USER MANUAL BOF-PSS3

Version 1 (Beta)

22.12.2020

Bank of Finland
PAYMENT AND
SETTLEMENT SYSTEM
SIMULATOR

Date	Description	Author
1.8.2013	New queue reason code for FIFO	Kasperi Korpinen
24.9.2013	Updating the output DB by deleting the deprecated one	Kasperi Korpinen
	Removal of chapter 5.7 Time transposition	Kasperi Korpinen
	Business day deduction dropdown	Kasperi Korpinen
7.8.2014	Examples removal of decimal comma	Kasperi Korpinen
	Log file conventions	
	Fujitsu contributors	
	My.cnf content	
	Execution of cross-checks before running	
7.12.2015	Stress Tester	Kasperi Korpinen
	Default directory for DB's	Kasperi Korpinen
	Joined the db manual to the same	Kasperi korpinen
4.1.2016	- notational 3.4.1	Kasperi Korpinen
21.9.2016	Version updating guidance	Kasperi Korpinen
	Task automation tools	
16.11.2016	Stress Tester	Kasperi Korpinen
27.6.2017	DB Rationalisation removed comparison views updated stress tester updated CCP task automation tools	Kasperi Korpinen
24.8.2017	Update the reference part dataset selection to the import functionality description	Kasperi Korpinen
21.9.2017	Changed tables etc. better	Maarit Aalto
5.10.2017	Added chapter on Import tools	Kasperi Korpinen
28.11.2017	New reference field into the database	Kasperi Korpinen
11.1.2018	Test table description update Run Benchmark task description.	Kasperi Korpinen
16.4.2018	Updated part data table	Kasperi Korpinen
30.5.2018	ABM algorithms	Kasperi Korpinen
24.8.2018	Updating the DB driver to a newer one	Kasperi Korpinen
3.1.2019	Installing instructions (Maria connector) and system requirements	Kasperi Korpinen
10.4.2019	Added descriptions to ABM	Kasperi Korpinen
8.10.2019	ABM	Kasperi Korpinen
3.12.2020	Stress tester report systemic unst	Kasperi Korpinen
3.12.2020	General description and architecture, Hardware recommendations, stress tester screen shots and text.	Kasperi Korpinen
4.12.2020	Remove mySQL installation instructions	Kasperi Korpinen
10.12.2020	Moved the HTTP API examples to annexes. Removed more MySQL references.	Harri Engblom
14.12.2020	Updated ABM description	Harri Engblom
15.12.2020	Updated ABM description	Kasperi Korpinen

Table of Contents

Quick installation guide

1) Download and install MariaDB 10.5
2) Request a simulator downloading link from BoF
3) Unzip the simulator package to `C:\BoF-PSS\`
4) Start the simulator by clicking `C:\BoF-PSS\startMin.cmd`

For more detailed installation instructions, please refer to CH 2.

1 Introduction

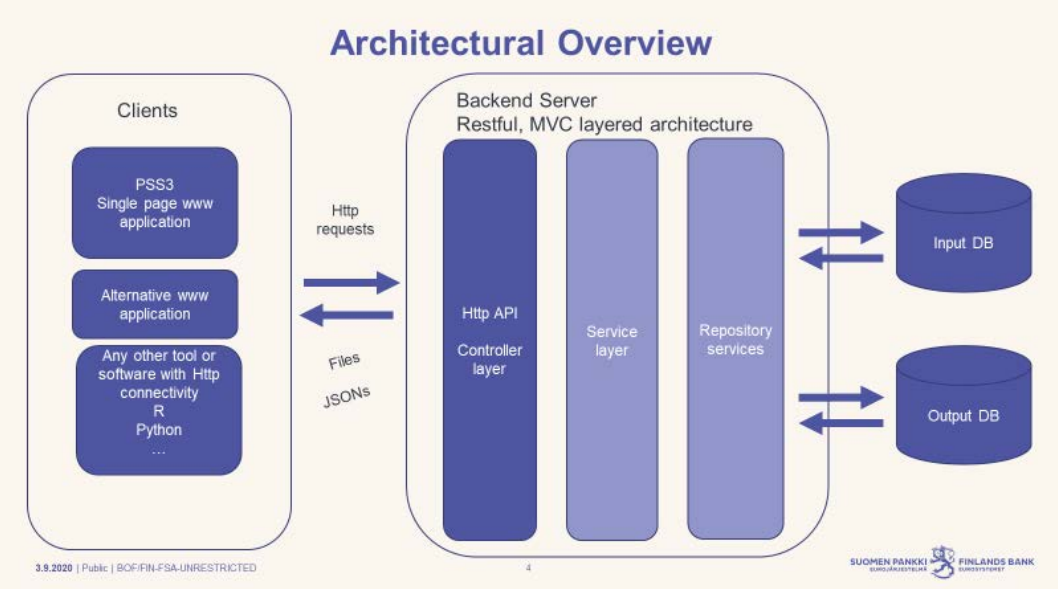
The Bank of Finland Payment and Settlement System Simulator (BoF-PSS3), is an analysis software designed for payment and settlement system simulations. The simulator can be used for studying liquidity needs and risks in payment and settlement systems. Special situations, which are often difficult or impossible to test in a real environment, can be simulated with this tool.

