TranSMART ETL Guide

eTRIKS

efpia

Platform and services for data staging, exploration, and use in translational research

Copyright © 2015 eTRIKS. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is available at http://www.gnu.org/licenses/fdl.html.

Contents

1 Introduction

This document is intended to be a guide for performing Extraction, Transformation and Loading (ETL) processes in tranSMART. It covers the ETL pipeline on Windows and Linux systems, for tranSMART instances running on a Postgres database. Chapter one in the Dataset Explorer ETL Guide [1] gives an excellent discussion on how to plan and build your ontologies, which will determine how data appears in tranSMART. This guide aims to combine the dispersed information on ETL from various sources, such as the tranSMART Foundation Wiki [2] and the ETL Guide mentioned above. **This guide assumes that you have a working instance of tranSMART installed.**

2 Dependencies

You will need to have several pieces of software installed before you can begin the ETL process. The next sections describe these dependencies and how to install them. The installation commands provided here for Linux systems work on Debian based distributions. For other distributions, please refer to your distribution's software installation documentation.

2.1 Pentaho Data-Integration

The ETL process makes use of the Pentaho Data-Integration software suite version 4.4.0. It is available at `http://sourceforge.net/projects/pentaho/files/Data%20Integration/4.4.0-stable/`. Extract this archive to an easily accessible location on your machine. In the remainder of this document, this directory will be referred to as `data-integration`. This software uses *Kettle*-scripts to read, transform, upload, and otherwise manipulate data. A Kettle script defines a workflow to be performed on the data. This workflow is a sequence of tasks, where each task can be a simple step (e.g. load data from file, check the data type of a field), a more complex transformation (e.g. transform an entire table into a standard format) or even another workflow. The goal of these scripts is to take data and configuration from multiple files, and transform them to an appropriate format to be loaded to the landing tables in the database.

2.2 Java Runtime Environment

The data integration software requires a Java Runtime Environment (JRE) to be installed on your system. You can find the latest version at `http://www.oracle.com/technetwork/java/javase/downloads/index.html`. On Debian-based operating systems like Ubuntu you can install the JRE directly by typing

```
sudo apt-get install openjdk-7-jre
```

at the command line. If you have a headless system (i.e. not running a graphical interface), you are probably better off installing the headless version:

```
sudo apt-get install openjdk-7-jre-headless
```

2.3 TranSMART ETL Scripts

The Kettle-scripts for tranSMART are maintained in a GitHub repository located at `https://github.com/transmart/tranSMART-ETL`. You can either download the repository as a zip file from the web page, or use a Git client to clone the repository. This directory will be referred to as `transmart-ETL`.

2.4 SSH Client

If you will be connecting from a remote host to the database, you will need an SSH client to set up a secure connection to the database. On a Linux host, chances are an SSH client is already installed. You can check this by typing `ssh -V` at the command line. You should see something like this:

```
user@hostname:~$ ssh -V
OpenSSH_6.6.1p1 Ubuntu-2ubuntu2, OpenSSL 1.0.1f 6 Jan 2014
```

If an SSH client is not available, now is the time to install one:

```
sudo apt-get install openssh-client
```

On a Windows host, you will need Putty and, depending on your situation, PuttyGen. Putty is an SSH client and PuttyGen can be used to convert OpenSSH keyfiles to Putty's own format. Both Putty and PuttyGen are available at http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html.

2.5 Postgres Client

Finally, you will need a Postgres client to if and when you need to talk directly to the database. You can check the availability of a client with the `psql -V` command. You should see something like this:

```
user@host:~$ psql -V
psql (PostgreSQL) 9.3.6
```

If the Postgres client is not available, you can install it with the following command:

```
sudo apt-get install postgresql-client
```

On a Windows host, download the installer at http://www.postgresql.org/download/windows/.

3 Set-Up

3.1 Database Connection Parameters

Create a directory called `.kettle` in your home directory. Copy the file `transmart-ETL/Kettle/postgres/kettle.properties` to this directory. An example of this file is shown in Appendix A. At the top of this file, there should be three variables. These variables contain the details for connecting to the database. Change them to match your situation. The variables are:

- `COMMON_DB_SERVER`: URL or IP-address of the database server,
- `COMMON_DB_PORT`: Port of the database server. The default port number for Postgres is `5432`,
- `COMMON_DB_NAME`: the database name. This is most likely `transmart`.

3.2 SSH Tunnel

By default Postgres only allows incoming connections from localhost. This means that only applications running on the same machine as Postgres can access the database. If you are performing ETL on the same machine as where the Postgres databases lives, you should have no issue connecting to the database. If you are on a different machine however, you will probably need to create an SSH tunnel to the Postgres database.

An SSH tunnel is a way to make a remote machine set up a connection to a server on your behalf, and act as a middle man between you and the server. We will use this to make the machine

where Postgres is hosted connect to itself (Postgres will allow this since the connection originates from localhost) and forward all traffic to us. This is visualized conceptually in Figure 1.

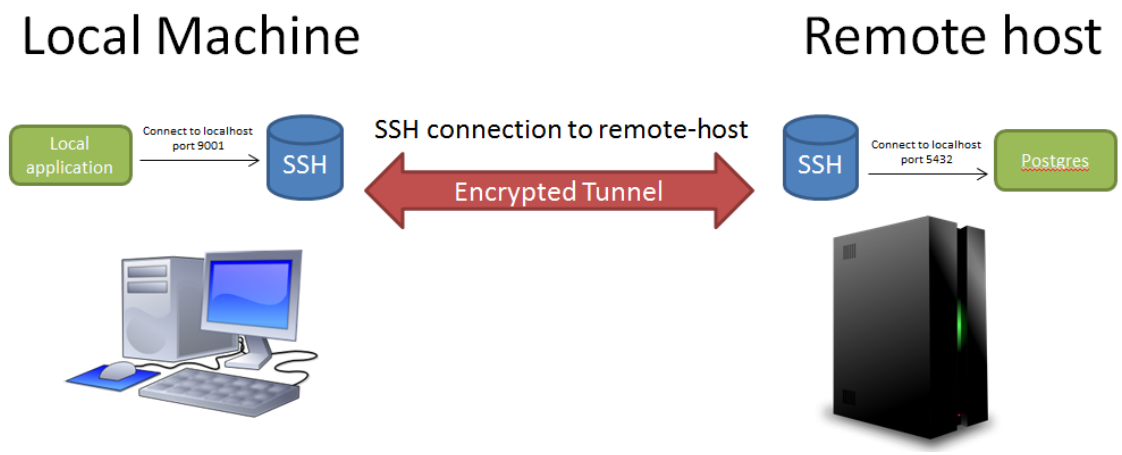

Figure 1: Conceptual representation of an SSH tunnel.

For this, we will use *Local forwarding*. This means that we will specify all components of the tunnel manually. The *Remote host* is the host we will connect to, and subsequently the host that will set up another connection on our behalf. The *Source port* identifies the entrance to the tunnel, i.e. the port number on your own machine. Any free local port could be used, 9001 being a safe choice. The *Destination* is where the remote host should connect to for us. This must point to the Postgres server, which is by default located at 127.0.0.1, port 5432. Once this is set up, any application on your machine can make a connection to local port 9001, and it will effectively be connected to the Postgres database on a remote host. Now that we know the components of an SSH tunnel, the following sections will describe how to set up this tunnel on Linux and Windows hosts.

3.2.1 Linux hosts

On a Linux system, you can set up a tunnel with the following command:

```
ssh -L 9001:127.0.0.1:5432 user@remote-host
```

where you should substitude `remote-host` with the URL of the host where Postgres lives, and `user` with your username on that host. This command will start a server listening on port `9001` on your machine.

3.2.2 Windows Hosts

This section assumes you have installed Putty as described above. Putty uses a different format for private keys than OpenSSH. The Putty private key files have a `.ppk` extension. You can convert between the two formats using PuttyGen. Open Putty and select the *Tunnels* subcategory from the *Connection/SSH* category. In the *Source port* input box, type `9001`. In the *Destination* box, type `127.0.0.1:5432`. Make sure the radio button *Local* is selected. Now click the *Add* button to add this tunnel to your connection. You can add more than one tunnel over the same SSH connection, if you need to. Your window should look like the example in Figure 2.

If you need to use a private key file for authentication, you can load your private key in the *Auth* category. Click the *Browse* button to select your private key file. As discussed above, this should be a `.ppk`-file. This is shown in Figure 3.

