

Installation and Administrators Guide of the openBIS and ETL Server 8.04.1



Date: 27.5.08

Author: CISD, <http://www.cisd.systemsx.ethz.ch/>

This manual is addressed to the administrators of openBIS and ETL Server. It is not a documentation for user, they should consult the available user manual.

Table of Contents

Installation and Administrators Guide of the openBIS and ETL Server 8.04.1.....	1
License.....	4
Software License.....	4
Get the software.....	5
openBIS 8.04.....	5
System Requirements.....	5
Installation.....	5
Installation steps.....	7
Update.....	8
Replace the SSL certificate.....	8
I don't have an existing certificate and want to create a new one.....	8
I already have a certificate and want to import it into the tomcat keystore.....	8
Update Tomcat configuration with a new keystore.....	9
Update Client keystore with your Certificate.....	9
Start Server.....	10
Stop Server.....	10
Authentication systems.....	10
The file based authentication system.....	10
The authentication interface to Crowd.....	11
ETL Server.....	12
Introduction.....	12
Functionality.....	12
How data sets get identified in the system.....	12
How data sets get linked to each other.....	12
Data Flow.....	13
File and Directory Naming Convention.....	14
Examples.....	14
Successful ETL Thread.....	14
Installation and Administration.....	15
System Requirements.....	15
Installation.....	15
Configuration.....	15
Structure of service.properties.....	17
Common configuration parameters.....	17
ETL thread configuration parameters.....	18
Data Set Information Extractor.....	18
ch.systemsx.cisd.etlserver.DefaultDataSetInfoExtractor.....	18
ch.systemsx.cisd.etlserver.DataSetInfoExtractorForPlatesOnDemand.....	19
ch.systemsx.cisd.etlserver.threev.DataSetInfoExtractorForDataAcquisition.....	19
ch.systemsx.cisd.etlserver.threev.DataSetInfoExtractorForImageAnalysis.....	20
Type Extractor.....	20
Storage Processor.....	21
Installation steps.....	21
Start Server.....	21
Stop Server.....	22
Data Set Download Server.....	23

Introduction.....	23
Installation.....	23
Configuration.....	23
Service Properties.....	23
Logging Definitions.....	24

License

Software License

Copyright 2008 ETH Zuerich, CISD

Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at

<http://www.apache.org/licenses/LICENSE-2.0>

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

Get the software

Our software is open source and developed under the Apache 2.0 License. You can get the software from <http://www.cisd.systemsx.ethz.ch/software/openBIS>.

openBIS 8.04

System Requirements

The requirements of the system on which the server should run are the following

- OS: Linux / MacOS X with
 - bash
 - unzip
 - awk
 - sed
- Java VM: JRE 1.5.0 or higher
- PostgreSQL 8.2 or higher
- SMTP server configured if you want to obtain notifications (note that you can also change the configuration to use another SMTP server if you don't have one configured on the server)

Because [Crowd](#) is used as authentication service the IP of the system and the openBIS application has to be registered at the Crowd server. On the other hand the openBIS server should be allowed to access Crowd via HTTPS (host: `source.systemsx.ch`, port: 8443)

Installation

The server distribution is a zip file named `openBIS-server-<version>-r<revision>.zip`. It contains:

- `openBIS-server/apache-tomcat.zip`: Tomcat 5.5 distribution.
- `openBIS-server/tomcat-version.txt`: Complete version number of the Tomcat distribution (needed by the `install.sh` script).
- `openBIS-server/server.xml`: Tomcat configuration file. The openBIS server uses the HTTPS connector at port 8443.
- `openBIS-server/openBIS.war`: openBIS Web Application.
- `openBIS-server/roles.conf`: Default role definitions.
- `openBIS-server/server.keystore`: Keystore with key for SSL connection between openBIS server and client.
- `openBIS-server/install.sh`: Installation script.
- `openBIS-server/passwd.sh`: To maintain the file used in the file based authentication system.
- `openBIS-server/service.properties`: Default openBIS configuration file.

In addition the properties file `service.properties` should be created from the following template:

service.properties

```
#####
# Template of service.properties for openBIS server
#####
# Supported: 'file-authentication-service' and 'crowd-authentication-service'
authentication-service = file-authentication-service

script-folder = .

# Supported: currently only 'postgresql' is supported
database.engine = postgresql
database.create-from-scratch = false
database.script-single-step-mode = false
# Leave empty to get the default for the db engine
database.url-host-part =
database.kind = demo
# Credentials of the database user which should own the database. Leave empty to use the OS user
database.owner =
database.owner-password =
# Credentials of a database user which is able to create a new database. Leave empty to use the
db engines default
database.admin-user =
database.admin-password =

# The URL that the Crowd service can be found at
crowd.service.url = some.url
# The port that the Crowd service can be found at
crowd.service.port = 443
# Name of the application when logging in into Crowd
crowd.application.name = openbis
# Password for which the openBIS service is registered at Crowd
crowd.application.password = <application password>

# SMTP properties
# Default values are 'localhost' for 'mail.smtp.host' and 'openbis@localhost' for 'mail.from'
mail.smtp.host = localhost
mail.from = openbis@localhost
mail.smtp.user =
mail.smtp.password =

# Properties for ProcessingPathValidator:
# path prefixes for each procedure type
processing-instruction.DATA_ACQUISITION.prefix-for-absolute-paths =
processing-instruction.DATA_ACQUISITION.prefix-for-relative-paths =
processing-instruction.IMAGE_ANALYSIS.prefix-for-absolute-paths =
processing-instruction.IMAGE_ANALYSIS.prefix-for-relative-paths =

# Group identifier (mandatory).
group-code = DEFAULT

# Workflow code name (mandatory; available values: strict, plates-on-demand)
workflow-code = strict

# The time after which an inactive session is expired by the service (in minutes).
session-timeout = 30

# Base URL of the data set download service (not used when empty)
dataset-download-service-base-url =
```

Notes:

- <application password>: Ask the administrator of the Crowd service.
- The credentials for the database user with the privilege to create a new database depends on the installation and configuration of the PostgreSQL database.
- Value specified as database-instance replaces the code (if and only if this one has the value CHANGEME) of the unique database instance found which has its flag is_original_source set to TRUE.

