User Manual for openBIS 8.04.1

Date: 26.5.08

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

This manual is addressed to the user of openBIS. It is not a documentation for administrators.

Table of Contents

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.

User Guide

The siRNA Image Mangement System is a client server application managing and processing image data generated in siRNA screens.

Parts

The current version consists of
- A DB Server
- An Application Server that controls authentication and database access via Web Services, provided over Secure Socket Layers
- The CISD authentication service (same accounts as for the Wiki and the Issue Tracker)
- A console client offering "commands" or "methods" to the user
- A buffer or overflow server infrastructure staging the image data locally if necessary due to network instabilities
- A Datamover moving file-based raw data produced by some local data producer (typically a measurement device) to a (remote) central storage, in this case to the NAS of ETHZ
- An ETL server that registers the images with the DB and thereby associating the data, i.e. images, with the corresponding experimental procedures
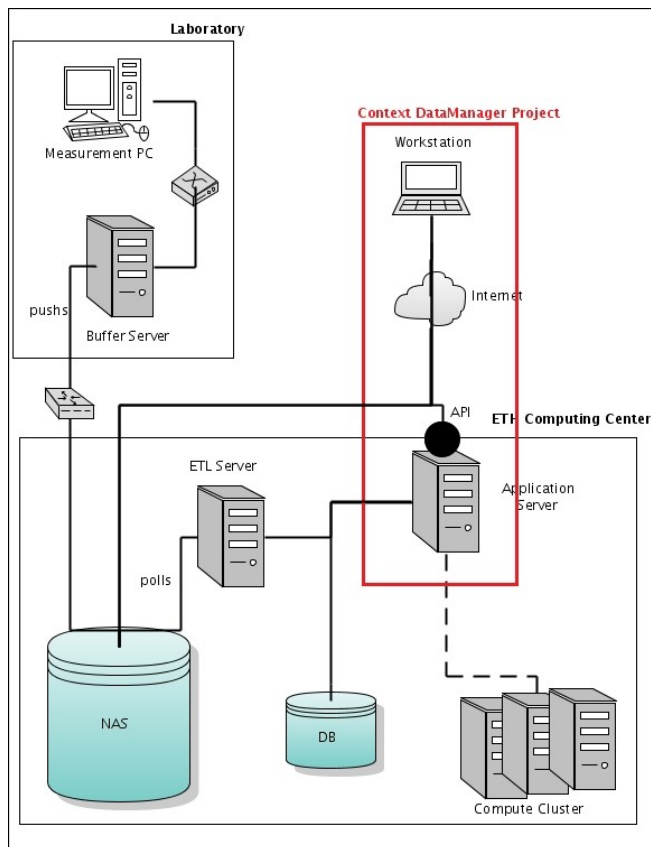
Prerequisites and Installation

To be able to run the command line client Java 5 (sometimes ambiguously called Java 1.5) or Java 6 (sometimes called Java 1.6) must be installed. To check, enter:

```
C:\> java -version
```

on the command line. The systems should return something like:

```
java version "1.6.0.01"
Java(TM) SE Runtime Environment (build 1.6.0_01-b06)
Java HotSpot(TM) Client VM ((build 1.6.0_01-b06, mixed mode, sharing)
```

> To be able to log in, you need a SystemsX.ch account and you need to be member of the `lims-users` group, unless that is the case please contact CISD for assistance. A SystemsX.ch account will give you access rights to all applications hosted by CISD, e.g. the SystemsX.ch Wiki and the issue tracking system.

1. The application server is available at <ins>https://sprint-openbis.ethz.ch/openbis</ins>. Just click on this link to sure that the server is available.
2. Download the current client release and unpack it.
3. Open a command line prompt and change into the `openBIS-client` directory.

```
D:\somewhere> cd openBIS-client
```

4. Open a session to the application server by typing:

```
D:\somewhere\openBIS-client> bis login
```

running the shell script, e.g. `bis.sh` (for OS Linux) or `bis.bat` (for OS Windows) calling the `login` method. Both scripts are provided by the zip file.

openBIS Tutorial

This tutorial presents an example for the standard workflow (so called *strict* workflow) for siRNA High Content Screening needed to get a new experiment registered.

First, a brief overview about the concepts and terms:

- An *experiment* is related to a *project*. A project can have many experiments.
- By registering an experiment *cell plates* or *re-infection plates* have to be specified. Optional also *processing instructions* can be specified.
- A re-infection plate is derived from a cell plate.
- A cell plate is derived from a *dilution plate*.
- A dilution plate is derived from a *master plate*.
- Each plate can be associated with a *control layout*.
- The generic term for plates and control layouts is *sample*.
- Samples are organized in *wells* which are ordered in a rectangular fashion called *plate geometry*.
- Well can be associated with a *material* of a certain *material type*. Control layout wells are associated with materials of type CONTROL. Wells of master plates are associated with materials of type OLIGO or COMPOUND. The wells for all other types of plates will be inherited from the master plate from which the plate is derived from.
- A materials, samples, and experiments can have zero or more *properties*.
- A property has a certain *property type* which specifies the unique property name (called *property code*), the data type, and whether it is mandatory or not.
- Materials of type OLIGO are associated with materials of type GENE.

Note:

- You have to login before you can do anything useful.
- You have to have at least USER privileges (for some commands even ADMIN).
- Often *tab-separated* files are needed in many commands. Such a file contains a table of values. Tab-separated files are easily created with Excel by just saving a spread sheet as 'Text (Tab delimited)'. Try to avoid special characters in the file naming. To avoid surprises, only uses letters, numbers and underscore. If you can not do so, put the file in quotes (").

Register Properties

For experiments, materials, and samples you can assign additional properties, beside of the already defined ones.

The following paragraphs show how you can assign properties to materials. But this also works for experiments and samples.

First, you should get an overview of the available types and their associated property types by using the commands `list-material-types` and `get-material-type-info`:

```
prompt> bis list-material-types -P
 Code      | Description
-----------+--------------------------------------------------------------------------
 BACTERIUM | Bacterium
 CELL_LINE | Cell Line or Cell Culture. The growing of cells under controlled conditions.
 COMPOUND  | Compound
 CONTROL   | Control of a control layout
 GENE      | Gene
 OLIGO     | Oligo nucleotide
 VIRUS     | Virus
(7 rows)

prompt> bis get-material-type-info CONTROL -P
                                Properties of materials of type CONTROL
 Code             | Label       | Data Type                         | M[...] | Description       | Registrator                | Registration Date
------------------+-------------+-----------------------------------+--------+-------------------+----------------------------+--------------------
 USER.COLOR       | Color       | VARCHAR                           | no | Material Color    | Darwin, Charles [darwin]   | 2008-02-19 13:23:59 GMT+01:00
 USER.DESCRIPTION | Description | VARCHAR                           | no | A Description     | System User [system]       | 2007-12-19 15:43:20 GMT+01:00
 USER.SKINCOLOR   | Color(1)    | CONTROLLEDVOCABULARY(USER.COLOR)  | no | Color of material | Einstein, Albert [einstein] | 2008-02-28 10:11:21 GMT+01:00
(3 rows)
prompt>
```

If the material type has already all properties you think are necessary for your purpose you can skip the following subsections.

Register Property Type

Registering properties and assigning them to certain material types needs ADMIN privileges.

The command `list-property-types` lists the available property types:

```
prompt> bis list-property-types -P
                                        Property Types
```

```
Code                        | Label              | Data Type                        | Registrator               | Registration Date
----------------------------+--------------------+----------------------------------+---------------------------+--------------------------
USER.CELLLINE_NAME          | Cell line          | VARCHAR                          | System User [system]      | 2007-12-05 13:33:34 GMT+01:00
USER.COLOR                  | Color              | VARCHAR                          | Darwin, Charles [darwin]  | 2008-02-19 12:35:03 GMT+01:00
USER.DESCRIPTION            | Description        | VARCHAR                          | System User [system]      | 2007-12-05 13:33:34 GMT+01:00
USER.GENE_SYMBOL            | Gene Symbol        | VARCHAR                          | System User [system]      | 2007-12-05 13:33:34 GMT+01:00
USER.NUCLEOTIDE_SEQUENCE    | Nucleotide Sequence| VARCHAR                          | System User [system]      | 2007-12-05 13:33:34 GMT+01:00
USER.OFFSET                 | Offset             | INTEGER                          | System User [system]      | 2007-12-05 13:33:34 GMT+01:00
USER.ORG                    | organism           | VARCHAR                          | System User [system]      | 2007-12-05 13:33:34 GMT+01:00
USER.REFSEQ                 | RefSeq             | VARCHAR                          | System User [system]      | 2007-12-05 13:33:34 GMT+01:00
USER.SKINCOLOR              | Color(1)           | CONTROLLEDVOCABULARY(USER.COLOR) | Einstein, Albert [einstein] | 2008-02-28 10:10:15 GMT+01:00
(9 rows)
prompt>
```

If none of them can be used to assign a new property type to a certain material type a new property type has to be defined by the command `register-property-types`. As a prerequisite a tab-separated file like the following has to be prepared:

property-types.txt

```
code    label   data_type       description
USER.VENDOR     Vendor  VARCHAR Vendor of a material
```

Note, that the code of a property has to start with `USER.` in order to distinguishes it from internally defined properties.

Now, executing the registration command with this file leads to:

```
prompt> bis register-property-types property-types.txt
Registering property types from 'D:\User\felmer\tmp\sprint-demo\openBIS-client\property-types.txt'.
  Following properties '[USER.VENDOR]' have been successfully registered.

prompt> bis list-property-types -P
                                        Property Types

Code                        | Label              | Data Type                        | Registrator               | Registration Date
----------------------------+--------------------+----------------------------------+---------------------------+--------------------------
USER.CELLLINE_NAME          | Cell line          | VARCHAR                          | System User [system]      | 2007-12-05 13:33:34 GMT+01:00
USER.COLOR                  | Color              | VARCHAR                          | Darwin, Charles [darwin]  | 2008-02-19 12:35:03 GMT+01:00
USER.DESCRIPTION            | Description        | VARCHAR                          | System User [system]      | 2007-12-05 13:33:34 GMT+01:00
USER.GENE_SYMBOL            | Gene Symbol        | VARCHAR                          | System User [system]      | 2007-12-05 13:33:34 GMT+01:00
USER.NUCLEOTIDE_SEQUENCE    | Nucleotide Sequence| VARCHAR                          | System User [system]      | 2007-12-05 13:33:34 GMT+01:00
USER.OFFSET                 | Offset             | INTEGER                          | System User [system]      | 2007-12-05 13:33:34 GMT+01:00
USER.ORG                    | organism           | VARCHAR                          | System User [system]      | 2007-12-05 13:33:34 GMT+01:00
USER.REFSEQ                 | RefSeq             | VARCHAR                          | System User [system]      | 2007-12-05 13:33:34 GMT+01:00
USER.SKINCOLOR              | Color(1)           | CONTROLLEDVOCABULARY(USER.COLOR) | Einstein, Albert [einstein] | 2008-02-28 10:10:15 GMT+01:00
```

```
 USER.VENDOR               | Vendor              | VARCHAR                        | Doe, John [system]        | 2008-04-24 11:08:17 GMT+02:00
(10 rows)

prompt>
```

If you want to see, to which materials, experiments, or samples the properties are assigned, you can call the following methods:

- list-experiment-property-types
- list-plate-property-types
- list-control-layout-property-types
- list-material-property-types

Assign Property Type to Material Type

Now, you are able to assign the new property type USER.VENDOR to the material type CONTROL with the command assign-material-property-type:

```
prompt> bis assign-material-property-type CONTROL USER.VENDOR
  Optional property 'USER.VENDOR' successfully assigned to the material type 'CONTROL'.

prompt> bis get-material-type-info -P CONTROL
                                           Properties of materials of type CONTROL
 Code            | Label       | Data Type                       | M[..] | Description         | Registrator             | Registration Date
-----------------+-------------+---------------------------------+-------+---------------------+-------------------------+------------------------------
 USER.COLOR      | Color       | VARCHAR                         | no    | Material Color      | Darwin, Charles [darwin] | 2008-02-19 13:23:59 GMT+01:00
 USER.DESCRIPTION | Description | VARCHAR                         | no    | A Description       | System User [system]    | 2007-12-19 15:43:20 GMT+01:00
 USER.SKINCOLOR  | Color(1)    | CONTROLLEDVOC[...](USER.COLOR)  | no    | Color of material   | Einstein, Albert [...]  | 2008-02-28 10:11:21 GMT+01:00
 USER.VENDOR     | Vendor      | VARCHAR                         | no    | Vendor of a material | Doe, John [doe]         | 2008-04-24 11:14:40 GMT+02:00
(4 rows)
prompt>
```

Later on you can change with the same command optional properties to mandatory ones and vice versa. Assignments can be removed by the command unassign-material-property-type.