Now return to the *Session* category to fill out the host name. Give this connection a name in the box right below *Saved sessions* and click *Save*. Now you can re-open this session anytime by double clicking its name, without having to re-enter all the parameters. This is shown in Figure 4.

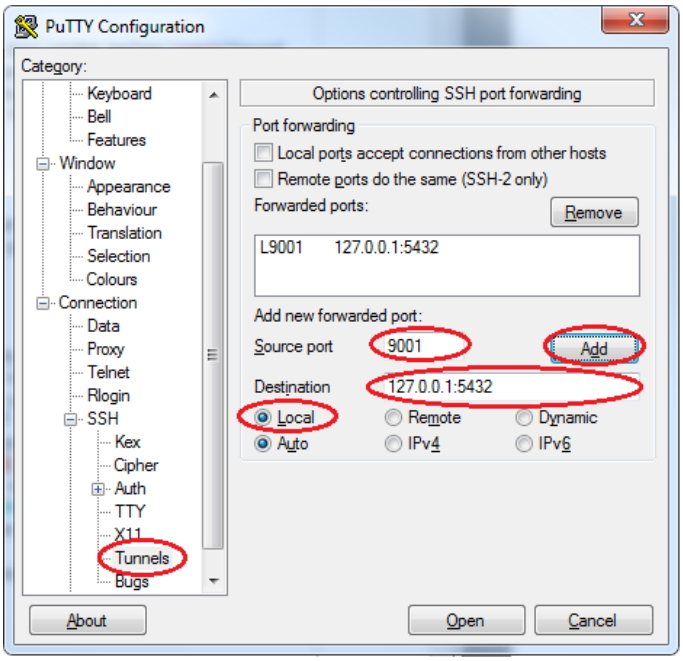

Figure 2: Filling out the tunnel configuration parameters in Putty.

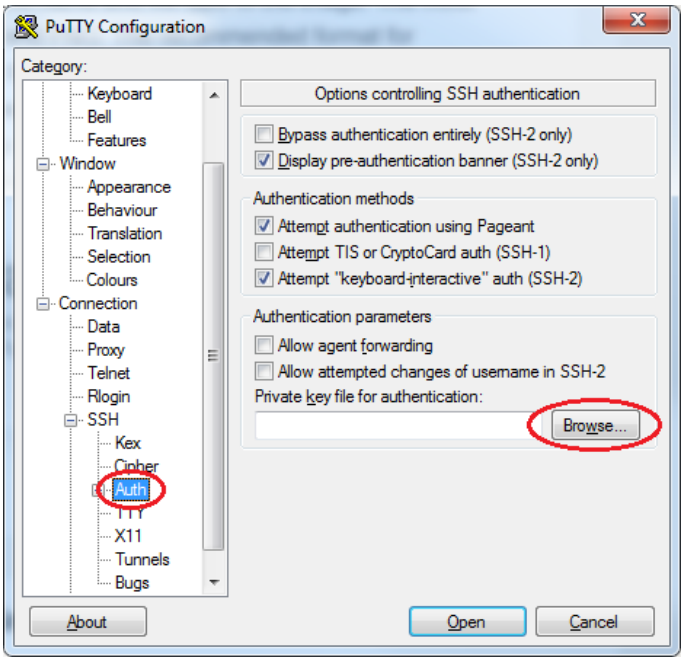

Figure 3: Selecting a private key file in Putty.

Finally click *Open* to open the connection. When the connection is established, Putty is listening for incoming connections on the local port `9001`.

3.2.3 Connecting Through The Tunnel

Now that you have a tunnel set up on port `9001`, any connections on that port will be forwarded to `postgres-host`. On the other side SSH will connect to `127.0.0.1:5432`, and allow traffic to

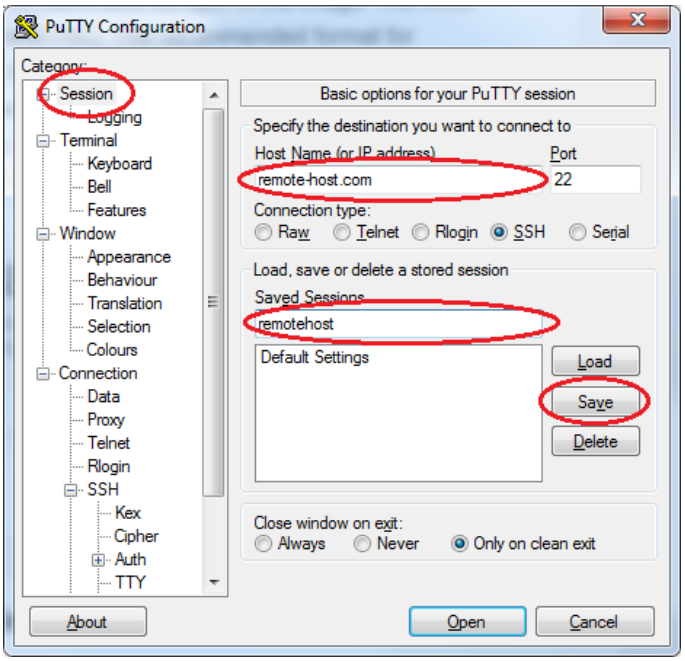

Figure 4: Entering the remote host and saving the session configuration in Putty.

flow from your machine to this connection. To Postgres it effectively looks as if you are connecting from localhost, so it will accept your connection. Security is maintained since all traffic between you and `postgres-host` is encrypted by SSH. In your `kettle.properties` file you should set `COMMON_DB_SERVER` to `127.0.0.1`, and `COMMON_DB_PORT` to `9001`, since this is now your entrance point to the Postgres server.

On both Windows and Linux systems, you can test your tunnel connection with the following command:

```
psql -h 127.0.0.1 -p 9001
```

On a Windows host you might need to write the full path to `psql.exe`, but the same arguments can be passed. If you get a password prompt, you have successfully connected and you can close the connection by hitting `Ctrl-C`. If your tunnel is not set up properly, you will get a connection refused message. In that case double-check your tunnel set-up.

Note You can choose any port number instead of 9001 for the local side of the tunnel. Although you should pick a number higher than 1023, since the first 1023 port numbers are reserved for system processes.

4 Kitchen/Spoon Basics

To perform the ETL processes, you can use either Kitchen or Spoon. Kitchen is the application that can read the Kettle-scripts that will define and execute the necessary transformations. Spoon provides a graphical interface for viewing a script and setting its parameters. Under the hood, Spoon calls Kitchen with the appropriate parameters. The folder `transmart-ETL/Kettle/postgres` contains example startup scripts for use in a bash-shell. The first section below describes the general procedure for loading a script and setting its parameters.

4.1 Input Files

In order to perform ETL, you will need to generate one or more input files. All input files should be tab delimited text files. The sections below will describe all of these files, and the meaning

of all columns. Input file columns that are marked as optional are not required to contain data, however the columns themselves (i.e. tab characters) should be there.

☞ You are only allowed to omit entire columns from the file if they are the last columns.

For example, the column mapping file consists of four required columns and four optional ones. Even if you will not use column five, seven and eight, you need to have a file with six columns in total. This is illustrated in Figure 5.

The column headers are only considered in the data files. E.g. in clinical data files they can be used as data labels, and in gene expression data files they identify the sample. For all other input files, i.e. the ones containing configuration rather than data, only the column position is considered. The sections below will clearly describe if a header row is required or not for all files.

Filename	Category	ColNb	DataLabel		CtrlVocabCode
GSEXXXX_clinical.txt		1	SUBJ_ID		
GSEXXXX_clinical.txt	Demographics+Age	2			L-123
GSEXXXX_clinical.txt		3	OMIT		

(a)

```
Filename→Category→ColNb→DataLabel→→CtrlVocabCode
GSEXXXX_clinical.txt→→1→SUBJ_ID→→
GSEXXXX_clinical.txt→Demographics+Age→2→→→L-123
GSEXXXX_clinical.txt→→3→OMIT→→
```

(b)

Figure 5: Example of a column map file with three unused columns. The final two columns can be omitted from the file, however column five should be included. (a): Structure of the example. (b): Tab seperated file. The right arrows indicate tab characters.

4.2 Loading A Script

When invoking kitchen you should pass at least the `-file=filename` option, where `filename` refers to the Kettle-script that you which to execute. In most cases you will also need to pass several `-param` options, which set parameters that are used in the script. The syntax for this option is `-param:PARAM_NAME=PARAM_VALUE`. You can instruct kitchen to write its output to a log file with the `-log=logfile` option. The default logging level is `Basic`. If you are having trouble uploading data, you can increase the logging level to e.g. `Debug` by passing the `-level=Debug` option. You can check the full list of available logging levels at http://wiki.pentaho.com/display/EAI/Kitchen+User+Documentation#KitchenUserDocumentation-Setthelogginglevel.

