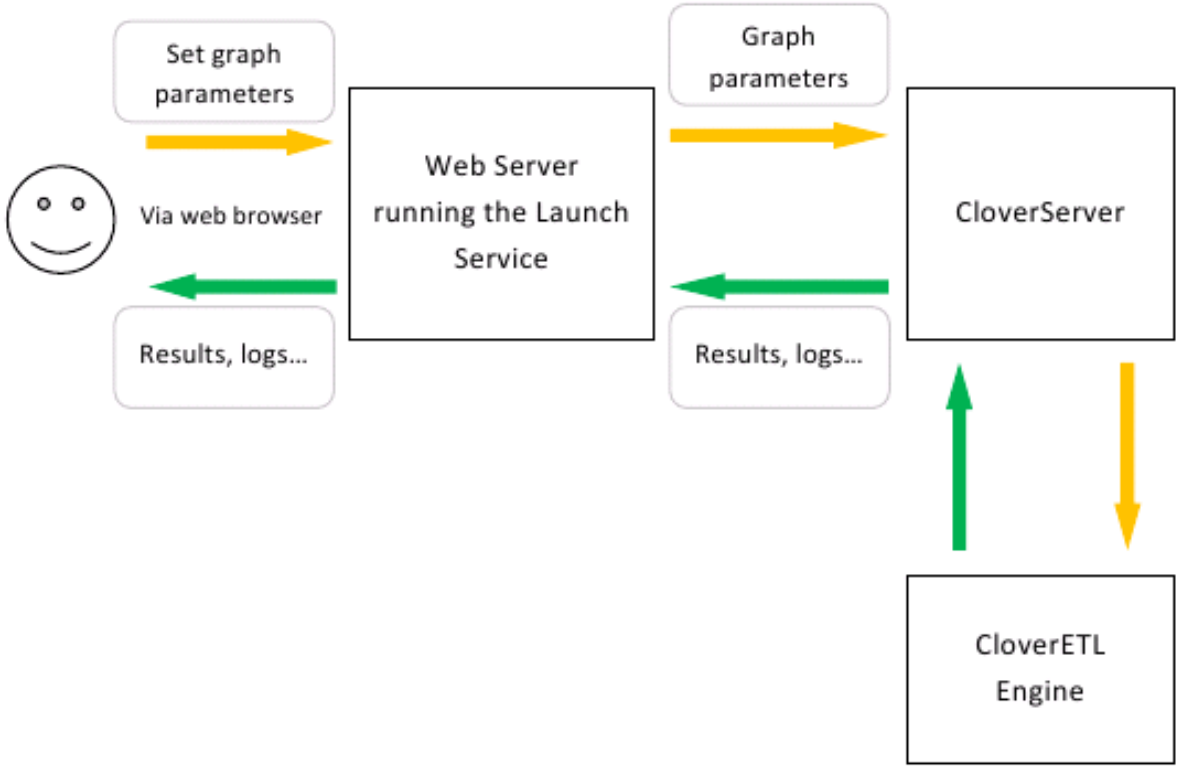



*Figure 17.1. Launch Services and CloverETL Server as web application back-end*

## Deploying Graph in Launch Service

To prepare a graph for publishing as a Launch Service, keep this in mind during the design process:

1. You can define a graph/jobflow listeners to create parameterized calls. Parameters are passed to the graph as Dictionary entries – so, design the graph so that it uses the Dictionary as input/output for parameters (e.g. file names, search terms, etc.)

2. The graph will need to be published in the Launch Services section, where you provide the configuration and binding for parameters to dictionary entries.

## Using Dictionary in ETL Graph/Jobflow for a Launch Services

A graph or a jobflow published as a service usually means that the caller sends request data (parameters or data) and the transformation processes it and returns back the results.

In a Launch Service definition, you can bind a service's parameters to Dictionary entries. These need to be predefined in the transformation.

Dictionary is a key-value temporary data interface between the running transformation and the caller. Usually, although not restricted to, Dictionary is used to pass parameters in and out the executed transformation.

For more information about Dictionary, read the "Dictionary" section in the CloverETL Designer User's Guide.

# Configuring the job in CloverETL Server Web GUI

Each Launch Service configuration is identified by its name, user, and group restriction. You can create several configurations with the same name, which is valid as long as they differ in their user or group restrictions.

User restrictions can then be used to launch different jobs for different users, even though they use the same launch configuration (i.e. name). For example, developers may want to use a debug version of the job, while the end customers will want to use a production job. The user restriction can also be used to prohibit certain users from executing the launch configuration completely.

Similarly, a group restriction can be used to differentiate jobs based on the user's group membership.

If multiple configurations match the current user/group and configuration name, the most specific one is picked. (The user name has higher priority than the group name.)

## Adding New Launch Configuration

Use the "New launch configuration" button on the Launch Services tab to create a new Launch Service. The name is the identifier for the service and will be used in the service URL. Then, select a sandbox and either a transformation graph or a jobflow that you want to publish.

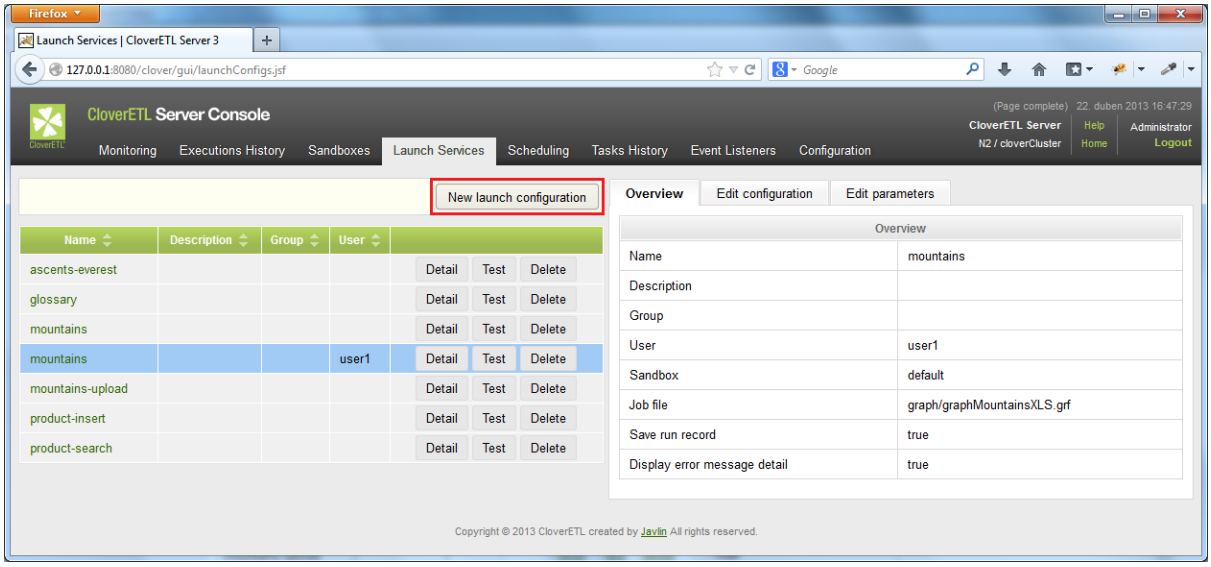



*Figure 17.2. Launch Services section*

Once you create the new Launch Service, you can set additional attributes like:

1. User and group access restrictions and additional configuration options (Edit Configuration)

2. Bind Launch Service parameters to Dictionary entries (Edit Parameters)

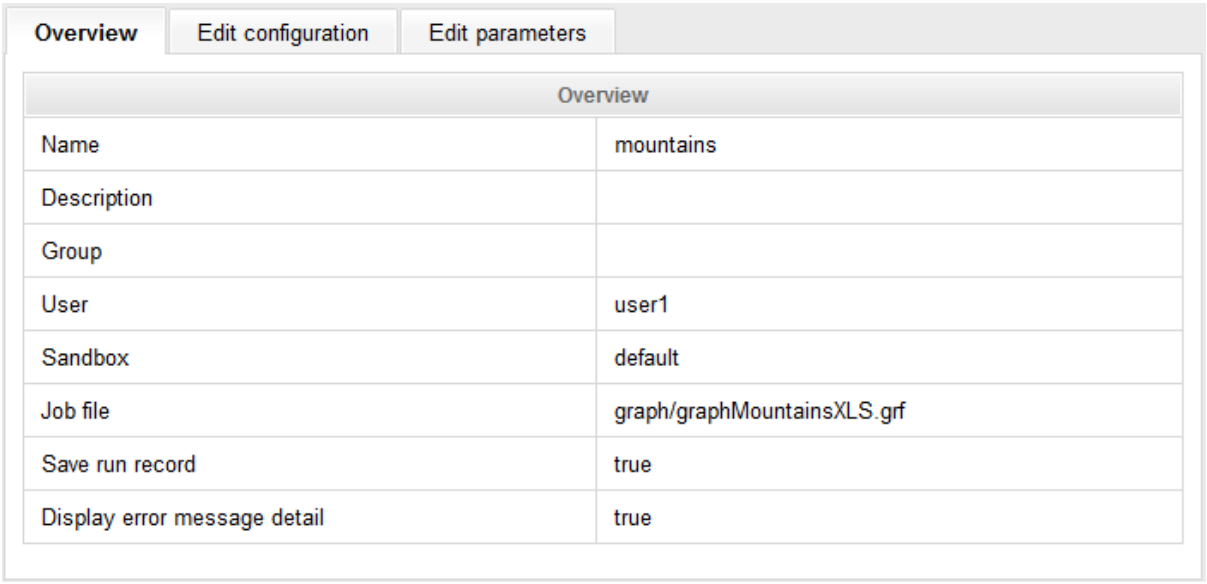



*Figure 17.3. Overview tab*

The Overview tab shows the basic details about the launch configuration. These can be modified in the Edit Configuration tab:

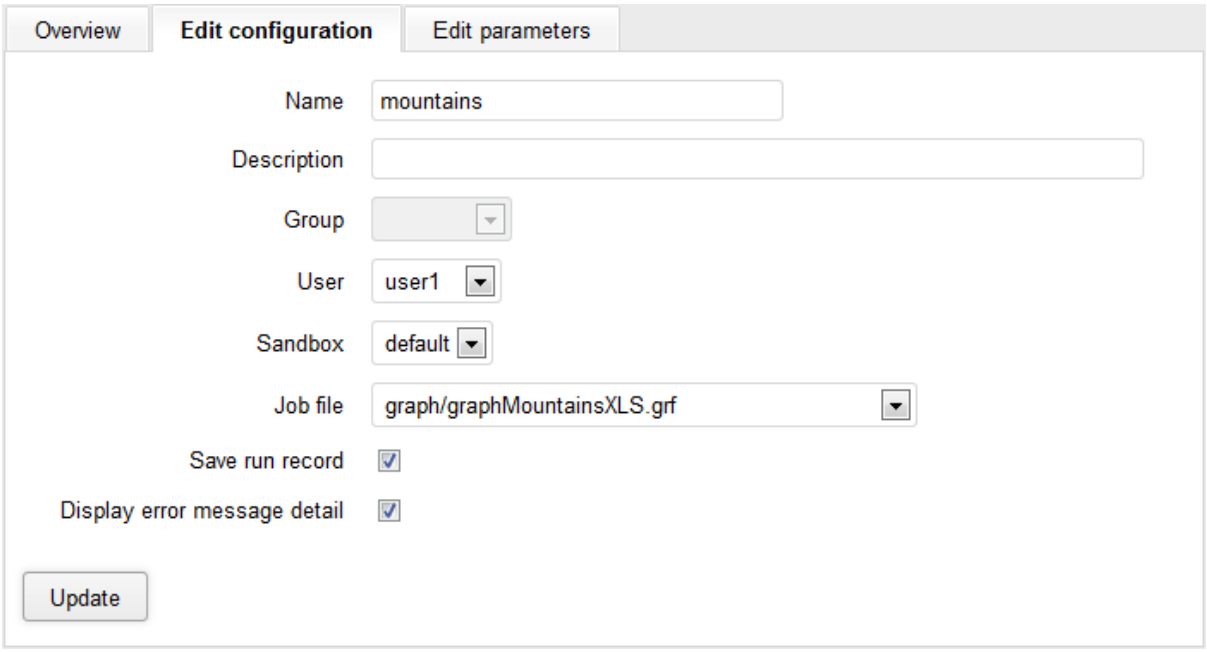



*Figure 17.4. Edit Configuration tab*

Editing configurations:

- *Name* - The name (identifier) under which the configuration will be accessible from the web.
- *Description* - The description of the configuration.
- *Group* - Restricts the configuration to a specific group of users.
- *User* - Restricts the configuration to a specified user.
- *Sandbox* - The CloverETL Sandbox where the configuration will be launched.

- *Job file* - Selects the job to run.
- *Save run record* - If checked, the details about the launch configuration will be stored in the Execution History. Uncheck this if you need to increase performance – storing a run record decreases response times for high frequency calls.
- *Display error message detail* - Check this if you want to get a verbose message in case the launch fails.

The “Edit parameters” tab can be used to configure parameter mappings for the launch configuration. The mappings are required for the Launch Service to be able to correctly assign parameters values based on the values sent in the launch request.