This document is the user manual of BoF-PSS3. It describes features of the software and their use. It also provides an overview of technical details of the simulator.

1.1 General overview

The BoF-PSS3 simulator consists of 3 main parts:
a) A graphical user interface implemented as a web-application using mainly techniques like html and javascript.
b) A back end server than can be installed on a regular windows PC.
c) Database storage. Currently MariaDB is used in development.

The architecture of the PSS3 program is pictured below:



1.1.1 Import and input functionalities

The simulator includes tools to import and validate transaction data, participant data, as well as data on daily account balances and credit limits. All data are stored

in database files. The user's main task is to check that the input data is formally valid and then import it into the simulator. The correctness of the input data is vital. Account ids in all files must correspond to the account ids in a participant dataset.

All input data must be presented in CSV (comma separated values) format, but it can be entered in a user-defined order. The input data can be edited by exporting them from the input database as CSV files to Excel. They can then be re-imported after the changes. Older Excel versions can handle about 65,000 rows. Excel 2010 is already able to handle ~1 000 000 rows. If larger files need to be edited, other tools (e.g. Python, Matlab, R, Access or SAS) or programming is usually needed. One option is to edit the data directly in the simulator's databases with SQL-queries. The use of SQL-queries requires some moderate technical skills. In rare situations, splitting tables in sub-tables may be a suitable solution. The simulator does not include a proprietary editor for this purpose.

1.1.2 Simulation execution

The simulator includes tools for configuring payment and settlement system setups and running simulations. The simulator records all events and bookings. Some premade reports and statistics on simulation runs are available. The simulator allows to set up and manage settlement structures, configure settlement rules and launch, monitor and control simulation runs. The simulator keeps a log file for the user of all simulations made.

1.1.3 Analysis functionalities and simulation results

The simulator has functionality for reporting basic statistics for common result parameters. The output database tables contain data amongst other for the booking order of transactions and balances of settlement accounts. The input database tables contain the transactions posted to the production system, while the output tables contain the settlement flow, i.e. settlement order and timing of submitted transactions.

Users typically perform many different simulations and want to compare the results of the different runs. When the simulator's basic reports are not enough, more complex or tailored analyses may require exporting CSV files for use with tools such as Excel or other statistical software. It is thus advisable to create a structure beforehand for simulation runs and determine which results are to be stored in databases for further analysis. The databases can become overly massive when

transaction volumes are high and all transaction-level events are retained in the databases. This is specifically the case when the automated stresstester is not used.

1.2 Supported system structures and simulation examples

BoF-PSS3 software supports a large variety of general system structures. It can model most of the payment and securities settlement system structures and processes found in real systems.

The simulator supports real-time gross settlement (RTGS), continuous net settlement (CNS) and deferred net settlement (DNS) systems and hybrid systems. The processing options for these systems are defined by selecting appropriate algorithms. For example, QUE algorithms define how transactions are released from queues, while PNS algorithms define when and how partial net settlement of queued transactions will be invoked.

The simulator also has multi-system capabilities, whereby a large number of interacting systems can be included in the same simulation (see chapter 3.4.3). When transactions occur between systems, they are booked in separate intersystem accounts. There are two types of intersystem transactions: straightforward participant-to-participant transactions or system invoked injection or settlement transactions between a main and ancillary system. In the straightforward case, the sending system's transaction data include a reference to a receiving participant in another system. It is possible in ancillary systems to define the end-of-day settlement system and accounts for each participant. Intraday injections may also be defined. These transfer liquidity between the main and ancillary system during the day according to participant needs.