The path prefixes have to be the same as for the ETL Server.

For adaption of the roles, look at the file `webapps/openBIS/WEB-INF/classes/roles.conf`. A typical role configuration file looks like

roles.conf

```
[DefaultRoles]
observer

[Roles]
etlserver = etlserver
admin = userA, userC
user = userB
observer = someVisitorUser
```

The section `[DefaultRoles]` defines the role(s) that a user should be assigned that is not listed explicitly in the `[Roles]` section. It contains one role in one line with no additions.

The section `[Roles]` has entries `<role> = <userlist>` which gives the users on the right hand side the role on the left hand side. A user can be in multiple user lists in which case he or she gets multiple roles.

You do not have to restart the *openBIS* server if you change the roles configuration file. After a certain amount of time (about 1mn), the server will notice the file timestamp change and will reload the roles configuration.

Installation steps

1. Unzip the distribution on the server machine into some temporary folder.
2. Copy an appropriated `service.properties` file into the same folder.
3. Change to the temporary folder.
4. Run the installation script as follows:

```
prompt> openBIS-server/install.sh [--nostartup] <server folder> [<service
properties file>] [<role config file>]
```

where:

- `<server folder>` needs to be specified as an absolute path.
- `--nostartup` if specified, then `_Tomcat_` is not automatically started after the installation.
- `<service properties file>` specifies a file containing key-value pairs as service properties (usually called `service.properties`). If not specified, the installation script loads the default one located in the installation folder.
- `[<role config file>]` specifies a file containing role definitions (usually `roles.conf`). If not specified, the installation script loads the default one located in the installation folder.

It does the following:

1. Unzips Tomcat (i.e. `apache-tomcat.zip`) into `<server folder>`.
2. Copies keystores to `<server folder>/apache-tomcat`.
3. Copies `server.xml` to `<server folder>/apache-tomcat/config`.
4. Modifies `<server folder>/apache-tomcat/bin/startup.sh`.
5. Unzips openBIS Web application (i.e. `openBIS.war`) into `<server folder>/apache-tomcat/webapps`.
6. Copies `<service properties file>` to `<server folder>/apache-tomcat/webapps/openBIS/WEB-INF/classes/service.properties`.

7. Copies <role config file> to <server folder>/apache-tomcat/webapps/openBIS/WEB-INF/classes/roles.conf.
8. Creates symbolic links to service.properties and roles.conf in <server folder>/apache-tomcat/etc, facilitating the access to them.
9. Finally starts Tomcat (if --nostartup is not specified).

On startup the openBIS service checks the connection with Crowd (if {{crowd-authentication-service}} is specified as {{authentication-service}} value) and creates the database on PostgreSQL. If something went wrong the administrator of the server machine will get an e-mail.

Update

In order to update an openBIS server do the following:

1. Stop the currently running server as described in section „Installation and Administrators Guide of the openBIS Server“.
2. Update accordingly one or both of the key files (service.properties resp. roles.conf) located at <server folder>/apache-tomcat/etc.
3. (Re)start the server as described in section „Installation and Administrators Guide of the openBIS Server“.

Replace the SSL certificate

You can change the existing certificate which comes with the installation with an own one. There are two ways doing this, depends if you already have a certificate for the domain or not:

I don't have an existing certificate and want to create a new one

Installation Guide from [Tomcat Homepage](#)

To create a new keystore from scratch, containing a single self-signed certificate, execute the following from a terminal command line:

```
%JAVA_HOME%\bin\keytool -genkey -alias tomcat -keyalg RSA \ -keystore /path/to/my/keystore
(Windows)
$JAVA_HOME/bin/keytool -genkey -alias tomcat -keyalg RSA \ -keystore /path/to/my/keystore (Unix)
```

After executing the keytool command, you will first be prompted for the keystore password. The default password used by Tomcat is "changeit" (all lower case), although you can specify a custom password if you like. Again, this will need to be reflected in the server.xml configuration file.

Answer all questions from the tool and fill it out to you needs.

Finally, you will be prompted for the key password, which is the password specifically for this Certificate (as opposed to any other Certificates stored in the same keystore file). You MUST use the same password here as was used for the keystore password itself. (Currently, pressing the ENTER at this prompt will automatically do this.)

The Keystore is generated in the path you specified (-keystore /path/to/my/keystore). If you didn't specify the path, the keystore is generated in the home directory of the user.

I already have a certificate and want to import it into the tomcat keystore

Installation Guide from [Tomcat Homepage](#)

It is possible to import certificates generated with OpenSSL.

To import the CERT in keystore, you just do next :

```
keytool -import -v -trustcacerts -alias tomcat -file CERT.pem -keystore /path/to/my/keystore
```

For more information, please read the documentation (in your JDK documentation package) about keytool.