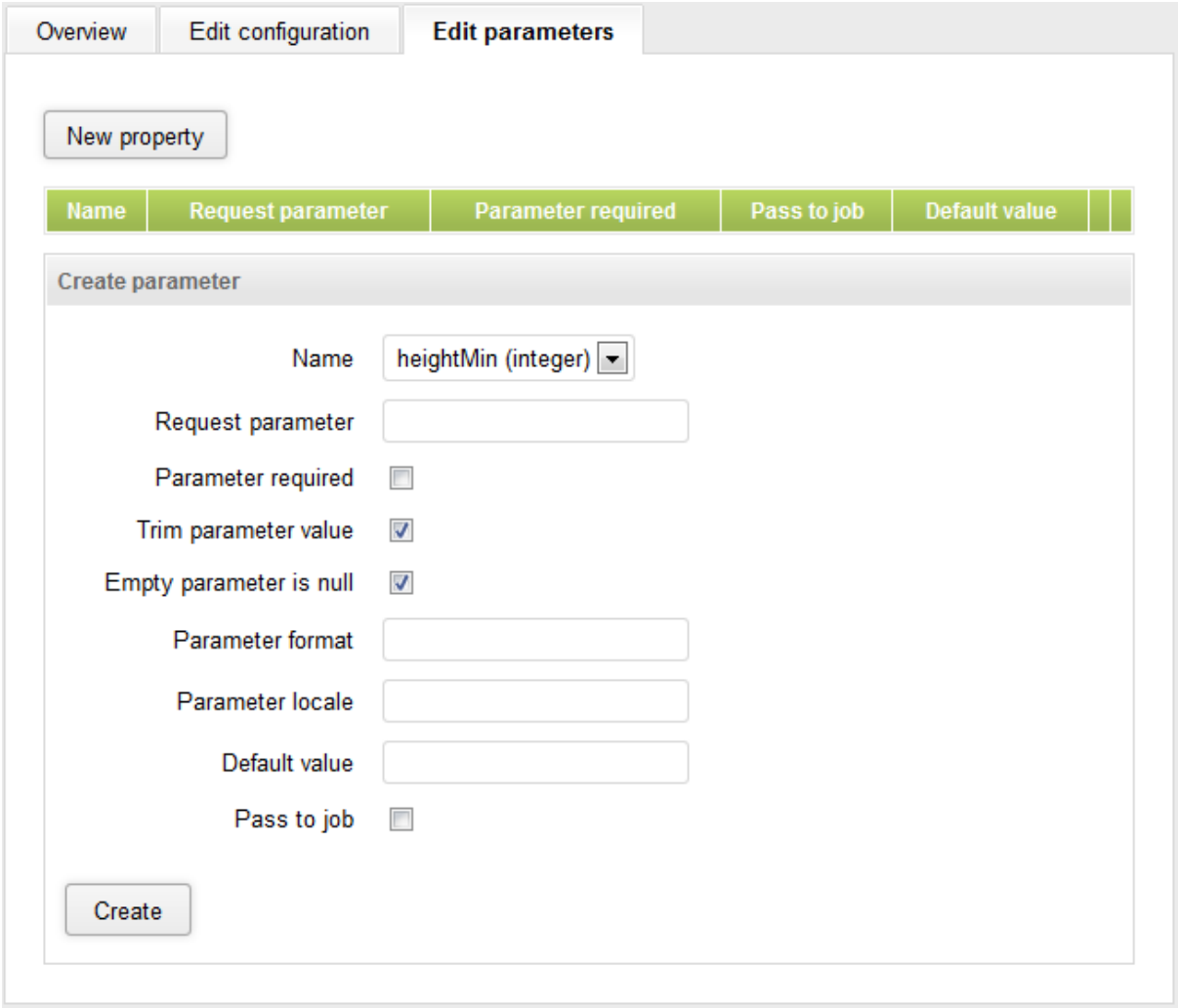



*Figure 17.5. Creating new parameter*

To add a new parameter binding, click on the “New property” button. Every required a graph/jobflow listenerproperty defined by the job needs to be created here.

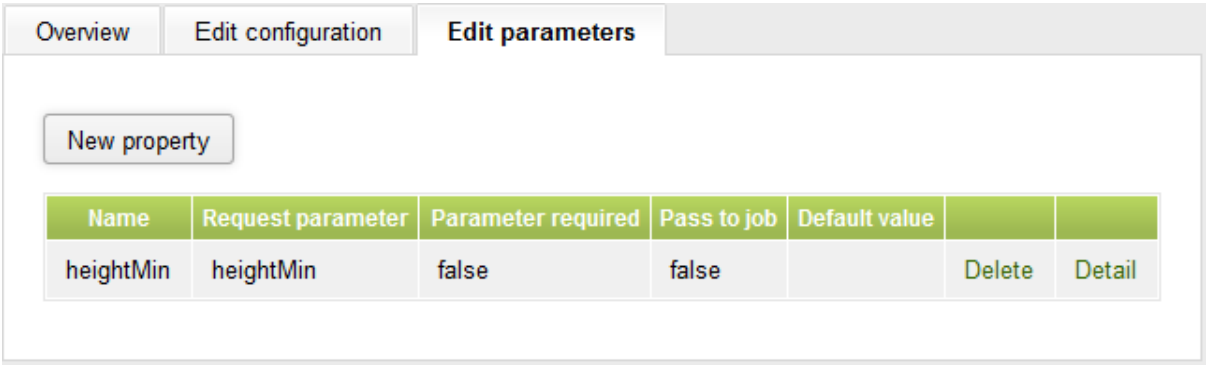



*Figure 17.6. Edit Parameters tab*

You can set the following fields for each property:

- *Name* - The name of the Dictionary entry defined in the graph/jobflow that you want to bind.
- *Request parameter* - The name of this property as it will be visible in the published service. This name can be different from Name.
- *Parameter required* - If checked, the parameter is mandatory and an error will be reported if it's omitted.
- *Pass to job* - If checked, the property value will be passed to the job also as a parameter (${ParameterName}, where ParameterName is equal to Name). This lets you use the parameter anywhere in the job definition (not just places that support Dictionary). However, parameters are evaluated during job initialization. Thus, such a job cannot be pooled which decreases performance for high frequency repetitive calls to the service. In this case, consider redesigning the transformation to use Dictionary instead, allowing for pooling.
- *Default value* - The default value applied in case the parameter is omitted in the launch request.

## Sending the Data to Launch Service

A launch request can be sent via HTTP GET or POST methods. A launch request is simply a URL which contains the values of all parameters that should be passed to the job. The request URL is composed of several parts:

(You can use a Launch Services test page, accessible from the login screen, to test drive Launch Services.)

```
[Clover Context]/launch/[Configuration name]?[Parameters]
```

- `[Clover Context]` is the URL to the context in which the CloverETL Server is running. Usually this is the full URL to the CloverETL Server (for example, for CloverETL Demo Server this would be http://server-demo.cloveretl.com:8080/clover).
- `[Configuration name]` is the name of the launch configuration specified when the configuration was created. In our example, this would be set to "mountains" (case-sensitive).
- `[Parameters]` is the list of parameters the configuration requires as a query string. It's a URL-encoded [RFC 1738] list of name=value pairs separated by the "&" character.

Based on the above, the full URL of a launch request for our example with mountains may look like this: http://server-demo.cloveretl.com:8080/clover/launch/NewMountains?heightMin=4000. In the request above, the value of heightMin property is set to 4000.

## Results of the Graph Execution

After the job terminates, the results are sent back to the HTTP client as content of an HTTP response.

Output parameters are defined in the job's Dictionary. Every Dictionary entry marked as "Output" is sent back as part of the response.

Depending on the number of output parameters, the following output is sent to the HTTP client:

- *No output parameters* - Only a summary page is returned. The page contains details such as: when the job was started, when it finished, the user name, and so on. The format of the summary page cannot be customized.
- *One output parameter* - In this case, the output is sent to the client as in the body of the HTTP response with its MIME content type defined by the property type in Dictionary.
- *Multiple output parameters* - In this case, each output parameter is sent to the HTTP client as part of multipart HTTP response. The content type of the response is either multipart/related or multipart/x-mixed-replace, depending on the HTTP client (the client detection is fully automatic). The multipart/related type is used for browsers based on Microsoft Internet Explorer and the multipart/x-mixed-replace is sent to browsers based on Gecko or Webkit.

Launch requests are recorded in the log files in the directory specified by the `launch.log.dir` property in the CloverETL Server configuration. For each launch configuration, one log file named [Configuration name]#[Launch ID].log is created. For each launch request, this file will contain only one line with following tab-delimited fields:

(If the property `launch.log.dir` is not specified, log files are created in the temp directory `[java.io.tmpdir]/cloverlog/launch` where "java.io.tmpdir" is system property)

- *Launch start time*
- *Launch end time*
- *Logged-in user name*
- *Run ID*
- *Execution status* FINISHED_OK, ERROR or ABORTED
- *IP Address* of the client
- *User agent* of the HTTP client
- *Query string* passed to the Launch Service (full list of parameters of the current launch)

In the case that the configuration is not valid, the same launch details are saved into the _no_launch_config.log file in the same directory. All unauthenticated requests are saved to the same file as well.

# Chapter 18. Configuration

We recommend the default installation (without any configuration) only for evaluation purposes. For production use, we recommend configuring a dedicated database and properly configuring the SMTP server for sending notifications.

## Config Sources and Their Priorities

There are several sources of configuration properties. If property isn't set, application default is used.

Warning: Do not combine sources specified below. Configuration becomes confusing and maintenance will be much more difficult.

### Context Parameters (Available on Apache Tomcat)

Some application servers allow to set context parameters without modification of WAR file. This way of configuration is possible and recommended for Tomcat.

Apache Tomcat may ignore some of context parameters in some environments, so this way isn't recommended, use of properties file is almost as convenient and muchg more reliable way.

#### Example for Apache Tomcat

On Tomcat it is possible to specify context parameters in context configuration file. `[tomcat_home]/conf/Catalina/localhost/clover.xml` which is created automatically just after deployment of CloverETL Server web application.

You can specify property by adding this element:

```
<Parameter name="[propertyName]" value="[propertyValue]" override="false" />
```

### Environment Variables

Set environment variable with prefix `clover.` , i.e. (`clover.config.file`)

Some operating systems may not use dot character, so also underlines (_) may be used instead of dots (`.`). So the `clover_config_file` works as well.

### System Properties

Set system property with prefix `clover.` , i.e. (`clover.config.file`)

Also underlines (_) may be used instead of dots (`.`) so the `clover_config_file` works as well.

### Properties File on default Location

Source is common properties file (text file with key-value pairs):

```
[property-key]=[property-value]
```

By default CloverETL tries to find config file `[workingDir]/cloverServer.properties`.

### Properties File on specified Location

The same as above, but properties file is not loaded from default location, because its location is specified by environment variable or system property `clover_config_file` or `clover.config.file`. This is recommended way of configuration if context parameters cannot be set in application server.

### Modification of Context Parameters in web.xml

Unzip `clover.war` and modify file `WEB-INF/web.xml`, add this code:

```
<context-param>
  <param-name>[property-name]</param-name>
  <param-value>[property-value]</param-value>
</context-param>
```

This way isn't recommended, but it may be useful when none of above ways is possible.

### Priorities of config Sources

Configuration sources have these priorities:

1. context parameters (specified in application server or directly in `web.xml`)
2. external config file CS tries to find it in this order (only one of them is loaded):
   - path specified by context parameter `config.file`
   - path specified by system property `clover_config_file` or `clover.config.file`
   - path specified by environment variable `clover_config_file` or `clover.config.file`
   - default location (`[workingDir]/cloverServer.properties`)
3. system properties
4. environment variables
5. default values

## Examples of DB Connection Configuration

In standalone deployment (non-clustered), configuration of DB connection is optional, since embedded Apache Derby DB is used by default and it is sufficient for evaluation. However, configuration of external DB connection is strongly recommended for production deployment. It is possible to specify common JDBC DB connection attributes (URL, username, password) or JNDI location of DB DataSource.

In clustered deployment, at least one node in cluster must have DB connection configured. Other nodes may have their own direct connection (to the same DB) or may use another node as proxy for persistent operations, however scheduler is active only on nodes with direct connection. See Clustering section for details about this feature, this section describes only direct DB connection configuration.

DB Configurations and their changes may be as follows:


- Embedded Apache Derby (p. 99)
- MySQL (p. 99)

## Embedded Apache Derby

Apache Derby embedded DB is used with default CloverETL Server installation. It uses working directory as storage directory for data persistence by default. This may be problem on some systems. In case of any problems with connecting to Derby DB, we recommend you configure connection to external DB or at least specify Derby home directory:

Set system property `derby.system.home` to set path which is accessible for application server. You can specify this system property by this JVM execution parameter:

```
-Dderby.system.home=[derby_DB_files_root]
```

If you use properties file for configuration, specify these parameters: `jdbc.driverClassName`, `jdbc.url`, `jdbc.username`, `jdbc.password`, `jdbc.dialect`. For example:

```
jdbc.driverClassName=org.apache.derby.jdbc.EmbeddedDriver
jdbc.url=jdbc:derby:databases/cloverDb;create=true
jdbc.username=
jdbc.password=
jdbc.dialect=org.hibernate.dialect.DerbyDialect
```

Take a closer look at the `jdbc.url` parameter. Part `databases/cloverDb` means a subdirectory for DB data. This subdirectory will be created in the directory which is set as `derby.system.home` (or in the working directory if `derby.system.home` is not set). Value `databases/cloverDb` is a default value, you may change it.

## MySQL

CloverETL Server requires MySql 5.x

If you use properties file for configuration, specify these parameters: `jdbc.driverClassName`, `jdbc.url`, `jdbc.username`, `jdbc.password`, `jdbc.dialect`. For example:

```
jdbc.driverClassName=com.mysql.jdbc.Driver
jdbc.url=jdbc:mysql://127.0.0.1:3306/clover?useUnicode=true&characterEncoding=utf8
jdbc.username=root
jdbc.password=
jdbc.dialect=org.hibernate.dialect.MySQLDialect
```

Since 3.0 JDBC driver is not included in CloverETL Server web archive, thus it must be added to the application server classpath.

Create DB with proper charset, like this:

```
CREATE DATABASE IF NOT EXISTS clover DEFAULT CHARACTER SET 'utf8';
```

# DB2

## DB2 on Linux/Windows

If you use properties file for configuration, specify these parameters: `jdbc.driverClassName`, `jdbc.url`, `jdbc.username`, `jdbc.password`, `jdbc.dialect`. For example:

```
jdbc.driverClassName=com.ibm.db2.jcc.DB2Driver
jdbc.url= jdbc:db2://localhost:50000/clover
jdbc.username=usr
jdbc.password=pwd
jdbc.dialect=org.hibernate.dialect.DB2Dialect
```

### Possible problems

#### Wrong pagesize

Database *clover* has to be created with suitable `PAGESIZE`. DB2 has several possible values for this property: 4096, 8192, 16384 or 32768.