Typical interacting system scenarios include:

- Several independent RTGS systems constituting a network of systems, e.g. TARGET,
- A domestic payment system environment consisting of an RTGS system and ancillary systems, e.g. a CNS and a DNS system settling in the RTGS system, and
- RTGS system settlement between an RTGS and a securities settlement system.

The simulator also supports multi-currency and multi-asset processing, which allows simulation of international payment systems and securities settlement systems. Assets are treated as book-entry currencies. Payment-versus-payment (PVP) and delivery-versus-payment (DVP) processing is supported. DVP/PVP

transaction pairs or groups should be connected via a DVP/PVP-link code. In addition to single intra-system DVP/PVP processing in RTGS or deferred net settlement mode, the simulator also supports RTGS DVP/PVP settlement between real-time systems.

The focal output factors in simulations are typically counterparty risk and overall risk, liquidity consumption, settlement volumes, gridlock situations and queuing time. Measures for these factors will be stored in the output database. In what-if simulations, the input parameters are modified to distinguish effects on output factors. The following input parameters are often used or modified in simulations:

- Input transaction flow (e.g. testing when a single counterparty or system has problems),
- Available liquidity,
- Credit limit/debit cap restrictions,
- Queuing and netting processes,
- Participant behaviour due to e.g. new pricing patterns,
- New settlement procedures, e.g. new algorithms,
- Structural changes, e.g. the merging of several systems,
- Changes in participant structure (e.g. introducing new participants, merging old participants), and
- New intersystem processes (e.g. a shift to RTGS-based DVP processing from end-of-day batch processing).

Liquidity is introduced to the simulations either by defining daily opening balances and/or intraday credit limits. Liquidity can also be introduced via repo-transactions and there are more alternatives available: introducing only the money legs between the participants and the central bank account (with abundant limit), introducing in DVP mode the money legs in the RTGS system and the asset legs in a separate securities settlement system or having a special collateral account (monetary value only) in the RTGS or securities settlement system.

Participant level risk management features can also be directly introduced in simulations by using bilateral limits (bilateral debit caps). These can be defined at bilateral and also at multilateral level separately from other liquidity arrangements.

Simulations may use available data from current systems or fictional, but representative, data. The simulator can be described as a deterministic model with stochastic input.

Data for some examples are distributed with the simulator software, e.g. an RTGS simulation, an RTGS system with an ancillary CNS or DNS system and a real-

time DVP securities settlement system. Some correct results are provided for all examples as illustration of what can be obtained as simulation output. All examples use semicolon as data separator points as decimal separator. The data for the examples and system descriptions are found in the directory C:\BoF-PSS\examples\DECIMAL_POINT.

2 Installation

Before using BoF-PSS software, you need to install MariaDB, MySQL (old) or MS SQL (less tested) database server. The simulator is distributed with a JDBC database connector. We recommend to use a MariaDB connector (mariadb-java-client-2.2.5.jar or newer) with MariaDB 10.5.

Information on how to order and download the BoF-PSS3 program is posted at https://www.suomenpankki.fi/en/financial-stability/bof-pss2-simulator/ordering/.

In case you are a user of BOF-PSS2 and wich to upgrade to PSS3. You should consider the following. We recommend to make a clean install. According to our instructions, PSS3 will anyway be installed in a different folder than PSS2. So in this sence you don't need to uninstall PSS2.

PSS3 uses the sam system database as PSS2 so here you might enter into conflicts. If you want to reset the `pss2_systemdb` you can simply make the SQl drop command on with Heidi SQL. On launch PSS3 will recreate the `pss2_systemdb` if it is missing. The early versions of PSS3 might still be compatible with the PSS2 `pss2_systemdb`.

2.1 Hardware and software requirements

Hardware

At least 64-bit system with a regular 4 core processor with 8 GB of memory is recommended. Sufficient main memory is essential for rapid execution of large transaction volumes. For large simulations 16 GB or more (>1,2 million transactions and >1000 participants) is recommended. Note that the 32-bit version of the Java virtual machine is able to use only approximately 1.5 GB of memory. In order to be able to allocate more memory than 1.5 GB to the Java virtual machine and the simulator, 64-bit versions of the Java Runtime Environment and the simulator are needed. Naturally the operating system also needs to be a 64-bit environment.

Since the version 9.3.0 the simulator supports parallelisation of simulations . To leverage this, you should run the simulator in an envaironement with at least 4 CPU's, preferably more than 6. Memory should be scaled up respectively. Approximately 1-2 GB per parallel simulation (eg. number of available cores). It has to be noted that bottle neck processes like DB operations do strongly benefit from higher performance cores and local SSD or NVMe hard drives.