Spoon works in nearly the same way, but provides a graphical interface for setting the script parameters. On the main window the script itself is graphically represented so it can be viewed and edited. Start up spoon by executing the spoon.bat (on Windows) or spoon.sh (on Linux) script. Choose *Open* under the *File* menu, and navigate to the script you which to execute. Click on the *Run this job* icon in the toolbar or choose *Run* from the *Action* menu (see Figure 6).

A dialog similar to the one in Figure 7 should appear. The section marked in green represent the parameters to be passed to the script, and should be filled out manually. These parameters will depend on the job and will be discussed in the following sections. The section marked in red

are global variables. The values for global variables used by the current script are taken from your `kettle.properties` file, and are shown in this section. Above the variables section there is a drop down box marked in blue which enables you to modify the logging level.

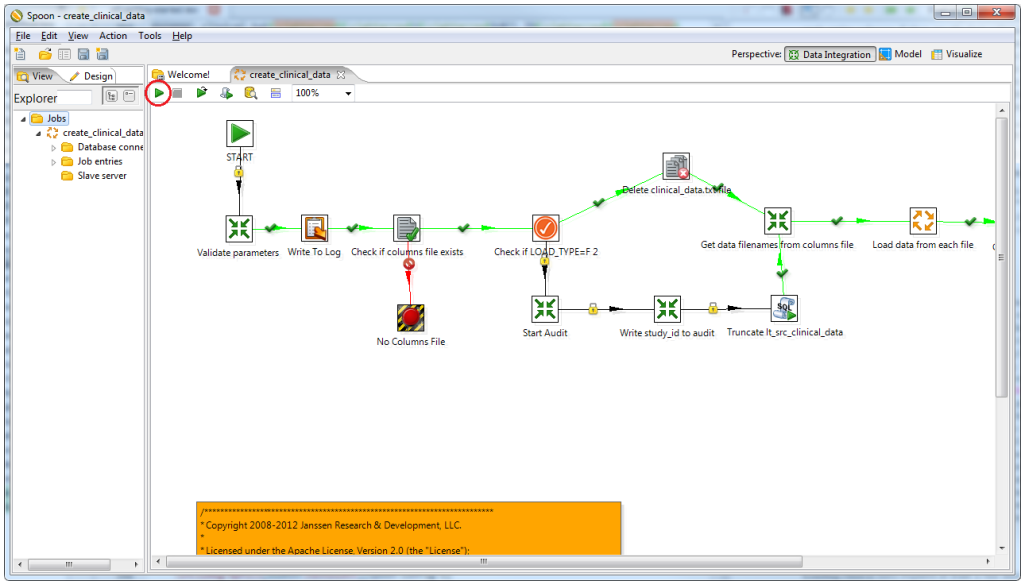

Figure 6: Starting a job with Spoon

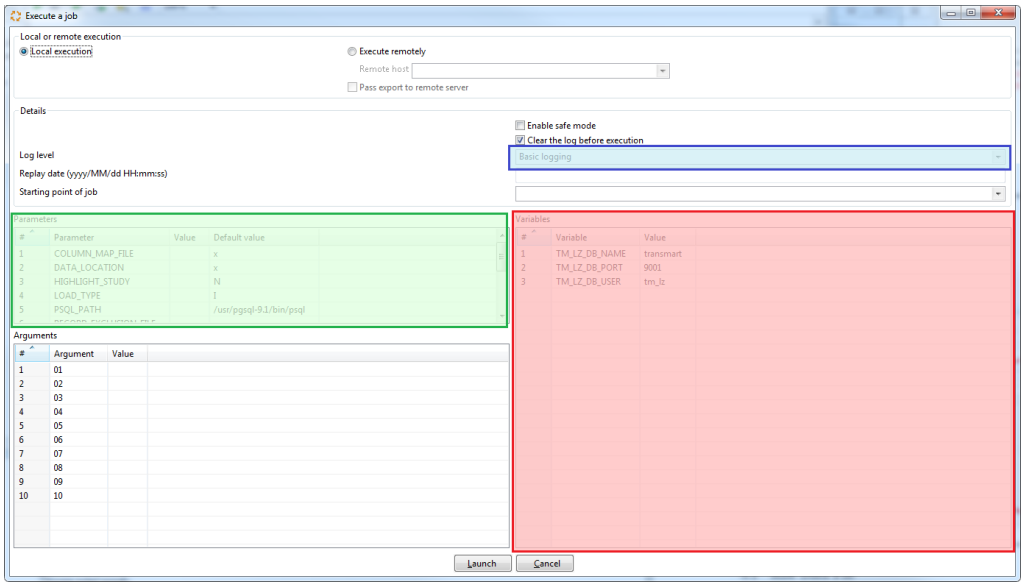

Figure 7: Configuring a job with Spoon

5 Clinical Data

Loading clinical data requires at least a *raw data* file and a *column mapping* file. The *word mapping* and *record exclusion* files are optional. Below follows a description of the input files, followed by information on invoking the clinical data upload script. The filenames do not need to follow a predefined convention, you can specify each filename as a parameter for the upload script. Since all input files will be referred to relative to a single source directory, it is advisable to keep all of the input files in the same directory.

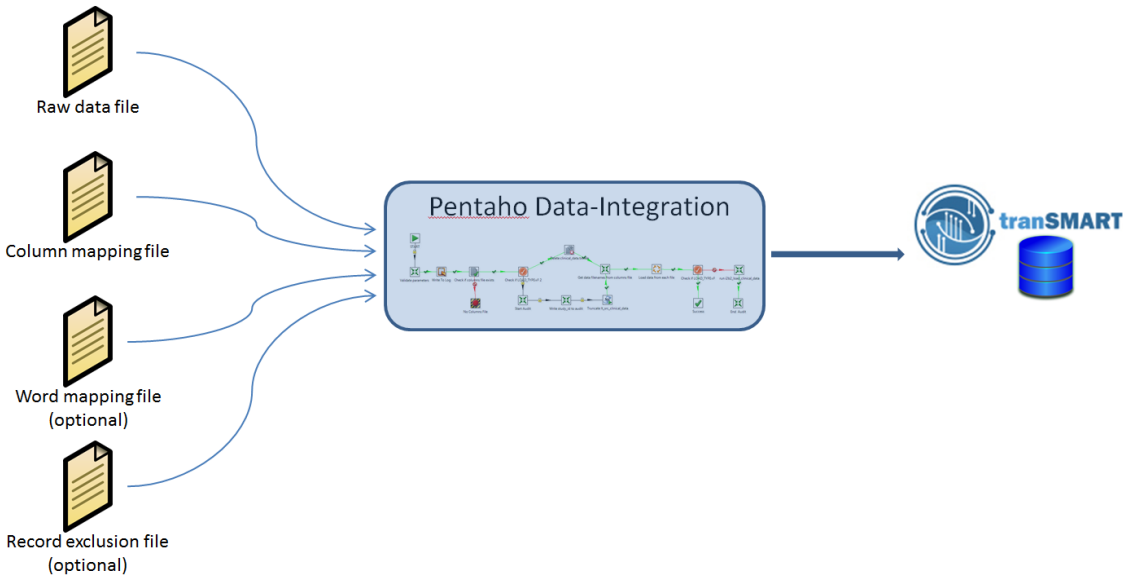

Figure 8: Conceptual representation of the ETL process for clinical data.

5.1 Raw Data File

Generate one or more files which contain the data that you wish to upload into tranSMART. Each file should be a tab-delimited text file with one header row, a column for the subject identifier and a column for each attribute. The following section describes the column mapping file, which will define the meaning of each column.

5.2 Column Mapping File

The column mapping file is a tab-delimited file with at least four columns and is used to map columns from the raw data files to concepts in the Dataset Explorer ontology tree. Table 1 describes the structure of this file. If you choose to include one or more of the optional columns listed below, they must appear as the n'th column, where n is the position listed below.