CloverETL Server should work on DB with `PAGESIZE` set to 16384 or 32768. If `PAGESIZE` value is not set properly, there should be error message in the log file after failed CloverETL Server startup:

```
ERROR:
DB2 SQL Error: SQLCODE=-286, SQLSTATE=42727, SQLERRMC=16384;
ROOT, DRIVER=3.50.152
```

SQLERRMC contains suitable value for `PAGESIZE`.

You can create database with proper `PAGESIZE` like this:

```
CREATE DB clover PAGESIZE 32768;
```

#### The table is in the reorg pending state

After some ALTER TABLE commands, some tables may be in "reorg pending state". This behaviour is specific for DB2. ALTER TABLE DDL commands are executed only during the first start of new CloverETL Server version.

Error message for this issue may look like this:

```
Operation not allowed for reason code "7" on table "DB2INST2.RUN_RECORD"..
        SQLCODE=-668, SQLSTATE=57016
```

or like this

```
DB2 SQL Error: SQLCODE=-668, SQLSTATE=57016, SQLERRMC=7;DB2INST2.RUN_RECORD, DRIVER=3.50.152
```

In this case "RUN_RECORD" is table name which is in "reorg pending state" and "DB2INST2" is DB instance name.

To solve this, go to DB2 console and execute command (for table run_record):

```
reorg table run_record
```

DB2 console output should look like this:

```
db2 => connect to clover1
```

```
Database Connection Information

Database server        = DB2/LINUX 9.7.0
SQL authorization ID   = DB2INST2
Local database alias   = CLOVER1

db2 => reorg table run_record
DB20000I  The REORG command completed successfully.
db2 => disconnect clover1
DB20000I  The SQL DISCONNECT command completed successfully.
```

"clover1" is DB name

#### DB2 does not allow ALTER TABLE which trims DB column length.

This problem depends on DB2 configuration and we've experienced this only on some AS400s so far. CloverETL Server applies set of DP patches during the first installation after application upgrade. Some of these patches may apply column modifications which trims length of the text columns. These changes never truncate any data, however DB2 does not allow this since it "may" truncate some data. DB2 refuses these changes even in DB table which is empty. Solution is, to disable the DB2 warning for data truncation, restart CloverETL Server which applies patches, then enable DB2 warning again.

### DB2 on AS/400

The connection on AS/400 might be slightly different.

If you use properties file for configuration, specify these parameters: `jdbc.driverClassName`, `jdbc.url`, `jdbc.username`, `jdbc.password`, `jdbc.dialect`. For example:

```
jdbc.driverClassName=com.ibm.as400.access.AS400JDBCDriver
jdbc.username=javlin
jdbc.password=clover
jdbc.url=jdbc:as400://host/cloversrv;libraries=cloversrv;date format=iso
jdbc.dialect=org.hibernate.dialect.DB2400Dialect
```

Use credentials of your OS user for `jdbc.username` and `jdbc.password`.

`cloversrv` in `jdbc.url` above is the name of the DB schema.

You can create schema in AS/400 console:

- execute command `STRSQL` (**SQL console**)
- execute `CREATE COLLECTION cloversrv IN ASP 1`
- `cloversrv` is the name of the DB schema and it may be at most 10 characters long

Proper JDBC driver must be in the application server classpath.

I use JDBC driver `jt400ntv.jar`, which I've found in `/QIBM/ProdData/Java400` on the server.

Use `jt400ntv.jar` JDBC driver.

Do not forget to add jar with JDBC driver to the Tomcat classpath.

## Oracle

If you use properties file for configuration, specify these parameters: `jdbc.driverClassName`, `jdbc.url`, `jdbc.username`, `jdbc.password`, `jdbc.dialect`. For example:

```
jdbc.driverClassName=oracle.jdbc.OracleDriver
jdbc.url=jdbc:oracle:thin:@host:1521:db
jdbc.username=user
jdbc.password=pass
```

```
jdbc.dialect=org.hibernate.dialect.Oracle9Dialect
```

Do not forget to add jar with JDBC driver to the application server classpath.

Since CloverETL Server version 1.2.1, dialect `org.hibernate.dialect.Oracle10gDialect` is no longer available. Please use `org.hibernate.dialect.Oracle9Dialect` instead.

These are privileges which have to be granted to schema used by CloverETL Server:

```
CONNECT
CREATE SESSION
CREATE/ALTER/DROP TABLE
CREATE/ALTER/DROP SEQUENCE

QUOTA UNLIMITED ON <user_tablespace>;
QUOTA UNLIMITED ON <temp_tablespace>;
```

## MS SQL

Ms SQL requires configuration of DB server.

- Allowing of TCP/IP connection:
- execute tool **SQL Server Configuration Manager**
- go to **Client protocols**
- switch on TCP/IP (default port is 1433)
- execute tool **SQL Server Management Studio**
- go to **Databases** and create DB clover
- go to **Security/Logins** and create user and assign this user as owner of DB *clover*
- go to **Security** and check **SQL server and Windows authentication mode**

If you use properties file for configuration, specify these parameters: `jdbc.driverClassName`, `jdbc.url`, `jdbc.username`, `jdbc.password`, `jdbc.dialect`. For example:

```
jdbc.driverClassName=com.microsoft.sqlserver.jdbc.SQLServerDriver
jdbc.url=jdbc:sqlserver://localhost:1433;databaseName=clover
jdbc.username=user
jdbc.password=pass
jdbc.dialect=org.hibernate.dialect.SybaseDialect
```

Do not forget to add jar with JDBC driver to the Tomcat classpath.

## Postgre SQL

If you use properties file for configuration, specify these parameters: `jdbc.driverClassName`, `jdbc.url`, `jdbc.username`, `jdbc.password`, `jdbc.dialect`. For example:

```
jdbc.driverClassName=org.postgresql.Driver
jdbc.url=jdbc:postgresql://localhost/clover?charSet=UTF-8
jdbc.username=postgres
jdbc.password=
jdbc.dialect=org.hibernate.dialect.PostgreSQLDialect
```

Do not forget to add jar with JDBC driver to the Tomcat classpath.

### JNDI DB DataSource

Server can connect to JNDI DB DataSource, which is configured in application server or container. However there are some CloverETL parameters which must be set, otherwise the behaviour may be unpredictable:

```
datasource.type=JNDI # type of datasource; must be set, because default value is JDBC
datasource.jndiName=# JNDI location of DB DataSource; default value is java:comp/env/jdbc/clover_server #
jdbc.dialect=# Set dialect according to DB which DataSource is connected to.
The same dialect as in sections above. #
```

The parameters above may be set in the same ways as other params (in properties file or Tomcat context file)

Example of DataSource configuration in Apache Tomcat. Add following code to context file.

```
<Resource name="jdbc/clover_server" auth="Container"
    type="javax.sql.DataSource" driverClassName="com.mysql.jdbc.Driver"
    url="jdbc:mysql://192.168.1.100:3306/clover?useUnicode=true&amp;characterEncoding=utf8"
    username="root" password="" maxActive="20" maxIdle="10" maxWait="-1"/>
```

**Note**

Special characters you type in the context file have to be specified as XML entities. E.g. ampersand "&" as "&" etc.

## List of Properties

*Table 18.1. General configuration*

| key | description | default |
|---|---|---|
| config.file | location of CloverETL Server configuration file | [working_dir]/ cloverServer.properties |
| license.file | location of CloverETL Server licence file (license.dat) | |
| engine.config.file | location of CloverETL engine configuration properties file | properties file packed with CloverETL |
| private.properties | List of server properties which are used only by CloverETL Server code. So these properties are not accessible outside of the ServerFacade. By default there are all properties which may contain password in the list. Basically it means, that their values are not visible for web GUI users. Values are replaced by single star "*". Changes in this list may cause unexpected behavior of some server API. | jdbc.password, executor.password, security.ldap.password, clover.smtp.password |
| engine.plugins.src | This property may contain absolute path to some "source" of additional CloverETL engine plugins. These plugins are not a substitute for plugins packed in WAR. "Source" may be directory or zip file. Both directory and zip must contain subdirectory for each plugin. Changes in the directory or the ZIP file apply only when the server is restarted. For details see section "Extensibility - engine plugins". | empty |
| datasource.type | Set this explicitly to JNDI if you need CloverETL Server to connect to DB using JNDI datasource. In such case, parameters "datasource.jndiName" and "jdbc.dialect" must be set properly. Possible values: JNDI \| JDBC | JDBC |

| key | description | default |
|---|---|---|
| datasource.jndiName | JNDI location of DB DataSource. It is applied only if "datasource.type" is set to "JNDI". | java:comp/env/jdbc/clover_server |
| jdbc.driverClassName | class name for jdbc driver name | |
| jdbc.url | jdbc url used by CloverETL Server to store data | |
| jdbc.username | jdbc database user name | |
| jdbc.password | jdbc database user name | |
| jdbc.dialect | hibernate dialect to use in ORM | |
| quartz.driverDelegateClass | SQL dialect for quartz. Value is automatically derived from "jdbc.dialect" property value. | |
| security_enabled | true \| false If it is set to false, then no authentication is required and anyone has admin privileges. | true |
| security.session.validity | Session validity in milliseconds. When the request of logged-in user/client is detected, validity is automatically prolonged. | 14400000 |
| security.default_domain | Domain which all new users are included in. Stored in user's record in the database. Shouldn't be changed unless the "clover" must be white-labelled. | clover |
| security.basic_authentication.features_list | Semi-colon separated list of features which are accessible using HTTP and which should be protected by Basic HTTP Authentication. Each feature is specified by its servlet path. | /request_processor;/simpleHttpApi;/launch;/launchIt;/downloadStorage;/downloadFile;/uploadSandboxFile;/downloadLog |
| security.basic_authentication.realm | Realm string for HTTP Basic Authentication. | CloverETL Server |
| security.digest_authentication.features_list | Semi-colon separated list of features which are accessible using HTTP and which should be protected by HTTP Digest Authentication. Each feature is specified by its servlet path. Please keep in mind, that HTTP Digest Authentication is feature added to the version 3.1. If you upgraded your older CloverETL Server distribution, users created before the upgrade cannot use the HTTP Digest Authentication until they reset their passwords. So when they reset their passwords (or the admin does it for them), they can use Digest Authentication as well as new users. | /webdav |
| security.digest_authentication.realm | Realm string for HTTP Digest Authentication. If it is changed, all users have to reset their passwords, otherwise they won't be able to access to the server features protected by HTTP digest Authentication. | CloverETL Server |
| security.digest_authentication.nonce_validity | Interval of validity for HTTP Digest Authentication specified in seconds. When the interval passes, server requires new authentication from the client. Most of the HTTP clients do it automatically. | 300 |
| clover.event.fileCheckMinInterval | Interval of file checkes (in milliseconds) See Chapter 12, File event listeners (p. 73) for details. | 1000 |
| clover.smtp.transport.protocol | SMTP server protocol. Possible values are "smtp" or "smtps". | smtp |

| key | description | default |
|---|---|---|
| clover.smtp.host | SMTP server hostname or IP address | |
| clover.smtp.port | SMTP server port | |
| clover.smtp.authentication | true/false If it is false, username and password are ignored | |
| clover.smtp.username | SMTP server username | |
| clover.smtp.password | SMTP server password | |
| clover.smtp.additional.* | Properties with prefix "clover.smtp.additional." are automatically added (without the prefix) to the Properties instance passed to the Mailer. May be useful for some protocol specific parameters. Prefix is removed. | |
| logging.project_name | used in log messages where it is neccessary to name the product name | CloverETL |
| logging.default_subdir | name of default subdirectory for all server logs; it is relative to the path specified by system property "java.io.tmpdir". Don't specify absolute path, use properties which are intended for absolute path. | cloverlogs |
| launch.log.dir | Location, where server should store launch requests logs. See Launch Services section for details. | ${java.io.tmpdir}/ [logging. default_subdir]/ launch where ${java.io.tmpdir} is system property |
| graph.logs_path | Location, where server should store Graph run logs. See Logging section for details. | ${java.io.tmpdir}/ [logging. default_subdir]/graph where ${java.io.tmpdir} is system property |
| temp.default_subdir | Name of default subdirectory for server tmp files; it is relative to the path specified by system property "java.io.tmpdir". | clovertmp |
| graph.debug_path | Location, where server should store Graph debug info. | ${java.io.tmpdir}/ [temp.default_subdir]/ debug where ${java.io.tmpdir} is system property |
| graph.pass_event_params _to_graph_in_old_style | Since 3.0. It is switch for backwards compatibility of passing parameters to the graph executed by graph event. In version prior to 3.0 all params has been passed to executed graph. Since 3.0 just specified parameters are passed. Please see Task - Execution of Graph (p. 49) for details. | false |
| threadManager.pool. corePoolSize | Number of threads which are always active (running or idling). Related to thread pool for processing server events. | 4 |

| key | description | default |
|---|---|---|
| threadManager.pool. queueCapacity | Max size of the queue(FIFO) which contains tasks waiting for thread. Related to thread pool for processing server events. It means, that there won't be more then "queueCapacity" waiting tasks. i.e. queueCapacity=0 - no waiting tasks, each task is immediatelly executed in available thread or in new thread. queueCapacity=1024 - up to 1024 tasks may be waiting in the queue for available thread from "corePoolSize". | 12 |
| threadManager.pool. maxPoolSize | Max number of active threads. If no thread from core pool is available and queue capacity is exceeded, pool creates new threads up to "maxPoolSize" threads. If there are more concurrent tasks then maxPoolSize, thread manager refuses to execute it. Thus keep queueCapacity or maxPoolSize big enough. | 1024 |
| task.archivator.batch_size | Max number of records deleted in one batch. It is used for deleting of archived run records. | 50 |
| launch.http_header_prefix | Prefix of HTTP headers added by launch services to the HTTP response. | X-cloveretl |
| task.archivator. archive_file_prefix | Prefix of archive files created by archivator. | cloverArchive_ |
| license.context_names | Comma separated list of web-app contexts which may contain license. Each of them has to start with slash! Works only on Apache Tomcat. | /clover-license,/ clover_license |
| license.display_header | Switch which specifies whether display license header in server web GUI or not. | false |

*Table 18.2. Defaults for job execution configuration - see section Job config properties for details*

| key | description | default |
|---|---|---|
| executor.tracking_interval | Interval in milliseconds for scanning current status of running graph. The shorter interval, the bigger log file. | 2000 |
| executor.log_level | Log level of graph runs. TRACE \| DEBUG \| INFO \| WARN \| ERROR | INFO |
| executor.max_running_concurrently | Amount of graph instances which may exist(or run) concurrently. 0 means no limits | 0 |
| executor.max_graph_instance_age | Interval in milliseconds. Specifies how long graph instance can be idling before it is released from memory. 0 means no limits. This property has been renamed since 2.8. Original name was executor.maxGraphInstanceAge | 0 |
| executor.classpath | Classpath for transformation/processor classes used in the graph. Directory [sandbox_root]/trans/ does not have to be listed here, since it is automatically added to graph run classpath. | |
| executor.skip_check_config | Disables check of graph configuration. Increases performance of graph execution, however may be useful during graph development. | true |
| executor.password | Password for decoding of encoded DB connection passwords. | |
| executor.verbose_mode | If true, more descriptive logs of graph runs are generated. | true |
| executor.use_jmx | If true, graph executor registers jmx mBean of running graph. | true |
| executor.debug_mode | If true, edges with enabled debug store data into files in debug directory. See property "graph.debug_path" | false |

See "Clustering" section for more properties.

