COMMAND-LINE INTERFACE USER MANUAL

Version 3.2.0
6.8.2012

Bank of Finland
PAYMENT AND SETTLEMENT
SYSTEM SIMULATOR

Contents

1. Introduction

The **command-line interface (CLI)** of the simulator can be used to configure and run simulations without starting the graphical user interface (GUI). It is useful when large number of similar tasks needs to be performed. Example of such situation could be repetition of a certain simulation for hundreds of times with different data sets. With the CLI it is also possible to use the BoF-PSS2 via other software responsible for creating the simulated data or commanding the execution.

CLI is designed to cover the most frequently used repetitive tasks. The graphical user interface is still used in tasks which are considered to be typically one off actions. Examples of these are creation of new projects, setting the data format defaults, definition of new templates and also definition of system data set contents. The operations currently supported by the CLI are importing datasets, configuration of simulations, executing simulations and exporting results.

In the command mode the simulator software is divided in two parts. First, there is server software, which is the actual simulator executing the given commands similarly as in the graphical mode. This server is displayed as a console window. The server is started by executing the Start-up-SERVER.bat file located in C:\BoF-PSS2\PROGRAM. The server must be running before any commands to the simulator can be given with the script language. The server is closed by executing the C:\BoF-PSS2\PROGRAM\PSS2_server_shutdown.bat file.

Second part is the client program, C:\BoF-PSS2\PROGRAM\BoF-PSS2.bat, which is called in the command prompt with the actual script command given in the argument. It passes on the individual command parameters to the server and collects the feedback of the commands submitted.

There are three general options for using the command interface:

a. Giving the individual commands directly in command prompt of the operating system, i.e. opening up the command prompt and executing the command.
b. Using a program which forms the script commands and submits them to the operating system to be executed in command prompt.
c. Writing a text file with a batch of sequential commands to be executed. The launch of the execution is done again via the command prompt. The text file can be created with Notepad, Excel or any other suitable program.

The commands for individual simulator tasks are the same in all options and are described in section 2 which covers option *a* for command launching. Using batch files and additional commands in this alternative, i.e. option *c*, are covered in section 3.

Calling the simulator CLI from a third party program, i.e. option *b*, is done by utilising either one of the other two alternatives. It depends more on usage of the third party program or the programming language that will be used for the task. Examples of programs to which the simulator could be connected are Matlab, Mathematica or any user created program.

Error handling and return values of the simulator command-line interface are described in section 4.

Note that you can use the graphical and command-line user interfaces in turns. Those tasks which are not repetitive are often easier to do using the graphical interface and those that have to be repeated several time with small changes in the parameters can be efficiently and rapidly executed using the command-line interface. **Using both interfaces simultaneously is however impossible.**

2. Giving individual commands

The simulator can execute individual commands which are entered one at a time with the relevant parameters. Results of simulations executed through the CLI are written to the same output database as those executed via GUI. The feedback related to interpretation and execution of commands given to the simulator can be directed to a separate feedback text file. The feedback file will contain any error messages due to errors in the command-line syntax or data and any error messages from the simulations tasks that normally would be presented via the graphical interface.

The individual commands can perform one of the following simulator tasks: import input data, configure a simulation, execute a simulation or export output data.

2.1. Notation syntax

Parameters are given as a text string, where reserved keywords are preceded with whitespace (space character) and a dash/minus sign (i.e. "-"). The command syntax is presented in this manual so that the **bold words** represent reserved keywords, which must be stated as such, and *italic words* represent variable user input such as names of templates or projects. Note that the commands and variables are given in plain text in the command prompt. The bold and italic style is just used for clarification in this manual.

Parameters, which are optional, are shown in this manual in square brackets []. The order of parameters in an actual command is arbitrary. However, in these instructions the parameter defining the main task is always stated in the beginning of each command.

Parameters can also be separated with comma or equal sign "=" instead of whitespace. Therefore space, dash, comma or equal sign can only be used in the variable user input, i.e. data set, template or file names, when placing the user input value in double quotes. For example referring to template *SYLS-ALL* is written as **-template:***"SYLS-ALL"*.

2.2. Import input file

Data can be imported from a CSV file to the input database of a project by using the **-import** command. When importing data from the command-line interface the data template defining the format of the CSV-file has to be created first in the graphical user interface. Note also that the CLI will use the same data format defaults (date, time, separators) as were last used in the

GUI. Possible errors in the input file are reported in error files in the ERRORLIST directory in the project folder.