Update Tomcat configuration with a new keystore

To change the keystore which is used you have to edit the `server.xml` file which is located in the `conf` directory of the openBIS installation. In this file is a section which looks like this:

```
<!-- Define a SSL HTTP/1.1 Connector on port 8443 -->
<Connector port="8443" maxHttpHeaderSize="8192"
  maxThreads="150" minSpareThreads="25" maxSpareThreads="75"
  keystoreFile="server.keystore"
  enableLookups="false" disableUploadTimeout="true"
  acceptCount="100" scheme="https" secure="true"
  clientAuth="false" sslProtocol="TLS" />
```

To use your generated keystore, specify the parameter `keystoreFile` with the path to your keystore, or just overwrite the existing keystore file.

If you specified a password for your keystore you also need to specify the password in the `server.xml` file with the parameter `keypass="mynewpass"`.

If you have a password for your keystore, please make sure that your certificate has the same password. Otherwise it will not work!

Update Client keystore with your Certificate

If you change the server with a self signed certificate you also have to update the keystore of the openBIS client, cause the client talks encrypted to the server and needs to know about the certificate.

```
First you need to export the certificate from the server (with keytool -list -v -keystore /path/to/my/keystore you get a list of all certificates in the keystore):
keytool -export -alias <CERT_ALIAS> -file /path/to/export/cert.crt -keystore /path/to/my/keystore
```

In the openBIS client, you need to import the cert, you exported before from the server.

```
keytool -import -alias <CERT_ALIAS> -file /path/to/export/cert.crt -keystore openBIS.keystore
```

For security reasons we recommend to delete (or move) the existing client keystore, a new keystore is automatically generated.

To check if certificate was properly imported, list its content:

```
keytool -list -v -keystore openBIS.keystore
```

If you change the server with a certificate which is signed by a trusted certificate authority (which is normally known from the Java), you have to update the openBIS client, not to use the keystore anymore. This can be done by removing the option `-Djavax.net.ssl.trustStore=etc/openBIS.keystore` from `bis.sh` (or `bis.bat`) which is located in the client distribution. If you do not remove this option, the authentication does not work.

If you are using the download server and change there the certificate, you should also import the certificate to the client (the same way it is done for the openBIS Server).

Unfortunately it is not possible to run one server with a self signed certificate and the other one with a trusted. So either both are self signed and you update your keystore on the client side, or both are trusted and you remove the option from `bis.sh` (or `bis.bat`).

Start Server

The server is started as follows:

```
prompt> cd <server folder>/apache-tomcat
prompt> bin/startup.sh
```

Stop Server

The server is stopped as follows:

```
prompt> cd <server folder>/apache-tomcat
prompt> bin/shutdown.sh
```

Authentication systems

openBIS currently supports two authentication systems: a self-contained system based on a UNIX-like passwd file and a system based on Crowd (see <http://www.atlassian.com/software/crowd>), an authentication middle ware sold by Atlassian.

The file based authentication system

By default, authentication-service = file-authentication-service is set. This authentication schema uses the file <server folder>/apache-tomcat/etc/passwd to determine whether an authentication to the system is successful or not.

The script {{<server folder>/apache-tomcat/bin/passwd.sh} can be used to maintain this file. This script supports the options:

```
passwd list | [remove|show|test] <user> | [add|change] <user> [option [...]]
--help                : Prints out a description of the options.
[-P,--change-password] : Read the new password from the console,
[-e,--email] VAL       : Email address of the user.
[-f,--first-name] VAL  : First name of the user.
[-l,--last-name] VAL   : Last name of the user.
[-p,--password] VAL    : The password.
```

A new user can be added with

```
prompt> passwd.sh add [-f <first name>] [-l <last name>] [-e <email>] [-p <password>] <username>
```

If no password is provided with the `-p` option, the system will ask for a password of the new user on the console. Please note that providing a password on the command line can be a security risk, because the password can be found in the shell history, and, for a short time, in the `ps` table. Thus `-p` is not recommended in normal operation.

The password of a user can be tested with

```
prompt> passwd.sh test <username>
```

The system will ask for the current password on the console and then print whether the user was authenticated successfully or not.

An account can be changed with

```
prompt> passwd.sh change [-f <first name>] [-l <last name>] [-e <email>] [-P] <username>
```

An account can be removed with

```
prompt> passwd.sh remove <username>
```

The details of an account can be queried with

```
prompt> passwd.sh show <username>
```

All accounts can be listed with

```
prompt> passwd.sh list
```

The authentication interface to Crowd

When setting `authentication-service = crowd-authentication-service` in `service.properties`, the `passwd` file has no effect. Instead, the following properties need to be configured via the following properties.

The URL (without port information):

```
crowd.service.url = https://crowd.your.org
```

The Port of the URL:

```
crowd.service.port = 443
```

The name of the application account in Crowd:

```
crowd.application.name = openbis
```

The password of the application account in Crowd:

```
crowd.application.password = <application password>
```

ETL Server

Introduction

ETL stands for *Extract*, *Transform* and *Load*. ETL is a process in data management that involves

- extracting data from outside sources,
- transforming it to fit business needs, and ultimately
- loading it into the data base.

See also http://en.wikipedia.org/wiki/Extract,_transform,_load for further information.

Functionality

In an ETL server several independent ETL threads run in parallel. Each thread

1. takes file-based raw data from a central storage,
2. checks with the openBIS server what experiment the raw data belong to and
3. makes sure the raw data are getting associated as external tables with the database of the openBIS application.

If any of the above steps fail, the ETL thread produces logs and assures that the raw data don't get lost by either moving them to a directory called `invalid`, `error`, or `unidentified`. If considered `invalid`, an notification email is sent to the user who has registered the experiment the data belong to.