# Chapter 19. Graph/Jobflow parameters

The CloverETL Server passes a set of parameters to each graph or jobflow execution.

Keep in mind that placeholders (parameters) ${paramName} are resolved only during the initialization (loading of XML definition file), so if you need the parameters to be resolved for each job execution, you cannot set the job to be pooled. However, current parameter values are always accessible by inline Java code like this:

```
String runId = getGraph().getGraphProperties().getProperty("RUN_ID");
```

Properties may be added or replaced like this:

```
getGraph().getGraphProperties().setProperty("new_property", value );
```

This is set of parameters which are always set by CloverETL Server:

*Table 19.1. Defaults for graph execution configuration - see section Graph config properties for details*

| key | description |
|---|---|
| SANDBOX_CODE | Code of sandbox which contains executed graph. |
| JOB_FILE | Path to the file, relative to sandbox root path. |
| SANDBOX_ROOT | Absolute path sandbox root. |
| RUN_ID | ID of the graph execution. In standalone mode or in cluster mode, it is always unique. It may be lower then 0 value, if the run record isn't persistent. See "Launch Services" for details. |

## Another sets of parameters according the type of execution

There are some more parameters in dependence of way, how the graph is executed.

### executed from Web GUI

no more parameters

### executed by Launch Service invocation

Service parameters which have attribute **Pass to graph** enabled are passed to the graph not only as "dictionary" input data, but also as graph parameter.

### executed by HTTP API run graph operation invocation

Any URL parameter with "param_" prefix is passed to executed graph but without "param_" prefix. i.e. "param_FILE_NAME" specified in URL is paased to the graph as property named "FILE_NAME".

### executed by RunGraph component

Since 3.0 only parameters specified by "paramsToPass" attribute are passed from the "parent" graph to the executed graph. However common properties (RUN_ID, PROJECT_DIR, etc.) are overwritten by new values.

## executed by WS API method executeGraph invocation

Parameters with values may be passed to the graph with the request for execution.

## executed by task "graph execution" by scheduler

*Table 19.2. passed parameters*

| key | description |
|---|---|
| EVENT_SCHEDULE_EVENT_TYPE | Type of schedule SCHEDULE_PERIODIC \| SCHEDULE_ONETIME |
| EVENT_SCHEDULE_LAST_EVENT | Date/time of previous event |
| EVENT_SCHEDULE_DESCRIPTION | Schedule description, which is displayed in web GUI |
| EVENT_USERNAME | User who "owns" the event. For schedule it is the user who created the schedule |
| EVENT_SCHEDULE_ID | ID of schedule which triggered the graph |

## executed from JMS listener

There are many graph parameters and dictionary entries passed, depending on the type of incomming message. See details in Chapter 9, JMS messages listeners (p. 65).

## executed by task "Start a graph" by graph/jobflow event listener

Since 3.0 only specified properties from "source" job are passed to executed job by default. There is server config property "graph.pass_event_params_to_graph_in_old_style" which can change this behavior so that ALL parameters from "source" job are passed to the executed job. This switch is implemented for backwards compatibility. Regarding the default behaviour: You can specified list of parameters to pass in the editor of graph event listener. Please see the section "Task - Execution of Graph" for details.

However following parameters with current values are always passed to the target job

*Table 19.3. passed parameters*

| key | description |
|---|---|
| EVENT_RUN_SANDBOX | Sandbox with graph, which is source of the event |
| EVENT_JOB_EVENT_TYPE | GRAPH_STARTED \| GRAPH_FINISHED \| GRAPH_ERROR \| GRAPH_ABORTED \| GRAPH_TIMEOUT \| GRAPH_STATUS_UNKNOWN, analogically JOBFLOW_* for jobflow event listeners. |
| EVENT_RUN_JOB_FILE | jobFile of the job, which is source of the event |
| EVENT_RUN_ID | ID of the graph execution, which is source of the event. |
| EVENT_TIMEOUT | Number of miliseconds which specifies interval of timeout. Makes sence only for "timeout" graph event. |
| EVENT_RUN_RESULT | Result (or current status) of the execution, which is source of the event. |
| EVENT_USERNAME | User who "owns" the event. For graph events it is the user who created the graph event listener |

### executed by task "graph execution" by file event listener

*Table 19.4. passed parameters*

| key | description |
|---|---|
| EVENT_FILE_PATH | Path to file, which is source of the event. Does not contain file name. Does not end with file separator. |
| EVENT_FILE_NAME | Filename of the file which is source of the event. |
| EVENT_FILE_EVENT_TYPE | SIZE \| CHANGE_TIME \| APPEARANCE \| DISAPPEARANCE |
| EVENT_FILE_PATTERN | Pattern specified in file event listener |
| EVENT_FILE_LISTENER_ID | |
| EVENT_USERNAME | User who "owns" the event. For file events it is the user who created the file event listener |

## How to add another graph parameters

### Additional "Graph Config Parameters"

It is possible to add so-called additional parameters in Web GUI - section **Sandboxes** for the selected graph or for all graphs in the selected sandbox. See details in the section called “Job config properties” (p. 31).

### Task "execute_graph" parameters

The "execute graph" task may be triggered by schedule, graph event listener or file event listener. Task editor allows you to specify key=value pairs which are passed to executed graph.

# Chapter 20. Recommendations for transformations developers

## Add external libraries to app-server classpath

Connections (JDBC/JMS) may require third-party libraries. We strongly recommended adding these libraries to the app-server classpath.

CloverETL allows you to specify these libraries directly in a graph definition so that CloverETL can load these libraries dynamically. However, external libraries may cause memory leak, resulting in "java.lang.OutOfMemoryError: PermGen space" in this case.

In addition, app-servers should have the JMS API on their classpath – and the third-party libraries often bundle this API as well. So it may result in classloading conflicts if these libraries are not loaded by the same classloader.

## Another graphs executed by RunGraph component may be executed only in the same JVM instance

In the server environment, all graphs are executed in the same VM instance. The attribute "same instance" of the RunGraph component cannot be set to false.

# Chapter 21. Logging

## Main logs

The CloverETL Server uses the log4j library for logging. The WAR file contains the default log4j configuration.

By default, log files are produced in the directory specified by system property "java.io.tmpdir" in the "cloverlogs" subdirectory.

"java.io.tmpdir" usually contains common system temp dir i.e. "/tmp". On Tomcat, it is usually "[TOMCAT_HOME]/temp"

The default logging configuration may be overridden by system property "log4j.configuration", which should contain the URL to log4j config file.

```
log4j.configuration=file:/home/clover/config/log4j.xml
```

Since such a configuration overrides the default configuration, it may have influence over Graph run logs. So your own log config has to contain following fragment to preserve Graph run logs

```
<logger name="Tracking" additivity="false">
  <level value="debug"/>
</logger>
```

These system properties allow for logging of HTTP requests/responses to stdout:

Client side:

`com.sun.xml.ws.transport.http.client.HttpTransportPipe.dump=true` (for more information consult CloverETL Designer Users's Guide - chapter Integrating CloverETL Designer with CloverETL Server)

Server side:

`com.sun.xml.ws.transport.http.HttpAdapter.dump=tru`

## Graph run logs

Each graph or jobflow run has its own log file – for example, in the Server Console, section "Executions History".

By default, these log files are saved in the subdirectory cloverLogs/graph in the directory specified by "java.io.tmpdir" system property.

It’s possible to specify a different location for these logs by the CloverETL property "graph.logs_path". This property does not influence main Server logs.

# Chapter 22. CloverETL Server API Extensibility

The CloverETL Server implements extensibility of its APIs, so the Server may expose additional features with a custom API.

There are two possibilities: The Groovy code API and the OSGi plugin.

## Groovy Code API

Since 3.3

The CloverETL Server Groovy Code API allows clients to execute Groovy code stored on the Server by an HTTP request. Executed code has access to the ServerFacade, instance HTTP request and HTTP response, so it's possible to implement a custom CloverETL Server API in the Groovy code.

To execute the code, call URL:

```
http://[host]:[port]/clover/groovy/[sandboxCode]/[pathToGroovyCodeFile]
```

Protocol HTTP can be changed to secured HTTPS according to the web server configuration.

The Server uses Basic or Digest authentication according to the configuration. So, the user must be authorized and must have permission to execute in the specified sandbox and permission to call "Groovy Code API".

Note that permission to call "Groovy Code API" (and edit them) is a very strong permission, since the Groovy Code can basically do the same as Java code and it runs as the same system process as a whole application container.

### Variables accessible in the Groovy code

By default, these variables are accessible to the Groovy code

*Table 22.1. Variables accessible in groovy code*

| type | key | description |
|---|---|---|
| javax.servlet.http.HttpServletRequest | request | Instance of HTTP request, which triggered the code. |
| javax.servlet.http.HttpServletResponse | response | Instance of HTTP response, which will be sent to the client when the script finishes. |
| javax.servlet.http.HttpSession | session | Instance of HTTP session. |
| javax.servlet.ServletConfig | servletConfig | instance of ServletConfig |
| javax.servlet.ServletContext | servletContext | instance of ServletContext |
| com.cloveretl.server.api.ServerFacade | serverFacade | Instance of serverFacade usable for calling CloverETL Server core features.<br><br>WAR file contains JavaDoc of facade API and it is accessible on URL: http://host:port/clover/javadoc/index.html |
| String | sessionToken | sessionToken, needed for calling serverFacade methods |

### Code examples

Code may return string that will be returned as content of the HTTP response – or it may construct the output itself to the output Writer

The following script writes its own output and doesn't return anything, so the underlying servlet doesn't modify the output at all. The advantage of this approach is that the output may be constructed on the fly and sent to the client continuously. However, when the output stream (or writer) is opened, the servlet won't send any error description in case of any error during the script processing.

```
response.getWriter().write("write anything to the output");
```

The following script returns String, so the underlying servlet puts the string to the output. The advantage of this approach is that in case of any error during code processing, the servlet returns a full stacktrace, so the script may be fixed. However, the constructed output may consume some memory.

```
String output = "write anything to the output";
return output;
```

The following script is little more complex. It returns XML with a list of all configured schedules. You need to have permission to list the schedules.

```
// uses variables: response, sessionToken, serverFacade
import java.io.*;
import java.util.*;
import javax.xml.bind.*;
import com.cloveretl.server.facade.api.*;
import com.cloveretl.server.persistent.*;
import com.cloveretl.server.persistent.jaxb.*;

JAXBContext jc = JAXBContext.newInstance( "com.cloveretl.server.persistent:com.cloveretl.server.persistent.jaxb" )
Marshaller m = jc.createMarshaller();
m.setProperty(Marshaller.JAXB_ENCODING, "UTF-8");
m.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, Boolean.TRUE);
m.setProperty(Marshaller.JAXB_SCHEMA_LOCATION, "/clover/schemas/cs.xsd");

Response<List<Schedule>> list = serverFacade.findScheduleList(sessionToken, null);
SchedulesList xmlList = new SchedulesList();
xmlList.setSchedulesList(list.getBean());
m.marshal(xmlList, response.getWriter());
```

# Embedded OSGi framework

Since 3.0

The CloverETL Server includes an embedded OSGi framework which allows for implementation of a "plugin" (OSGi bundle). It can add a new API operation or even extend the Server Console UI. It is independent of the standard clover.war.

CloverETL itself isn't based on OSGi technology, OSGi is used only optionally for extending server APIs. OSGi framework is completely disabled by default and is enabled only when the property "plugins.path" is set as described below.

Embedded OSGi framework Equinox uses some invasive techniques, so it may interfere with other applications or even some CloverETL features. Generic recommendation is to use Groovy API explained above instead, however there are still use cases when OSGi plugin is better choise. E.g. custom API has to use different libraries then the ones on the server classpath. Whereas groovy uses the same classpath as CloverETL, OSGi plugin has it's own isolated classpath. So in such case, when the OSGi plugin has to be used, CloverETL should be deployed in the environment which is as simple as possible, e.g. Clover is the only app deployed on the container, which should be also lightweight - Tomcat or Jetty. Such deployment would minimalize chance of possible conflicts.