Import command syntax:

BoF-PSS2 **-import -table**:*table name* **-project**:*project id* **-dataset**:*data set*
-inputfile:*file path* **-system**:*system id* **-template**:*template name* [**-update**]
[**-insert**] [**-feedbackfile***: file path*]

Parameter code	**Parameter value**
-import	command for importing input data
-table	the database table to import in (e.g. TRAN or PART)
-project	project ID
-dataset	data set to be created or updated
-inputfile	path to the CSV-file to input
-system	system ID
-template	name of the template to be used
-update	When this parameter is given, the imported data is used to update existing data set. The -update parameter does not allow the insertion of new rows. This parameter is alternative to -insert. If none of these are given, default action is to create a new data set by replacing the existing one. ***Optional***.
-insert	When this parameter is used, the imported data is inserted as new rows in an existing data set. This parameter is alternative to -update. If none of these are given, default action is to create a new data set. ***Optional***.
-feedbackfile	feedback file name and path for storing the result of the import operation. Feedback is also displayed in the simulator server window. ***Optional***

Example:

BoF-PSS2 **-import -table**:*TRAN* **-project**:*Example* **-dataset**:*tr01*
-inputfile:*c:\temp\trans11.csv* **-system**:*rtgs* **-template**:*TRAN_ALL*

This example command would import new transaction data to the TRAN table in the database of the project "Example" and its system “rtgs” from the input file “trans11.csv” using template “TRAN_ALL”. A new data set named “tr01” would be created.

2.3. Simulation configuration

Simulations can be configured using the command "**-configure**", which basically generates a simulation ID and attaches the given data sets to the simulation ID.

Configure command syntax:

BoF-PSS2 **-configure -project**:*project id* **-simulationID**:*simulation id*
[**-name**:*simulation name*] [**-description:***simulation description*]
[**-feedbackfile**:*file path*] [**-skipcrosscheck**] [**-submission**:*algorithm name*]
-system(ID:*system id*/**sycd**: *data set*/**tran**:*data set*/**part**:*data set*
[/**iccl**:*data set*][/**dbal**:*data set*][/**blim**:*data set*] **)**

<table>
<tr><th>Parameter code</th><th>Parameter value</th></tr>
<tr><td>-configure</td><td>command for configuring simulations</td></tr>
<tr><td>-project</td><td>project ID</td></tr>
<tr><td>-simulationID</td><td>simulation ID to create or modify</td></tr>
<tr><td>-name</td><td>simulation name (optional)</td></tr>
<tr><td>-description</td><td>simulation description (optional)</td></tr>
<tr><td>-skipcrosscheck</td><td>The cross-check will be skipped if this parameter is given. Without it the cross-check will be executed. (optional)</td></tr>
<tr><td>-submission</td><td>The name of the submission algorithm to use. If the parameter is not given, default algorithm SUFIFOPR is used. (optional)</td></tr>
<tr><td>-system()</td><td>The data sets to be used in an individual system. Values of sub-parameters are given as a list separated with slashes "/" within the parenthesis. This parameter can be given multiple times if the simulation includes multiple systems.
Sub-parameters:
<table>
<tr><td>ID</td><td>System ID</td></tr>
<tr><td>sycd</td><td>existing system data set to use</td></tr>
<tr><td>tran</td><td>existing transaction data set</td></tr>
<tr><td>part</td><td>existing participant data set</td></tr>
<tr><td>iccl</td><td>existing intraday credit limit data set (optional)</td></tr>
<tr><td>dbal</td><td>existing daily balances data set (optional)</td></tr>
<tr><td>blim</td><td>existing bilateral credit limit data set (optional)</td></tr>
</table></td></tr>
<tr><td>-feedbackfile</td><td>feedback file name and path for storing the result of the configuration operation. Feedback is also displayed in the simulator server window (optional).</td></tr>
</table>

Note: If there are more systems belonging to the same simulation the system parameter and its sub-parameters are stated multiple times.

Also note that the feedback file includes important information on how successful the cross-check has been. In a situation in which the cross-check is unsuccessful the simulation ID will not be created or modified and saved.

Example:

BoF-PSS2 **-configure -project**:*Example* **-simulationID**:*sim1*
-system(**ID**:*rtgs*/**sycd**:*syset1*/**tran**:*tset2*/**part**:*banks*)

This example command would create a new simulation ID “sim1” in the project “Example”. The simulation has one system “rtgs” and uses system data set “syset1”, transaction data set “tset2” and participant data set “banks”.

2.4. Simulation execution

Simulations can be executed using the "**-execute**" command. The simulation IDs executed must be available in the input database.