The **BoF-PSS3** simulator can process massive transaction flows effectively with adequate available main memory resources. The complexity of the algorithms used and the selected output tables to be computed during the simulations strongly influence the running times and memory usage of simulations.

The **BoF-PSS3** simulator keeps all transactions and other input data to be processed during a simulation in the main memory. The amount of transactions is the decisive factor in main memory use. When there are more transactions than space in the main memory, system performance is likely to degrade strongly due to necessary disk swaps. Even then, the simulator continues processing during such circumstances until the limit of 1.5 GB is achieved for the 32-bit version.

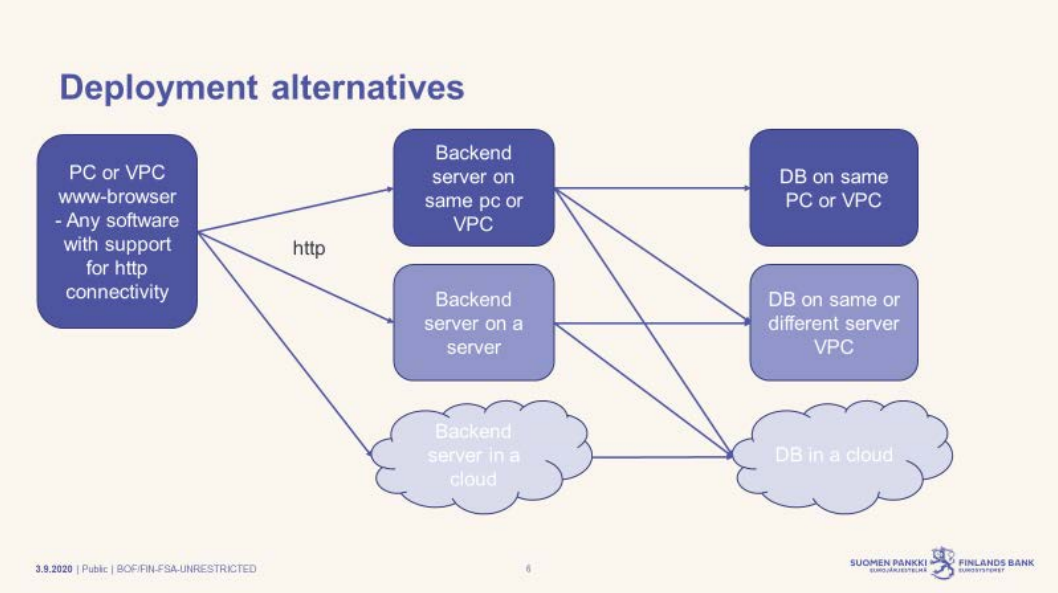
If you try to run too many simulations in parallel you might encounter crashes. In these cases it is simply recommended to try with less simulations at a time. As at 10.12.2020 we have successful experience on running a 10 day simulation with 1,5 mlj. transactions per day in 5 parallel threads in 10 CPU machine with 4-5 minutes average processing time per day. This is not a explicit limit but just one observation.

Needed Software
Windows 10 (newer DB software might not work with older windows versions)
Microsoft Excel installed (required to open reports from the user interface)
MariaDB/ MySQL /MS SQL-SERVER database server.
Sun Microsystem's Java Runtime Environment (JRE) 11 (distributed and installed with the **BoF-PSS3** program).
The **BoF-PSS3** program should work with limitations in Linux, although this is yet to be tested. Please contact the Bank of Finland if you are interested in running the software in a Linux environment.

2.2 Possible deployment setups

The architecture of the simulator allows several installation possibilities. Because BOF-PSS3 simulator engine has been implemented as a back end server application with an http API, the simulator engine can be installed locally on a PC or on a remote server. As at 4.12.2020 we still have little experience on running the

simulator on a server, we recommend to deploy it on a PC or on a virtual PC (VPC). The picture bellow illustrates the different configuration possibilities.



2.3 Installing a database server

The **BoF-PSS3** program assumes that Microsoft Excel and MariaDB, MySQL, or MS SQL Server are installed before the installation of the Simulator. Currently the main database version used for development and by the simulator team is **MariaDB 10.5 (**as at 4.12.2020**)**.