If you want to assigned to certain experiment or sample types a property, you need to call the following methods:

- assign-experiment-property-type
- assign-plate-property-type
- assign-control-layout-property-type

Register Materials

Registering materials is a prerequisite for registering control layouts and plates. With the command `list-materials` you get a list of already registered materials. For large sets of materials of a certain type (e.g. `GENE` and `OLIGO`) it is convenient to reduce the output by the following command line options

- `-b, --buffer`: If specified, the output on the console is stopped when the screen is full. By pressing space key or enter key output will be continued. By pressing 'q' output is quited.
- `-l <number of lines>, --limit=<number of lines>`: Maximum number of lines to be shown.
- The arguments specified after the material type code restrict the query to those plates whose code matches the specified wild card pattern (* = any string, ? = any character).

In order to register materials you need ADMIN privileges. A tab-separated file like the following has to be prepared before registration:

controls.txt

```
code    user.vendor     user.description
WATER   Stadt Basel     Plain water
TX      VuDu    Toxicase
```

The column `code` is mandatory. Other columns are determined by the assignment of the property types to the material type as discussed above.

The two new materials defined in this example file will be registered for the material type `CONTROL` by executing the command `register-materials`:

```
prompt> bis register-materials CONTROL controls.txt
Register materials of type 'CONTROL' from 'controls.txt'.
  2 materials of type 'CONTROL' successfully registered.

prompt> bis list-materials -P CONTROL
 Code                 | Registrator          | Registration Date             | Description      | Color | Vendor
----------------------+----------------------+-------------------------------+------------------+-------+-------------
 EMPTY                | Doe, John [doe]      | 2008-02-19 13:52:47 GMT+01:00 | empty well       | -     | -
 GFP                  | System User [system] | 2007-08-24 00:00:00 GMT+02:00 | -                | -     | -
 GREEN                | Doe, John [doe]      | 2008-02-19 13:52:47 GMT+01:00 | my green control | green | -
 SCRAM                | System User [system] | 2007-08-24 00:00:00 GMT+02:00 | -                | -     | -
 TX                   | Doe, John [doe]      | 2008-04-24 11:28:01 GMT+02:00 | Toxicase         | -     | VuDu
 WATER                | Doe, John [doe]      | 2008-04-24 11:28:01 GMT+02:00 | Plain water      | -     | Stadt Basel
(6 rows)

prompt>
```

In order to register plates you have to register also materials of type `COMPOUND` or `GENE` and `OLIGO`. When an experiment will be registered you have to

specify a material which is called *study object*. Usual a study object is a `VIRUS` or a `BACTERIUM`. It might be therefore necessary to register also materials of this type.

Register Control Layouts

As a prerequisite materials of type `CONTROL` have to be registered because they will be associated with wells in a control layout. A tab-separated file like the following defines a control layout:

TEST_CL.txt

```
code    material
A01     GREEN
A02     EMPTY
B01     GREEN
B02     EMPTY
```

The first column denotes the well codes. It has to be a letter for the row followed by two digits for the column. The second column denotes the corresponding code of a registrated material of type `CONTROL`.

Control layouts can be attached to any plate, be it either a master plate, dilution plate, cell plate or reinfection plate. The attachment can be made when plates are registered to the system.

This control layout is registered by the command `register-control-layout`:

```
prompt> bis register-control-layout -p PLATE_GEOMETRY=384_WELLS_16X24 TEST_CL.txt
Register control layout from 'D:\User\felmer\tmp\openBIS-client\TEST_CL.txt'.
  Control layout 'TEST_CL' successfully registered.

prompt> bis list-control-layouts -P
[This table is modified for clarity reasons]
        Control Layouts
 Code | Type Code | Invalidated | Registrator | Registration Date | Control Layout | Master Plate | Control Layout Master P. | Dilution Plate | Control
Layout Dilution P. | Cell Plate | Control Layout Cell P. | Experiment | Project | Plate Geometry
[...-+--+-+-...]
 TEST_CL | CONTROL_LAYOUT | no | [a] | 2008-04-24 11:40:06 GMT+02:00 | - | - | -  | - | - | - | - | - | - | 384_WELLS_16X24
(1 rows)

prompt>
```

Note, that

- the name of the file (without extension) is used to define the code of the control layout. The code has to be unique not only among all control layouts but also among all plates.
- you have to specify plate geometry. Available plate geometries can be listed using `get-vocabulary-info PLATE_GEOMETRY` command.

```
prompt> bis get-vocabulary-info PLATE_GEOMETRY -P
                    Vocabulary Terms
 Code              | Registrator         | Registration Date
-------------------+---------------------+------------------------------
 1536_WELLS_32X48  | System User [system] | 2008-04-18 14:16:00 GMT+02:00
 384_WELLS_16X24   | System User [system] | 2008-04-18 14:16:00 GMT+02:00
 96_WELLS_8X12     | System User [system] | 2008-04-18 14:16:00 GMT+02:00
(3 rows)
```

Register Plates

Plates are related to each other in a hierarchical way: A plate can have zero or many *derived plates*. The root in this hierarchy is a *master plate*. The whole hierarchy is as follows: master plate > dilution plate > cell plate > reinfection plate. During plate registration this relationship is established.

As a consequence master plates are the first plates to be registered. Registering of a master plate is also different from registering all other types of plates because a registered master plate defines the materials associated with non-control wells for all derived plates.

Register Master Plates

This needs ADMIN privileges. A tab-separated file as the following has to be prepared:

MASTER-PLATE1.txt

```
code    material    material_type
A03     1_A     OLIGO
A04     2_A     OLIGO
B03     12_A    OLIGO
B04     13_A    OLIGO
```

As in the control layout file the first column specifies the wells. The material is specified by the second and third column. Note, that the second column is not enough because the same material code can be used for different material types. Thus, only the combination of material code and material type codes is unique.

This master plate is registered by the command `register-master-plate`:

```
prompt> bis register-master-plate -p PLATE_GEOMETRY=384_WELLS_16X24 MASTER-PLATE1.txt batch7
Registering master plate, loading data from 'D:\User\felmer\tmp\sprint-demo\openBIS-client\MASTER-PLATE1.txt'.
  Master plate 'MASTER-PLATE1' successfully registered.

prompt> bis get-plate-info -P MASTER-PLATE1
<<<Plate Information for code 'MASTER-PLATE1'>>>
  Type Code: MASTER_PLATE
  Registrator Login: a
  Registrator Name: -
  Registrator Email: -
  Registration Date: 2008-04-24 11:52:56 GMT+02:00
  Control Layout: -
  Master Plate: -
  Control Layout Master P.: -
  Dilution Plate: -
  Control Layout Dilution P.: -
  Cell Plate: -
  Control Layout Cell P.: -
  Generated Plates: -
  Experiment: -
  Project: -
  Consistent: -
  Invalidated: no
  Plate Geometry: 384_WELLS_16X24
                          Plate layout
 Gene Id | Gene Symbol | Material Batch | Nucleotide Sequence   | Oligo | Well
---------+-------------+---------------+----------------------+------+------
 1       | A1BG        | BATCH7        | DDDBBHDUUDUUDDDBDHHBH | 1_A  | A03
 2       | A2M         | BATCH7        | UDUDBDUBDBBUHHBDUBDHD | 2_A  | A04
 12      | SERPINA3    | BATCH7        | UUUBDUUBBUDDUUUUUBUHD | 12_A | B03
 13      | AADAC       | BATCH7        | HDBDUDUBHUDUUDHBDDDUB | 13_A | B04
(4 rows)

prompt>
```

The material batch can be used as a kind of version of a master plate. As for control layouts the name of the file (without extension) is used to define the code of the master plate. The code has to be unique among all samples.

Register Other Types of Plates

Registering other plates than master plates needs only `USER` privileges.

First, dilution plates will be registered. A tab-separated file as the following has to be prepared:

dilution-plates.txt

```
code     parent
DP01a    MASTER-PLATE1
DP01b    MASTER-PLATE1
```

These dilution plates are registered by the command `register-dilution-plates`:

```
prompt> bis register-dilution-plates dilution-plates.txt
Register dilution plates from 'D:\User\felmer\tmp\SPRINT~1\OPENBI~1\dilution-plates.txt'.
  New dilution plate '[dilutionPlate=DP01a,masterPlate=MASTER-PLATE1]' successfully registered.
  New dilution plate '[dilutionPlate=DP01b,masterPlate=MASTER-PLATE1]' successfully registered.

prompt> bis list-plates DILUTION_PLATE -P DP01?
[This table is modified for clarity reasons]
                                          Dilution Plates
Code | Type Code | Invalidated | Registrator | Registration Date | Control Layout | Master Plate | Control Layout Master P. | Dilution Plate | Control
Layout Dilution P. | Cell Plate | Control Layout Cell P. | Experiment | Project
[...+---+---+---+...]
 DP01A | DILUTION_PLATE | no | [a] | 2008-04-24 11:58:45 GMT+02:00 | -  | MASTER-PLATE1 | - | - | - | - | - | - | -
 DP01B | DILUTION_PLATE | no | [a] | 2008-04-24 11:58:45 GMT+02:00 | -  | MASTER-PLATE1 | - | - | - | - | - | - | -
(2 rows)

prompt> bis get-plate-info -P DP01a
<<<Plate Information for code 'DP01A'>>>
  Type Code: DILUTION_PLATE
  Registrator Login: a
  Registrator Name: -
  Registrator Email: -
  Registration Date: 2008-04-24 11:58:45 GMT+02:00
  Control Layout: -
  Master Plate: MASTER-PLATE1
  Control Layout Master P.: -
  Dilution Plate: -
  Control Layout Dilution P.: -
  Cell Plate: -
  Control Layout Cell P.: -
  Generated Plates: -
  Experiment: -
  Project: -
  Consistent: -
  Invalidated: no
                        Plate layout
```

```
 Gene Id | Gene Symbol | Material Batch | Nucleotide Sequence  | Oligo | Well
---------+-------------+----------------+----------------------+-------+------
 1       | A1BG        | BATCH7         | DDDBBHDUUDUUDDDBDHHBH | 1_A   | A03
 2       | A2M         | BATCH7         | UDUDBDUBDBBUHHBDUBDHD | 2_A   | A04
 12      | SERPINA3    | BATCH7         | UUUBDUUBBUDDUUUUUBUHD | 12_A  | B03
 13      | AADAC       | BATCH7         | HDBDUDUBHUDUUDHBDDDUB | 13_A  | B04
(4 rows)

prompt>
```

Registering of cell plates is similar. Again a tab-separated file as the following has to be prepared:

cell-plates.txt

```
Code    Parent
CP01a1  DP01a
CP01a2  DP10a
CP01b1  DP01b
```

These cell plates are registered by the command `register-cell-plates`:

```
prompt> bis register-cell-plates cell-plates.txt
Register cell plates from 'D:\User\felmer\tmp\SPRINT~1\OPENBI~1\cell-plates.txt'.
  New cell plate '[cellPlate=CP01a1,dilutionPlate=DP01a]' successfully registered.
  New cell plate '[cellPlate=CP01a2,dilutionPlate=DP01a]' successfully registered.
  New cell plate '[cellPlate=CP01b1,dilutionPlate=DP01b]' successfully registered.

prompt>
```

For registering reinfection plates see command `register-reinfection-plates`.

Register Experiments

After registration of cell plates (and at least one study object) experiments can be registered.

Register Projects

First, a project has to be chosen or created. All current projects are listed by the command `list-projects`:

```
prompt> bis list-projects -P
        Projects
```

```
 Code             | Group
------------------+-------
 DRUGGABLE GENOME | CISD
 NEPTUN           | CISD
(2 rows)

prompt>
```

A new project is created by the command `register-project`:

```
prompt> bis register-project example-project
New project 'example-project' has been successfully stored in the database.

prompt> bis list-projects -P
        Projects
 Code             | Group
------------------+-------
 DRUGGABLE GENOME | CISD
 EXAMPLE-PROJECT  | CISD
 NEPTUN           | CISD
(3 rows)

prompt>
```

Note: This needs ADMIN privileges.

Register an Experiment

As a minimum two files have to be created in the same folder:

experiments.txt

```
project code      user.description       study_object     study_object_type      cell_plates
example-project experiment1     My first experiment     HIV-1   VIRUS   >cell-plates-for-experiment1.txt
```

and

cell-plates-for-experiment1.txt

```
CP01a1
CP01b1
```

The second file is a list of all cell plates used in `experiment1`. The name of the file is marked in `experiments.txt` by the symbol '>'.