Table 1: Description of columns in the column mapping file

Pos	Column	Description	Example
1	FILENAME	The name of the raw data file containing the data. If your raw data is distributed across multiple files, they can all be referenced with one column mapping file.	GSEXXXX_clinical.txt
2	CATEGORY_CD	This describes the position of the concept in the study structure within tranSMART	Clinical_Data+Alcohol_Habits
3	COL_NBR	The column number in the datafile that should be mapped	3

Table 1: (continued)

Pos	Column	Description	Example
4	DATA_LABEL	The data label you wish to assign to this column of the data file. Kettle will automatically pull the column heading and use it as the data label if you leave this empty. If you would like to map the column heading to a new label, enter it here.	The reserved words below instruct Kettle to perform specific actions. Reserved words must be fully capitalized to be recognized. • OMIT: Skip the column. • SUBJ_ID: The data in this column is the subject ID. • SITE_ID: The ID of the site where this data was acquired. • VISIT_NAME: Use the data in this column as the visit name. A subject's attributes can be measured at multiple points in time. Each concept will be extended with the distinct values found in this column. • DATA_LABEL: Treat the data in this column as a data label for another column. • \: Maps the data as a data value of a column designated as a DATA_LABEL column. • VISIT_NAME_2: A second level of visit name. The total visit name will become VISIT_NAME\VISIT_NAME_2. • SEQ_COL: If a subject's attribute was measured multiple times, this column indicates the sequence number of the measurement. The values in this column should therefore be integer and strictly positive. • UNITS: Units for the value in column ALT_DATA_LABEL. In addition to these reserved words, there exist the following reserved concept names: Age, Race, Sex and Gender. Sex and Gender are synonyms. If a column carries this label, it will be automatically incorporated in the summary statistics page of the Dataset Explorer. In contrast to the reserved keywords, the match on these concept names is not case sensitive. The concept will appear in the ontology tree with the same capitalization as you have written it here. **Note** Each raw data file must include one column that carries the label SUBJ_ID. All other reserved words are optional.

Table 1: (continued)

Pos	Column	Description	Example
5	ALT_DATA_LABEL (optional)	Use this column if you wish to use the data in a column as a data label for another column, or if you wish to aggregate multiple data values. Aggregate functions are applied over rows which have the same values in all columns except for COL_NBR.	This field specifies the number of the column that should be used as the source for the data label. This column cannot be left blank for any row where the \ is used in the data label field. In rare circumstances where multiple columns must be integrated together, use the following convention: • Append an A to the column number if you want to have the value from the other column added as a level in the ontology after the data label of the column. • Append a B to the column number if you want to have the value from the other column added as a level in the ontology before the data label of the column (for example, 4A, 6B, etc.). The default is A if not specified. If you want aggregation, set this field to MIN, MEAN or MAX for the minimum, mean or maximum respectively.
6	CTRL_VOCAB_CODE (optional)	Use this column if you wish to map the record to a controlled terminology (for example, SNOMED or MedDRA).	L-85B02
7	SEQ_FORMAT_FROM (optional)	Undocumented	
8	SEQ_FORMAT_TO (optional)	Undocumented	

5.3 Word Mapping File

The word mapping file is an optional file that allows a data value to be transformed into another data value. Any exact match in the specified column of the raw data file will be transformed to the new data value. The word mapping file is primarily used to map categorical values to a controlled vocabulary, and also to change unknown and null values into a value that can be displayed in tranSMART. A word mapping file must be a tab-delimited text file with four columns, which are described in Table 2.

Table 2: Description of columns in the word mapping file

Pos	Column	Description	Example
1	FILENAME	The name of the file containing the data you wish to remap.	GSEXXXX_clinical.txt
2	COL_NBR	The column number within the raw clinical data file that should be mapped.	3
3	FROM_VALUE	The original data value of the record	RA
4	TO_VALUE	The new data value you wish to display	Rheumatoid Arthritis

5.4 Record Exclusion File

The record exclusion file is a tab delimited text file that is used to exclude records with a specific value in a particular column of the raw data file from the upload process. If the column value for a particular line matches the exclude value, the entire line is skipped. No data for that subject will be found in the database. Table 3 describes the columns that make up this file.

Table 3: Description of columns in the record exclusion file

Pos	Column	Description	Example
1	FILENAME	The name of the file containing the data you want to filter.	GSEXXXX_clinical.txt
2	COL_NBR	The column number within the raw clinical data file that should be checked.	3
3	VALUE	The data value you wish to exclude	N/A

5.5 Upload Script

Make sure you have prepared your data in the formats listed above. The Kettle-script for clinical data is located at `transmart-ETL/Kettle/postgres/Kettle-ETL/create_clinical_data.kjb`. Table 4 describes the parameters that can be passed to the `create_clinical_data.kjb` script. Use either Kettle or Spoon as described above to load the script and set its parameters.

Table 4: Clinical data upload parameters

Name	Default	Description
COLUMN_MAP_FILE	x	Filename of the column mapping file
DATA_LOCATION	x	Full path to the input files. All other filenames are relative to this path.
PSQL_PATH	x	Path to your psql executable. E.g. `C:\Program Files\PostgreSQL\bin\psql.exe` or `/usr/bin/psql`. Only used if LOAD_TYPE = L.

Table 4: (continued)

Name	Default	Description
HIGHLIGHT_STUDY	N	Y will cause the study name to be shown in green in the ontology
LOAD_TYPE	I	I: load the data by generating an insert statement for each row. This is the preferred method for loading to the database. L: load the data through the Postgres bulk loader. This can be more efficient than the I option, although in most cases the performance difference is negligible. You need te set PSQL_PATH as well to use this option. F: instead of loading to the database, write to a file. The data will be written to a file called <STUDY_ID>_clinical_data, where <STUDY_ID> is the study ID you configured with the STUDY_ID parameter.
SECURITY_REQUIRED	N	N: Indicates this is a public study. Any user logged in to tranSMART can view this study. Y: A tranSMART administrator needs to give explicit access to each user who requires access to this study.
SORT_DIR	$HOME	Full path to a directory where temporary files can be stored for sorting
STUDY_ID	x	Unique identifier of the study. ☞ The study ID will be transformed to uppercase during ETL. This means that all future references to the study ID must be in uppercase as well.
TOP_NODE	x	The string that defines the top node of the ontology, including the full name of the study. For example: \Public Studies\Breast_Cancer_Kao_GSE20685\
WORD_MAP_FILE	x	Optional file that remaps values when reading source files
RECORD_EXCLUSION_FILE	x	Optional file defining filters to exclude records

5.6 Example

In this example we will upload a small mock study, using the aspects defined above. The files can be found in the Clinical data directory included with this document. The clinical data upload example consists of four files: GSEXXXX_clinical.txt, GSEXXXX_columns.txt, GSEXXXX_words.txt and GSEXXXX_record_exclusion.txt. It is recommended to study the contents of each file, in order to understand the upload process.

GSEXXXX_clinical.txt contains the raw clinical data. Notice that the DAS28 column has one missing value, and one value set to NA. In this example, we want to exclude the subject for which the DAS28 score is NA. We create a record exclusion file with one line defining the exclusion

parameters. Additionally, we want to rename the value of RA in the disease_state column to Rheumatoid Arthritis. The word mapping file can help us do that. Finally, we use the column mapping file to define the study ontology.

Now we are ready to start the ETL process. As the top node of our study, we choose \Public studies\GSEXXXX\. You can review the meaning of all parameters in table 4. Since typing out all parameters every time can be tedious, we recommend keeping a small script file within the study directory. If you need to re-upload the data at a later point, you can just run this script. The Clinical data directory contains scripts for both Linux and Windows environments.

On a Linux machine, open the file load_clinical.sh in the study directory. On a Windows machine, open the file load_clinical.bat in the study directory. Change the DATA_INTEGRATION_PATH and TRANSMART_ETL_PATH variables to the location of the Data Integration software suite and the tranSMART-ETL repository respectively. Notice that when keeping the upload script together with the data files, we can set the DATA_LOCATION parameter to . (current directory). Finally check the location of your psql executable and make sure it corresponds to the SQLLDR_PATH parameter. In a terminal, navigate to the Clinical data directory and start the upload script. On Linux, type bash load_clinical.sh, on Windows, type load_clinical.bat.

Alternatively, you can use Spoon as described in section 4. Just hit the run button and copy all parameter values over from the script to the appropriate input fields in Spoon.