Execute command syntax:

BoF-PSS2 **-execute -simulationID:***simulation id* **-project:***project id*
[**–statistics(***SYLS/ACST/TEST/NEST/AVST/BEST/UNST/SUST/QUST/BIST/QURE***)**]
[**-skipcrosscheck**][**-feedbackfile:***file path*]

Parameter code	**Parameter value**
-execute	command for executing simulations
-project	project ID
-simulationID	simulation ID to execute
-skipcrosscheck	The cross-check will be skipped if this parameter is given. Without it the cross-check will be executed. (**optional**)
-statistics()	States which statistics are saved (**optional**). Required output tables are given within parenthesis as a list of table names in capital letters separated with slashes ("/").
-feedbackfile	feedback file name and path for storing the result of the execution operation. Feedback is also displayed in the simulator server window (**optional**).

Example:

BoF-PSS2 **-execute -simulationID**:*sim1* **-project**:*Example* **-statistics**(*ACST*/*TEST*)

This example command would execute simulation ID "sim1" in the project "Example". Account level statistics and transaction event statistics would be written into the output database.

2.5. Export output file

Data can be exported from the output database using the command "**-export**". (Another advanced output option is to read directly the data from the database using the SQL-interface of MySQL, see chapter on MySQL interfaces in the BoF-PSS2 User Manual).

The data templates to be used when creating the output CSV-files using the CLI have to be first created using the graphical user interface as the CLI does not support template creation. Note also that the CLI will use the same data format defaults (date, time, separators) as were last used in the GUI.

Export command syntax:

BoF-PSS2 **-export -table**:*table name* **-simulationID**:*simulation id* **-project**:*project id*
-system:*system id* **-template**:*template name* **-exportfile**:*file path*
[**-createnames**][**-feedbackfile**:*file path*]

Parameter code	Parameter value
-export	command for exporting output file
-project	project ID
-simulationID	simulation ID
-system	system ID of the relevant system
-table	table to export
-exportfile	file path of the export output file to create
-template	name of the data template to use
-createnames	No value. When present, the names of the columns will be written in the output file. If not present, the names will not be written. (**optional**)
-feedbackfile	feedback file name and path for storing information about the export operation. Feedback is also displayed in the simulator server window (**optional**).

Example:
BoF-PSS2 **-export** **-table**:*SYLS* **-simulationID**:*sim1* **-project**:*Example*
-exportfile:*c:\temp\syls.csv* **-template**:*st02* **-createnames**

This example command would export system level statistics from the simulation ID “sim1” in the project “Example” into a file named “syls.csv”. Template used for export is “st02” and names of columns are written into the file.

2.6. Exporting all simulation output

The "**-writeresult**" command exports the contents of all output databases to CSV files in the folder path given with the **-feedbackfile** parameter.

Parameter code	Parameter value
-writeresult	command for exporting all output databases into CSV-files
-project	project ID
-simulationID	simulation ID
-feedbackfile	Path to the folder where the CSV files will be written (**optional**). If not given, the files will be written into C:\ (i.e. drive root).

Example:

BoF-PSS2 **-writeresult** **-project**:*valid1* **-simulationID**:*sim1* **-feedbackfile**:*C:\results*

This example command would export all results from the simulation ID "sim1" in the project "valid1". The CSV-files would be written into the directory "C:\results". Note that even the statistics that have not been calculated will be written into files, resulting in empty files.

2.7. ,Stopping an operation

The "**-stop**" command can be used to stop a long-running operation or a sequence of command given via a command file. This is analogous to pressing the "stop simulation" /

"stop import" / "stop export" etc. buttons in the graphical user interface. The response to this command is not always immediate. It depends on the state of the currently performed task and size of data.

If a task is stopped, the already finished results are saved. For example during a simulation, results from those days, which were fully processed and written to the output database are not affected, when the simulation is aborted.

3. Using a command file

The command-line interface can also be used with a command file as parameter. It is a text file which contains individual commands separated by line breaks, i.e. each on its own line. The commands are the same as described in chapter 2. Alternatively the separate commands can be placed in an operating system batch file (e.g. .BAT). The command file offers some advantages compared to the use of operating system's batch file.

The command file alternative gives the possibility to create a large batch of commands to be executed by the simulator. The file can be created by using a text editor or any other program capable of writing text files. Lines are executed one by one starting from top.

Execution of the command file is by default aborted if an erroneous line is encountered. By using the **-ignore** parameter the simulator can be asked to ignore erroneous lines and only skip to the next command in case of an error.