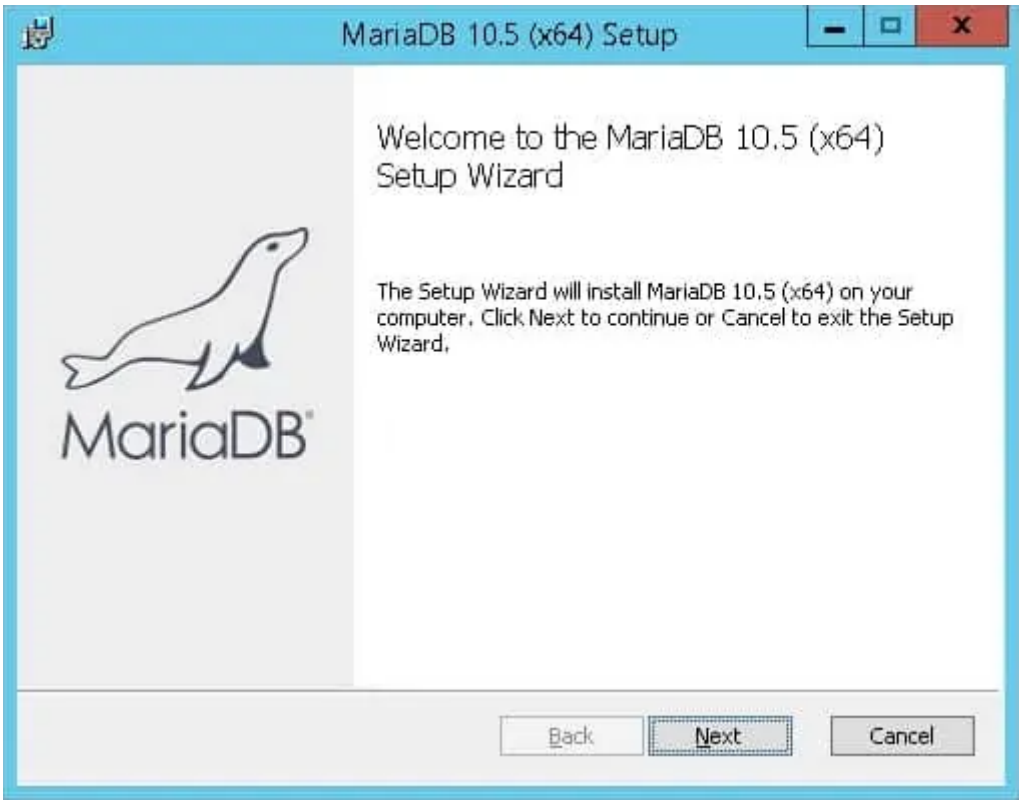
2.3.1 Installing MariaDB

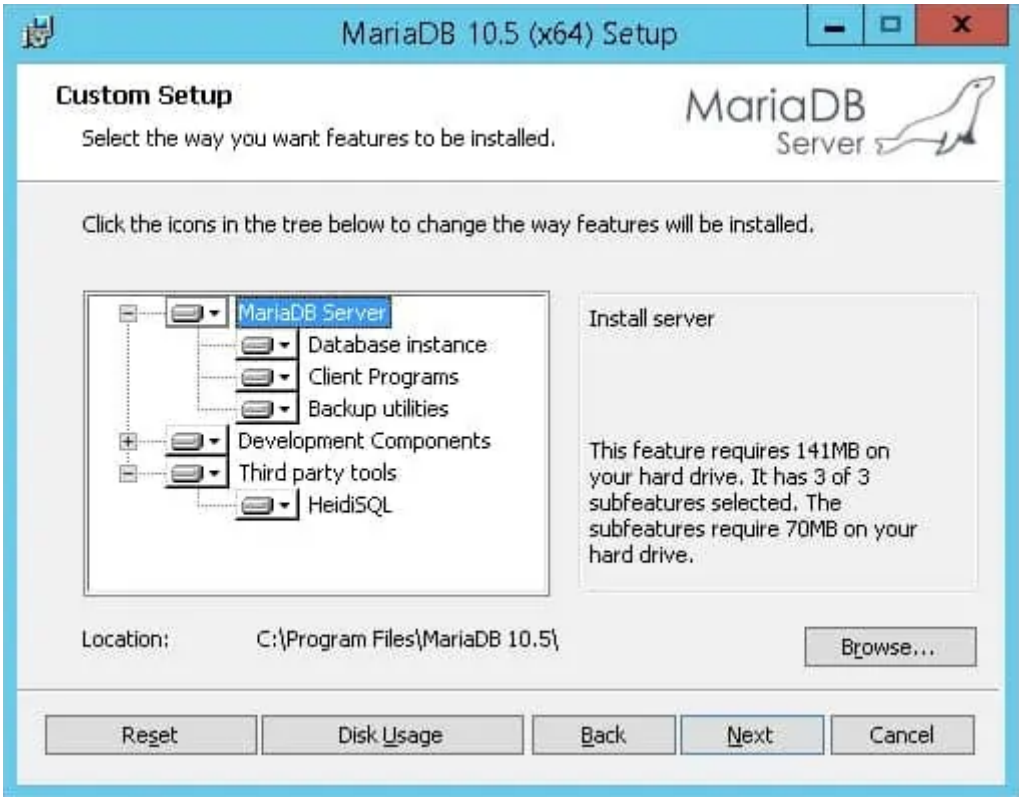
MariaDB database server offers a drop-in replacement functionality for MySQL. It is built by some of the original authors of MySQL together with assistance of free and open source software developers. Note that all DB versions do not allow to save database files to other directories than the data directory of the DB engine. We know that 5.2 and 5.3 do allow saving database table files freely, but for instance version MariaDB 10 does not. This is not an issue for the simulator as the simulator has always allowed to store databases to arbitrary locations. The simulator has an automated database storing policy recognition system which will guide project creation. MariaDB 10 is better performing than older versions in terms of speed.