Examples of interferences with embedded OSGi framework (only when framework is explicitly configured and initialized)

- OSGi framework causes malfunction of CloverETL components WebServiceClient and EmailReader on IBM Websphere
- OSGi framework causes failure of CloverETL startup on Weblogic 10.3.5 running on Oracle JRockit JVM
- OSGi framework can't initialize itself properly on Weblogic 12

## Plugin possibilities

Basically, the plugin may work as new server API similarly as Launch Services, HTTP API, or WebServices API. It may just be simple JSP, HttpServlet or complex SOAP Web Services. If the plugin contains some HTTP service, it’s registered to listen on a specified URL during the startup and incoming HTTP requests are "bridged" from the web container to the plugin. The plugin itself has access to the internal CloverETL Server interface called "ServerFacade". ServerFacade offers methods for executing graphs, obtaining graph status and executions history, and manipulation with scheduling, listeners, configuration, and many more. So the API may be customized according to the needs of a specific deployment.

## Deploying an OSGi bundle

There are two CloverETL Server configuration properties related to the OSGi framework:

- plugins.path - Absolute path to the directory containing all your plugins (JAR files).
- plugins.autostart - It is a comma-separated plugin names list. These plugins will be started during the server startup. Theoretically, the OSGi framework can start the OSGi bundle on demand; however, it is unreliable when the servlet bridge to the servlet container is used, so we strongly recommended naming all your plugins.

To deploy your plugin: set the two config properties, copy the plugin to the directory specified by "plugins.path" and restart the server.

# Chapter 23. Extensibility - CloverETL Engine Plugins

Since 3.1.2

The CloverETL Server can use external engine plugins loaded from a specified source. The source is specified by config property "engine.plugins.src"

See details about the possibilities with CloverETL configuration in *Chapter 18,* Configuration *(p. 97)*

This property must be the absolute path to the directory or zip file with additional CloverETL engine plugins. Both the directory and zip must contain a subdirectory for each plugin. These plugins are not a substitute for plugins packed in WAR. Changes in the directory or the ZIP file apply only when the server is restarted.

Each plugin has its own class-loader that uses a parent-first strategy by default. The parent of plugins' classloaders is web-app classloader (content of [WAR]/WEB-INF/lib). If the plugin uses any third-party libraries, there may be some conflict with libraries on parent-classloaders classpath. These are common exceptions/errors suggesting that there is something wrong with classloading:

- java.lang.ClassCastException
- java.lang.ClassNotFoundException
- java.lang.NoClassDefFoundError
- java.lang.LinkageError

There are a couple of ways you can get rid of such conflicts:

- Remove your conflicting third-party libraries and use libraries on parent classloaders (web-app or app-server classloaders)
- Use a different class-loading strategy for your plugin.
  - In the plugin descriptor plugin.xml, set attribute greedyClassLoader="true" in the element "plugin"
  - It means that the plugin classloader will use a self-first strategy
- Set an inverse class-loading strategy for selected java packages.
  - In the plugin descriptor plugin.xml, set attribute "excludedPackages" in the element "plugin".
  - It's a comma-separated list of package prefixes – like this, for example: excludedPackages="some.java.package,some.another.package"
  - In the previous example, all classes from "some.java.package", "some.another.package" and all their sub-packages would be loaded with the inverse loading strategy, then the rest of classes on the plugins classpath.

The suggestions above may be combined. It's not easy to find the best solution for these conflicts and it may depend on the libraries on app-server classpath.

For more convenient debugging, it’s useful to set TRACE log level for related class-loaders.

```
<logger name="org.jetel.util.classloader.GreedyURLClassLoader">
 <level value="trace"/>
</logger>
<logger name="org.jetel.plugin.PluginClassLoader">
 <level value="trace"/>
</logger>
```

See the "Logging" section for details about overriding a server log4j configuration.

# Chapter 24. Clustering

There are two common Cluster features: high availability and scalability. Both are implemented by the CloverETL Server on different levels. This section should clarify the basics of CloverETL Clustering.

The CloverETL Server only works in the Cluster if your license allows it.

## High Availability

Since version 3.0, CloverETL Server does not recognize any differences between cluster nodes. Thus, there are no "master" or "slave" nodes meaning all nodes can be virtually equal. There is no single point of failure(SPOF) in the CloverETL cluster itself, however SPOFs may be in the input data or some other external element.

Clustering offers high availability(HA) for all features accessible through HTTP. This includes sandbox browsing, modification of services configuration (scheduling, launch services, listeners) and primarily job executions. Any cluster node may accept incoming HTTP requests and process them itself or delegate it to another node.

Since all nodes are equal, almost all requests may be processed by any cluster node:

- All job files, metadata files, etc. are located in shared sandboxes. Thus all nodes have access to them. A shared filesystem may be a SPOF, thus it is recommended to use a replicated filesystem instead.
- The database is shared by all cluster nodes. Again, a shared DB might be a SPOF, however it may be clustered as well.

But there is still a possibility, that a node cannot process a request by itself. In such cases, it completely and transparently delegates the request to a node which can process the request.

These are the requests which are limited to one (or more) specific node(s):

- a request for the content of a partitioned or local sandbox. These sandboxes aren't shared among all cluster nodes. Please note that this request may come to any cluster node which then delegates it to a target node, however, this target node must be up and running.
- A job is configured to use a partitioned or local sandbox. These graphs need nodes which have a physical access to the required sandboxes.
- A job has allocation specified by specific cluster nodes. Concept of "allocation" is described in the following sections.

Thus an inaccessible cluster node may cause a failure of the request, so if it's possible, it's better to avoid using specific cluster nodes or resources accessible only by specific cluster node.

CloverETL itself implements a load balancer for executing jobs. So a job which isn't configured for some specific node(s) may be executed anywhere in the cluster and the CloverETL load balancer decides, according to the request and current load, which node will process the job. All this is done transparently for the client side.

To achieve HA, it is recommended to use an independent HTTP load balancer. Independent HTTP load balancers allow transparent fail-overs for HTTP requests. They send requests to the nodes which are running.

## Scalability

There are two independent levels of scalability implemented. Scalability of transformation requests(and any HTTP requests) and data scalability (parallel data processing).

Both of these "scalability levels" are "horizontal". Horizontal scalability means adding nodes to the cluster, whereas vertical scalability means adding resources to a single node. Vertical scalability is supported natively by the CloverETL engine and it is not described here.

## Transformation Requests

Basically, the more nodes we have in the cluster, the more transformation requests (or HTTP requests in general) we can process at one time. This type of scalability is the CloverETL server's ability to support a growing number of clients. This feature is closely related to the use of an HTTP load balancer which is mentioned in the previous section.

## Parallel Data Processing

This type of scalability is currently available only for ETL graphs. Jobflow and Profiler jobs can't run in parallel.

When a transformation is processed in parallel, the whole graph (or its parts) runs in parallel on multiple cluster nodes having each node process just a part of the data.

So the more nodes we have in the cluster, the more data can be processed in the specified time.

The data may be split(partitioned) before the graph execution or by the graph itself on the fly. The resulting data may be stored in partitions or gathered and stored as one group of data.

The curve of scalability may differ according to the type of transformation. It may be almost linear, which is almost always ideal, except when there is a single data source which cannot be read by multiple readers in parallel limiting the speed of further data transformation. In such cases it is not beneficial to have parallel data processing since it would actually wait for input data.

### ETL Graph Allocation

Each ETL graph executed in cluster environment is automatically subjected to transformation analysis. The main goal of this analysis is to find so called ETL graph **allocation**. The graph allocation is set of instructions for cluster environment how the transformation should be executed. For better understanding how the parallel data processing works, it is necessary to get deeper information about the graph analysis and resulted allocation.

First of all, analysis needs to find allocation for each individual component. The component allocation is set of cluster nodes where the component should be running. There is several ways how the component allocation can be specified, see following section of the documentation. But important information for now is, that a component can be requested to run in multiple instances - that is necessary for parallel data processing. Next step of analysis is to find optimal graph decomposition to ensure all component allocation will be satisfied and tries to minimise number of remote edges between graph instances.

Resulted analysis says how many instances (workers) of the graph needs to be executed, on which cluster nodes these instances will be running and which components will be present in the instances. In other words, one executed graph can be running in many instances, each instance can be processed on arbitrary cluster node and moreover each instance contains only convenient components.

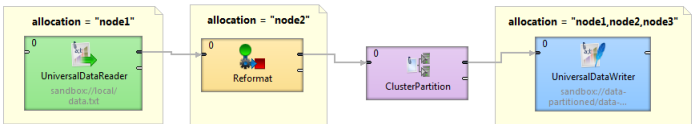



*Figure 24.1. Component allocations example*

This figure shows sample graph with few components with various component allocations. First component UniversalDataReader requests to be executed on node1, following Reformat component should be running on cluster node2, the ClusterPartition component is special component which makes possible to change cardinality of allocation of two interconnected components (detailed description of cluster partitioning and gathering follows this section). The last component UniversalDataWriter requires to be executed right on three cluster nodes node1, node2 and node3. Visualisation of transformation analysis shows the following figure. Three workers (graphs) will be executed, each on different cluster node (which is not necessary, even multiple workers can be associated with a single node). Worker on cluster node1 contains only UniversalDataReader and first of three instances

of UniversalDataWriter component. Both components are connected by remote edges with components, which are running on node2. The worker running on node3 contains only UniversalDataWriter fed by data remotely transfered from ClusterPartitioner running on node2.

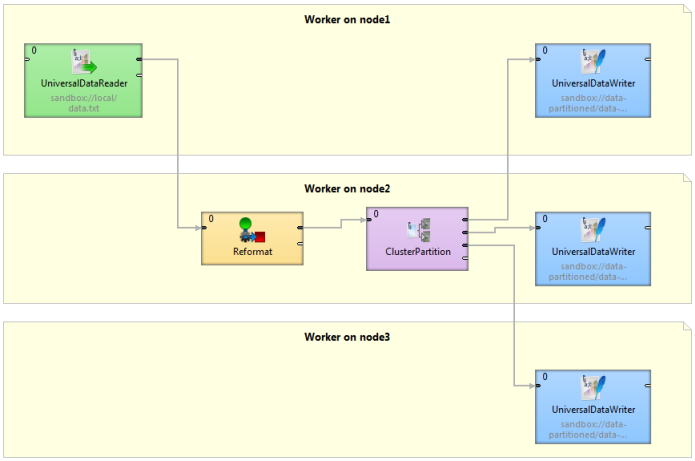



*Figure 24.2. Graph decomposition based on component allocations*

### Component Allocation

Allocation of a single component can be derived in several ways (list is ordered according priority):

- **Explicit definition** - all components have common attribute **Allocation**. CloverETL Designer allows user to use convenient dialog.

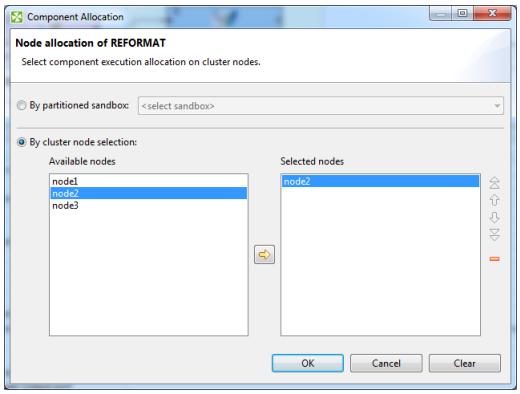



*Figure 24.3. Component allocation dialog*

Three different approaches are available for explicit allocation definition:

- Allocation based on number of workers - component will be executed in requested instances on some cluster nodes, which are preferred by CloverETL Cluster. Server can use build-in loadbalancing algorithm to ensure fastest data processing.
- Allocation based on reference on a partitioned sandbox - component allocation corresponds with locations of given partitioned sandbox. Each partitioned sandbox has a list of locations, each bound to specific cluster node. Thus allocation would be equivalent to list of locations. See "Partitioned sandbox" in Partitioned and Local Sandboxes (p. 120) for details.
- allocation defined by list of cluster node identifiers (single cluster node can be used more times)

- **Reference to a partitioned sandbox** UniversalDataReader, UniversalDataWriter and ParallelReader components derives theirs allocation from fileURL attribute. In case the URL refers to a file in a partitioned sandbox, the component allocation is automatically derived from locations of the partitioned sandbox. So in

case you manipulate with one of these components with a file in partitioned sandbox suitable allocation is used automatically.

- **Adoption from neighbour components** By default, allocation is inherited from neighbour components. Components on the left side have higher priority. Cluster partitioners and cluster gathers are nature bounds for recursive allocation inheritance.

## Partitioning/gathering Data

As mentioned before, data may be partitioned and gathered in multiple ways. It may be prepared before the graph is executed or it may be partitioned on the fly.

**Partitioning/gathering "on the fly"**

There are six special components to consider: ClusterPartition, ClusterLoadBalancingPartition, ClusterSimpleCopy, ClusterSimpleGather, ClusterMerge and ClusterRepartition. All work similarly to their non-cluster variation. But their splitting or gathering nature is used to change data flow allocation, so they may be used to change distribution of the data among workers.

**ClusterPartition** and **ClusterLoadBalancingPartition** work similar to a common partitioner, change the data allocation from 1 to N. Component preceding the ClusterPartitioner run on just one node, whereas component behind the ClusterPartitioner run in parallel according to node allocation. **ClusterSimpleCopy** component can be use in similar locations, this component does not distribute the data records, but copies them to all output workers.

**ClusterGather** and **ClusterMerge** work in the opposite way. They change the data allocation from N to 1. Component preceding the gather/merge run in parallel while component behind the gather run on just one node.

**Partitioning/gathering data by external tools**

Partitioning data on the fly may in some cases be an unnecessary bottleneck. Splitting data using low-level tools can be much better for scalability. The optimal case being, that each running worker reads data from an independent data source. Thus there does not have to be a ClusterPartitioner component and the graph runs in parallel from the beginning.

Or the whole graph may run in parallel, however the results would be partitioned.

## Node Allocation Limitations

As described above, each component may it's own node allocation specified which may result in some conflicts.