Command file execution syntax:

BoF-PSS2 –**commandfile:***file path* [**-feedbackfile:***file path*] [**-ignore**]

Parameter code	**Parameter value**
-commandfile	command for executing a command file
-ignore	If this parameter is used, the execution of the command file is continued after encountering an erroneous command. (**optional**)
-feedbackfile	feedback file name and path for storing information about the execution of the commands (**optional**). Feedback is also displayed in the simulator server window.

The command file consists of a number of individual commands. These have to be separated by line breaks. Each individual command must be on its own line.

If a feedback file is defined together with -commandfile argument, this setting will override the possible definitions of feedback files in the individual command file commands. As a result all feedback will be written into the one general feedback file.

The command file can also contain comment rows marked by double slashes (//) in the beginning of the row. Comment rows can be placed between individual commands but not in the middle of a single command.

3.1. Example command file contents

Please note that line breaks in the middle of a command are not allowed and are present here only because of space constraints.

```
// import participant data, this is comment line.
bof-pss2 -import –table:PART –project:LVPS –dataset:banksA
–inputfile:C:\BoF-PSS2\input\part.csv –system:SYS1 –template:PTMP

// import transaction data
bof-pss2 -import –table:TRAN –project:LVPS –dataset:tranDat1
–inputfile:C:\BoF-PSS2\input\tran_sep04.csv –system:SYS1 –template:TTMP

// Configure two simulations from existing data sets with different system
// data sets
bof-pss2 –configure –project:LVPS –simulationID:sim1
–system(ID:rtgs/sycd:syst1/tran:tranDat1/part:banksA)

–configurate –project:LVPS –simulationID:sim2
–system(ID:rtgs,sycd:syst2,tran: tranDat1,part:banksA)

// execute first simulation. Ignore failures
-execute –simulationID:sim1 –project:LVPS –statistics(SYLS/ACST/TEST)

// execute another simulation. Skip cross-check and ignore failures
-execute -simulation:sim2 -project:LVPS -statistics(SYLS/ACST/TEST)
-skipcrosscheck
```

4. Error handling and return values

The CLI executes only those commands which have correct and faultless syntax. If the command is erroneous, an error message is displayed in the BoF-PSS2 server console window or in the feedback file, if such was defined.

The errors are displayed in following cases:

- The command is missing a mandatory parameter.
- The command has unknown parameters which cannot be parsed.
- The parameter values used in the command are incorrect.

The first class of errors is encountered if the main parameter (e.g. import, configure) or the parameters required by the main parameter (e.g. -project) are missing from the command. The second class of errors is encountered e.g. with mistyped parameters. The third class of errors is encountered if the command syntax is correct, but e.g. the project or template name is not present in the database.

Most error messages are displayed only one by one, i.e. if a command contains multiple errors, only the first one encountered is displayed.

All CLI commands r**eturn a binary value** depending on the outcome of the command. Successful commands return 1 and unsuccessful commands return 0. In case of unsuccessful commands an error message is also displayed in the server window. These values can be used

for example with third-party programs to verify that a command has been successfully executed.
Commands with return value 1 can still lead to errors in the actual task performed in the simulator. For example simulation configuration command syntax can be correct and thus return value 1 even if the defined simulation includes cross-check errors. Similarly the import command will return value 1 even if some of the rows in the input file contained errors.

5. Known bugs of the BoF-PSS2 CLI

At the time of writing there are some known bugs in the command-line interface of the simulator. These will be fixed in later updates.

- The **-configure** command does not support multiple systems. Only the last system defined in the command will actually be stored in the simulation. Workaround is to define multisystem simulations via GUI.
- If errors are found in cross-check when using the **-configure** command, only the number of errors is reported. To see detailed cross-check error information, you should use the GUI for cross-checking.
- New simulations are created by the **-configure** command even if errors are found in the cross-check. You can however configure the simulation again with corrected datasets.
- If cross-check founds errors when using the **-execute** command, the simulator will crash. The workaround is to use datasets without errors or to use the -skipcrosscheck parameter.
- When executing a command file and specifying a feedback file together with the **-commandfile** command, the main feedback file will not contain the feedback from individual commands. You can specify feedback files for the individual commands in the command file.
- Feedback file may display the same message multiple times. The error messages in the feedback file are not always as detailed as in the server window.
- When using the **-import** command the simulator may crash when the CSV-files to be imported contain incorrectly formatted data, e.g. the separator character used is different than was last defined when importing in the GUI. This happens when all of the rows in the input file are erroneous.
- Double quote usage as described in chapter 2.1 does not work in command files. This means that when using a command file you cannot use e.g. file or template names containing space, dash, comma or equal sign.
- When doing multiple **import** commands in a row with small input files containing errors, the error file produced by the last import command may overwrite previous error files.