How data sets get identified in the system

There are 2 ways to link data sets to unique identifiers ('data set codes').

1. The openBIS server generates the data set code once the data set gets registered. The openBIS server will create the code if the Data Set Info Extractor does not set the data set code in the `DataSetInformation` and will ensure that the data set codes are unique. This is the recommended way to identify data sets.
1. The Data Set Info Extractor sets the data set code in the `DataSetInformation`. In this case the creator of the codes needs to ensure that the codes are unique.

How data sets get linked to each other

Data sets can be in a parent-child relationship which means that the child data set is produced by processing the parent data set. In order to allow tracking this relationship information, an Data Set Info Extractor can specify a parent data set code in the `DataSetInformation`.

Placeholders

If the parent data set identified by the parent data set code is not yet known to the openBIS server (which can happen if the child data set arrives before the parent data set), a placeholder entry will be created for the parent data set.

Data Flow

An ETL thread requires 2 directories, to be specified in the `service.properties` file, see also [Configuration](#) section:

1. The `<incoming-dir>` is a directory on a central file system to watch for incoming data. Each thread has different folder.
2. The `<storeroot-dir>` is the root directory of the data store. It is the same for all threads.

Note that both directories must reside in the same file system.

The ETL thread creates the following directories:

1. `<storeroot-dir>/Instance_<instance code>/Group_<group code>/Project_<project code>/Experiment_<experiment code>/ObservableType_<observable type code>/Barcode_<barcode>` is the target directory of all data sets belonging to a certain sample (specified barcode), experiment, project, and group. The instance is defined in the configuration file (`service.properties`) of the ETL server. For each of these data sets a numbered subdirectory is created. The first one is 1, the second 2, etc.
2. `<storeroot-dir>/invalid` is the target directory for all data which can be associated with an experiment, but come from a plate with an invalid bar code. This can happen only in the plates-on-demand workflow.
3. `<storeroot-dir>/unidentified` is the target directory for all data which can not be associated with an experiment.
4. `<storeroot-dir>/error` is the target directory for all data which causing an error when been extracted and/or transformed.

The workflow is as follows:

1. The *Data Mover* moves data sets (files or directories) from the microscope acquisition PC to `<incoming-dir>` of one of the ETL threads (see documentation on the [Data Mover](#)).
2. An ETL thread checks `<incoming-dir>` every `check-interval` seconds for marker files associated with these data sets. A marker file is an empty with the name `.MARKER_is_finished_<data set file name>`.
3. `DataSetInformation` is extracted from the incoming data set by using one of the various configurable Data Set Info Extractors. Currently all extractors only need the data set file name.
4. Next the *openBIS* server is asked for the experiment to which this data set can be associated. This step depends on the kind of workflow:
 - *strict* workflow: Only sample code (i.e. bar code) is needed. It has to denote a sample registered for an experiment.
 - *plates-on-demand* workflow: In addition project code and experiment code has to be specified. In accordance to some rule the code of the master plate can be derived from the sample code.
5. If the data set
 - can be associated with a registered experiment and bar code, it is moved from `<incoming-dir>` to `<storeroot-dir>` and registered with at the *openBIS* server.
 - can be associated neither with a registered experiment nor with a registered bar code, it is moved from `<incoming-dir>` to `<storeroot-dir>/unidentified`.

- No registration with the openBIS service takes place.
 - can be associated with a registered experiment, but refers to a bar code not associated with this particular experiment, it is moved from `<incoming-dir>` to `<storeroot-dir>/invalid`. No registration with the openBIS service takes place. An email notification is sent to the person who registered the experiment the data set could be associated with.
 - can be associated with a registered experiment, but refers to an invalid bar code, a file `.faulty-paths` is created in `<incoming-dir>` listing the file name of the data set. No registration with the openBIS service takes place.
 - can be associated with a registered experiment and bar code, but couldn't be extracted and/or transformed, new entry is moved to `<storeroot-dir>/error/ObservableType_<observable type code>`. In addition a file `<new entry>.exception` is created in this directory. No registration with the openBIS service takes place.
6. If transformation/extraction and registration are performed successfully the ETL thread is asking *openBIS server* for processing instruction. If there is a processing instruction for the registered experiment and the procedure type the ETL thread does the following:
- Copies the data set to the directory specified by the path attribute processing instruction. If possible hard links are created for file. Only the directory structure is copied.
 - Writes the processing parameters (if any) to a file in that directory.
 - Optional a marker file is created.

Data Set Codes and Data Provenence

The ETL server will ensure that the data set code is used as a prefix for the processing directory. The Data Set Info Extractor run by the ETL server thread that deals with the incoming processed data will use this data set code later on to identify the parent data set of the incoming data set.

File and Directory Naming Convention

The following Data Set Info Extractors

- `ch.systemsx.cisd.etlserver.DefaultDataSetInfoExtractor`
- `ch.systemsx.cisd.etlserver.DataSetInfoExtractorForPlatesOnDemand`
- `ch.systemsx.cisd.etlserver.threev.DataSetInfoExtractorForDataAcquisition`
- `ch.systemsx.cisd.etlserver.threev.DataSetInfoExtractorForImageAnalysis` assume that the name of a data set file or directory contains all necessary data set information. The name has to be a concatenation of entities with an entity separator specified in the `service.properties` file.

