**Command line Data Dumper Tool**

**User Documentation**

*Last Modified: 01/02/10*

*Author: Guillermo C. Martínez*

*Email:* [*gcomesana@cnio.es*](mailto:gcomesana@cnio.es)

*Team: -*

*Client: Genetic & Molecular Epidemiology Group*

logoElixirINB.jpg

Table of Contents

1. Introduction 3

2. Features 3

3. How to use 3

3.1 Command line options 4

3.2 Examples of use 5

**Standalone Data Dumper Tool.   
User Documentation**

# 1. Introduction

As the data is introduced in the database, the same data has to be retrieved. At this point and under the current circumstances by Jan 2010, an easy but practical solution was came up with. Then, a standalone command-line program was implemented to yield *comma-separated values* files with the requested data.

The main purpose of this standalone program is, first of all, to retrieve data to curate the database. Further purposes are oriented to statistical calculations. Since the tool is not available online, user and role checking when retrieving data was considered not necessary as the tool is distributed in a controlled way. This is a simplification over the ideal dump web application.

This web application, implementation due to a near future, should look an improved user interface and likely more queries as requested by customer.

# 2. Features

The dumper tool is a java program based on the same development as the main form builder tool. It is bundled as a *jar* file with all necessary custom and third party libraries packed in the jar file. So, this is an easy way to distribute the tool as it is independent on libraries and the only necessary requirement is to have Java 1.5 or later installed on the system (which is the default in many systems, otherwise it can be downloaded from <http://java.sun.com/javase/downloads/index.jsp>, choosing JRE for better).

The result is a simple program which yields *csv* files, inside which each row holds the requested data for one subject or sample, and each column represents the values for one question. So, *one cell* in the *csv* file is the value of one subject for one question. There can be columns with many blank values for many subjects, which means subjects with more answers for repeatable questions than others (when repeatable questions, there will be different amount of answers among all subjects, and subjects with less answers than the subject(s) with highest amount, will have their cell as blanks, in order to make match further column questions). “Missing” values are displayed as they are in the database (‘9999’ by convention). So, the difference between “missing” values and null (blank) cells is that a missing value -9999- is set by database when the field exist in the questionnaire and is not filled in. But a blank or null cell is generated when, for a subject or sample, a field (or group of fields) is repeatable, but not for others. In this case it is necessary in order to create all fields (columns) for the download to get match between all patients/samples, that these fields appear as null.

The yielded file is suitable to be loaded in any statistical program or spreadsheet. A recent request was the chance to add as input a two-column file mapping the question codes to variable names in order to make the columns more descriptive. The column header for the output data file is the question codes; by adding the *mapping file*, the column header will be the *variable names*, which are intended to be more descriptive.

This mapping file is just a text file with a header line and lines forming pairs “*question code; header column label*”. The header column labels will be the labels displayed in the columns for the right question code. For example, lets the line in a variable names file “*A3D;COB\_MGF*”. This means the column for the question with question code *A3D* will be replaced with *COB\_MGF*, which is a more meaninful label for the curator.

# 3. How to use

First of all, a console is necessary as the program runs on command line. This is trivial for Unix, as the system is mostly controlled by command line. For Mac users, the *terminal* tool is available with Mac Os X, and the *iterm* tool is available from <http://iterm.sourceforge.net/>. For Windows users, the *command prompt* or similar tool has to be run.

Once the proper tool is started the next steps are the same no matter the operating system or tool. Go to the directory where the jar file is located in your system. Assuming Java1.5 is correctly installed on the system, the program is ready to be run as:

jpop:appform bioinfo$ java -jar appform-dump-padme.jar

Usage is:

java -jar customp-dump.jar -p project\_name -i questionnaire [-g hostpial/group name] -s section order [-o sort order] -f filename.csv

Database dumper:

Options are (order is irrelevant):

-p project name

-i questionnaire name

-g hospital/group name

-s section order in questionnaire

-f file name (absolute or relative)

Spaces between switches and names are relevant.

## 3.1 Command line options

The program has to be parametrized in order to know which data to be retrieved. The following are the meaning of the mandatory options:

* -p <project name>, this is the *name of the project* which the interview belongs to.
* -i <questionnaire name>, the *name of the questionnaire* belonging to the project.
* -s <section order>, this parameter is a number, which is the position of the section in the questionnaire, starting with 1. No more than one section can be retrieved at this point and that particular feature was not requested.
* -h <variable names file>, the name of the file mapping question codes into names. These names will be the labels displayed as header columns.
* -batch <dump config filename>, perform in batch (unattended) mode. Useful when scheculing dumps in a time period basis.
* -f <file out name>, the *name of the output csv file*. The path can be absolute or relative. The output file must not exist. If the path is relative, the directory where the output file will be created will be relative to the application path directory.