- *Node allocation of neighbouring components must have the same cardinality* So it doesn't have to be the same allocation, but the cardinality must be the same. E.g. There is an ETL graph with 2 components: DataGenerator and Trash. DataGenerator allocated on nodeA sending data to Trash allocated on nodeB works fine. DataGenerator allocated on nodeA sending data to Trash allocated on nodeA and nodeB fails.
- *Node allocation behind the ClusterGather and ClusterMerge must have cardinality 1* So it may be any allocation, but the cardinality must be just 1.
- *Node allocation of components in front of the ClusterPartition, ClusterLoadBalancingPartition and ClusterSimpleCopy must have cardinality 1*

## Partitioned and Local Sandboxes

Partitioned and local sandboxes were mentioned in previous sections. These new sandbox types were introduced in version 3.0 and they are vital for parallel data processing.

Together with shared sandboxes, we have three sandbox types in total.

**Shared sandbox**

This type of sandbox must be used for all data which is supposed to be accessible on all cluster nodes. This includes all graphs, metadata, connections, classes and input/output data for graphs which should support HA, as described

above. All shared sandboxes reside in the directory, which must be properly shared among all cluster nodes. You can use suitable sharing/replicating tool according to the operating system and filesystem.

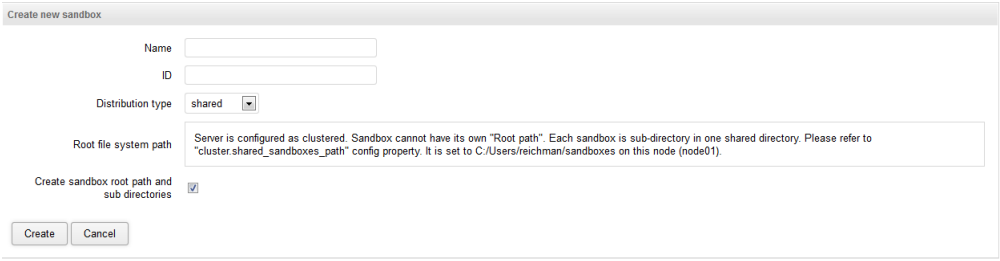



*Figure 24.4. Dialog form for creating new shared sandbox*

As you can see in the screenshot above, you cannot specify any root path on the filesystem. Shared sandboxes are stored in the directory specified by "cluster.shared_sandboxes_path". Each shared sandbox has its own subdirectory in it, which is named by sandbox ID.

**Local sandbox**

This sandbox type is intended for data, which is accessible only by certain cluster nodes. It may include massive input/output files. The purpose being, that any cluster node may access content of this type of sandbox, but only one has local(fast) access and this node must be up and running to provide data. The graph may use resources from multiple sandboxes which are physically stored on different nodes since cluster nodes are able to create network streams transparently as if the resource was a local file. See Using a Sandbox Resource as a Component Data Source (p. 122) for details.

Do not use local sandbox for common project data (graphs, metadata, connections, lookups, properties files, etc.). It would cause odd behaviour. Use shared sandboxes instead.

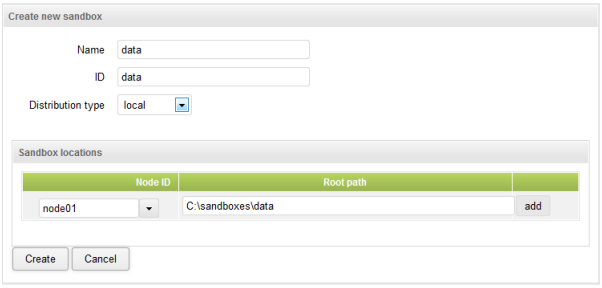



*Figure 24.5. Dialog form for creating new local sandbox*

**Partitioned sandbox**

This type of sandbox is actually an abstract wrapper for a couple of physical locations existing typically on different cluster nodes. However, there may be multiple locations on the same node. A partitioned sandbox has two purposes which are both closely related to parallel data processing.

1. **node allocation** specification - locations of a partitioned sandbox define the workers which will run the graph or its parts. So each physical location will cause a single worker to run. This worker does not have to actually store any data to "its" location. It is just a way to tell the CloverETL Server: "execute this part of ETL graph in parallel on these nodes"

2. **storage for part of the data** during parallel data processing. Each physical location contains only part of the data. In a typical use, we have input data split in more input files, so we put each file into a different location and each worker processes its own file.

As you can see on the screenshot above, for a partitioned sandbox, you can specify one or more physical locations on different cluster nodes.

Do not use partitioned sandbox for common project data (graphs, metadata, connections, lookups, properties files, etc.). It would cause odd behavior. Use shared sandboxes instead.

### Using a Sandbox Resource as a Component Data Source

A sandbox resource, whether it is a shared, local or partitioned sandbox (or ordinary sandbox on standalone server), is specified in the graph under the fileURL attributes as a so called sandbox URL like this:

```
sandbox://data/path/to/file/file.dat
```

where "data" is a code for sandbox and "path/to/file/file.dat" is the path to the resource from the sandbox root. URL is evaluated by CloverETL Server during graph execution and a component (reader or writer) obtains the opened stream from the server. This may be a stream to a local file or to some other remote resource. Thus, a graph does not have to run on the node which has local access to the resource. There may be more sandbox resources used in the graph and each of them may be on a different node. In such cases, CloverETL Server would choose the node with the most local resources to minimalize remote streams.

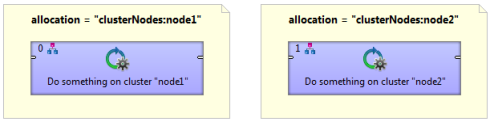
The sandbox URL has a specific use for parallel data processing. When the sandbox URL with the resource in a *partitioned sandbox* is used, that part of the graph/phase runs in parallel, according to the node allocation specified by the list of partitioned sandbox locations. Thus, each worker has it is own local sandbox resource. CloverETL Server evaluates the sandbox URL on each worker and provides an open stream to a local resource to the component.

The sandbox URL may be used on standalone server as well. It is excellent choice when graph references some resources from different sandboxes. It may be metadata, lookup definition or input/output data. Of course, referenced sandbox must be accessible for the user who executes the graph.

## Graph allocation examples

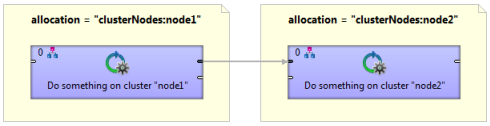
#### Basic component allocation

This example shows two component graph, where allocation ensures the first component will be executed on cluster node1 and the second component will be executed on cluster node2.



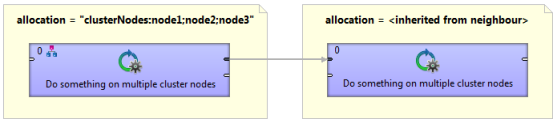

#### Basic component allocation with remote data transfer

Two components connected with an edge can have different allocation. The first is executed on node1 and the second is executed on node2. Cluster environment automatically ensures remote data records transfer.



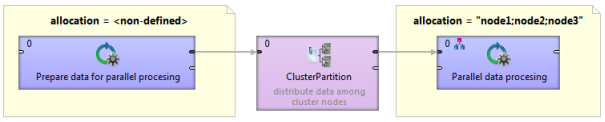

#### Multiple execution

Graph with multiple node allocation is executed in parallel. In this example both components have same allocation, so three identical transformation will be executed on cluster node1, node2 and node3.



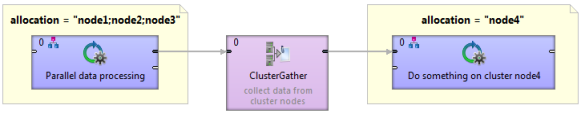

#### Cluster data partitioning

Graph with two allocations. First component has single node allocation, which is not specified and is automatically derived to ensure minimal number of remote edges. The ClusterPartition component distribute records for further data processing on cluster node1, node2 and node3.




### Cluster data gathering

Graph with two allocations. Resulted data records of parallel data processing in the first component are collected in ClusterGather component and passed to cluster node4 for further single node processing.




## Recommendations for Cluster Deployment

1. All nodes in the cluster should have a synchronized system date-time.

2. All nodes share sandboxes stored on a shared or replicated filesystem. The filesystem shared among all nodes is single point of failure. Thus, the use of a replicated filesystem is strongly recommended.

3. All nodes share a DB, thus it must support transactions. I.e. The MySQL table engine, MyISAM, may cause strange behaviour because it is not transactional.

4. All nodes share a DB, which is a single point of failure. Use of a clustered DB is strongly recommended.

5. Configure the license as "license.file" for this property on Tomcat. Do not use `clover_license.war`. Tomcat loads web-apps in an unpredictable order and for the cluster, the license must be loaded before CloverETL Server itself.

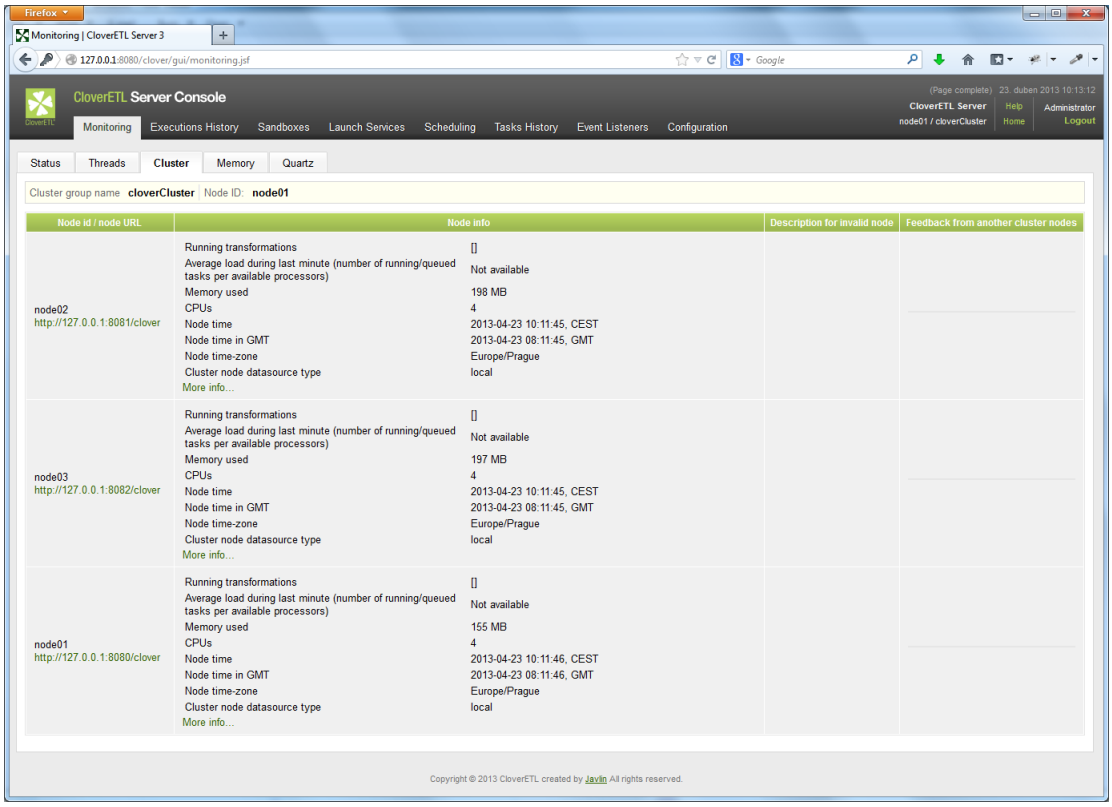
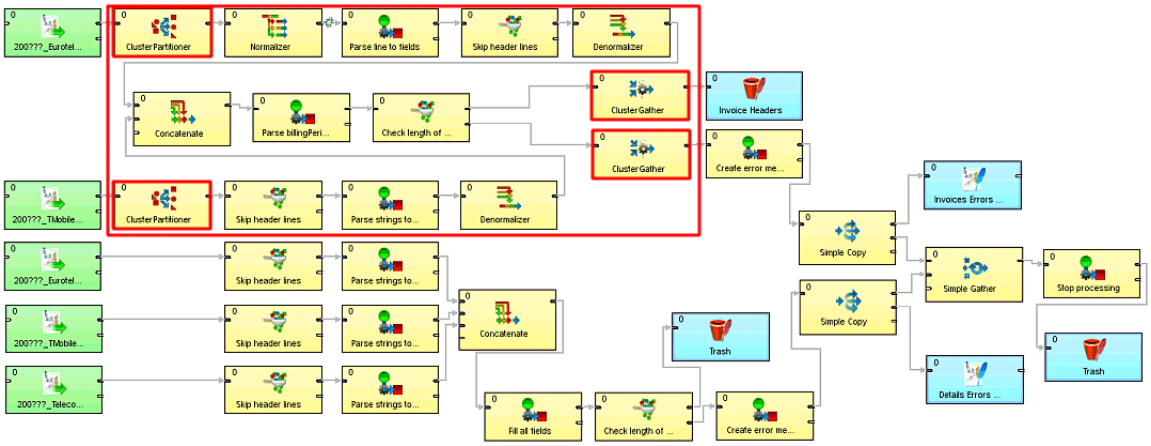



*Figure 24.6. List of nodes joined to the cluster*

# Example of Distributed Execution

The following diagram shows a transformation graph used for parsing invoices generated by a few cell phone network providers in Czech Republic.




The size of these input files may be up to a few gigabytes, so it is very beneficial to design the graph to work in the cluster environment.