Examples

Successful ETL Thread

We assume the *Data Mover* has moved the following data to `PATH/incoming`:

```
PATH/incoming/EXP4::CP001-1AI> ls
A10.tif A11.tif A12.tif
```

When the process has been completed, the *Data Mover* creates the file:

```
PATH/incoming> ls -a
. . . .MARKER_is_finished_EXP4::CP001-1AI
```

The *ETL server* is searching in PATH/incoming for the file .MARKER_is_finished_EXP4::CP001-1AI. After less than check-interval seconds, the *ETL server* will find the file .MARKER_is_finished_EXP4::CP001-1AI and will start to recursively move the directory EXP4::CP001-1AI and all files underneath to

```
PATH/storerooot/Group_MY-
GROUP/Project_NEW/Experiment_EXP4/ObservableType_IMAGE/Barcode_CP001-1AI/1/1EXP4::CP001-1AI> ls
A10.tif A11.tif A12.tif
```

In addition the *ETL server* will register all the data sets with the openBIS service call the method register-data-set.

Installation and Administration

System Requirements

The requirements of the system on which the server should run identical to [openBIS Server](#) system requirements.

Installation

The ETL server distribution is a zip file named etl-server-<version>-r<revision>.zip. It contains:

- etlserver/etlserver.sh: Script to start the server in a Unix shell.
- etlserver/etlserver.bat: Script to start the server in a Windows Command shell.
- etlserver/lib: Folder containing all necessary Java libraries to run the server.
- etlserver/data/store: Default root directory of the data to be stored.
- etlserver/data/incoming: Default directory of incoming data
- etlserver/etc/openBIS.keystore: Key store needed for an HTTPS connection with openBIS WebService.
- etlserver/etc/log.xml: Log4J logging properties.
- etlserver/etc/service.properties: ETL Server configuration file.

Configuration

The ETL Server is completely configured by the configuration file service.properties. Here is an example:

service.properties

```
# Code of openBIS instance
instance-code = Lab42

# The root directory of the data store
storerooot-dir = targets/store

# The check interval (in seconds)
check-interval = 5
```

```

# The time-out for clean up work in the shutdown sequence (in seconds).
# Note that that the maximal time for the shutdown sequence to complete can be as large
# as twice this time.
shutdown-timeout = 30

# If free disk space goes below value defined here, a notification email will be sent.
# Value must be specified in kilobytes (1048576 = 1024 * 1024 = 1GB). If no high water mark is
# specified or if value is negative, the system will not be watching.
highwater-mark = 1048576

# If a data set is successfully registered it sends out an email to the registrator.
# If this property is not specified, no email is sent to the registrator. This property
# does not affect the mails which are sent, when the data set could not be registered.
notify-successful-registration = false

# The URL of the openBIS server
server-url = http://localhost:8080/openbis

# The username to use when contacting the openBIS server
username = etlserver

# The password to use when contacting the openBIS server
password = doesnotmatter

# SMTP properties (must start with 'mail' to be considered).
# mail.smtp.host = localhost
# mail.from = etlserver@localhost

# Maximum number of retries if renaming failed.
# renaming.failure.max-retries = 12

# The number of milliseconds to wait before retrying to execute the renaming process.
# renaming.failure.millis-to-sleep = 5000

# Globally used separator character which separates entities in a data set file name
data-set-file-name-entity-separator = _

# Prefixes for processing paths for all procedure types.
# default-prefix-for-absolute-paths is the key for paths starting with '/'.
# default-prefix-for-relative-paths is the key for paths not starting with '/'.
#
default-prefix-for-absolute-paths =

# Processors of processing instructions.
#
# processors: comma separated list of procedure type codes
# processor.<procedure type code>.prefix-for-absolute-paths: Key for a processing path starting
# with '/'.
# processor.<procedure type code>.prefix-for-relative-paths: Key for a processing path not
# starting with '/'.
# processor.<procedure type code>.parameters-file: Name of the file containing the processing
# parameters.
# processor.<procedure type code>.finished-file-template: Name of the marker file which finishes
# processing.

processors = DATA_ACQUISITION
processor.DATA_ACQUISITION.prefix-for-absolute-paths = ${default-prefix-for-absolute-paths}
processor.DATA_ACQUISITION.prefix-for-relative-paths = targets/processing
processor.DATA_ACQUISITION.parameters-file = parameters
processor.DATA_ACQUISITION.data-set-code-prefix-glue = ${data-set-file-name-entity-separator}
processor.DATA_ACQUISITION.finished-file-template = .MARKER_is_finished_{0}
processor.DATA_ACQUISITION.input-storage-format = BDS_DIRECTORY
# time after which the copy of a single file for processing should complete.
# If that will not happen, operation will be terminated and relaunched.
#processor.DATA_ACQUISITION.data-copy-timeout = 2

# Comma separated names of processing threads. Each thread should have configuration properties
# prefixed with its name.
# E.g. 'code-extractor' property for the thread 'my-etl' should be specified as 'my-etl.code-
# extractor'
inputs=main-thread

# -----
# 'main-thread' thread configuration
# -----