If only the above described options are used, the result file will contain the data for all subjects who were interviewed for the <questionnaire name> for the project <project name>. Now, to fine the output, a couple of no mandatory options can be added:

* -g <hospital name>, restrict the subjects only those who belong to the hospital <hospital name>

## 3.2 Examples of use

* Get all data for the answers to the questions in section *Water* for *all subjects* in the project *PanGene-Eu* and questionnaire *QES\_EnglishUK* (in bold the command line sentence)

java -jar appform-dump-padme.jar -p PanGen-Eu -s 5 -i QES\_EnglishUK -f exampleUk.csv

3.1.1|4.1.1|5.1.1|6.1.1|9.1.1|10.1.1|11.1.1|11.1.2|12.1.1|14.1.1|15.1.1|16.1.1|17.1.1|18.1.1|19.1.1|20.1.1|20.1.2|21.1.1|24.1.1|25.1.1|27.1.1|28.1.1|30.1.1|31.1.1|34.1.1|37.1.1|37.1.2|38.1.1|40.1.1|40.1.2|41.1.1|43.1.1|43.1.2|44.1.1|47.1.1|50.1.1|50.1.2|51.1.1|51.1.2|54.1.1|55.1.1|57.1.1|58.1.1|60.1.1|62.1.1|64.1.1|66.1.1|67.1.1|68.1.1|

Writing file header as:

subject|group|interview|section|D1\_1-1-1|D1\_2-1-1|D1\_3-1-1|D1\_4-1-1|D1B\_1-1-1|D1B\_2-1-1|D2-1-1|D2-1-2|D3-1-1|D4\_0-1-1|D4\_1-1-1|D4\_2-1-1|D4\_3-1-1|D4\_4-1-1|D4\_5-1-1|D5-1-1|D5-1-2|D6-1-1|D7\_1A-1-1|D7\_1B-1-1|D7\_2A-1-1|D7\_2B-1-1|D7\_3A-1-1|D7\_3B-1-1|D8-1-1|D9A-1-1|D9A-1-2|D10A-1-1|D9B-1-1|D9B-1-2|D10B-1-1|D9C-1-1|D9C-1-2|D10C-1-1|D11-1-1|D12A-1-1|D12A-1-2|D12B-1-1|D12B-1-2|D13A-1-1|D13B-1-1|D14A-1-1|D14B-1-1|D15A-1-1|D15B-1-1|D16A-1-1|D16B-1-1|D1\_6-1-1|D4\_6-1-1

Total subjects collected: 21

End!!

This is the screen output so far. First is the header for the output csv file. The separator is the pipe (|) and it was chosen to prevent the same character is found in the raw data.

* The same example restricting the output to the Royal London Hospital subject. The input and console output are as is showed:

java -jar appform-dump-padme.jar -p PanGen-Eu -s 5 -i QES\_EnglishUK -o 1 -g Royal London Hospital -f exampleUk-inv.csv

3.1.1|4.1.1|5.1.1|6.1.1|9.1.1|10.1.1|11.1.1|11.1.2|12.1.1|14.1.1|15.1.1|16.1.1|17.1.1|18.1.1|19.1.1|20.1.1|20.1.2|21.1.1|24.1.1|25.1.1|27.1.1|28.1.1|30.1.1|31.1.1|34.1.1|37.1.1|37.1.2|38.1.1|40.1.1|40.1.2|41.1.1|43.1.1|43.1.2|44.1.1|47.1.1|50.1.1|50.1.2|51.1.1|51.1.2|54.1.1|55.1.1|57.1.1|58.1.1|60.1.1|62.1.1|64.1.1|66.1.1|67.1.1|68.1.1|

Writing file header as:

subject|group|interview|section|D1\_1-1-1|D1\_2-1-1|D1\_3-1-1|D1\_4-1-1|D1B\_1-1-1|D1B\_2-1-1|D2-1-1|D2-1-2|D3-1-1|D4\_0-1-1|D4\_1-1-1|D4\_2-1-1|D4\_3-1-1|D4\_4-1-1|D4\_5-1-1|D5-1-1|D5-1-2|D6-1-1|D7\_1A-1-1|D7\_1B-1-1|D7\_2A-1-1|D7\_2B-1-1|D7\_3A-1-1|D7\_3B-1-1|D8-1-1|D9A-1-1|D9A-1-2|D10A-1-1|D9B-1-1|D9B-1-2|D10B-1-1|D9C-1-1|D9C-1-2|D10C-1-1|D11-1-1|D12A-1-1|D12A-1-2|D12B-1-1|D12B-1-2|D13A-1-1|D13B-1-1|D14A-1-1|D14B-1-1|D15A-1-1|D15B-1-1|D16A-1-1|D16B-1-1|D1\_6-1-1|D4\_6-1-1

Total subjects collected: 3

End!!