Kitchen will produce a large amount of output. Check the last few lines output. If everything went well, these lines should look like this:

```
INFO  07-05 08:56:52,708 - Kitchen - Finished!
INFO  07-05 08:56:52,708 - Kitchen - Start=2015/05/07 08:56:45.339,
    Stop=2015/05/07 08:56:52.708
INFO  07-05 08:56:52,708 - Kitchen - Processing ended after 7 seconds.
```

Kitchen will also tell you if something went wrong:

```
INFO  07-05 08:56:19,419 - Kitchen - Finished!
ERROR 07-05 08:56:19,419 - Kitchen - Finished with errors
INFO  07-05 08:56:19,419 - Kitchen - Start=2015/05/07 08:56:18.982,
    Stop=2015/05/07 08:56:19.419
INFO  07-05 08:56:19,419 - Kitchen - Processing ended after 0 seconds.
```

Notice the extra line warning you about errors. If there are errors, double-check the script parameters and variables.

Log in to your tranSMART instance to admire your work! Open the study node. The ontology should look like the one shown in Figure 9. Notice that altough we have 20 subjects in our input file, only 19 were uploaded to tranSMART. Remember that we excluded the subject with the NA value for DAS28 using the record exclusion file. Additionally, there was a subject with a missing DAS28 score. As a result, we can see that there are 19 subjects in the study, of which only 18 have the DAS28 concept. Finally, we can see that the RA concept was mapped to Rheumatoid Arthritis.

6 Gene Expression Data

Loading gene expression data requires at least a *raw data* file, a *subject sample mapping* file and it requires the platform definition to be present in tranSMART. Uploading the platform definition is discussed in Section 7. The *sample remapping* file is optional. Below follows a description of the input files, followed by information on invoking the gene expression data upload script. If you have clinical data from the same study as well, it is important to upload the clinical data first. Gene expression data can be linked to subjects who already have clinical data in the database, but not the other way around.

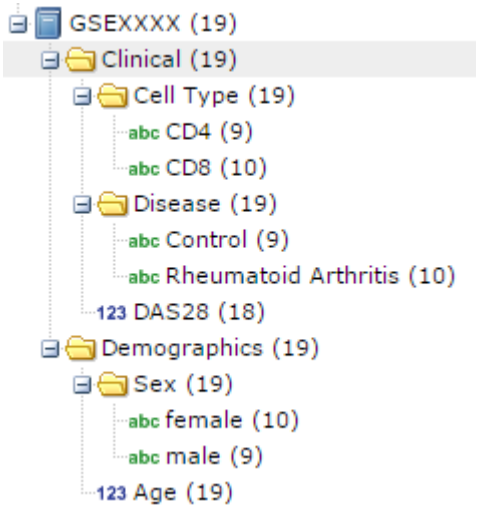

Figure 9: Ontology of the example study

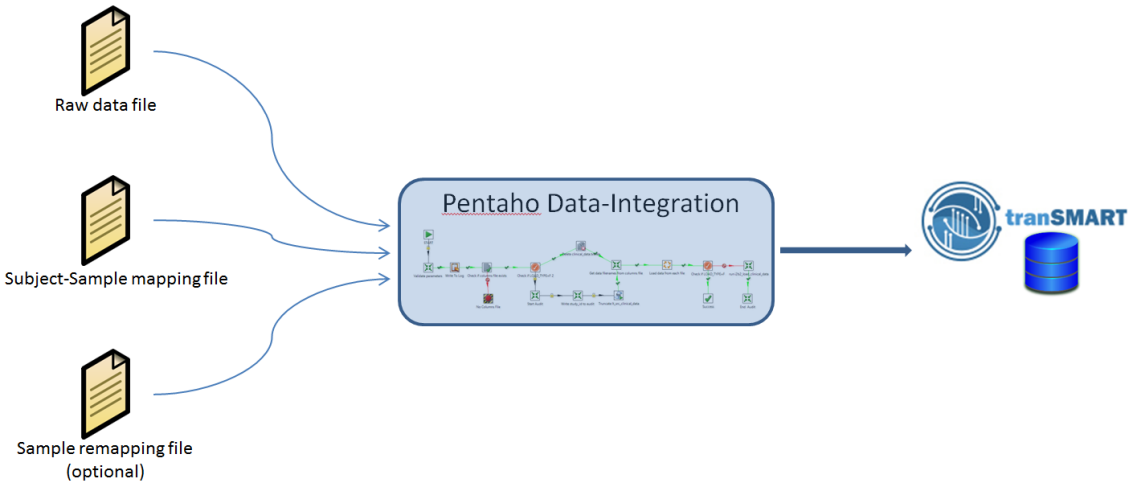

Figure 10: Conceptual representation of the ETL process for gene expression data.

6.1 Raw Intensity Input Files

The raw intensity values can be located in one or more tab delimited text files. The first row denotes the column headers. The first column should be named `ID_REF`, and refers to the probe ID from which the intensity was measured. All subsequent columns names are sample IDs. These sample IDs will be used in the subject sample mapping file to map sample IDs to subject IDs from clinical data.

6.2 Subject Sample Mapping File

The subject sample mapping file is a tab delimited text file. It holds information on each of the samples in the raw data files. Table 5 describes the columns that make up this file.

Table 5: Description of columns in subject sample mapping file

Pos	Column	Description	Example
1	STUDY_ID	Identifier of the study. Remember that study IDs are always uppercase	GSEXXXX
2	SITE_ID	Identifier of the site where the samples were acquired	
3	SUBJECT_ID	Identifier of the subject. This should correspond to the identifier of a subject in the clinical data file	SUBXXXX
4	SAMPLE_ID	Identifier of the sample. This should correspond to a column name in the raw intensity data file	SAMXXXX
5	PLATFORM	The platform identifier	GPL201
6	TISSUETYPE	Tissue type from which the sample was collected	Synovial Tissue
7	ATTR1	Custom attribute 1. The value of this field can be used in the category code to make it part of the ontology	
8	ATTR2	Custom attribute 2. The value of this field can be used in the category code to make it part of the ontology	
9	CATEGORY_CD	Category code where this gene expression data will be inserted in the ontology. You can use the keywords PLATFORM, TISSUETYPE, ATTR1 and ATTR2 here as well. PLATFORM will be replaced by the description of the platform in the PLATFORM column, the other keywords will be replaced by their values in the corresponding columns. Use + to seperate ontology levels and _ for spaces.	Biomarker_Data+ Gene_Expression+ PLATFORM+ TISSUETYPE
10	SOURCE_CD	Identifier of data source	GEO

6.3 Sample Remapping File

The sample remapping file is optional and is a tab delimited text file. You can use this file to rename specific sample IDs in particular input files to something else. Table 6 describes the columns that make up this file.

Table 6: Description of columns in the sample remapping file

Pos	Column	Description	Example
1	REMAP_DATA_FILENAME	Name of the file that holds the data to be remapped	raw.GSEXXXX_expression.txt
2	CURRENT_SAMPLE_ID	The sample ID to be renamed	

Table 6: (continued)

Pos	Column	Description	Example
3	NEW_SAMPLE_ID	The new name of the sample ID	

6.4 Upload Script

Make sure you have prepared your data in the formats listed above. The Kettle-script for gene expression data is located at `transmart-ETL/Kettle/postgres/Kettle-ETL/load_gene_expression_data.kjb`. It is important to know that tranSMART stores three *projections* of the data: the *raw* data, the *log-transformed* data and the *z-score*. When exporting data you can choose which projection you want. Therefore it is important to set the `DATA_TYPE` parameter (described below) to the correct value.

Table 7 describes the parameters that can be passed to the `load_gene_expression_data.kjb` script. Use either Kettle or Spoon as described above to load the script and set its parameters.