```



```
# The directory to watch for incoming data.
main-thread.incoming-dir = targets/incoming

# The store format that should be applied in the incoming directory.
main-thread.incoming-dir.format =

# ----- Plugin properties

# The extractor plugin class to use for code extraction
main-thread.data-set-info-extractor = ch.systemsx.cisd.etlserver.DefaultDataSetInfoExtractor
# Separator used to extract the barcode in the data set file name
main-thread.data-set-info-extractor.entity-separator = ${data-set-file-name-entity-separator}

main-thread.type-extractor = ch.systemsx.cisd.etlserver.SimpleTypeExtractor
main-thread.type-extractor.file-format-type = TIFF
main-thread.type-extractor.locator-type = RELATIVE_LOCATION
main-thread.type-extractor.observable-type = IMAGE
main-thread.type-extractor.procedure-type = DATA_ACQUISITION

# The storage processor (IStorageProcessor implementation)
#main-thread.storage-processor = ch.systemsx.cisd.etlserver.DefaultStorageProcessor
main-thread.storage-processor = ch.systemsx.cisd.etlserver.BDSStorageProcessor
main-thread.storage-processor.version = 1.0
main-thread.storage-processor.sampleTypeCode = CELL_PLATE
main-thread.storage-processor.sampleTypeDescription = Screening Plate
main-thread.storage-processor.format = HCS_IMAGE V1.0
main-thread.storage-processor.number_of_channels = 2
main-thread.storage-processor.contains_original_data = TRUE
main-thread.storage-processor.well_geometry = 3x3
main-thread.storage-processor.file-extractor =
ch.systemsx.cisd.etlserver.imsb.HCSImageFileExtractor
```

Structure of `service.properties`

- The section above property inputs contains configuration parameters common for all ETL threads.
- The property inputs is a comma-separated list of the names of the ETL threads. Note, that log entries show the name of the ETL thread causing the entry.
- The name of all configuration parameters for a certain ETL thread starts with `<thread name>..`

Common configuration parameters

Most of the common parameters are explained by short commons in the example above.

The common part also contains optional configuration parameters for the processors for each procedure type. These processors initiating processing specified by the processing instruction registered in openBIS Server.

- `processors`: Comma-separated list of procedure type codes.
- `processor.<procedure type code>.hard-link-instead-of-copy`: If `false` the incoming new data set entry will not be copied to the folder specified by the processing path (a part of the processing instruction). Otherwise a hard link will be created. Default: `true`
- `processor.<procedure type code>.prefix-for-absolute-paths`: Prefix of processing paths starting with `'/'`. Default: Empty string.
- `processor.<procedure type code>.prefix-for-relative-paths`: Prefix of processing paths not starting with `'/'`. Default: Empty string.
- `processor.<procedure type code>.data-set-code-prefix-glue`: Character which will be used to concatenate the data set code with the data set name in order to create the data set name for processing.

- `processor.<procedure type code>.parameters-file`: Name of the file created in the folder specified by the processing path which will contain the processing parameters (an optional part of the processing instruction). This file will not be created if this configuration parameter is not present or if there are no processing parameters.
- `processor.<procedure type code>.finished-file-template`: Template for the name of the marker file (empty file) created in the folder specified by the processing path. A `{0}` in this template will be replaced by the name of incoming new data set entry. This marker file will not be created if this parameter is not present.

Note, that

- Path prefixes which are non empty-string have to specify existing and accessible directories. This is checked on server start up.
- The processors are used by all ETL threads.

ETL thread configuration parameters

Beside of the simple configuration parameters which are explained in the example above the configuration parameters for an ETL thread contains three sections: Code extractor, type extractor, and storage processor.

Data Set Information Extractor

The data set information extractor can extract the following informations from an incoming new data set entry:

- Sample Code: Code of the sample to which this data set belongs.
- Experiment Identifier: Project code and experiment code together to which the sample belongs.
- Data Set Code: Unique code of the data set.
- Parent Data Set Code: Code of the data set from which this data set is derived from.
- Data Producer Code: Code of the producer (e.g. instrument, analyser) of the data set.
- Data Production Date: Timestamp of data production.

The property `<thread name>.data-set-info-extractor` is the fully-qualified name of the Java class which implements `ch.systemsx.cisd.etlserver.IDataSetInfoExtractor`. Currently there are three implementations which all extract the informations from the name of the incoming new data set entry.

`ch.systemsx.cisd.etlserver.DefaultDataSetInfoExtractor`

This is the default extractor for the case that data set code is created by the openBIS server. It extracts the following data:

- Sample Code
- Parent Data Set Code
- Data Producer Code
- Data Production Date

It has the following configuration parameters:

- `<thread name>.data-set-info-extractor.entity-separator`: Character which separates entities in the file name. Whitespace characters are not allowed. Default: `'.'`

- `<thread name>.data-set-info-extractor.index-of-sample-code`: Index of the entity which is interpreted as the sample code. Default: -1
- `<thread name>.data-set-info-extractor.index-of-parent-data-set-code`: Index of the entity which is interpreted as the parent data set code. If not specified no parent data set code will be extracted.
- `<thread name>.data-set-info-extractor.index-of-data-producer-code`: Index of the entity which is interpreted as the data producer code. If not specified no data producer code will be extracted.
- `<thread name>.data-set-info-extractor.index-of-data-production-date`: Index of the entity which is interpreted as the data production date. If not specified no data production date will be extracted.
- `<thread name>.data-set-info-extractor.data-production-date-format`: Format of the data production date. For the correct syntax see [SimpleDateFormat](#). Default: 'yyyyMMddHHmmss'

Note: Indices can be positive (0 first entity from left, 1 second entity from left, etc.) or negative (-1 first entity from right, -2 second entity from right, etc.).

ch.systemsx.cisd.etlserver.DataSetInfoExtractorForPlatesOnDemand

This is the extractor for *plates-on-demand* workflow. It extracts the following data:

- Sample Code
- Parent Data Set Code
- Data Producer Code
- Data Production Date
- Project Code
- Experiment Code
- `<thread name>.data-set-info-extractor.entity-separator`: Character which separates entities in the file name. Whitespace characters are not allowed. Default: '.'
- `<thread name>.data-set-info-extractor.index-of-project-code`: Index of the entity which is interpreted as the project code. Default: -3
- `<thread name>.data-set-info-extractor.index-of-experiment-code`: Index of the entity which is interpreted as the experiment code. Default: -2
- `<thread name>.data-set-info-extractor.index-of-sample-code`: Index of the entity which is interpreted as the sample code. Default: -1
- `<thread name>.data-set-info-extractor.index-of-parent-data-set-code`: Index of the entity which is interpreted as the parent data set code. If not specified no parent data set code will be extracted.
- `<thread name>.data-set-info-extractor.index-of-data-producer-code`: Index of the entity which is interpreted as the data producer code. If not specified no data producer code will be extracted.
- `<thread name>.data-set-info-extractor.index-of-data-production-date`: Index of the entity which is interpreted as the data production date. If not specified no data production date will be extracted.
- `<thread name>.data-set-info-extractor.data-production-date-format`: Format of the data production date. For the correct syntax see [SimpleDateFormat](#). Default: 'yyyyMMddHHmmss'

ch.systemsx.cisd.etlserver.threev.DataSetInfoExtractorForDataAcquisition

This is the extractor which assumes that the data set code is created by the data set producer. It extracts the following data:

- Sample Code
- Data Set Code
- Parent Data Set Code
- Data Producer Code
- Data Production Date

It has the following configuration parameters:

- `<thread name>.data-set-info-extractor.entity-separator`: Character which separates entities in the file name. Whitespace characters are not allowed. Default: '.'
- `<thread name>.data-set-info-extractor.index-of-sample-code`: Index of the entity which is interpreted as the sample code. Default: -1
- `<thread name>.data-set-info-extractor.index-of-parent-data-set-code`: Index of the entity which is interpreted as the parent data set code. If not specified no parent data set code will be extracted.
- `<thread name>.data-set-info-extractor.index-of-data-producer-code`: Index of the entity which is interpreted as the data producer code. If not specified no data producer code will be extracted.
- `<thread name>.data-set-info-extractor.index-of-data-production-date`: Index of the entity which is interpreted as the data production date. If not specified no data production date will be extracted.
- `<thread name>.data-set-info-extractor.data-production-date-format`: Format of the data production date. For the correct syntax see [SimpleDateFormat](#). Default: 'yyyyMMddHHmmss'
- `<thread name>.data-set-info-extractor.indices-of-data-set-code-entities`: Space or comma separated list of entity indices which define the data set code uniquely. This is a mandatory property.
- `<thread name>.data-set-info-extractor.data-set-code-entities-glue`: Symbol used to concatenate entities defining the data set code. Default: '.'

ch.systemsx.cisd.etlserver.threev.DataSetInfoExtractorForImageAnalysis

This is a variant of

`ch.systemsx.cisd.etlserver.DefaultDataSetInfoExtractor`. The difference is that the parent data set code can be defined more than one entity of the data set name.

It has the following configuration parameters:

- `<thread name>.data-set-info-extractor.entity-separator`: Character which separates entities in the file name. Whitespace characters are not allowed. Default: '.'
- `<thread name>.data-set-info-extractor.index-of-sample-code`: Index of the entity which is interpreted as the sample code. Default: -1
- `<thread name>.data-set-info-extractor.indices-of-parent-data-set-code-entities`: Space or comma separated list of entity indices which define the parent data set code uniquely. This is a mandatory property.

- `<thread name>.data-set-info-extractor.data-set-code-entities-glue`: Symbol used to concatenate entities defining the parent data set code. Default: '.'
- `<thread name>.data-set-info-extractor.index-of-data-producer-code`: Index of the entity which is interpreted as the data producer code. If not specified no data producer code will be extracted.
- `<thread name>.data-set-info-extractor.index-of-data-production-date`: Index of the entity which is interpreted as the data production date. If not specified no data production date will be extracted.
- `<thread name>.data-set-info-extractor.data-production-date-format`: Format of the data production date. For the correct syntax see [SimpleDateFormat](#). Default: 'yyyyMMddHHmmss'

Type Extractor

Extractor for

- procedure type code
- observable type code
- file format type code
- locator type code

from the incoming new data set entry.

The property `<thread name>.type-extractor` is the fully-qualified name of the Java class which implements

`ch.systemsx.cisd.etlserver.IProcedureAndDataTypeExtractor`. Currently there is only one implementation:

- `ch.systemsx.cisd.etlserver.SimpleTypeExtractor`: It does not extract these codes from the incoming new data set entry. These codes always the codes provided by the following configuration parameters:
 - `<thread name>.type-extractor.procedure-type`: Allowed values: DATA_ACQUISITION and IMAGE_ANALYSIS
 - `<thread name>.type-extractor.observable-type`: Allowed values: IMAGE and IMAGE_ANALYSIS_DATA
 - `<thread name>.type-extractor.file-format-type`: Allowed values: TIFF, 3VPROPRIETARY, and PLKPROPRIETARY
 - `<thread name>.type-extractor.locator-type`: Allowed value: RELATIVE_LOCATION

Storage Processor

A storage processor handles incoming new data set entries which can be identified by the code extractor and which can be registered in the openBIS Server.

The property `<thread name>.storage-processor` is the fully-qualified name of the Java class which implements `ch.systemsx.cisd.etlserver.IStorageProcessor`.