## Details of the Example Transformation Design

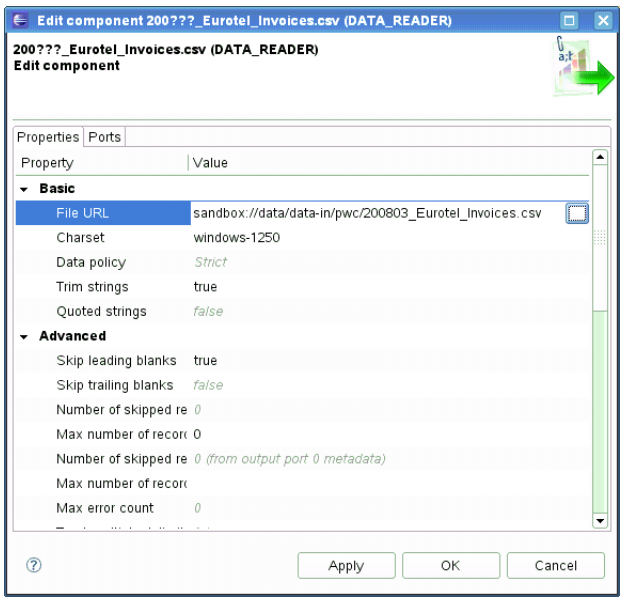
Please note there are four cluster components in the graph and these components define a point of change "node allocation", so the part of the graph demarcated by these components is highlighted by the red rectangle. Allocation of these component should be performed in parallel. This means that the components inside the dotted rectangle should have convenient allocation. The rest of the graph runs just on single node.

### Specification of "node allocation"

There are 2 node allocations used in the graph:

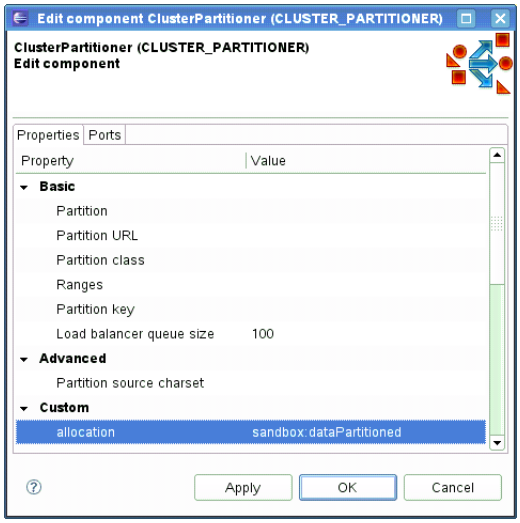
- node allocation for components running in parallel (demarcated by the four cluster components)
- node allocation for outer part of the graph which run on a single node

The single node is specified by the sandbox code used in the URLs of input data. The following dialog shows the File URL value: "sandbox://data/path-to-csv-file", where "data" is the ID of the server sandbox containing the specified file. And it is the "data" *local* sandbox which defines the single node.




The part of the graph demarcated by the four cluster components may have specified its allocation by the file URL attribute as well, but this part does not work with files at all, so there is no file URL. Thus, we will use the "node allocation" attribute. Since components may adopt the allocation from their neighbours, it is sufficient to set it only for one component.

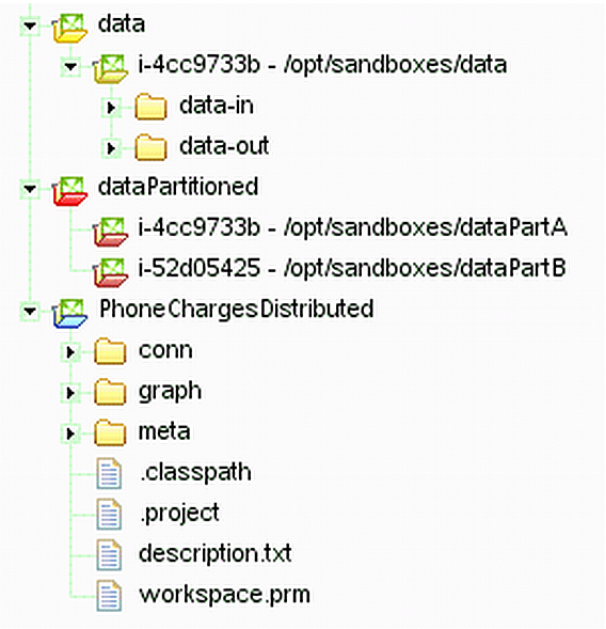
Again, "dataPartitioned" in the following dialog is the sandbox ID.




Let's investigate our sandboxes. This project requires 3 sandboxes: "data", "dataPartitioned" and "PhoneChargesDistributed".

- data
  - contains input and output data
  - local sandbox (yellow folder), so it has only one physical location
  - accessible only on node "i-4cc9733b" in the specified path
- dataPartitioned
  - partitioned sandbox (red folder), so it has a list of physical locations on different nodes

  - does not contain any data and since the graph does not read or write to this sandbox, it is used only for the definition of "nodes allocation"
  - on the following figure, allocation is configured for two cluster nodes
- PhoneChargesDistributed
  - common sandbox containing the graph file, metadata, and connections
  - shared sandbox (blue folder), so all cluster nodes have access to the same files




If the graph was executed with the sandbox configuration of the previous figure, the node allocation would be:

- components which run only on single node, will run only on the "i-4cc9733b" node according to the "data" sandbox location.
- components with allocation according to the "dataPartitioned" sandbox will run on nodes "i-4cc9733b" and "i-52d05425".

## Scalability of the Example Transformation

The example transformation has been tested in the Amazon Cloud environment with the following conditions for all executions:

- the same master node
- the same input data: 1,2 GB of input data, 27 million records
- three executions for each "node allocation"
- "node allocation" changed between every 2 executions
- all nodes has been of "c1.medium" type

We tested "node allocation" cardinality from 1 single node, all the way up to 8 nodes.

The following figure shows the functional dependence of run-time on the number of nodes in the cluster:

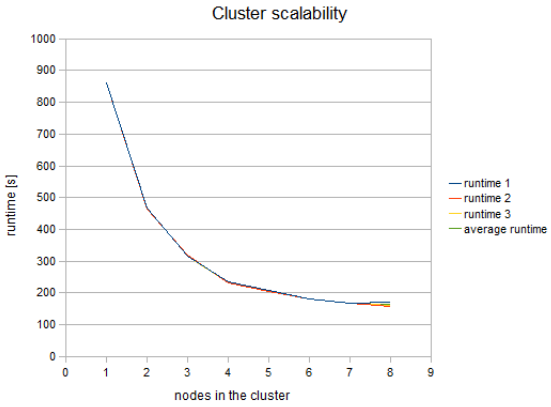



*Figure 24.7. Cluster Scalability*

The following figure shows the dependency of "speedup factor" on the number of nodes in the cluster. The speedup factor is the ratio of the average runtime with one cluster node and the average runtime with x cluster nodes. Thus:

```
speedupFactor = avgRuntime(1 node) / avgRuntime(x nodes)
```

We can see, that the results are favourable up to 4 nodes. Each additional node still improves cluster performance, however the effect of the improvement decreases. Nine or more nodes in the cluster may even have a negative effect because their benefit for performance may be lost in the overhead with the management of these nodes.

These results are specific for each transformation, there may be a transformation with much a better or possibly worse function curve.

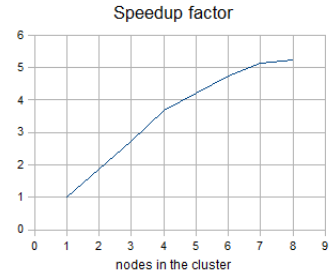



*Figure 24.8. Speedup factor*

Table of measured runtimes:

| nodes | runtime 1 [s] | runtime 2 [s] | runtime 3 [s] | average runtime [s] | speedup factor |
|---|---|---|---|---|---|
| 1 | 861 | 861 | 861 | 861 | 1 |
| 2 | 467 | 465 | 466 | 466 | 1.85 |
| 3 | 317 | 319 | 314 | 316.67 | 2.72 |
| 4 | 236 | 233 | 233 | 234 | 3.68 |
| 5 | 208 | 204 | 204 | 205.33 | 4.19 |
| 6 | 181 | 182 | 182 | 181.67 | 4.74 |
| 7 | 168 | 168 | 168 | 168 | 5.13 |
| 8 | 172 | 159 | 162 | 164.33 | 5.24 |

# Cluster configuration

Cluster can work properly only if each node is properly configured. Clustering must be enabled, nodeID must be unique on each node, all nodes must have access to shared DB (direct connection or proxied by another cluster node) and shared sandboxes, and all properties for inter-node cooperation must be set according to network environment.

Properties and possible configuration are the following:



## Mandatory properties

Besides mandatory cluster properties, you need to set license.file and other necessary properties which are not specifically related to the cluster environment. Database connection must be also configured, however besides direct connection it's alternatively possible to configure proxing using another cluster node/nodes. See property "cluster.datasource.type" for details.

*Table 24.1. Mandatory properties - these properties must be properly set on each node of the cluster*

| property | type | default |
|---|---|---|
| cluster.enabled | boolean | false |
| *description:* | switch whether server is connected to the cluster or not | |
| cluster.node.id | String | node01 |
| *description:* | each cluster node must have unique ID | |
| cluster.shared_sandboxes_path | String, path | |
| *description:* | Path, where all shared sandboxes are stored on this node. If cluster is enabled, all sandboxes containing job files (graphs, metadata, connection configs, etc.) must be shared, thus instead of separated "rootPath" specified for each sandbox, all shared sandboxes are stored in the filesystem folder specified by "cluster.shared_sandboxes_path". Path to the root directory of the sandbox is constructed like this: [shared_sandboxes_path]/[sandboxID] Also sharing must be properly configured, so the content of the directory is accesible on filesystems of all cluster nodes. | |
| cluster.jgroups.bind_address | String, IP address | 127.0.0.1 |
| *description:* | IP address of ethernet interface, which is used for communication with another cluster nodes. Necessary for inter-node messaging. | |
| cluster.jgroups.start_port | int, port | 7800 |
| *description:* | Port where jGroups server listens for inter-node messages. | |
| cluster.jgroups.tcpping.initial_hosts | String, in format: "IPaddress1[port1],IPaddress2[port2]" | 127.0.0.1[7800] |

| property | type | default |
|---|---|---|
| *description:* | List of IP addresses(with ports) where we expect running and listening nodes. It is related to another nodes "bind_address" and "start_port" properties. I.e. like this: bind_address1[start_port1],bind_address2[start_port2],... It is not necessary to list all nodes of the cluster, but at least one of listed host:port must be running. Necessary for inter-node messaging. | |
| cluster.http.url | String, URL | http://localhost:8080/clover |
| *description:* | URL of the CloverETL cluster node. It must be HTTP/HTTPS URL to the root of web application, thus typicaly it would be "http://[hostname]:[port]/clover". Primarily it's used for synchronous inter-node communication from other cluster nodes. It's recommended to use a fully qualified hostname or IP address, so it's accessible from client browser or CloverETL Designer. | |

## Optional properties

| property | type | default | description |
|---|---|---|---|
| cluster.node.sendinfo.interval | int | 5000 | time interval in ms; each node sends info about itself to another nodes; this interval specified how often the info is sent |
| cluster.node.remove.interval | int | 15000 | time interval in ms; if no node info comes in this interval, node is considered as lost and it is removed from the cluster |
| cluster.max_allowed_time_shift_between_nodes | int | 2000 | Max allowed time shift between nodes. All nodes must have system time synchronized. Otherwise cluster may not work properly. So if this threshold is exceeded, node will be set as invalid. |
| cluster.group.name | String | cloverCluster | Each cluster has its unique group name. If you need 2 clusters in the same network environment, each of them would have its own group name. |
| cluster.datasource.type | String | local | Change this property to "remote" if the node doesn't have direct connection to the CloverETL Server database, so it has to use some other cluster node as proxy to handle persistent operations. In such case, also property "cluster.datasource.delegate.nodeIds" must be properly configured. Properties jdbc.* will be ignored. Please note, that scheduler is active only on nodes with direct connection. |
| cluster.datasource.delegate.nodeIds | String | | List of cluster node IDs (separated by comma ",") which this node may use as proxy to handle persistent operations. At least one of the listed node IDs must be running, otherwise this node will fail. All listed node IDs must have direct connection to CloverETL Server database properly configured. Property "cluster.datasource.delegate.nodeIds" is ignored by default. Property "cluster.datasource.type" must be set to "remote" to enable the feature. |

## Example of 2 node cluster configuration

This section contain example of CloverETL cluster nodes configuration. In addition it is necessssary to configure:

- sharing or replication of directory /home/clover/nfs_shared/sandboxes
- connection to the same database from both nodes
- HTTP load balancer

Configuration of node on 192.168.1.131

```
jdbc.dialect=org.hibernate.dialect.MySQLDialect
datasource.type=JNDI
datasource.jndiName=java:comp/env/jdbc/clover_server

cluster.enabled=true
cluster.node.id=node01
cluster.shared_sandboxes_path=/home/clover/nfs_shared/sandboxes

license.file=/home/clover/license/license.dat

cluster.group.name=cloverCluster
cluster.jgroups.bind_address=192.168.1.131
cluster.jgroups.start_port=7800
cluster.jgroups.tcpping.initial_hosts=192.168.1.13[7800]

cluster.http.url=http://192.168.1.131:8080/clover
```

Configuration of node on 192.168.1.13

```
   # this node will use node01 as proxy for its persistent operations above shared DB
      cluster.datasource.type=remote
cluster.datasource.delegate.nodeIds=node01

   # direct connection to the shared DB isn't configured
#jdbc.dialect=org.hibernate.dialect.MySQLDialect
#datasource.type=JNDI
#datasource.jndiName=java:comp/env/jdbc/clover_server

cluster.enabled=true
cluster.node.id=node02
cluster.shared_sandboxes_path=/home/clover/nfs_shared/sandboxes

license.file=/home/clover/license/license.dat

cluster.group.name=cloverCluster
cluster.jgroups.bind_address=192.168.1.13
cluster.jgroups.start_port=7800
cluster.jgroups.tcpping.initial_hosts=192.168.1.131[7800]

cluster.http.url=http://192.168.1.13:8080/clover
```

## Load balancing properties

Multiplicators of load balancing criteria. Load balancer decides which cluster node executes graph. It means, that any node may process request for execution, but graph may be executed on the same or on different node according to current load of the nodes and according to these multiplicators.

The higher number, the higher relevance for decision. All multiplicators must be greater then 0.