The experiment is registered by the command `register-experiments`. The first parameter should be the code of the experiment type. The allowed experiment types can be obtained by the command `list-experiment-types`.

```
prompt> bis register-experiments SIRNA_HCS experiments.txt
Register experiments from 'D:\User\felmer\tmp\SPRINT~1\OPENBI~1\experiments.txt'.
  Experiment '[Experiment=experiment1,Project=example-project]' successfully registered.

prompt> bis list-experiments SIRNA_HCS -P -p example-project
 Project Code    | Code        | Invalidated | Study Object | Registrator    | Registration Date           | Last Dataset Date | Description
-----------------+-------------+-------------+--------------+----------------+-----------------------------+-------------------+-------------------
 EXAMPLE-PROJECT | EXPERIMENT1 | no          | HIV-1        | Doe, John [doe] | 2008-03-20 09:31:13 GMT+01:00 | -               | My first experiment
(1 rows)

prompt>
```

Add Documents to Experiments

After an experiment has been registered any kind of document, such as a protocol or a change from the default protocol e.g. can be attached to the experiment by the command `add-experiment-attachment`:

```
prompt> bis add-experiment-attachment example-project experiment1 ..\revision2873.diff
Successfully stored attachment 'D:\User\felmer\tmp\SPRINT~1\OPENBI~1\..\revision2873.diff' for experiment '[Experiment=experimnt1,Project=example-project]'.

prompt> bis get-experiment-info example-project experiment1
<<<Experiment Information for experiment code 'EXPERIMENT1'>>>
  Experiment Type: SIRNA_HCS
  Registrator Login: u
  Registrator Name: The User, U
  Registrator Email: u@users.com
  Registration Date: 2008-04-24 13:08:12 GMT+02:00
  Study Object: HIV-1
  Study Object Type: VIRUS
  Project Code: EXAMPLE-PROJECT
  Project Group: CISD
  Description: Experiment without processing instructions

 Attachment        | Version | Registration Date           | Registrator
-------------------+---------+-----------------------------+---------------
 revision2873.diff |       1 | 2008-04-24 13:16:48 GMT+02:00 | The User, U [u]
(1 rows)
```

```
prompt>
```

An attached document can be downloaded with the command `get-experiment-attachment`:

```
prompt> bis get-experiment-attachment example-project experiment1 revision2873.diff
Successfully retrieved attachment 'D:\User\felmer\tmp\SPRINT~1\OPENBI~1\revision2873.diff' (version: 1) from experiment
'[Experiment=experiment1,Project=example-project]'.

prompt>
```

Advanced Topics of openBIS

This chapter explains topics which are beyond the scope of a tutorial.

Tab-separated Input and Output

Input

There are various commands needing files in a tab-separated format as input. Such a file contains a table of values. Each row of the table is a line in the file. The values of the different cells in the table are separated by a TAB character. The line before the first row contains the names of each column. They are not case sensitive. The order of the columns can be arbitrary. Tab-separated files are easily created with Excel by just saving a spread sheet as 'Text (Tab delimited)'.

Such tab-separated input files can also be created by other means. The following variation are recognized by openBIS:

- Column headers in first line: Example:

```
code    material        material_type
A03     1_A     OLIGO
A04     2_A     OLIGO
B03     12_A    OLIGO
B04     13_A    OLIGO
```

- Comment section: An arbitrary number of lines before the line with column headers. All lines start with a hash symbol '#'. Example:

```
# Example of a master plate
# used by some tests
code    material        material_type
A03     1_A     OLIGO
A04     2_A     OLIGO
B03     12_A    OLIGO
B04     13_A    OLIGO
```

- Column headers at the end of the comment section: Here the last line of comment section contains the tab-separated column names. The line before contains only a hash symbol '#'. Example:

```
    # Example of a master plate
    # used by some tests
    #
    #code   material        material_type
    A03     1_A     OLIGO
    A04     2_A     OLIGO
    B03     12_A    OLIGO
    B04     13_A    OLIGO
```

Output

All list commands and some get commands outputs there data in a tab-separated format. The output on the console can be redirected into a file as in the following example:

```
prompt> bis list-material-types > material-types.txt
```

If the type of the file is `.txt` it can be easily import into Excel.

If you want the output more human readable, you can add the option `--pretty-print` or `-P` to the list and get commands. In this case, the output is pretty formated, as pretty as it is possible on the command line.

Properties

OpenBIS offers its users a mechanism of assigning new property types to the chosen types of materials, experiments, plates and control layouts. In this section it will be described how to manipulate on these properties. For the reason of simplicity the special case of material properties was described, but everything except small differences in command names holds true for all other entities which can have properties.

The system stores information about materials studied in experiments, stored in plate's well etc. During the usage of the system it can turn out, that some new and specific material properties should be stored. E.g. we can specify that all GENE materials should have a SIZE property. From that point it is possible to store a property value for each material.

To get the property types that are currently assigned to a given material type (e.g. OLIGO), enter:

```
prompt> bis get-material-type-info --pretty-print OLIGO
```

The output should like as following:

```
                            Properties of materials of type OLIGO
Code                   | Label        | Data Type | Manda... | Description                | Registrator   | Registration Date
```

```
-------------------------+--------------------+----------+---------+------------------------------------+-----------------+----------------
 USER.DESCRIPTION         | Description        | VARCHAR  | no      | A Description                      | System User [...] | 2008-[...]GMT+02:00
 USER.NUCLEOTIDE_SEQUENCE | Nucleotide Sequence | VARCHAR | yes     | A sequence of nucleotides          | System User [...] | 2008-[...] GMT+02:00
 USER.OFFSET              | Offset             | INTEGER  | no      | Offset from the start of the sequence | System User [...] | 2008-[...]GMT+02:00
 (3 rows)
```

It means that each OLIGO material has besides the code and inhibitor code three properties: description, nucleotide sequence and the offset. Value of each property for each material is displayed together with other information when `list-materials` command is executed.

Registering materials with properties

When new materials are registered in the system, values for their properties should be specified in the file together with the material codes and other fields. Values for a particular property should be placed in one column. The name of the column should be the same as the property code displayed by the `get-material-type-info` command (see 'Code' column in the listing above). If the property is mandatory then its value has to be specified for all materials of the given type. In the example table above we see, that for OLIGO materials the nucleotide sequence property is mandatory and all others are optional.

Displaying existing property types

If we want to assign a new property to a chosen type of material, first we have to make sure, that the appropriate property type is defined. To get all available property types and their description enter:

```
prompt> bis list-property-types --pretty-print
```

You should get something like:

```
                                    Property Types
 Code                    | Label              | Data Type | Registrator         | Registration Date
-------------------------+--------------------+-----------+---------------------+----------------------------
 USER.DESCRIPTION         | Description        | VARCHAR   | System User [system] | 2008-04-18 14:16:00 GMT+02:00
 USER.GENE_SYMBOL         | Gene Symbol        | VARCHAR   | System User [system] | 2008-04-18 14:16:00 GMT+02:00
 USER.NUCLEOTIDE_SEQUENCE | Nucleotide Sequence | VARCHAR  | System User [system] | 2008-04-18 14:16:00 GMT+02:00
 USER.OFFSET              | Offset             | INTEGER   | System User [system] | 2008-04-18 14:16:00 GMT+02:00
 USER.REFSEQ              | RefSeq             | VARCHAR   | System User [system] | 2008-04-18 14:16:00 GMT+02:00
 (5 rows)
```

The 'Data Type' column explains what kind of values a given property can store. Following data types are currently supported:

1. `VARCHAR`: text, cannot be longer than 1024 characters.
2. `INTEGER`: integer number. Can be positive or negative (..., -1, -2, -3, 0, 1, 2, 3, ...).
3. `REAL`: a real number, such as 2.4871773339 or -5.0.
4. `BOOLEAN`: value can be `true` or `false`
5. `TIMESTAMP`: date and time

The command `list-property-types` shows all available property types.

If you want to see, to which Material, Experiment or Sample the properties are assigned, you can call the following methods:

- list-experiment-property-types
- list-plate-property-types
- list-control-layout-property-types
- list-material-property-types

Defining a new property type

If the property type which we want to assign to a material type does not yet exist, it can be defined using the `register-property-types` command. Note that an Administrator role is required to do that. In the example below 3 new property types are registered:

```
prompt> bis register-property-types examplePropertyTypes.txt
```

where examplePropertyTypes.txt is the path to a tab separated file composed of exactly 4 columns:

1. `code` - the code of the new property type, used when new materials are defined
2. `description` - text which can be helpful later on to determine what is this property useful for,
3. `label` - used when displaying property values,
4. `dataTypeCode` - what kind of data a property should store (e.g. text or numbers). Available values are `VARCHAR`, `INTEGER`, `REAL`, `BOOLEAN` and `TIMESTAMP`.

The file which is given as a parameter could look like that:

```
code        description             label       data_type
USER.EYE_COLOR The color of the eyes     Eye color VARCHAR
USER.SIZE       The size of the object    Size        INTEGER
USER.VOLUME     The volume of this person Volume      REAL
```

```
USER.IS_VALID   Can the material be used? Valid?    BOOLEAN
USER.PURCHASE_DATE      Date of purchase    Purchased TIMESTAMP
```

Assigning an optional property

It is possible to assign a new property type to a given material type using the command `assign-material-property-type` (only for users having `ADMIN` role). To assign SIZE property to all materials of type GENE, enter:

```
prompt> bis assign-material-property-type GENE USER.SIZE
```

Of course, property type with a given name (here `SIZE`) must already be defined. From this point it is possible to specify SIZE value for each gene during gene registration. By default the assigned property is optional, so for some materials the value of the property can be unspecified.

Assigning a mandatory property

By using `assign-material-property-type` command with `-m` parameter a mandatory property can be assigned:

```
prompt> bis assign-material-property-type GENE USER.SIZE -m
```

Assigning a mandatory property to materials of some type causes that the value of the property has to be specified for all the materials of the chosen type. That is why the command above works only if no GENE materials have been defined in the system till that point. If this is not the case, then the file with values for all currently existing genes have to be provided:

```
prompt> bis assign-material-property-type GENE USER.SIZE -m -f gene-size-values.txt
```

The file has to contain one row with two columns for each gene material. The first column specifies material code and the second is the value of the property for that material.

```
code    value
1       13
2       98
3       12
```

To check for which material codes the value of the property has to be specified, execute the command without specifying the file after `-f` option. All the codes will be displayed on the console. To save all those codes in the file, use output redirection:

```
prompt> bis assign-material-property-type GENE USER.SIZE -m > missing-gene-codes.txt
```

Changing a 'mandatory' flag for a property

The `assign-material-property-type` command can be also used to change the 'mandatory' flag of properties, which were already assigned to material types. If the property is mandatory, it can be made optional by executing the command again, but without `-m` parameter.

```
prompt> bis assign-material-property-type GENE USER.SIZE
```

Making the optional property mandatory may require specifying all missing property values in the file:

```
prompt> bis assign-material-property-type GENE USER.SIZE -m -f missing-gene-size-values.txt
```

Removing a property

To declare that a given property type is no longer applicable to a chosen material type, enter the command `unassign-material-property-type` and specify material type and property code:

```
prompt> bis unassign-material-property-type GENE USER.SIZE
```

If the values for a property which is about to be removed were already specified for some materials, then after user's confirmation all those values will be deleted too.

openBIS Commands

In this section all commands are listed (in alphabetical order) and described. By just typing `bis` (or `bis.sh` for Linux and MacOS) all commands are listed.

The following options are applicable for all commands and therefore not mentioned in the descriptions of the commands below:

- `-h, --help`: Shows a help text with a brief description of all options and arguments.
- `-s <base URL>, --server-base-url=<base URL>`: The base URL of the openBIS server. If not specified the URL from the file `.openBIS-default` will be used.

- Browsing commands
    - get-control-layout-info
    - get-control-layout-type-info
    - get-experiment-attachment
    - get-experiment-info
    - get-experiment-type-info
    - get-material-type-info
    - get-plate-info
    - get-plate-locations
    - get-plate-type-info
    - get-vocabulary-info
    - list-contacts
    - list-experiment-attachments
    - list-experiment-datasets
    - list-experiment-plates
    - list-experiment-property-types
    - list-experiment-types
    - list-experiments
    - list-material-property-types
    - list-material-types
    - list-materials
    - list-plate-types
    - list-plates
    - list-processing-instructions
    - list-projects
    - list-property-types
    - list-plate-property-types
    - list-control-layout-property-types
    - list-control-layouts
    - list-vocabularies
    - login
    - logout
- Commands for Users and Administrators

- add-experiment-attachment
  - register-cell-plates
  - register-control-layout
  - register-dilution-plates
  - register-experiments
  - register-reinfection-plates
- Commands for Administrators only
  - assign-control-layout-property-type
  - assign-experiment-property-type
  - assign-material-property-type
  - assign-plate-property-type
  - invalidate-experiment
  - invalidate-plates
  - register-master-plate
  - register-materials
  - register-project
  - register-property-types
  - register-vocabulary
  - unassign-control-layout-property-type
  - unassign-experiment-property-type
  - unassign-material-property-type
  - unassign-plate-property-type

Browsing commands

`get-control-layout-info`

Description: Shows the code of the material of type `CONTROL` for each well (specified in the format `<row letter><two colum digits>`) of the control layout.

Usage:

```
prompt> bis get-control-layout-info [--pretty-print] <control layout code>
```

Options:

- `-P, --pretty-print`: Shows the data in a pretty printed format which is more human readable than the tab-separated one.