Table 7: Gene expression data upload parameters

Name	Default	Description
BULK_LOADER_PATH	x	Path to your psql executable. E.g. `C:\Program Files\PostgreSQL\bin\psql.exe` or `/usr/bin/psql`. Only required if `LOAD_TYPE` set to L.
DATA_FILE_PREFIX	x	Prefix for the filenames of raw gene expression data files
DATA_LOCATION	x	Full path to the input files
DATA_TYPE	L	Can be R, L or T. • R: The data are raw intensity values, no transformation has occured. In this case the log (base `LOGBASE`) and z-score is calculated. The z-score is trimmed to the interval $[-2.5, 2.5]$ and calculated from the log values. • L: The data has been log transformed. It is uploaded to the log projection and a z-score is calculated from this. The zscore is trimmed to the interval $[-2.5, 2.5]$. The raw intensity is derived using `LOGBASE`. • T: data will be uploaded with no additional transformation to the log-projection. Data is also loaded to the z-score projection, but there it is trimmed to the interval $[-2.5, 2.5]$.
FilePivot_LOCATION	x	Full path to directory where `FilePivot.jar` is located. This file should be located in `transmart-ETL/Kettle/postgres/`

Table 7: (continued)

Name	Default	Description
JAVA_LOCATION		Full path to the directory where java (or java.exe) is located.
LOAD_TYPE	I	I: load the data by generating an insert statement for each row. This is the preferred method for loading to the database. L: load the data through the Postgres bulk loader. This can be more efficient than the I option, although in most cases the performance difference is negligible. You need te set BULK_LOADER_PATH as well to use this option. F: instead of loading to the database, write to a file. The data will be written to a file called <STUDY_ID>_clinical_data, where <STUDY_ID> is the study ID you configured with the STUDY_ID parameter.
LOG_BASE	2	The log base to use when log transforming raw data. Also used to derive the raw intensity value from already log transformed data.
MAP_FILENAME	x	Filename of the subject-to-sample mapping file
SAMPLE_REMAP_FILENAME		Filename of the sample remapping file. Omit this parameter or set it to NOSAMPLEREMAP if there is no sample remapping.
SAMPLE_SUFFIX		If all sample_cds have a common suffix that you wish to remove, you can specify that suffix here. **Note:** The suffix will not be removed if a SAMPLE_REMAP_FILENAME is specified
SECURITY_REQUIRED	N	N: Indicates this is public data. Any user logged in to tranSMART can view this data. Y: A tranSMART administrator needs to give explicit access to each user who requires access to this data.
SORT_DIR	x	Full path to a directory where temporary files can be stored for sorting
SOURCE_CD	STD	Only samples with a matching DATA_SOURCE_CD column in the subject sample mapping file will be imported. Samples with an empty (null) DATA_SOURCE_CD will be imported regardless of the value of this parameter
STUDY_ID	x	Unique identifier of the study. This will be transformed to all caps before being used
TOP_NODE	x	The string that defines the node under which this data will be inserted. For example: \Public Studies\Breast_Cancer_Kao_ GSE20685\Gene Expression

6.5 Example

In this example we will add gene expression data to the mock study. If you have not uploaded the mock study yet, please refer to section 5.6. The gene expression data can be found in the `Expression data` directory included with this document. The file `raw.GSEXXXX_expression.txt` contains the raw expression values. The subject sample mapping is defined in `Subject_Sample_Mapping.txt`.

If you are on a Linux system, open the `load_gene_expression.sh` file. On a Windows system, open the `load_gene_expression.bat` file. Change the `DATA_INTEGRATION_PATH` and `TRANSMART_ETL_PATH` variables to the location of the Data Integration software suite and the tranSMART-ETL repository respectively. In contrast to the clinical data upload script, we can not refer to the data location by relative path. The data location is also passed to file pivoter, so it needs to be an absolute path. Complete the file by defining the `DATA_LOCATION` parameter. If the Java executable can not be found automatically by your system, you must specify its location with the `JAVA_LOCATION` parameter. In a terminal, navigate to the Expression data directory and start the script. On Linux, type `bash load_gene_expression.sh`, on Windows, type `load_gene_expression.bat`.

Alternatively, you can use Spoon as described in section 4. Just hit the run button and copy all parameter values over from the script to the appropriate input fields in Spoon.

Just like uploading clinical data, kitchen will produce a large amount of output. Check the last few lines output. If everything went well, these lines should look like this:

```
INFO  07-05 08:56:52,708 - Kitchen - Finished!
INFO  07-05 08:56:52,708 - Kitchen - Start=2015/05/07 08:56:45.339,
Stop=2015/05/07 08:56:52.708
INFO  07-05 08:56:52,708 - Kitchen - Processing ended after 7 seconds.
```

Kitchen will also tell you if something went wrong:

```
INFO  07-05 08:56:19,419 - Kitchen - Finished!
ERROR 07-05 08:56:19,419 - Kitchen - Finished with errors
INFO  07-05 08:56:19,419 - Kitchen - Start=2015/05/07 08:56:18.982,
Stop=2015/05/07 08:56:19.419
INFO  07-05 08:56:19,419 - Kitchen - Processing ended after 0 seconds.
```

Notice the extra line warning you about errors. If there are errors, double-check the script parameters and variables.

Log in to your tranSMART instance to admire your work! Open the study node. You should now see an extra node compared to the ontology shown in Figure 9. The gene expression data has been added to the study and linked to the existing subjects thanks to the subject sample mapping file. Your ontology should like shown in Figure 11.

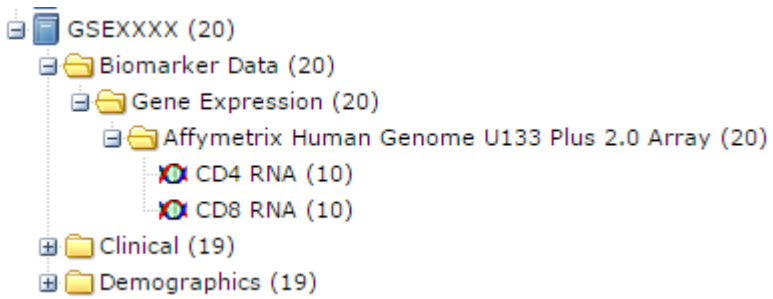

Figure 11: Ontology of the example study, after uploading gene expression data

7 Gene Expression Platform Definitions

Before you can upload gene expression data, you have to define the platform that was used to generate the intensity values. NCBI's Gene Expression Omnibus [3] has a large collection of platform definitions available.

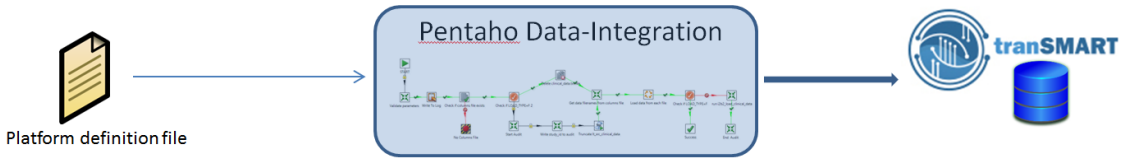

Figure 12: Conceptual representation of the ETL process for gene expression platform definition.

7.1 Platform Definition File

The platform definition file is a tab delimited text file. The necessary columns are described in Table 8. The position of these columns in the text file does not matter, you will need to define their positions in the Kettle script used to upload platform definitions. Lines starting with !, # or ~ are treated as comments and will be ignored.

If you need to preprocess the platform definition file, it is advised to put the columns shown in Table 8 in the order they appear in that table. Then you can set the `DATA_SOURCE` parameter of the Kettle script to P, which will upload the platform definition file directly to the correct database table.

Table 8: Description of columns in the platform file

Column	Description	Example
`GPL_ID`	GPL ID of the platform this probe belongs to	GPL201
`PROBE_ID`	ID of the probe. These IDs should correspond to probe IDs in gene expression data files.	1007_s_at
`GENE_SYMBOL`	Gene symbol corresponding to this probe. This field can be left empty, e.g. when a particular probe has not yet been mapped to a gene.	DDR1
`GENE_ID`	Numeric ID for this gene. If omitted, tranSMART will look for the gene symbol in other platform definitions that are already on the system and fill out this field if the gene symbol is found.	780
`ORGANISM`	The organism this platform operates on	Homo Sapiens

A second format for platform definition files is supported as well. This second format embeds a table in the gene symbol column. Columns of this embedded table should be separated by the string defined in the `GENETAB_DELIM` Kettle script parameter, and rows should be separated by the string defined in the `GENETAB_REC_DELIM` parameter. The embedded table should contain at least two columns, one for the gene symbol and one for the gene ID. It can contain multiple rows to represent that the probe corresponds to multiple genes.

7.2 Upload Script

The Kettle script used to upload platform definitions is located at `transmart-ETL/Kettle/postgres/Kettle-ETL/load_annotation.kjb`. Its parameters are listed in Table 9.