Each node of the cluster may have different load balancing properties. Any node may process incomming requests for transformation execution and each may apply criteria for loadbalancing in a different way according to its own configuration.

These properties aren't vital for cluster configuration - default values are sufficient

*Table 24.3. Load balancing properties*

| property | type | default | description |
|---|---|---|---|
| cluster.lb.balance.running_graphs | float | 3 | Specify importance of running graphs for load balancing. |
| cluster.lb.balance.memused | float | 0.5 | Specify importance of used memmory for load balancing. |
| cluster.lb.balance.cpus | float | 1.5 | Specify importance of number of CPUs for load balancing. |
| cluster.lb.balance.request_bonus | float | 2 | Specify importance of the fact, that the node is the same which processes request for execution. The same node, which decides where to execute graph. If you specify this multiplicator great enough, it will cause, that graph will be always executed on the same node, which processes request for execution. |
| cluster.lb.balance.node_bonus | float | 1 | Overall ratio bonus for configured node. Values greater then "1" increase probability the node will be choosen by the loadbalancer. Value "1" means no bonus or penalty. "0" means that the node will be never choosen by the loadbalancer, however it still may execute graphs, e.g. when there is no other node in cluster, or when the graph is designed to run on the node. |

# Chapter 25. Temp Space Management

Many of the components available in the CloverETL Server require temporary files or directories in order to work correctly. *Temp space* is a physical location on the local file system where these files or directories are created and maintained.

## Overview

The overview of temp spaces defined in CloverETL Server is available under *Configuration* > *Temp space management* > *Overview*

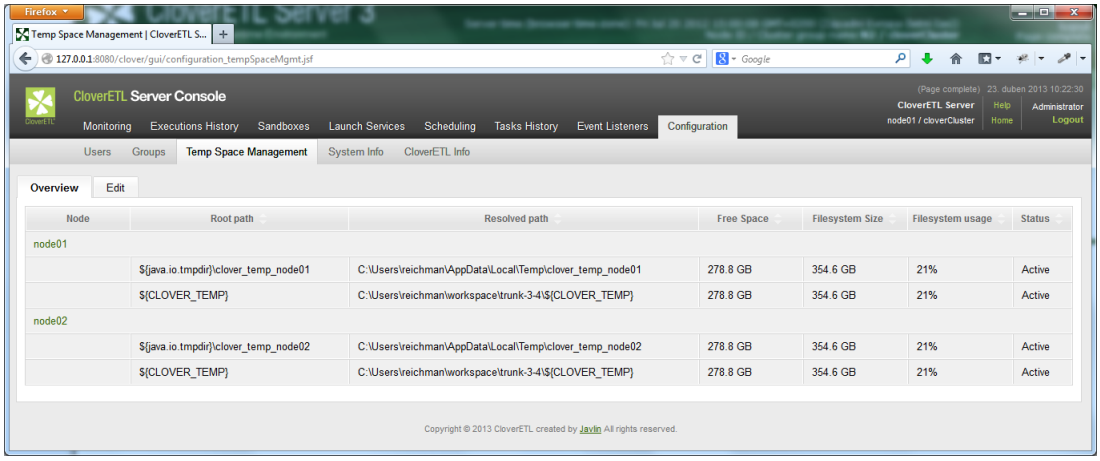



*Figure 25.1. Configured temp spaces overview - one default temp space on each cluster node*

## Setup

Temp space management offers an iterface to add, suspend, resume and delete a temp space. It is accessible under *Configuration* > *Temp space management* > *Edit.*

The screen is divided in two drop-down areas: Global Configuration and Per Node Configuration. The *Global configuration* manages temp spaces of standalone server or in case of a server cluster temp spaces on all its nodes. The *Per Node Configuration* allows to maintain temp spaces separately on each node.

### Initialization

When CloverETL Server is started the system checks temp space configuration: in case no temp space is configured a new default temp space is created in the directory where `java.io.tmpdir` system property points. The directory are named as follows:

- `${java.io.tmpdir}/clover_temp` in case of a standalone server
- `${java.io.tmpdir}/clover_temp_<node_id>` in case of server cluster

## Adding Temp Space

In order to define new temp space enter its path into text field under lastrow in the table and click the "Add" link. Note that environment variables and system properties may be used in the path, e.g.: `${VARIABLE_NAME}/temp_space` If the directory entered does not exist, it will be created.

### Note

The environment variables have higher priority than system properties of the same name. The path with varibles are resolved after system has added new temp space and while the server is starting. In case the variable value has been changed it is necessary to restart the server to such change take effect.

## Tip

The main point of adding additional temp spaces is to enable higher system throughtput - therefore the paths entered should point to directories residing on different physical devices to achieve maximal I/O performance.

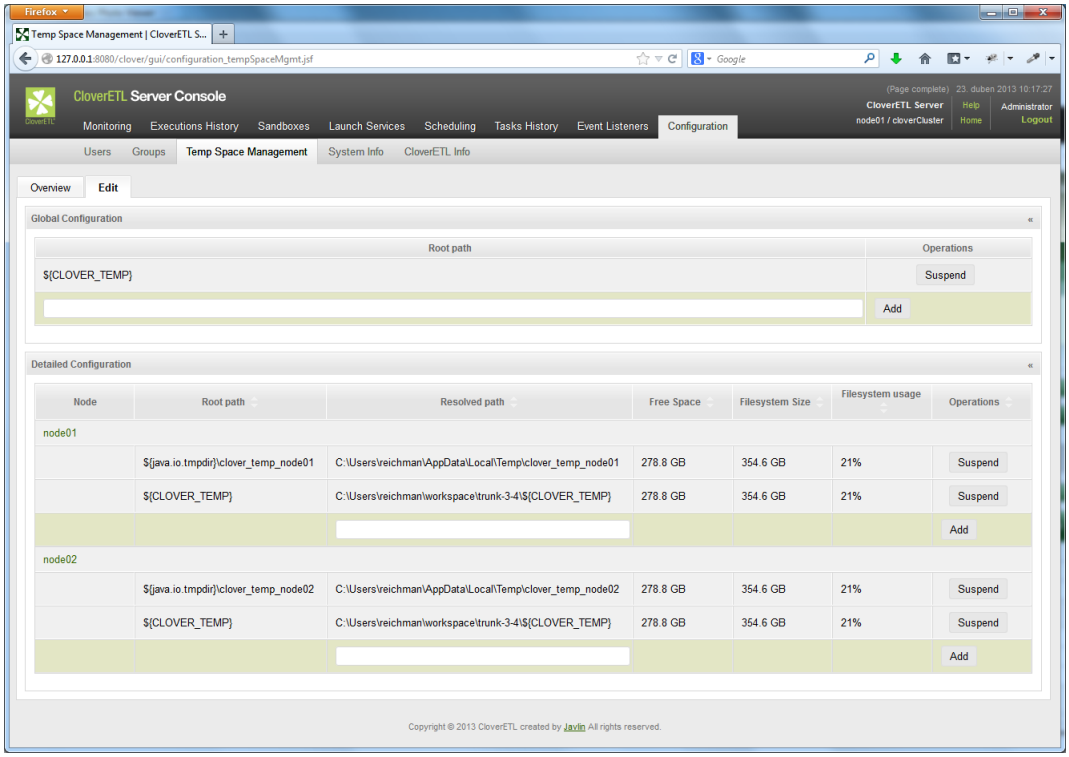



*Figure 25.2. Newly added global temp space using environment property set on both nodes.*

### Suspending Temp Space

To suspend a temp space click on "Suspend" link in the panel. In case there are files left from previous or current graph executions a notification is displayed. Once the temp space has been suspended, no new temporary files will be created in it, but the files already created may be still used by running jobs.

## Note

The system ensures that at least one active (i.e. not suspended) temp space is available.

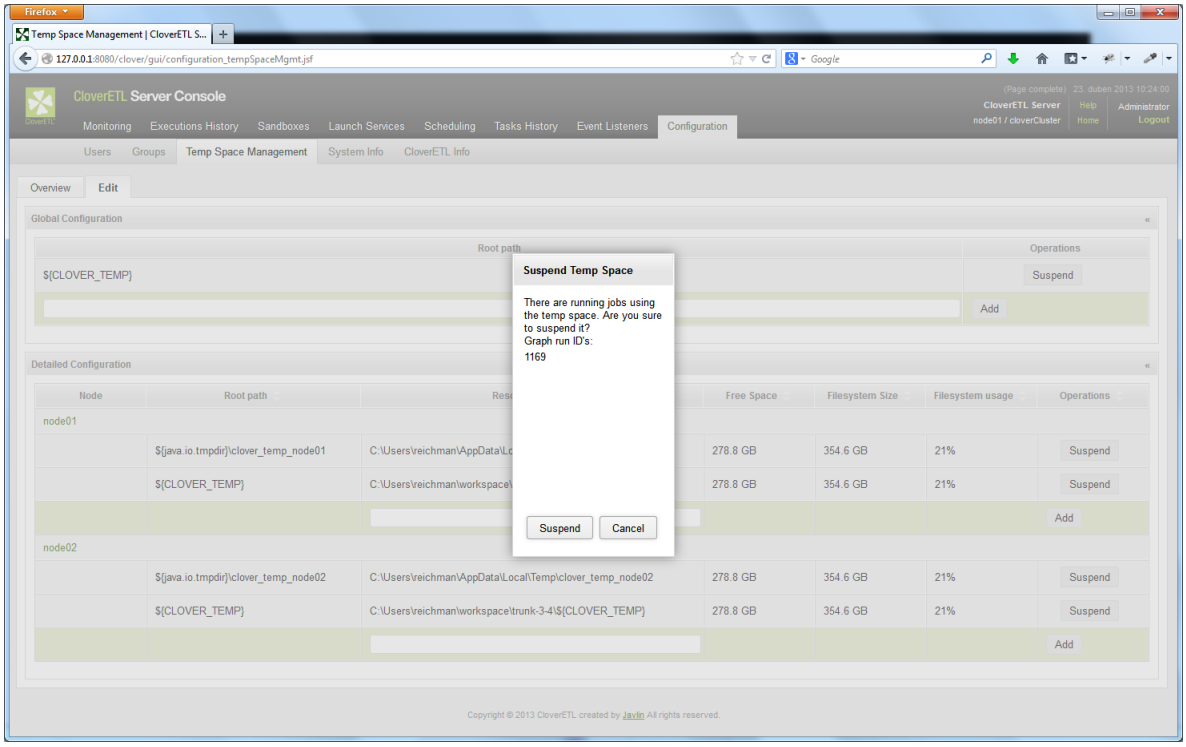



*Figure 25.3. Suspend operation asks for confirmation in case there are data present from running jobs.*

### Resuming Temp Space

To resume a temp space click on "Resume" link in the panel. Resumed temp space is active, i.e. availabale for temporal files and directories creation.

### Removing Temp Space

To remove a temp space click on "Remove" link in the panel. Only suspended temp space may be removed. Should be there any running jobs using the temp space, system will not allow its removal. In case there are some files left in the the temp space directory, it is possible to remove them in the displayed notification panel. The available options are:

- *Remove* - remove temp space from system, but keep its content
- *Remove and delete* - remove the temp space from system and its content too
- *Cancel* - do not proceed with operation

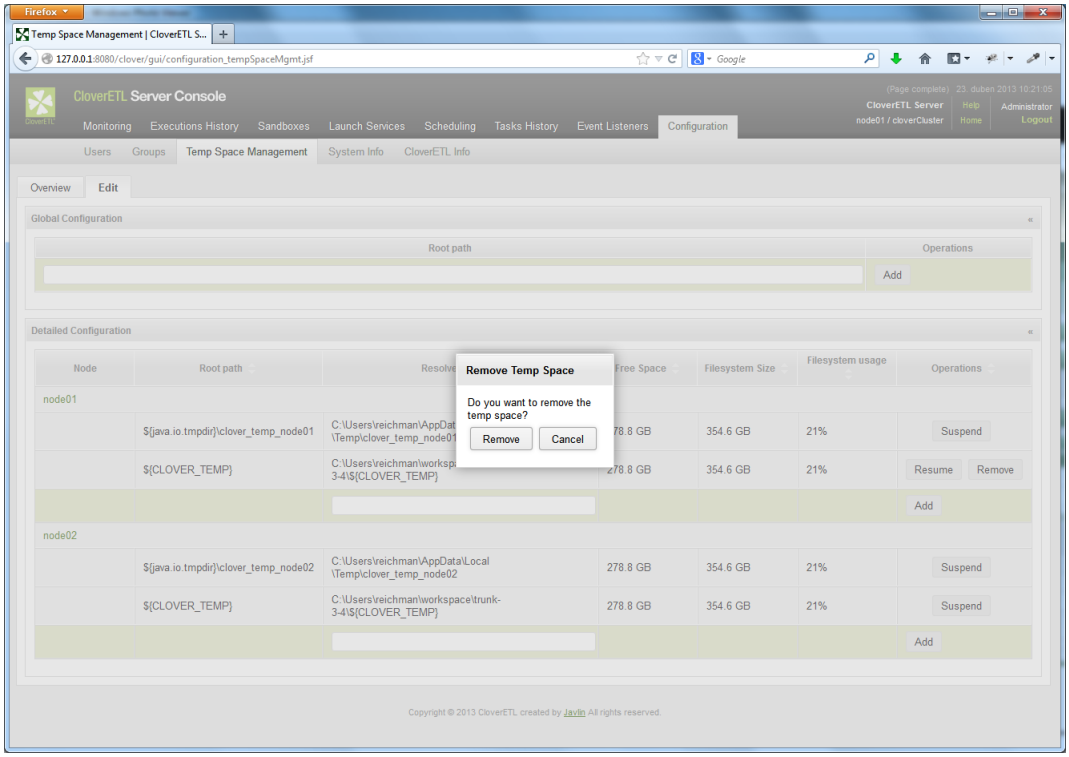



*Figure 25.4. Remove operation asks for confirmation in case there are data present in the temp space.*

# List of Figures

# List of Tables