Parameter:

- `<control layout code>`: Code of the control layout. The command `list-control-layouts` lists all available values.

`get-control-layout-type-info`

Description: Lists the property types assigned to control layouts.

Usage:

```
prompt> bis get-control-layout-type-info [--pretty-print]
```

Options:

- `-P, --pretty-print`: Shows the data in a pretty printed format which is more human readable than the tab-separated one.

get-experiment-attachment

Description: Downloads a specified file which has been attached to a specified experiment by the command `add-experiment-attachment`.

Usage:

```
prompt> bis get-experiment-attachment [-v <version number>] <project
code> <experiment code> <file name>
```

Options:

- `-v <version number>, --attachmentVersion=<version number>`: Version of the requested file. Has to be a number greater or equals 1. Without this option the latest version will be downloaded.

Parameter:

- `<project code>`: Code of the project to which the experiment belongs.
- `<experiment code>`: Code of the experiment. Available experiments can be listed by the command `list-experiments`.
- `<file name>`: Name of the file to be downloaded.

get-experiment-info

Description: Gets basic information about a specified experiments.

Usage:

```
prompt> bis get-experiment-info [--pretty-print] <project code> <experiment code>
```

Other detailed information about the experiment can be obtained by commands:

- `list-experiment-attachments`
- `list-processing-instructions`
- `list-experiment-datasets`
- `list-experiment-plates`

Options:

- `-P, --pretty-print`: Shows the data in a pretty printed format which is more human readable than the tab-separated one.

Parameter:

- `<project code>`: Code of the project to which the experiments belong.
- `<experiment code>`: Codes of the experiments. Exactly one experiment has to be specified. Available experiments can be listed by the command

```
list-experiments.
```

get-experiment-type-info

Description: Lists the property types assigned to a specified experiment type.

Usage:

```
prompt> bis get-experiment-type-info [--pretty-print] <experiment type code>
```

Options:

- `-P, --pretty-print`: Shows the data in a pretty printed format which is more human readable than the tab-separated one.

Parameter:

- `<experiment type code>`: The experiment. The command `list-experiments` lists all available values.

get-material-type-info

Description: Lists the property types assigned to a specified material type.

Usage:

```
prompt> bis get-material-type-info [--pretty-print] <material type code>
```

Options:

- `-P, --pretty-print`: Shows the data in a pretty printed format which is more human readable than the tab-separated one.

Parameter:

- `<material type code>`: Code of the material type. The command `list-material-types` lists all available values.

get-plate-info

Description: Gets detailed information about a plate.

Usage:

```
prompt> bis get-plate-info [-b] [--pretty-print] <plate code>
```

Options:

- `-b, --buffer`: If specified, the output on the console is stopped when the screen is full. By pressing space key or enter key output will be continued. By pressing 'q' output is quited.
- `-P, --pretty-print`: Shows the data in a pretty printed format which is more human readable than the tab-separated one.**Parameter:**
- `<plate code>`: Code of the plate to be shown.

`get-plate-locations`

Description: Lists for a specified gene its locations in all plates for all valid experiments. The set of experiments can be restricted by various options.

Usage:

```
prompt> bis get-plate-locations [-a] [-p <project code>] [-e <experiment code>] [-
i] [-o <study object code>] [-t [<from>]:[<until>]] <gene-symbol or id>
```

Options:

- `-a, --show-also-invalid`: Also locations of invalid plates are shown if this flag is specified.
- `-i, --gene-id`: Parameter `<gene-symbol or id>` is interpreted as gene id if this flag is specified.
- `-p <project code>, --project=<project code>`: Code of a project. This restricts the query. Available projects can be listed by the command `list-projects`.
- `-e <experiment code>, --experiment=<experiment code>`: Code of an experiment. Will be ignored if option `-p` is not also specified. This restricts the query to one specific experiment. Available experiments can be listed by the command `list-experiments`.
- `-o <study object code>, --study-object=<material code>`: Material code of the object to be studied. This restricts the query.
- `-t <time interval>, --time-interval=<time interval>`: A query restriction on the time interval. Only those experiments are taken into account which have an overlap of the specified time interval with the time interval defined by the experiment registration date and the registration date of the last registered data set.

Parameter:

- `<gene-symbol or id>`: The gene symbol. Because the gene symbol is not unique the gene id can be used if the option `-i` is specified.

`get-plate-type-info`

Description: Lists the property types assigned to a specified plate type.

Usage:

```
prompt> bis get-plate-type-info [--pretty-print] <sample type code>
```

Options:

- `-P, --pretty-print`: Shows the data in a pretty printed format which is more human readable than the tab-separated one.

Parameter:

- `<sample type code>`: The plate type. The command `list-plate-types` lists all available values.

`get-vocabulary-info`

Description: Lists all terms in a given controlled vocabulary set. To see available vocabulary sets, use `list-vocabularies` command.

Usage:

```
prompt> bis get-vocabulary-info [--pretty-print] <vocabulary-code>
```

Options:

- `-P, --pretty-print`: Shows the data in a pretty printed format which is more human readable than the tab-separated one.

Parameter:

- `<vocabulary-code>`: Code of the controlled vocabulary set.

`list-contacts`

Description: Lists all persons who used the system.

Usage:

```
prompt> bis list-contacts [--pretty-print]
```

Options:

- `-P, --pretty-print`: Shows the data in a pretty printed format which is more human readable than the tab-separated one.

Parameter: No parameters.

`list-experiment-attachments`

Description: Prints information about all the documents attached to a specified experiment. Additionally basic information about the experiment (as in `get-experiment-info`) are listed.

Usage:

```
prompt> bis list-experiment-attachments [--pretty-print] <project code>
<experiment code>
```

Other detailed information about the experiment can be obtained by commands:

- list-processing-instructions
- list-experiment-datasets
- list-experiment-plates

Options:

- `-P, --pretty-print`: Shows the data in a pretty printed format which is more human readable than the tab-separated one.

Parameter:

- `<project code>`: Code of the project to which the experiments belong.
- `<experiment code>`: Codes of the experiments. Exactly one experiment has to be specified. Available experiments can be listed by the command

```
list-experiments.
```

list-experiment-datasets

Description: Prints information about all datasets produced in a specified experiment. Additionally basic information about the experiment (as in `get-experiment-info`) are listed.

Usage:

```
prompt> bis list-experiment-datasets [--pretty-print] <project code> <experiment code>
```

Other detailed information about the experiment can be obtained by commands:

- list-experiment-attachments
- list-processing-instructions
- list-experiment-plates

Options:

- `-P, --pretty-print`: Shows the data in a pretty printed format which is more human readable than the tab-separated one.

Parameter:

- `<project code>`: Code of the project to which the experiments belong.
- `<experiment code>`: Codes of the experiments. Exactly one experiment has to be specified. Available experiments can be listed by the command `list-experiments`.

list-experiment-plates

Description: Prints information about all plates that are connected with a specified experiment. Additionally basic information about the experiment (as in `get-experiment-info`) are listed.

Usage:

```
prompt> bis list-experiment-plates [--pretty-print] <project code> <experiment code>
```

Other detailed information about the experiment can be obtained by commands:

- list-experiment-attachments
- list-processing-instructions
- list-experiment-datasets

Options:

- `-P, --pretty-print`: Shows the data in a pretty printed format which is more human readable than the tab-separated one.

Parameter:

- `<project code>`: Code of the project to which the experiments belong.
- `<experiment code>`: Codes of the experiments. Exactly one experiment has to be specified. Available experiments can be listed by the command

```
list-experiments.
```

list-experiment-property-types

Description: Lists all property types for experiment and also show the assignment, which property type is used for which assignment and if the property type is mandatory.

Usage:

```
prompt> bis list-experiment-property-types [--pretty-print]
```

If you want to see the other assignent:

- list-experiment-attachments
- list-processing-instructions
- list-experiment-datasets

Options:

- `-P, --pretty-print`: Shows the data in a pretty printed format which is more human readable than the tab-separated one.

Parameter: No parameters.

list-experiment-types

Description: Lists all experiment types.

Usage:

```
prompt> bis list-experiment-types [--pretty-print]
```

Options:

- `-P, --pretty-print`: Shows the data in a pretty printed format which is more human readable than the tab-separated one.

Parameter: No parameters.

list-experiments

Description: Lists all valid experiments. The set of experiments can be restricted by various options.

Usage:

```
prompt> bis list-experiments [-a] [--pretty-print] [-p <project code>] [-o <study
object code>] [-t [<from>]:[<until>]] <experiment type code>
```

Options:

- `-a, --show-also-invalid`: Also invalid experiments are shown if this flag is specified.
- `-P, --pretty-print`: Shows the data in a pretty printed format which is more human readable than the tab-separated one.
- `-p <project code>, --project=<project code>`: Code of a project. This restricts the query. Available projects can be listed by the

command `list-projects`.

- `-o <study object code>, --study-object=<material code>`: Material code of the object to be studied. This restricts the query.
- `-t <time interval>, --time-interval=<time interval>`: A query restriction on the time interval. Only those experiments are taken into account which have an overlap of the specified time interval with the time interval defined by the experiment registration date and the registration date of the last registered data set.

Parameter: `<experiment type code>`: Code of the experiment type. The command `list-experiment-types` lists all available values.

list-material-property-types

Description: Lists all property types for material and also show the assignment, which property type is used for which assignment and if the property type is mandatory.

Usage:

`prompt> bis list-material-property-types [--pretty-print]`

If you want to see the other assigment:

- list-experiment-attachments
- list-processing-instructions
- list-experiment-datasets

Options:

- `-P, --pretty-print`: Shows the data in a pretty printed format which is more human readable than the tab-separated one.

Parameter: No parameters.

list-material-types

Description: Lists all material types.

Usage:

`prompt> bis list-material-types [--pretty-print]`

Options:

- `-P, --pretty-print`: Shows the data in a pretty printed format which is more human readable than the tab-separated one.

Parameter: No parameters.

list-materials

Description: Lists all materials of a specified type.

Usage:

```
prompt> bis list-materials [-b] [--pretty-print] [-l <number of lines>] <material
type code> ["<pattern>" ...]
```

Options:

- `-b, --buffer`: If specified, the output on the console is stopped when the screen is full. By pressing space key or enter key output will be continued. By pressing 'q' output is quited.
- `-P, --pretty-print`: Shows the data in a pretty printed format which is more human readable than the tab-separated one.
- `-l <number of lines>, --limit=<number of lines>`: Maximum number of lines to be shown.
- `["<pattern>" ...]`: Restricts the query to those materials whose code matches the specified wild card pattern (* = any string, ? = any character). Pattern should be placed inside the quotation marks.

Parameter:

- `<material type code>`: Code of the material type. The command `list-material-types` lists all available values.

list-plate-types

Description: Lists all plate types.

Usage:

```
prompt> bis list-plate-types [--pretty-print]
```

Options:

- `-P, --pretty-print`: Shows the data in a pretty printed format which is more human readable than the tab-separated one.

Parameter: No parameters.

list-plates

Description: Lists all plates of a specified type. Various command line options may restrict or expand the list.

Usage:

```
prompt> bis list-plates [-a] [-b] [--pretty-print] [-l <number of lines>] [-n]
<plate type code> ["<pattern>" ...]
```

Options:

- `-a, --show-also-invalid`: Also invalid plates are shown if this flag is specified.
- `-b, --buffer`: If specified, the output on the console is stopped when the screen is full. By pressing space key or enter key output will be continued. By pressing 'q' output is quited.
- `-P, --pretty-print`: Shows the data in a pretty printed format which is

more human readable than the tab-separated one.

- `-l <number of lines>, --limit=<number of lines>:` Maximum number of lines to be shown.
- `-n, --no-experiment:` Only plates registered to an experiment are shown.
- `["<pattern>" ...]:` Restricts the query to those plates whose code matches the specified wild card pattern (* = any string, ? = any character). Pattern should be placed inside the quotation marks.

Parameter: `<plate type code>`: Code of the plate type. The command `list-plate-types lists` all available values.

`list-processing-instructions`

Description: Prints information about all processing instructions in a specified experiment. Additionally basic information about the experiment (as in `get-experiment-info`) are listed.

Usage:

```
prompt> bis list-processing-instructions <project code> <experiment code>
```

Other detailed information about the experiment can be obtained by commands:

- list-experiment-attachments
- list-processing-instructions
- list-experiment-datasets

Options:

- `-P, --pretty-print:` Shows the data in a pretty printed format which is more human readable than the tab-separated one.

Parameter:

- `<project code>`: Code of the project to which the experiments belong.
- `<experiment code>`: Codes of the experiments. Exactly one experiment has to be specified. Available experiments can be listed by the command `list-experiments`.

`list-projects`

Description: Lists all projects.

Usage:

```
prompt> bis list-project [--pretty-print]
```

Options:

- `-P, --pretty-print:` Shows the data in a pretty printed format which is more human readable than the tab-separated one.