Currently there are only two implementations:

- `ch.systemsx.cisd.etlserver.DefaultStorageProcessor`: The incoming new data set is just moved into the storage at place described above.
- `ch.systemsx.cisd.etlserver.BDSStorageProcessor`: It is responsible for

processing the incoming data in accordance to the rules defined in [CISD Biological Data Standards](#). For the configuration parameters of the BDSStorageProcessor see ETL Server and BDS.

Highwater mark settings

The server can be configured to not start processing a new incoming data set if the available free disk space on the data store drops below a certain threshold. This threshold is set by the option `highwater-mark` and is specified in kilo-bytes. Once the file system got cleaned up and the amount of free space grows above the threshold, the ETL server will resume operation automatically.

Installation steps

1. Unzip the distribution on the server machine into some temporary folder.
2. Edit `etc/service.properties` file if necessary.
3. In case you need your own implementation of one or more of the plugins described above, you need to package them into a jar file `etlserver-plugins.jar` and then you need to copy this jar file into the `lib/` subdirectory of the ETL server directory. Then you can specify your own classes in the `service.properties` file the same way described above for the plugin classes written by CISD. *Note that adding a plugin jar file requires a restart of the ETL server.*

Start Server

The ETL jar file can be run by executing a shell command `etlserver.sh` (for Unix) or `etlserver.bat` (for Windows). Several properties of the configuration file `service.properties` can be overridden by command line options:

```
prompt> ./etlserver.sh --help
etlserver <required options> [option [...]]
--help                               : Prints out a description of the options.
--version                             : Prints out the version information.
--test-notify                         : Tests the notify log (i.e. that an email is
                                     sent out).
--notify-successful-registration      : If true, a notification is sent on
                                     successful registration of a data set in
                                     openBIS [default: false]
[-c,--check-interval] N              : The interval to wait between two checks (in
                                     seconds) [default: 120]
[-p,--password] VAL                  : User login password
[-s,--server-url] URL                 : URL of the server
[-t,--shutdown-timeout] N            : The time-out for clean up work in the
                                     shutdown sequence (in seconds) [default: 30]
[-u,--username] VAL                  : User login name

Example: etlserver --notify-successful-registration -c N -p VAL -s URL -t N -u VAL
```

Stop Server

On server shutdown (corresponds to sending a SIGTERM signal to the Java process), a shutdown sequence is initiated. If currently no data are processed, this sequence will be finished immediately. In case of one or more threads doing some work at the time, they will be given the opportunity to clean-up (i.e. to perform a roll-back on a file system operation). The time that is maximally allowed for any shutdown sequence to complete can be influenced by the `shutdown-timeout`. Note that the

total maximal time of the shutdown sequence can be as long as twice the time specified for the shutdown-timeout.

Data Set Download Server

Introduction

The Data Set Download Server allows to download files from a data set registered by the ETL Server. It is a simple file server. All necessary metadata of a data set are obtained from the openBIS Server.

Installation

1. Unzip the distribution `download-server-<version>-r<release>.zip`. A folder `download-server` with following content will be created
 - `download-service.sh`
 - `etc/jetty-keystore`
 - `etc/log.xml`
 - `etc/openBIS.keystore`
 - `etc/service.properties`
 - `lib/` contains all Java classes.
 - `log/`
2. Adapt `etc/service.properties` in accordance with the next section.
3. Start the server with

```
prompt> ./download-service.sh start
```

4. Server can be stopped with

```
prompt> ./download-service.sh stop
```

Configuration

The Data Set Download Server is configured by two files:

- `etc/service.properties`: Service properties
- `etc/log.xml`: Logging definitions

Note: In the `service.properties` file for openBIS the following line is needed for allowing user access to the Data Set Download Server:

```
dataset-download-service-base-url = https://<host>:<port>/dataset-download
```

where `<host>` is the host name (or IP address) of the host of the Data Set Download Server and `<port>` is the port as specified in the file `etc/service.properties` of the Data Set Download Server.

Service Properties

The content of `etc/service.properties` as shipped with the distribution reads

`service.properties`

```
# The root directory of the data store
storeroot-dir = targets/store

# Port
```



```

port = <enter port>

# The URL of the openBIS server
server-url = https://localhost:8443/openbis

# Session timeout in minutes
session-timeout = 30

# Path to the keystore
keystore.path = etc/jetty-keystore

# Password of the keystore
keystore.password = OBF:lvny1zlo1x8e1vnwlvn6lx8g1zlu1vn4

# Key password of the keystore
keystore.key-password = OBF:lu2ulwml1z7slz7alwnl1u2g

```

- **storeroot-dir**: This is the root directory of the store of all data sets. It corresponds to the property of same name for the ETL Server. In fact, both paths should point to the same location in the filesystem. But they can be different because both servers can run on different machines.
- **port**: The port used by the Data Set Download Server. Note, that the server can only be accessed via HTTPS.
- **server-url**: Base URL of the openBIS Server. It should be of the form `https://<host>:<port>/openbis`.
- **session-timeout**: Session timeout in minutes.
- **keystore.path**: Path to the keystore needed by the Data Set Download Server for HTTPS.
- **keystore.password**: Password of the keystore specified by `keystore.path`. In the shipped `service.properties` the password for `etc/jetty-keystore` is specified.
- **keystore.key-password**: Password of the key in the keystore. In the shipped `service.properties` the password for `etc/jetty-keystore` is specified.

It is strongly recommended **not** to use `etc/jetty-keystore`.

Logging Definitions

The file `etc/log.xml` contains the specification for the [log4j](#) loggers used by the Data Set Download Server. Only parameters `SMTPHost` (host name or address of the SMTP server) and `to` (e-mail address of the receiver of serious error messages) might be adapted.