Table 9: Description of parameters in the platform upload file

Name	Default	Description
ANNOTATION_DATE		Release date of the platform
ANNOTATION_RELEASE		Release number of the platform
ANNOTATION_TITLE	NOTITLE	Platform title. E.g. Affymetrix Human Genome U95 Version 2 Array
DATA_LOCATION		Full path to the platform definition file
DATA_SOURCE	A	A: Platform definition file needs to be preprocessed. This option will extract the necessary columns based on the information below. P: Platform definition file is a preprocessed definition file, and can be directly loaded.
EMBEDDED_GENE_TABLE	N	Y or N, if the platform definition file contains an embedded gene table (see above) or not, respectively. This parameter is only considered if DATA_SOURCE is set to A.
GENETAB_DELIM	//	Delimiter of the columns in the embedded gene table
GENETAB_ID_COL	-1	Column number of the gene ID in the embedded gene table
GENETAB_REC_DELIM	///	Delimiter of the rows in the embedded gene table. If the platform definition file does not contain a gene table, use this string as a delimiter in the gene symbol and gene ID columns
GENETAB_SYMBOL_COL	-1	Column number of the gene symbol in the embedded gene table
GENE_ID_COL	19	Column index of the gene ID
GENE_SYMBOL_COL	15	Column index of the gene symbol
GPL_ID	GPLXXX	GPL ID of the platform
LOAD_TYPE	I	I: load the data by generating insert statements for each row. L: load the data through the Postgres bulk loader. F: instead of loading to the database, write to a file. **Note:** The F option can only be applied when DATA_SOURCE is set to A.
ORGANISM_COL	3	Column index of the organism
PROBE_COL	1	Column index of the probe ID
SKIP_ROWS	1	Number of rows to skip. **Note:** This script does not assume a header row is present. If a header row exists, this should be set to one.

Table 9: (continued)

Name	Default	Description
SOURCE_FILENAME	x	Filename of the platform definition file
SQLLDR_PATH		Full path to your psql executable

7.3 Example

☞ This section has yet to be completed.

8 Uploading Data with ICE

The FCL4tranSMART application provides an Integrated Curation Environment (ICE) where you can design your ontology, column mappings and other aspects in a GUI application. The application was developed for tranSMART version 1.1, and is currently incompatible with version 1.2. When a new version of this application is released, this document will be updated as well. For future reference, the most recent version of the application is available at https://github.com/transmart/tranSMART-ETL/tree/master/FCL4tranSMART/Postgres.

9 Removing Data

Sometimes it will be necessary to remove an entire study from the database, e.g. to re-upload it with a different tree structure. You need to know the study ID of the study you which to remove. Remember that study IDs are always uppercase. The commands below are for Linux hosts. On a Windows host, just replace `psql` by `psql.exe`, and make sure your terminal is in the directory containing `psql.exe`.

If you have a tunneled connection to the database, use the following command to log in to an interactive shell as the user `tm_cz`:

```
psql -U tm_cz -h 127.0.0.1 -p 9001 transmart
```

When you are on the machine hosting the database itself, you will most likely still need to connect through a network socket. By default Postgres will look at your system username when not connecting through a network socket. This is easily avoided by passing the `-h` parameter just like when connecting from a remote host:

```
psql -U tm_cz -h 127.0.0.1 transmart
```

When asked for a password, enter `tm_cz` and press enter. You should now see the prompt `transmart=>`, indicating that you are connected.

The clinical data, metadata and gene expression data are stored in a number of different tables. Execute the following commands, where you replace `{STUDYID}` by the actual study ID that you want to remove. Remember that the study ID is always uppercase. On Linux, you can also use the script provided in Appendix B.

```
DELETE FROM i2b2metadata.i2b2_tags WHERE path=
 (SELECT c_fullname FROM i2b2metadata.i2b2
  WHERE sourcesystem_cd='{STUDYID}' ORDER BY c_hlevel ASC LIMIT 1);

DELETE FROM i2b2demodata.concept_dimension
WHERE sourcesystem_cd='{STUDYID}';
```

```
DELETE FROM i2b2demodata.concept_counts
WHERE concept_path IN
 (SELECT c_fullname FROM i2b2metadata.i2b2
  WHERE sourcesystem_cd='{STUDYID}');

DELETE FROM i2b2demodata.patient_dimension
WHERE sourcesystem_cd LIKE '{STUDYID}:%';

DELETE FROM i2b2demodata.observation_fact
WHERE sourcesystem_cd='{STUDYID}';

DELETE FROM deapp.de_subject_microarray_data
WHERE trial_name='{STUDYID}';

DELETE FROM deapp.de_subject_sample_mapping
WHERE TRIAL_NAME='{STUDYID}';

DELETE FROM i2b2metadata.i2b2 WHERE sourcesystem_cd='{STUDYID}';

DELETE FROM i2b2metadata.i2b2_secure WHERE c_fullname='{STUDYID}';
```

10 Troubleshooting

During an ETL process, a lot can go wrong. The cause can sometimes be difficult to determine. This section provides some ways to help you figure out where the problem might be.

Your first step should always be to increase the log level of Kitchen. This was described in Section 4. We recommend the `debug` log level. The `rowlevel` log level will output lines for each individual row inserted to the database and therefore, this level of verbosity can hinder your search for the cause of the error.

If you can not find a cause in the output of Kitchen, it is time to look at the audit logs. During ETL, most actions are logged to tables in the database, these logs can provide clues as to what is going wrong. First, connect to the database as the `tm_cz` user as described in Section 9. We need to find out the job ID of the job that is causing the error. This is probably the most recent job that was carried out. Execute the following SQL statement:

```
SELECT * FROM cz_job_master ORDER BY job_id DESC LIMIT 10;
```

The most recent job is now at the top. The `job_status` column should show the value `FAIL`. You can also check the start date to make sure this is the job you want to investigate. Now we can go to the audit log to view the steps taken in this job. Suppose the job ID was XXXX, then execute the following query:

```
SELECT * FROM cz_job_audit WHERE job_id=XXXX;
```

You will now get a list of steps taken, this shows you which steps are completed and which steps fail. To get to the specific error message, execute the following query:

```
SELECT * FROM cz_job_error WHERE job_id=XXXX;
```

You will then be able to see the error message. One possible error message is `invalid input syntax for type numeric: ""`. This indicates that an empty string is being inserted in the place where a numeric value should go. You probably have an empty cell somewhere in your data that got overlooked during the curation process.

As an alternative to using the command-line based `psql` program, you can use any graphical Postgres administration tool. Here we will describe how to set up pgAdmin, available from http://www.pgadmin.org/download/.

First, we need to set up our database connection. Under the File menu, choose *Add server*. Give the connection a meaningful name, and fill in the rest of the details. Remember to connect as user `tm_cz`, with password `tm_cz`. If you are using a tunnel, choose use your tunnel entrance point as the host and port. The add server dialog should look like shown in Figure 13. Now click OK.

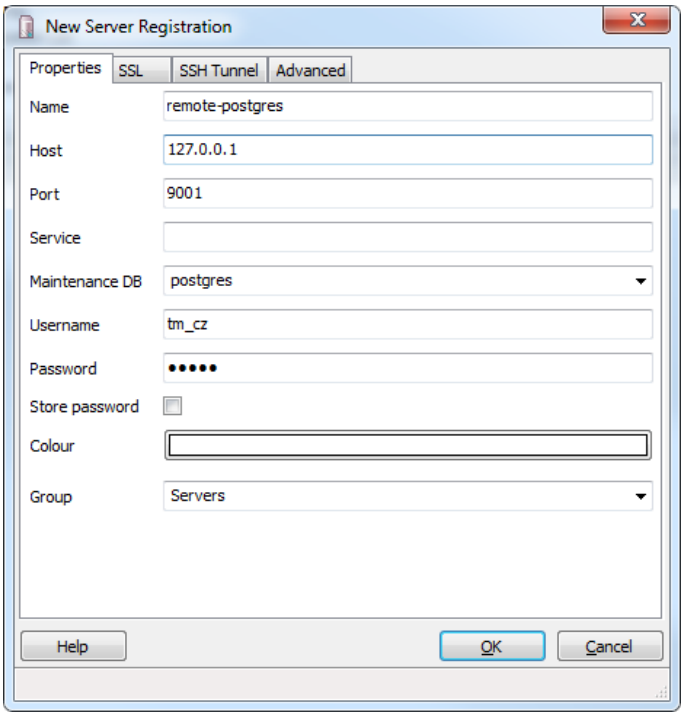

Figure 13: Add a server in pgAdmin.

On the left side of the main pgAdmin window is the *Object browser*. You should see your newly created server connection here. Double-clicking the server name makes pgAdmin connect to the server. A tree structure is shown representing the contents of the database server. Open the *Databases* node, then open the *transmart* node and the *schemas* node. This shows all schemas currently in the transmart database (see Figure 14).

Open the *Tables* node under the *tm_cz* schema node and find the `cz_job_master`, `cz_job_audit` and `cz_job_error` tables. You can right-click on any of these tables and choose *View data/View Last 100 rows* to see the most recent information, as shown in Figure 15. Alternatively you can open the *Query tool* from the *Tools* menu, and enter the SQL queries described above.