Parameter: No parameters.

list-property-types

Description: Lists all property types.

Usage:

```
prompt> bis list-property-types [--pretty-print]
```

Options:

- -P, --pretty-print: Shows the data in a pretty printed format which is more human readable than the tab-separated one.

Parameter: No parameters.

list-plate-property-types

Description: Lists all property types for plate and also show the assignment, which property type is used for which assignment and if the property type is mandatory.

Usage:

```
prompt> bis list-plate-property-types [--pretty-print]
```

If you want to see the other assignment:

- list-experiment-attachments
- list-processing-instructions
- list-experiment-datasets

Options:

- -P, --pretty-print: Shows the data in a pretty printed format which is more human readable than the tab-separated one.

Parameter: No parameters.

list-control-layout-property-types

Description: Lists all property types for control layout and also show the assignment, which property type is used for which assignment and if the property type is mandatory.

Usage:

```
prompt> bis list-control-layout-property-types [--pretty-print]
```

If you want to see the other assignment:

- list-experiment-attachments
- list-processing-instructions
- list-experiment-datasets

Options:

- -P, --pretty-print: Shows the data in a pretty printed format which is more human readable than the tab-separated one.

Parameter: No parameters.

list-control-layouts

Description: Lists all control layouts.

Usage:

```
prompt> bis list-control-layouts [--pretty-print]
```

Options:

- `-P`, `--pretty-print`: Shows the data in a pretty printed format which is more human readable than the tab-separated one.

Parameter: No parameters.

list-vocabularies

Description: Lists all available vocabularies. A vocabulary is a collection of terms. This set can be used as a data type for a property type. To see terms available in a given vocabulary set, use `get-vocabulary-info` command.

Usage:

```
prompt> bis list-vocabularies [--pretty-print]
```

Options:

- `-P`, `--pretty-print`: Shows the data in a pretty printed format which is more human readable than the tab-separated one.

Parameter: No parameters.

login

Description: Authenticates the specified user and establishes a connection to the openBIS server. A connection will be automatically closed after some idle time.

An authenticated user has a role which may restrict the set of commands he or she is allowed to execute.

Usage:

```
prompt> bis login [-u <login name>] [-p <password>]
```

Options:

- `-u <login name>`, `--username=<login name>`: The login name of the user to be authenticated. If not specified the user will be asked.
- `-p <password>`, `--password=<password>`: The password of the user to be authenticated. If not specified the user will be asked. **This option should only be used in scripts calling this command.**

> Blank option value (uniquely composed of whitespace, empty ("")) are considered by the system as *not provided* and will throw an error.

Parameter: No parameters.

logout

Description: Closes the connection to the openBIS server.

Usage:

```
prompt> bis logout
```

Options: No additional options.

Parameter: No parameters.

Commands for Users and Administrators

add-experiment-attachment

Description: Uploads and attaches an arbitrary file to a specified experiment. The file name is used to identify the attachment and to download it with command `get-experiment-attachment`.

A new version can be uploaded by this command. All versions are available for downloading.

Usage:

```
prompt> bis add-experiment-attachment <project code> <experiment code> <file path>
```

Options: no additional options.

Parameter:

- `<project code>`: Code of the project to which the experiment belongs.
- `<experiment code>`: Code of the experiment. Available experiments can be listed by the command list-experiments.
- `<file path>`: Path to the file to be uploaded.

register-cell-plates

Description: Registers all cell plates specified in a file. For each cell plate a corresponding dilution plate has to be specified.

Usage:

```
prompt> bis register-cell-plates [-c <control layout code>] <cell plate file>
```

Options:

- -c <control layout code>, --control-layout=<control layout code>: Control layout code which will be used for all plates whose control layout has not been specified in the plate file.

Parameter:

- `<cell plate file>`: A tab-separated file with at least two columns: `code` and `parent`. The first column contains unique and currently not registered cell plate codes (maximum length: 20 characters). The second

column contains codes of already registered dilution plates (registered by the command `register-dilution-plates`). In an optional column `control_layout` individual control layout codes (registered by the command `register-control-layout`) can be specified for chosen cell plates. exampleCellPlates.txt or exampleCellPlatesWithControl.txt can serve as templates. Additional columns may or must appear if properties have been assigned to cell plates.

register-control-layout

Description: Registers a control layout specified in a file. This command specifies for certain wells a certain material of type `CONTROL`.

Note:

- The file name without the extension serves as the code of the control layout to be registered.
- Property `PLATE_GEOMETRY` is mandatory for control layouts and must be specified. Available plate geometries can be listed using `get-vocabulary-info PLATE_GEOMETRY` command.

Usage:

```
prompt> bis register-control-layout [-p <property code>=<property value>] [-f
<properties file>] <control layout file>
```

Options:

- -p <property code>=<property value>, --property=<property code>=<property value>: Setting the value of the property named <property code>. This can only be used for setting one property. If you want or have to set more than one property, use the `-f` option with a properties file.
- -f <properties file>, --properties-file=<properties file>: A tab-separated file with two columns (named `code` and `value`) for all properties to be set.

Parameter:

- `<control layout file>`: A tab-separated file with at two columns: `code` and `materialCode`. The first column contains well codes in the format `<row letter><two colum digits>`. The second column contains codes of already registered materials of type `CONTROL` (registered by the command register-materials). CL1.txt can serve as template. As the file name (without the extension) specifies the control layout code, only valid characters can be used (letters, numbers and underscore).

register-dilution-plates

Description: Registers all dilution plates specified in a file. For each dilution plate a corresponding master plate has to be specified. Optional a control layout can be associated.

Usage:

```
prompt> bis register-dilution-plates [-c <control layout code>] [-c <control
layout code>] <dilution plate file>
```

Options:

- -c <control layout code>, --control-layout=<control layout code>: Control layout code which will be used for all plates whose control layout has not been specified in the plate file.

Parameter:

- `<dilution plate file>`: A tab-separated file with at least two columns: `Dilution Plate` and `Master Plate`. The first column contains unique and currently not registered dilution plate codes (maximum length: 20 characters). The second column contains codes of already registered master plates (registered by the `command register-master-plate`). In an optional column `Control Layout` individual control layout codes (registered by the command `register-control-layout`) can be specified for chosen dilution plates. exampleDilutionPlates.txt or exampleDilutionPlatesWithControl.txt can serve as templates. Additional columns may or must appear if properties have been assigned to dilution plates.

register-experiments

Description: Registers one or more experiments as specified in the file.

Usage:

```
prompt> bis register-experiments [-d <root directory>] <experiment-type-code>
<experiment file>
```

Options:

- -d <root directory>, --root-dir=<root directory>: Root directory of <experiment file> and all files referred inside an experiment file.

Parameter:

- `<experiment-type-code>`: Code of an existing experiment type. The allowed experiment types can be obtained by the command `list-experiment-types`.
- `<experiment file>`: File specifying the experiments to be registered. A tab-separated experiment file contains the following columns (order is irrelevant):

Column name	Mandatory?	Description
project	yes	Code of a registered project.
code	yes	Experiment code which has to be unique in the specified project. It has to be no longer than 20 characters and cannot contain any dashes.

Column name	Mandatory?	Description
`study_object`	yes	Code of a registered material of the type specified by `study_object_type`.
`study_object_type`	yes	Code of a registered material type.
`cell_plates`	yes/no	File of cell plates associated with this experiment. The file path has to be relative to the above-mentioned root directory. The path has to be prefixed by '>'. The file has no header and contains one cell plate code per line. This column is optional if the `workflow-code` is `plates-on-demand` (i.e. old IMSB workflow). cellPlates.txt is an example.
`control_layout`	no/yes	Code of a registered control layout. This is a mandatory column if the `workflow-code` is `plates-on-demand`. Otherwise this column is not allowed.
`user.description`	no	A brief description of the purpose of the experiment to be registered.
`processing_instructions`	no	Processing instructions for the ETL Server. The instructions are specified in a file whose path is relative to the root directory. The path has to be prefixed by '>'. A processing instruction file is a tab-separated file with the mandatory columns `procedure_type` and `path`. The procedure type code has to be either `DATA_ACQUISITION` or `IMAGE_ANALYSIS`. Optional columns are `description` and `parameters`. The last one denotes the path (relative to the above-mentioned root directory and with prefix '>') of the processing parameters which will be uploaded. exampleProcessingInstructions.txt can serve as a template.

exampleExperiments.txt can serve as a template if the workflow-code is plates-on-demand. Otherwise exampleExperimentsWithCellPlates.txt can serve as a template.

register-reinfection-plates

Description: Registers reinfection plates specified in a file. For each reinfection plate a corresponding cell plate has to be specified. Optionally a control layout can be associated.

Usage:

```
prompt> bis register-reinfection-plates [-c <control layout code>] <reinfection
plate file>
```

Options:

- -c <control layout code>, --control-layout=<control layout code>: Code of the control layout which will be used for all reinfection plates whose control layout has not been specified in the reinfection plate file.

Parameter:

- `<reinfection plate file>`: A tab-separated file with at least two columns: `code` and `parent`. The first column contains unique and currently not registered reinfection plate codes (maximum length: 20 characters). The second column contains codes of already registered cell plates (registered by the command `register-cell-plates`). In an optional column `control_layout` individual control layout codes (registered by the command `register-control-layout`) can be specified for chosen cell plates. exampleReinfectionPlates.txt or exampleReinfectionPlatesWithControl.txt can serve as templates. Additional columns may or must appear if properties have been assigned to reinfection plates.

Commands for Administrators only

`assign-control-layout-property-type`

Description: Assigns a property type to a control layout. The assigned property type can be optional or mandatory. This can be changed by executing this command again.

If an optional property type is changed to a mandatory one it might be necessary to specify a file with values for this property. This tab-separated file has to provide a value for all control layouts which do not have a value for the property which had become mandatory.

Usage:

```
prompt> bis assign-control-layout-property-type [-f <file path>] [-g <global
default value>] [-m[=<mandatory flag>]] <property type code>
```

Options:

- `-f <file path>, --load-values=<file path>`: A tab-separated text file (specified by `<file path>` with two colums: `code` (here control layout code) and `value`.
- `-g <global default value>, --global=<global default value>`: A global default value for entities that are not specified in `<file path>`.
- `-m[=<mandatory flag>], --mandatory[=<mandatory flag>]`: If `<mandatory flag>` is `true` or missing the assign property type will be a mandatory one. If this option is not specified it will be optional.

Parameter:

- `<property type code>`: Code of the property type. The command `list-property-types` lists all available values.

`assign-experiment-property-type`

Description: Assigns a property type to an experiment type. The assigned property type can be optional or mandatory. This can be changed by executing this command again.

If an optional property type is changed to a mandatory one it might be necessary to specify a file with values for this property. This tab-separated file has to provide a value for all experiments of all projects which do not have a value for the property which had become mandatory.

Usage:

```
prompt> bis assign-experiment-property-type [-f <file path>] [-g <global default
value>] [-m[=<mandatory flag>]] <experiment type code> <property type code>
```

Options:

- `-f <file path>`, `--load-values=<file path>`: A tab-separated text file (specified by `<file path>` with three colums: `code` (here experiment code), `project` and `value`.
- `-g <global default value>`, `--global=<global default value>`: A global default value for entities that are not specified in `<file path>`.
- `-m[=<mandatory flag>]`, `--mandatory[=<mandatory flag>]`: If `<mandatory flag>` is `true` or missing the assign property type will be a mandatory one. If this option is not specified it will be optional.

Parameter:

- `<experiment type code>`: Code of the experiment type. The command `list-experiment-types` lists all available values.
- `<property type code>`: Code of the property type. The command `list-property-types` lists all available values.

`assign-material-property-type`

Description: Assigns a property type to a material type. The assigned property type can be optional or mandatory. This can be changed by executing this command again.

If an optional property type is changed to a mandatory one it might be necessary to specify a file with values for this property. This tab-separated file has to provide a value for all materials which do not have a value for the property which had become mandatory.

Usage:

```
prompt> bis assign-material-property-type [-f <file path>] [-g <global default
```

```
value>] [-m[=<mandatory flag>]] <material type code> <property type code>
```

Options:

- `-f <file path>`, `--load-values=<file path>`: A tab-separated text file (specified by `<file path>` with two colums: `code` (here material code) and `value`.
- `-g <global default value>`, `--global=<global default value>`: A global default value for entities that are not specified in `<file path>`.
- `-m[=<mandatory flag>]`, `--mandatory[=<mandatory flag>]`: If `<mandatory flag>` is `true` or missing the assign property type will be a mandatory one. If this option is not specified it will be optional.

Parameter:

- `<material type code>`: Code of the material type. The command `list-material-types` lists all available values.
- `<property type code>`: Code of the property type. The command `list-property-types` lists all available values.

`assign-plate-property-type`

Description: Assigns a property type to a plate type. The assigned property type can be optional or mandatory. This can be changed by executing this command again.