As it can be seen, the number of subjects retrieved is just 3, those ones filtered by hospital which belong to.

## 3.3 Enhancement: Batch operation

In order to provide an unattended way to get backups in a period basis (hourly, daily, …), a batch feature was developed. However, a cron or similar facility is necessary in order to schedule the execution of the backup tasks, as a custom scheduler is not part of the application (it can be provided upon future demands).

To set the **batch mode**, the parameter –batch has to be provided. This parameter is exclusive from those ones previously described. Following, the configuration or properties file has to be provided describing the backups. Following is an example of the command line for a batch processing:

:$ java -jar ${**JAR\_PATH**}/**padme-batchdumper.jar** -batch ${**PROPS\_DIR**}/**bulks.props**

**JAR\_PATH** and **PROPS\_DIR** are environment variables to locate the jar file (in this case **padme-batchdumper.jar**) and the properties file (**bulks.props**).

The backup configuration file is just like a java properties file. The keys are as follows:

* **qrytype**: the type of bulk to be retrieved. It can be one of four options:
  + **normal** or nothing. It is the default option. It retrieves a set of patient data according to project, interview and section parameters
  + **questotals**. Retrieves the number of answers for a set of questions of a questionnaire grouping by the possible values.
  + **totals**. It gives the number of subjects with at least one answer for any question for all interviews. Intended for curation purposes.
  + **alltotals**. It retrieves the number of subjects which have a performance done, and the number of answers for any question in the performances. This is addressed to see how many subjects with performances but no answer are held in the database. It is more detailed than the previous one
* **filename**: the name of the file where the results will be dumped
* **prj**: the name of the project where the data is to be retrieved from
* **intrv**: the name of the interview belonging to the project prj the data is to be retrieved from
* **sec**: the name of the section inside de interview intrv. This is necessary to get an homogeneus and size limited bulk of data. If several sections want to be retrieved, different backup configuration blocks has to be declared for every section
* **grp**: the group the patients belong to in order to retrieve the data
* **ques**:this parameter applies only when *qrytype* is *questotals*. The value of this parameter will be the code of the question (e.g. C2) the total data wants to be retrieved.

As several custom backups can be defined in a single configuration file, a **postfix** number has to be added to the parameters in order to group those ones in right blocks. This feature is showed in the next example:

# this is a comment line

qrytype3: normal

prj3:PanGen-Eu

grp3:Hospital de la Santa Creu I San Pau

sec3:4

intrv3:QES\_Español

filename3: padme-listSantpauSec4.csv

# para tu pregunta

qrytype4: questotals

prj4: PanGen-Eu

intrv4: QES\_Ireland

sec4: 3

grp4: Mater Hospital Dublin

ques4: A1,A3

filename4: padme-a1a3TotalsIEPangen.csv

# no section or group needed for question totals

qrytype8: questotals

prj8: ISBlaC

intrv8: SampleCollection\_FRANCE

# sec8: 1

# grp8: Hospital del Mar

ques8: A1

filename8: padme-a1Totals-ISBLaC-France.csv

# minimun requirements for alltotals

qrytype5:alltotals

filename5: padme-alltotals.csv

At the example above, the postfix numbers group different backups (four backups are showed here). When qrytype is questotals, more than one question code can be specified in a comma-separated list. The postfix numbers are arbitrary as their only purpose is to assemble the different parameters.

So, the recommended procedure to set up automatic backups is as follows:

1. Set a directory with necessary privileges to dump the backup files yielded by the backup tool
2. Write the configuration file
3. It is recommended to write a system script file in order to gather and, eventually, compress or perform whatever post-processing task on the generated files. An example of this file is showed next:

#!/bin/bash

DAY=$(date +%d%m%Y)

JAR\_PATH=/home/gcomesana/Development/cron

PROPS\_DIR=/home/gcomesana/Development/cron

TARGZ\_DIR=/home/gcomesana/Development/backup/custom

# DUMP\_DIR=/home/gcomesana/Development/backup

source /usr/local/Modules/3.2.0/init/bash

module add java/java32vm

java -jar ${JAR\_PATH}/**padme-batchdumper.jar** -batch ${PROPS\_DIR}/**bulks.props**

tar zcf ${TARGZ\_DIR}/padme-bulks-${DAY}.tar.gz \*-${DAY}\*.csv

rm \*-${DAY}\*.csv

# cd ${DUMP\_DIR}

Basically, in this script the java program (under the **padme-batchdumper.jar** name) is run taking a configuration file named **bulks.props**. Then, the files yielded are tar-gzipped into a new tar-gz file supposing the generated filenames match with the pattern **\*-${DAY}\*.csv**.