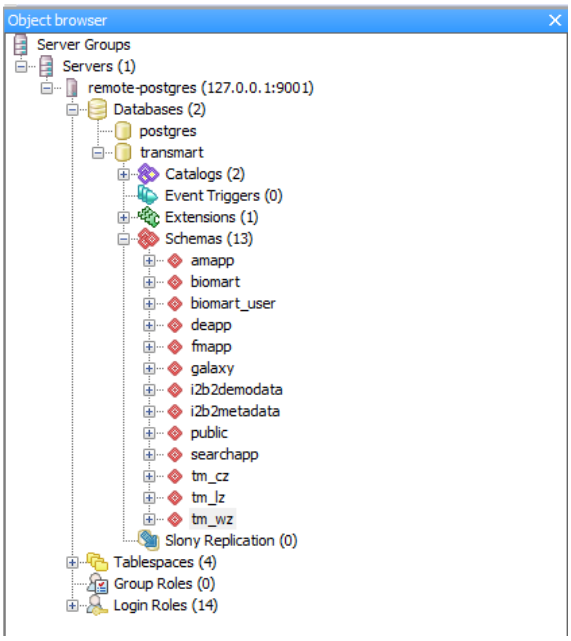

Figure 14: The Object browser in pgAdmin.

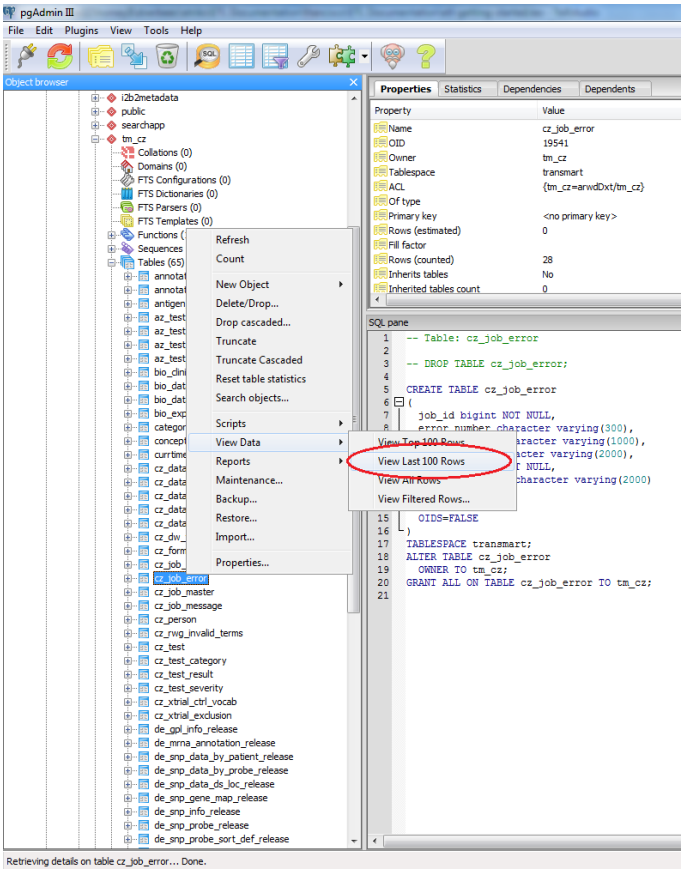

Figure 15: Retrieve the most recent information from a table in pgAdmin.

Appendix

A Sample kettle.properties file

```
# Note: lines like these with a # in front of it are comments
# Replace XXX by the server name or ip address, this is in most cases 127.0.0.1
#  e.g. COMMON_DB_SERVER=127.0.0.1
# Replace YYY by the port number, for Postgres the default is 5432
#  if you are using a tunnel, write your local tunnel port here (e.g. 9001)
#  e.g. COMMON_DB_PORT=9001

COMMON_DB_SERVER=XXX
COMMON_DB_PORT=YYY
COMMON_DB_NAME=transmart

# Set schema variables

TM_CZ_DB_NAME=${COMMON_DB_NAME}
TM_CZ_DB_PORT=${COMMON_DB_PORT}
TM_CZ_DB_SERVER=${COMMON_DB_SERVER}
TM_CZ_DB_PWD=tm_cz
TM_CZ_DB_USER=tm_cz

TM_LZ_DB_NAME=${COMMON_DB_NAME}
TM_LZ_DB_PORT=${COMMON_DB_PORT}
TM_LZ_DB_SERVER=${COMMON_DB_SERVER}
TM_LZ_DB_PWD=tm_lz
TM_LZ_DB_USER=tm_lz

DEAPP_DB_NAME=${COMMON_DB_NAME}
DEAPP_DB_PORT=${COMMON_DB_PORT}
DEAPP_DB_SERVER=${COMMON_DB_SERVER}
DEAPP_DB_PWD=deapp
DEAPP_DB_USER=deapp

BIOMART_DB_NAME=${COMMON_DB_NAME}
BIOMART_DB_PORT=${COMMON_DB_PORT}
BIOMART_DB_SERVER=${COMMON_DB_SERVER}
BIOMART_DB_PWD=biomart
BIOMART_DB_USER=biomart

I2B2DEMODATA_DB_NAME=${COMMON_DB_NAME}
I2B2DEMODATA_DB_PORT=${COMMON_DB_PORT}
I2B2DEMODATA_DB_SERVER=${COMMON_DB_SERVER}
I2B2DEMODATA_DB_PWD=i2b2demodata
I2B2DEMODATA_DB_USER=i2b2demodata

BIOMART_STAGE_DB_NAME=${COMMON_DB_NAME}
BIOMART_STAGE_DB_PORT=${COMMON_DB_PORT}
BIOMART_STAGE_DB_SERVER=${COMMON_DB_SERVER}
BIOMART_STAGE_DB_PWD=biomart_stage
BIOMART_STAGE_DB_USER=biomart_stage
```

B Removing a study

Save the following script to a file, e.g. `remove_study.sh`.

```
#!/bin/bash

if [ $# -ne 1 ]; then
  echo "Completely remove a study. This includes clinical and omics data."
  echo "Usage: $0 <STUDY_ID>"
  echo "  STUDY_ID: e.g. GSEXXXX, i.e. all caps"
  exit
fi

sed "s/{STUDYID}/$1/" <<EOD
DELETE FROM i2b2metadata.i2b2_tags WHERE path=
 (SELECT c_fullname FROM i2b2metadata.i2b2
  WHERE sourcesystem_cd='{STUDYID}' ORDER BY c_hlevel ASC LIMIT 1);
DELETE FROM i2b2demodata.concept_dimension WHERE sourcesystem_cd='{STUDYID}';
DELETE FROM i2b2demodata.concept_counts WHERE concept_path IN
 (SELECT c_fullname FROM i2b2metadata.i2b2   WHERE sourcesystem_cd='{STUDYID}');
DELETE FROM i2b2demodata.patient_dimension WHERE sourcesystem_cd LIKE '{STUDYID}:%';
DELETE FROM i2b2demodata.observation_fact WHERE sourcesystem_cd='{STUDYID}';
DELETE FROM i2b2demodata.patient_trial WHERE trial='{STUDYID}';
DELETE FROM deapp.de_subject_microarray_data WHERE trial_name='{STUDYID}';
DELETE FROM deapp.de_subject_sample_mapping WHERE TRIAL_NAME='{STUDYID}';
DELETE FROM i2b2metadata.i2b2 WHERE sourcesystem_cd='{STUDYID}';
DELETE FROM i2b2metadata.i2b2_secure WHERE c_fullname='{STUDYID}';
EOD
```

Now you can generate the necessary SQL commands by running the script, giving the study ID as a parameter:

```
bash remove_study.sh GSEXXXX
```

You can also directly execute the generated SQL:

```
bash remove_study.sh GSEXXXX | psql transmart
```

Version History

Version 1.0	May 13, 2015	Initial Version

Contributors

- Denny Verbeeck, J&J. dverbeec@its.jnj.com
- Francisco Bonachela Capdevilla, J&J. fcapdevi@its.jnj.com

References

[1] Recombinant Data Corp., *Dataset Explorer ETL Guide*, 2012, Available at https://wiki.transmartfoundation.org/download/attachments/131201/tranSMART_DSE_ETL_Guide.pdf.

[2] tranSMART Foundation Wiki, ETL Section, Available at https://wiki.transmartfoundation.org/display/TSMTGPL/Data+Sharing+and+Loading.

[3] NCBI-GEO, Available at http://www.ncbi.nlm.nih.gov/geo/query/acc.cgi.