If an optional property type is changed to a mandatory one it might be necessary to specify a file with values for this property. This tab-separated file has to provide a value for all plates which do not have a value for the property which had become mandatory.

Usage:

```
prompt> bis assign-plate-property-type [-f <file path>] [-g <global default
value>] [-m[=<mandatory flag>]] <plate type code> <property type code>
```

Options:

- `-f <file path>`, `--load-values=<file path>`: A tab-separated text file (specified by `<file path>` with two colums: `code` (here plate code) and `value`.
- `-g <global default value>`, `--global=<global default value>`: A global default value for entities that are not specified in `<file path>`.
- `-m[=<mandatory flag>]`, `--mandatory[=<mandatory flag>]`: If `<mandatory flag>` is `true` or missing the assign property type will be a mandatory one. If this option is not specified it will be optional.

Parameter:

- `<plate type code>`: Code of the sample type. The command `list-plates-types` lists all available values.
- `<property type code>`: Code of the property type. The command `list-`

property-types lists all available values.

`invalidate-experiment`

Description: Invalidates a specified experiment. After invalidation the plates which were used in the invalidated experiment can be reused in another experiment.

Usage:

```
prompt> bis invalidate-experiment [-r <invalidation-reason>] <project code>
<experiment code>
```

Options:

- `-r <invalidation-reason>, --invalidation-reason=<invalidation-reason>`: Brief description why the experiment is invalidated. If the description contains spaces, it should be surrounded by quote signs, e.g. "operator error".

Parameter:

- `<project code>`: Code of the project to which the experiment belongs.
- `<experiment code>`: Code of the experiment. Available experiments can be listed by the command `list-experiments`.

`invalidate-plates`

Description: Invalidates one or more plates. The plates are specified by their codes in the command line and/or in a specified file.

Note:

- Control layouts can not be invalidated.
- All subordinated plates will also be invalidated. Plates hierarchy is as follows: master plate > dilution plate > cell plate > reinfection plate. For example, invalidation of a dilution plate means also invalidation of all cell plates derived from this dilution plate. In addition all reinfection plates of those cell plates are also invalidated.

Usage:

```
prompt> bis invalidate-plates [-r <invalidation-reason>] [-f <file with plate
codes>] [<plate code> ...]
```

Options:

- `-r <invalidation-reason>, --invalidation-reason=<invalidation-reason>`: Brief description why the plates will be invalidated.
- `-f <file with plate codes>, --plate-names-file=<file with plate codes>`: File with codes (one per line) of all plates to be invalidated. Invalid_plates.txt can serve as template.

Parameter:

- `<plate code>`: Zero or more plate codes.

register-master-plate

Description: Registers a master plate from a specified file.

Note:

- The name of the file without extension serves as the code of the master plate.
- Property `PLATE_GEOMETRY` is mandatory for master plates and must be specified. Available plate geometries can be listed using `get-vocabulary-info PLATE_GEOMETRY` command.

Usage:

```
prompt> bis register-master-plate [-p <property code>=<property value>] [-f
<properties file>] [-c <control layout code>] <master plate file> <material batch
code>
```

Options:

- `-c <control layout code>, --control-layout=<control layout code>`: Code of a control layout which is optionally associated with the master plate to be registered. The command `list-control-layouts` lists all available values.
- `-p <property code>=<property value>, --property=<property code>=<property value>`: Setting the value of the property named <property code>. This can only be used for setting one property. If you want or have to set more than one property, use the `-f` option with a properties file.
- `-f <properties file>, --properties-file=<properties file>`: A tab-separated file with two columns (named `code` and `value`) for all properties to be set.

- `--empty <empty-plate-name>`: should be used to create an empty master plate. When this option is given, master plate file should not be specified.

Parameter:

- `<master plate file>`: A tab-separated file with the following mandatory colums (see MP001-1.txt as an example):
    - `code`: The well code specified in the format `<row letter><two colum digits>`.
    - `material`: The code of the material of type specified in column `material_type`.
    - `material_type`: The code of a material type. The command `list-material-types` lists all available values.
- `<material batch code>`: An arbitrary text specifying the instance of all materials associated in the master plate to be registered.

register-materials

Description: Registers materials from a specified file.

Usage:

```
prompt> bis register-materials <material type code> <material file>
```

Options: No additional options.

Parameter:

- `<material type code>`: The code of the type of materials to be registered. The command `list-material-types` lists all available values.
- `<material file>`: Tab-separated file which contains one material and its properties per line. It has the mandatory column `code` which contains the material code which has to be unique for the specified material type (maximum length: 20 characters). The column `inhibitor_code` contains the code of the material which is inhibited by the material to be registered. This is a mandatory column if the material type is `OLIGO`. Additional columns are determined by the property types assigned to the specified material type. The command `get-material-type-info` lists all available columns. The name of the column in the material file has to be the same (ignoring case) as the property type code. Columns of non-mandatory properties are optional in the material file. See exampleGenes.txt, exampleOligos.txt, controls.txt, viruses.txt for example input files.

register-project

Description: Registers a new project.

Usage:

```
prompt> bis register-project <project code>
```

Options: No additional options.

Parameter:

- `<project code>`: Code of the new project. It has to be unique and not more than 20 characters long.

register-property-types

Description: Registers property types from a specified file.

Usage:

```
prompt> bis register-property-types <property types file>
```

Options: No additional options.

Parameter:

- `<property types file>`: A tab-separated file with the following mandatory columns (see examplePropertyTypes.txt for an example):
    - `code`: Code (i.e. name) of the property type to be registered. It has to be unique and not more than 20 characters long.

- `label`: Label to be used for human readable output.
- `description`: A brief description.
- `data_type`: Code of the type of data associated with the property type to be registered. The following data types are available:

Data type code	Description
VARCHAR	Text with not more than 1024 characters.
INTEGER	32-bit integer number.
REAL	Double precision floating point number.
BOOLEAN	Boolean value: Either `true` or `false` ignoring case.
TIMESTAMP	Time stamp (i.e. date and time).
CONTROLLED VOCABULARY	This type is useful when you want to have a text property, but its values should be restricted to a certain set of controlled vocabularies. In the `vocabulary` column the name of the set should be specified if this type is used.

When the property has a `TIMESTAMP` type, it's value can be specified in following formats:

- full format with time zone (yyyy-MM-dd HH:mm:ss Z), e.g. 2007-12-24 16:59:59 +0200.
- without a time zone (yyyy-MM-dd HH:mm:ss), e.g. 2007-12-24 16:59:59. The local timezone will be assumed.
- without specifying seconds (yyyy-MM-dd HH:mm), e.g. 2007-12-24 16:59.
- specyfing just a date, without the time (yyyy-MM-dd), e.g. 2007-12-24. The midnight of that day in the local time zone will be assumed as the time. where yyyy is the year, MM is the month (as number), dd is the day in the month, HH is the hour (0-23), mm is the minute, ss is the second, and Z is the time zone.

- `vocabulary`: Optional, should be specified only when `data_type` is CONTROLLEDVOCABULARY. The value should be the name of the controlled vocabulary set which restricts the set of allowed values (see examplePropertyTypesWithVocabularies.txt for an example). The available vocabularies can be displayed by `list-vocabularies` command.

register-vocabulary

Description: Registers controlled vocabulary set with given terms. To browse

existsting controlled vocabulary set see `list-vocabularies` and `get-vocabulary-info` commands.

Usage:

```
prompt> bis register-vocabulary <vocabulary-name> <vocabulary-terms-file>
```

Options:

- `--add-terms`: If specified, the terms from `<vocabulary-terms-file>` will be added to already existing `<vocabulary-name>`.

Parameter:

- `<vocabulary-name>`: Name of the controlled vocabulary set
- `<vocabulary-terms-file>`: The file with vocabulary terms. Each term should be in one line. There should be no header. The following file can serve as an example: exampleVocabularyTermsOrganism.txt

unassign-control-layout-property-type

Description: Unassigns a property type from control layout.

Note: This automatically deletes all values of the specified property type for all control layouts.

Usage:

```
prompt> bis unassign-control-layout-property-type [--force] <property type code>
```

Options:

- `--force`: Attempts to unassign the property type without asking the user for confirmation.

Parameter:

- `<property type code>`: Code of the property type. The command `list-property-types` lists all available values.

unassign-experiment-property-type

Description: Unassigns a property type from an experiment type.

Note: This automatically deletes all values of the specified property type for all experiments of the specified type.

Usage:

```
prompt> bis unassign-experiment-property-type [--force] <experiment type code>
<property type code>
```

Options:

- `--force`: Attempts to unassign the property type without asking the user for confirmation.

Parameter:

- `<experiment type code>`: Code of the experiment type. The command

`list-experiment-types` lists all available values.
- `<property type code>`: Code of the property type. The command `list-property-types` lists all available values.

unassign-material-property-type

Description: Unassigns a property type from a material type.

Note: This automatically deletes all values of the specified property type for all materials of the specified type.

Usage:

```
prompt> bis unassign-material-property-type [--force] <material type code>
<property type code>
```

Options:
- `--force`: Attempts to unassign the property type without asking the user for confirmation.

Parameter:
- `<material type code>`: Code of the material type. The command `list-material-types` lists all available values.
- `<property type code>`: Code of the property type. The command `list-property-types` lists all available values.

unassign-plate-property-type

Description: Unassigns a property type from a plate type.

Note: This automatically deletes all values of the specified property type for all plates of the specified type.

Usage:

```
prompt> bis unassign-plate-property-type [--force] <plate type code> <property
type code>
```

Options:
- `--force`: Attempts to unassign the property type without asking the user for confirmation.

Parameter:
- `<plate type code>`: Code of the plate type. The command `list-plate-types` lists all available values.
- `<property type code>`: Code of the property type. The command `list-property-types` lists all available values.

Command line options

General Options

Short Option	Long Option Name	Description
-h	--help	Shows a help text with a brief description of all options and arguments.
-s \<base URL\>	--server-base-url=\<base URL\>	The base URL of the openBIS server.

Command specific options

Short Option	Long Option Name	Description	Command(s)
	--force	Attempts to unassign the property type without asking the user for confirmation.	`unassign-material-property-type`
	--add-terms	If specified, terms read from the input file will be added to existing vocabulary.	`register-vocabulary`
-a	--show-also-invalid	Also invalid plates are shown if this flag is specified.	`get-plate-locations`, `list-cell-plates`, `list-dilution-plates`, `list-experiments`, `list-master-plates`, `list-reinfection-plates` `list-experiment-plates` `list-experiment-datasets`
-b	--buffer	If specified, the output on the console is stopped when the screen is full.	`get-plate-info`, `list-cell-plates`, `list-control-layouts`, `list-dilution-plates`, `list-master-`

Short Option	Long Option Name	Description	Command(s)
			`plates`, `list-materials`, `list-reinfection-plates`
-c <control layout code>	--control-layout=<control layout code>	Control layout code	`register-cell-plates`, `register-dilution-plates`, `register-master-plate`, `register-reinfection-plates`
-d <root directory>	--root-dir=<root directory>	Root directory of <experiment file> and all files	`register-experiments`
-e <experiment code>	--experiment=<experiment code>	Code of an experiment.	`get-plate-locations`
-f <file path>	--load-values=<file path>	A tab-separated text file (specified by <file path> with two colums.	`assign-material-property-type`
-f <file with plate codes>	--plate-names-file=<file with plate codes>	File with codes (one per line) of all plates to be invalidated.	`invalidate-plates`
-g <global value>	--global=<global value>	Global default value for entities that are not specified in an external file.	`assign-experiment-property-type`, `assign-material-property-type`, `assign-sample-property-type`
-i	--gene-id	Parameter <gene-symbol or id> is interpreted as gene id if this flag is specified.	`get-plate-locations`
-l <number of lines>	--limit=<number of lines>	Maximum number of lines to be shown.	`list-cell-plates`, `list-control-layouts`, `list-dilution-plates`, `list-master-plates`, `list-materials`, `list-reinfection-plates`
-m[<mandatory flag>]	--mandatory[=<mandatory flag>]	If <mandatory flag> is true or missing the assign property type will be a mandatory one.	`assign-material-property-type`

Short Option	Long Option Name	Description	Command(s)
-n	--no-experiment	Only plates registered to an experiment are shown.	`list-cell-plates`, `list-reinfection-plates`
-o \<study object code\>	--study-object=\<material code\>	Material code of the object to be studied.	`get-plate-locations`, `list-experiments`
-o \<organization code\>	--organization-code=\<organization code\>	Code of an organization.	`login`
-p \<project code\>	--project=\<project code\>	Code of a project.	`get-plate-locations`, `list-experiments`
-p \<property code\>=\<property value\>	--property=\<property code\>=\<property value\>	Setting the value of the property \<property code\>.	`register-control-layout`, `register-master-plate`
-p \<password\>	--password=\<password\>	The password of the user to be authenticated.	`login`
-P	--pretty-print	Shows the data in a pretty printed format which is more human readable than the tab-separated one.	`get-control-layout-info`, `get-material-type-info`, `get-plate-info`, `get-vocabulary-info`, `list-cell-plates`, `list-contacts`, `list-control-layouts`, `list-dilution-plates`, `list-experiment-types`, `list-experiments`, `list-master-plates`, `list-material-types`, `list-materials`, `list-plate-types`, `list-projects`, `list-property-types`, `list-reinfection-plates`, `list-vocabularies`, `list-experiment-attachments`, `list-experiment-datasets`, `list-processing-`

Short Option	Long Option Name	Description	Command(s)
			`instructions`
-r \<invalidation-reason>	--invalidation-reason=\<invalidation-reason>	Brief description why the experiment is invalidated.	`invalidate-experiment`, `invalidate-plates`
-t \<time interval>	--time-interval=\<time interval>	A query restriction on the time interval.	`get-plate-locations`, `list-experiments`
-u \<login name>	--username=\<login name>	The login name of the user to be authenticated.	`login`
-v \<version number>	--attachmentVersion=\<version number>	Version of the requested file.	`get-experiment-attachment`

Time interval specification

For some commands of the command line client a time interval can be specified by the option `-t`. This time interval has to be specified as follows

```
[<from>]:[<until>]
```

A missing `<from>` specification means minus infinity. A missing `<until>` specification means now.

The time itself is specified either absolutely in the form

```
<4 digit year>-<2 digit month>-<2 digit day>
```

or relatively in the form

```
now-<number><unit>
```

where `<unit>` is either

Unit	Description
h	hour
d	day
w	week = 7 days
m	month = 30 days
y	year = 365 days

Examples:

| Example	Meaning
`:2007-12-10`	Before 10th December 2007
`2007-08-01:2007-12-01`	Between 1st August 2007 until 1st December 2007
`now-2d:`	The last two days
`now-3m:now-5h`	The last three months but not in the last 5 hours

Biological Data Standard

This page describes the Biological Data Standard (BDS) used in the CISD software systems for data not stored in the database.

Motivation and Goals

The ETL Server receives sets of large measurement or analysis data. These data sets are usually to large to be stored completely in the database. When the data set is registered at the application server only meta data are stored in the database. What to do with data? They has to stored in the file system in a way that

- User can get an identifier for the corresponding experiment meta data stored in the database.
- Some of the meta data are available even if the database isn't accessible. This is not only for the sake of the user but also for some external software doing further analysis of the data.
- The data should be organized in a standardized way independent of the usually non-standard original organization of the data. For each domain (e.g. high-content screening, proteomics mass-spectrometry, etc.) the standard is of course different.
- The data should be organized in a way as they are usually needed by analysis programs.
- The way data is stored and organized should be consistent across different domains. For example, a date/timestamp should be always stored in the same format.
- The standard should be versioned. If possible new versions should be backwards compatible.

The solution is to store the data and additional meta data in a container (called **BDS Container**). The ETL Server creates such a container from the received data. The relative path to the container is registered in the database. The application server (or any other program) can access the data in the BDS Container.
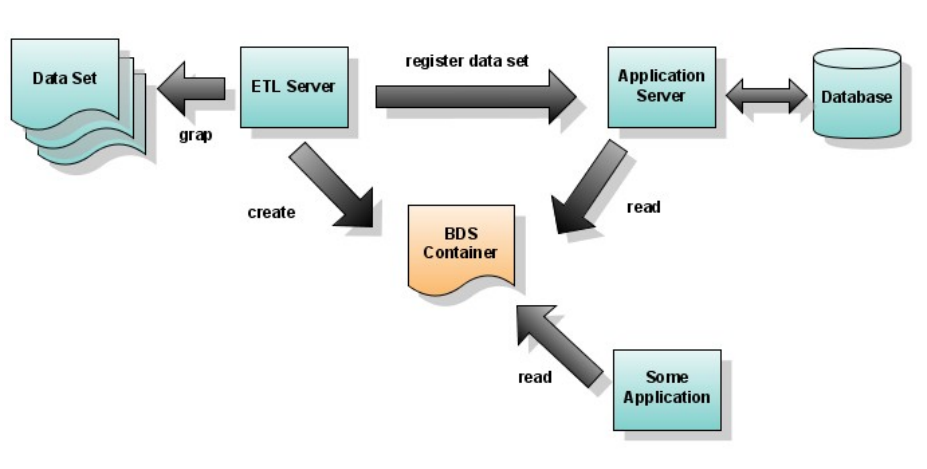
This wiki space contains all the specification of the Biological Data Standard. There is also Java API developed for creating and reading BDS Containers.

Concepts

Important concepts and terms of BDS are defined here.

Data, Metadata and Annotation

- **Data** are any kind of data produced by some measurements, analyses or calculations.
- **Metadata** are data about the above mentioned data. Metadata are partially bundled with the data. Some metadata are stored in the openBIS database. If this is the case the metadata bundled with the data contains information which allows to retrieve the missing part of the metadata from openBIS.
- **Annotation** are additional data which are not required to make sense of the data but might be interesting anyway.

Directory, File, and Link

The data is organized hierarchical as a **tree**. There are three different types of **nodes**:

- **Directory**: A node with children.
- **File**: A leaf node (i.e. childless node). A file has a **value** which can be either textual or binary data.
- **Link**: A reference to another node of any type anywhere in the tree. Cyclic dependencies are not allowed. An additional file with the relevant information of all links has to be provided in order to restore broken links.

All nodes have in common that they have a **name** which has to be unique in their parent directory. The tree itself is a nameless root directory. How structures are represented in this document see Structure Representation Syntax and Semantics.

Container

A **container** is a physical representation of a tree. The following representations will be supported:

- **File system**: Directories and files are easily translated into they corresponding file system representatives. For links hard link should be used if possible. All files (except the original data) should be ASCII files and optimized for reading by programs (e.g. using TAB-separated value format for tables)
    - Advantage:
        - Easy access by scripts and programs in any language (Perl, Python, Java, C, Matlab, R, S, Spotfire, etc.).
    - Disadvantages:
        - Not efficient concerning memory consumption
        - Bad performance in Hierarchical Storage Management (HSM)

solutions because of lots of small files
- No platform-independent solution for links

- **HDF5**: A BDS Container is a HDF5 file. Directories are represented by HDF5 groups. Files are represented as binary data like vectors, matrices etc.
  - Advantages:
    - Efficient use of memory and CPU resources
    - Suitable for HSM solutions because only one (big) file has to be handled.
    - Available on many platforms
    - Independent from the file system of the platform
  - Disadvantages:
    - Needs HDF5 library.
    - Programming non-trivial.

To over come the disadvantages of an HDF5 container a conversion program will be provided which converted one BDS Container representation into another one.

Structure

A **structure** is a definition of how the data are organized in a tree. It is a kind of template. It has a version which is stored in the root directory.

Format

As **format** is a set of definitions needed in order to make sense of the data itself.

Even though this is a rather vague definition a concrete format is very specific. It may consists of:

- directory layout (this means for a given format one knows what sub-directories to expect in the /data directory
- what entity the data set refers to (e.g. a screening plate)
- what entities sub sets contained in this data set refers to (e.g. wells or tiles on a screening plate)
- whether it is raw or derived data
- if applicable: what coordinate system is used to organize the data (e.g. wells and tiles for RNAi images)
- if applicable: what experimental condition description system is used to organize the data (e.g. the channels of a microscope); note that this is usually kind of a template that needs to have parameters to be filled in (e.g. the wave-lengths of the channels)
- the format(s) of a single entity in the data set (e.g. a tiff image or a float vector)

Format Variant

A format can contain variants. A **variant** leaves some room to customize a format in that it allows to decide on parts of the format on a case by case basis. All variant

parameters need to be fully specified for a given instance of a data set of a given format.

An example of a variant is whether a raw data set contains the original data as measured by the instrument or only the standardized form. Another example could be the image format of RNAi screening plates, i.e. whether they are tiff or png.

Note that the number of variants should be kept to a minimum, because standards with lots of variants are difficult to implement and are quite often implemented incompletely (just assuming that it will be always one of the variants and neglecting the other variants).

Format Building Block

A **format building block** is a re-usable format (e.g. the coordinate system for screening plate). It has its own page in the wiki an can be referred by other formats or format building blocks.

Version

Because the organization of data will change over time a structure and a format will have a **version**. It is specified by two non-negative integers: **major version** and **minor version**. Whenever changes to a definition (structure or format) need to be made the corresponding version will be increased. There are two types of increases:

- New minor version: The minor version number is increased by one. A minor version must be *backward compatible* to all previous minor versions of the same major version. A version is called backward compatible if software written for an earlier version can also handle any data structured/formatted in accordance with the newer version. Backward compatible versions typically extend the structure/format by additional stuff.
- New major version: The major version number is increased by one and the minor version is set to zero. A major version is not required to be backward compatible to any previos version.

The structure of the version information is as follows:

```
version/
    major
    minor
```

The version structure itself is not versioned.

In texts a version should be written as follows:

```
V<major>.<minor>
```

For example, V2.1 means major version 2 and minor version 1.

Data types

Flags

Flags can only have one of the following values: `TRUE` (meaning `yes`) or `FALSE` (meaning `no`).

XFlags

Extended Flags (XFlags) can only have one of the following values: `TRUE` (meaning `yes`), `FALSE` (meaning `no`) or `UNKNOWN` (meaning it's value is not known to the system).

Enumerations

The allowed items of an enumeration are written in capital letters. No spaces will be used, but instead an underscore. Only characters and numbers are allowed here. A correct example is `ORIGINAL_DATA`.

Dates

Dates are always given in the following format:

```
yyyy-MM-dd HH:mm:ss Z
```

where:

- `yyyy` is the year (4 digits)
- `MM` is the month (2 digits}
- `dd` is the day of the month (2 digits)
- `HH` are the hours (counting from 0 to 23, 2 digits)
- `mm` are the minutes (2 digits), `ss` are the seconds (2 digits)
- `Z` is the time zone relative to GMT (4 digits and a sign symbol in accordance with [RFC 822](#). For central Europe it would be +0100)

Lists

Lists are ordered collections of items. In a *file system container* a list will be represented as a file which contains one item per line, i.e. different items of the list will be separated by a `LF` character.

Units

When a quantity requires a unit, this unit has to be specified in the format. For example, a format will specify whether a wavelength is given in `nanometers` or in `angstroms`. There will be no shortcuts for units appended to the quantities itself.

Structure Representation: Syntax and Semantics

In this documentation a textual representation of the tree or a subtree is used. Its syntax and semantics is best explained by an example:

```
version/
   major = 2
   minor = 1
data/
   original/
      xyz08-001.tiff
      xyz08-002.tiff
   standard/
      xyz08/
         001.tiff -> data/original/xyz08-001.tiff
         002.tiff -> data/original/xyz08-002.tiff
metadata/
```

Here are the rules:

- Directories end with '/'.
- Child nodes are intended by 3 spaces relative to their parent directory.
- If a file is followed by an equals sign '=' it means that the file has the value as specified what follows the equal sign.
- Links have a '->' after their name followed by the full path to their referring node.

Naming Conventions

Directories, files, and links created to organize the data have to fulfill a naming convention. The name of original data files and folders are left unchanged even though they are not conform to the naming convention.

Rules:

- A name contains only lower-case letters, decimal digits, and '_'.
- A file or a link referring to a file can contain one dot character '.'.
- A name starts either with a letter or a digit.

BDS Structure V1.0

> With all definitions we follow the **"principle of least surprise"**. So if for example a standard defines one way to store a structure for wells, and such a structure is needed in two places, they are supposed to look the same.

Template

```
version/
   major = 1
   minor = 0
data/
   original/
   standard/
metadata/
   data_set/
      code = 20080402142616324-198
      production_timestamp = 2008-04-02 08:03:23 +0100
      producer_code = M1
      observable_type = HCS_IMAGE
      is_measured = TRUE
      is_complete = TRUE
      parent_codes = 20080402142616674-197
                     20080402142656712-196
   format/
      version/
         major = 2
         minor = 1
      code
      variant
   experiment_identifier/
      instance_code = IMSB
      group_code = LP
      project_code = NEMO
      experiment_code = EXP1
   experiment_registration_timestamp = 2007-06-23 21:07:08 +0100
   experiment_registrator/
      first_name = John
      last_name = Doe
      email = John.Doe@organization.org
   sample/
      type_description = Screening Plate
      type_code = CELL_PLATE
      code = CP001A-3AB
   md5sum/
      original
   standard_original_mapping = ch1/row3/col4/row1_col1.tiff   I
NEMO.EXP1::CP001A-3AB/xyz08-001.tiff
                              ch1/row3/col4/row1_col2.tiff   I
NEMO.EXP1::CP001A-3AB/xyz08-002.tiff
                              ch1/row3/col4/row1_col12.tiff  I
NEMO.EXP1::CP001A-3AB/xyz08-012.tiff
   parameters/
      plate_geometry/
      well_geometry/
      contains_original_data = TRUE
annotations/
   channel1/
      wavelength = 123
   channel2/
      wavelength = 300
```

Explanations

The meaning of the various nodes in this structure are explained in the following sections. For more information about terms, concepts, and structure representation see Introduction and Concepts and Structure Representation Syntax and Semantics.

`version`

This directory specifies the version of the structure used.

`data`

This directory contains the measured or computed data.

`data/original`

This directory will contain the original data as they have been received from the ETL server. These files are mainly data files (e.g. images) but they also could be accompanied by some log or other helper files that should not be mapped in the `standard` directory. Although one could argue that we have redundancy between the name of the folder received `NEMO.EXP1::CP001A-3AB` and the values found in experiment identifier, we want to keep these original data as *original* as possible.

Example

Assuming the following folder has been received:

```
NEMO.EXP1::CP001A-3AB/
    xyz08.log
    xyz08-001.tiff
    xyz08-002.tiff
    xyz09.log
    xyz09-001.tiff
```

It will be imported as follows:

```
data/
    original/
        NEMO.EXP1::CP001A-3AB/
            xyz08.log
            xyz08-001.tiff
            xyz08-002.tiff
            xyz09.log
            xyz09-001.tiff
```

`data/standard`

This directory contains the data structured in a standard way defined by the format. In order to save storage space data files are not duplicated but referred by hardlinks.

To continue the example from above the structure in `data/standard` might look like as follows:

```
data/
    original/
        NEMO.EXP1::CP001A-3AB/
            xyz08.log
            xyz08-001.tiff
```

```
            xyz08-002.tiff
            xyz09.log
            xyz09-001.tiff
    standard/
        ch1/
            row3/
                col4/
                    row1_col1.tiff ->
data/original/NEMO.EXP1::CP001A-3AB/xyz08-001.tiff
                    row1_col2.tiff ->
data/original/NEMO.EXP1::CP001A-3AB/xyz08-002.tiff
                    row1_col3.tiff ->
data/original/NEMO.EXP1::CP001A-3AB/xyz09-001.tiff
```

> The actual structure in `data/standard` depends on the format which is specified by `metadata/format/format_code`. It also specifies the link label format and answers questions like: which part comes first, space coordinates or experiment conditions?

metadata

This directory contains some metadata of the data stored in `data` and informations which allows to query *openBIS* for additional metadata.

metadata/data_set

Nodes in this directory specify information about the data set.

- `metadata/data_set/code`: is a unique identifier that allows to find the data set in the database.
- `metadata/data_set/production_timestamp`: Date, in the default format, which provides the information when the data set has been created (either measured or calculated).
- `metadata/data_set/producer_code`: identifies the "device" that produced the data set (which can be a measurement device like a microscope or a software program).
- `metadata/data_set/observable_type`: is a code that describes the type of data that is stored in this standard. It is an Enumeration and possible values are `HCS_IMAGE` or `HCS_IMAGE_ANALYSIS_DATA`. Usually one BDS standard will always have one observable type.
- `metadata/data_set/is_measured`: Flag specifying whether the data set has been *measured* from a sample (`TRUE`) or whether it has been *derived* by means of some calculation from another data set (`FALSE`).
- `metadata/data_set/is_complete`: Xflag specifying whether the data set is complete or not, i.e. whether it contains all files that it is expected to contain. Not all storage processors can assess whether a data set is complete. Thus the value of can be `TRUE`, `FALSE` or `UNKNOWN`.
- `metadata/data_set/parent_codes`: is only non-empty if `is_measured` is `FALSE`. In this case, it will contain a List) of codes that this data set has been derived from.

`metadata/format`

Nodes in this directory define the format of the data in `data/standard`.

- `metadata/format/code`: specifies the code of the format definition.
- `metadata/format/version`: specifies the format version. It is completely independent to the structure version of the data structure.
- `metadata/format/variant`: This is optional in general (though for a given data standard it will be either alway present or always absent, depending on whether the data standard has variants or not) and specifies a variant of the format. Its value has to be from a restricted vocabulary which is specified in the format definition (defined by the format code and version). A format variant can be thought of a name space of the `metadata/parameters` directory, so a certain value of `format_variant` will tell the user what entries he or she can expect in `metadata/parameters`. Note that there may be flags in `metadata/parameters` which may indicate further variations to a format, as defined in the format definition.

Depending on the format code this directory may contain additional nodes.

`metadata/experiment_identifier`

The *experiment identifier* is used to get all the metadata from the *openBIS* database. Four codes are needed to specify it uniquely:

- `instance_code`: identifier of the openBIS server instance
- `group_code`: identifier of the research group / lab
- `project_code`: identifier of the research project that this experiment has been performed in
- `experiment_code`: identifier of the experiment

`metadata/experiment_registration_timestamp`

The date, in the default format (see Dates), when the experiment specified by `metadata/experiment_identifier` was registered.

`metadata/experiment_registrator`

The name of the registrator of the experiment. It contains the mandatory files `first_name`, `last_name`, and `email`.

`metadata/sample`

The *sample* that the values of this data set are either *measured from* or *derived from* via processing, e.g. a screening plate.

- `type_code`: is a unique identifier of the type of the sample. It is an Enumeration and current possible values are `CELL_PLATE` or `REINFECT_PLATE`.

- `type_description`: is a free text description of the type of sample intended to be used by the human reader only. Note that these data are redundant, because a data format implicitly specifies this type.
- `code` can be tracked back to the database, e.g. `CP001A-3AB` is a barcode of a screening plate.

metadata/md5sum/original

The `original` folder should be checked for integrity (i.e., to verify that a file has not changed as a result of file transfer, disk error, human meddling, etc.). This can be achieved by calling `gmd5sum` on *Mac OS X*. For an introduction to `md5sum` and an 'HOWTO' to install `gmd5sum` on *Mac OS X*, have a look at http://www.answers.com/topic/md5sum. For a fast *Java* implementation of MD5, see http://www.twmacinta.com/myjava/fast_md5.php.

Typically you will perform following command to create the desired `md5sum` file (in our case for the `original` directory):

```
find . -type f -exec gmd5sum {} \; > original
```

The output file (`original`) will have the following format:

```
a7efaa48fdf59e0ee4308df7e5252bf4   NEMO.EXP1::CP001A-3AB/xyz08.log
7687bc40fe2af4f80c7f8fede5c9e16c   NEMO.EXP1::CP001A-3AB/xyz08-001.tiff
b157aa7e1909089f04cf58a1bb752b54   NEMO.EXP1::CP001A-3AB/xyz08-002.tiff
934df91fe90108a9a12858aeed23e779   NEMO.EXP1::CP001A-3AB/xyz09.log
05098dc194e09e24334298c32ca87538   NEMO.EXP1::CP001A-3AB/xyz09-001.tiff
```

Between the checksum and the file path are two space characters. The file path (with '/' as name separator) is relative to `data/original`.

metadata/standard_original_mapping

This is a file containing information about the relationship between a file in `data/standard` and `data/original`. Each line of this file contains three items separated by a TAB character:

1. Path in `data/standard`: Existing path. Occurs only once in this file. The file path separator is system independent and is '/'.
2. Type of Relationship: `I` or `T`.
3. Path in `data/original`: this can be any valid and existing path (may contain TAB characters). The file path separator is system independent and is '/'.

The meaning of the type of relationship is as follows:

- `I` (IDENTICAL): The first path is pointing to a hardlink in `data/standard` and the second path is pointing to the orginale file in `data/original`. Such mapping entry is only necessary if the hardlink is broken for some reason.
- `T` (TRANSFORMED): Both paths pointing to files with different content. The first path is pointing to a file that is the transformation resp. standardization form of the file referred by the second path. This is a typical situation for derived data because we want to standardize these data. On the other one may

still want to know the corresponding original data file because it could contain information which is lost due to standardization.

Note: This file can be empty.

`metadata/parameters`

The `parameters` sub-directory contains all parameters that are specific to a certain data format. Consequently, the content of this directory needs to be defined in the data format definition. With *parameters* we mean the information one needs to make sense of the *format* of the data in `data/` directory. An example of parameters is the plate dimension and the well dimension for tiled plate-based data.

For information that is needed to make sense of the *content* of the data, see `annotations/` instead. So for example for plate-based data the channel wavelengths or the mapping table from biological materials to wells wouldn't belong to `metadata/standard` but to `annotations/`.

`annotations`

The directory `annotations/` is reserved for all information that is specific to a data standard but that is neither measured data (which is found in `data/`) nor is it required to make sense of the data structure in `data` (which can be found in `metadata/parameters`. Rather it is information that is needed to make sense of the content of the data or is needed for quality control. An example of information that should be placed here is the wavelength of a fluorescent measurement or the id of the device that was used to measure it.

The content of the `annotations/` directory is completely defined in each data standard.

Unknown (UNKNOWN) 1.0

This format is used when nothing is known about the format of the data. In this case the directory `data/standard` of the structure does not exists (see BDS Structure V1.0).

HCS IMAGE 1.0

HCS (High-Content Screening) with Images

For the general description of the biological data standards, please refer to BDS Structure V1.0

Description of data stored in this format

This data standard is supposed to store image data from High Content Screening

(HCS) experiments. Such experiments are performed on *plates* which consist of *wells* containing the system to be tested. Typical plate geometries have 8x12 or 16x24 wells.

In HCS experiments, an automated microscope takes images of every well. Since in the common case the display window of the microscope is too small to cover a complete well in the required magnification, the microscope takes several images of every well, that, when put together, depict the complete well. Such images which show only a part of a well are called a *tile* in the following.

HCS fluorescence microscopes are typically able to measure more than one color (or wavelength). Common devices can measure up to 4 colors in 4 *channels*. One *channel* thus represents one color which is described by its wavelength.

One data set as described below represents the image data taken from one plate.

Directory `metadata/data_set`

For the description of the `metadata` directory, please be referred to metadata.

The `metadata/data_set/observable_type` for this format is `HCS_IMAGE`.

Directory `metadata/format`

For the description of the `metadata` directory, please be referred to metadata.

The `metadata/format/format_code` for this format is `HCS_IMAGE`, the `metadata/format/version` is 1.0 (for the structure of format specifiers, see metadata/format).

This format does not have any variants.

Directory `metadata/parameters`

The `metadata` subdirectory `parameters` contains the following children (all parameters are mandatory):

1. **`plate_geometry`**. This is the geometry of a *plate*, i.e. the number of wells per column and per well. A typical value is 16x24.
2. **`well_geometry`**. This is the geometry of a *well*, i.e. the number of tile images per column and per well. A typical value is 3x3.
3. **`number_of_channels`**. It specifies the number of *channels* we are going to find here.
4. **`contains_original_data`**. Flag specifying whether the `data` directory contains the original data or not.
   In the later case only the standardized data are available (located in `data/standard` directory). No file is located in `data/original` directory and the mapping file in `metadata/standard_original_mapping` is thus empty.

An example structure of the `metadata/parameters` hierarchy looks like this:

```
metadata/
    parameters/
        plate_geometry/
            rows = 16
            columns = 24
        well_geometry/
            rows = 3
            columns = 3
        contains_original_data = TRUE
        number_of_channels = 2
```

Directory `annotations`

The directory `annotations` contains the following children (all entries are mandatory):

1. **`channelN`**. In this format, the only setting is the `wavelength` of the channel. It is provided in the unit of *nanometers*. We may have **one or more** *channels* (see `metadata/parameters/number_of_channels` for how many there are). The channels are serially numbered starting with 1.

An example structure of the `annotations` hierarchy looks like this:

```
annotations/
    channel1/
        wavelength = 123
    channel2/
        wavelength = 200
```

Directory `data/standard`

In the `data/standard` all images of one plate are stored, hierarchically ordered by:

1. *Channel* denoted by `channelL`
2. *Plate Row* denoted by `rowN`
3. *Plate Column* denoted by `columnM`

One directory thus corresponds to one well and one color. It contains image data for all tiles of the particular well and color. Each image is named according to the schema `rowN_columnM`, where `N` and `M` are the row and column number of the tile. The image is stored in standard **tiff** format.

   The numbering of rows and columns always starts with 1 (rather than 0).

Example (here in the case where original data are stored, the dependency between standardized data and original data are symbolically depicted by "`->`"):

```
data/
    standard/
        channel1/
            row1/
                column1/
                    row1_column1.tiff -> NEMO.EXP1::CP001A-3AB/xyz08-001.tiff
                    row1_column2.tiff -> NEMO.EXP1::CP001A-3AB/xyz09-001.tiff
        channel2/
            row1/
                column1/
                    row1_column1.tiff -> NEMO.EXP1::CP001A-3AB/xyz08-002.tiff
                    row1_column2.tiff -> NEMO.EXP1::CP001A-3AB/xyz09-002.tiff
```

```
                ...
```

The reference which the dependency points to is relative to the `original` directory. Note that depending on the value of `contains_original_data` flag you will have here (hard) links (as depicted now) or independent (standardized) data sets.