Installation steps of the MariaDB 10 database server:

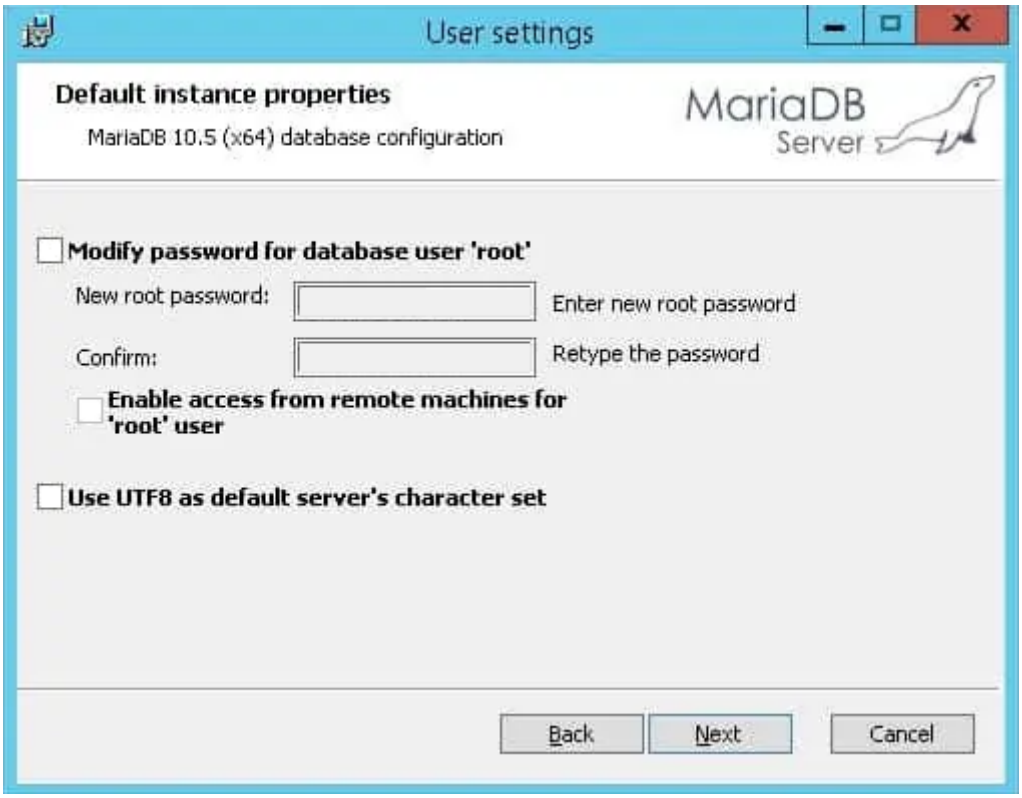
1. Download the latest MariaDB 10 installation utility corresponding to your operation system.

2. Double click the installation file to start the installation and click **Next** in the setup wizard window.

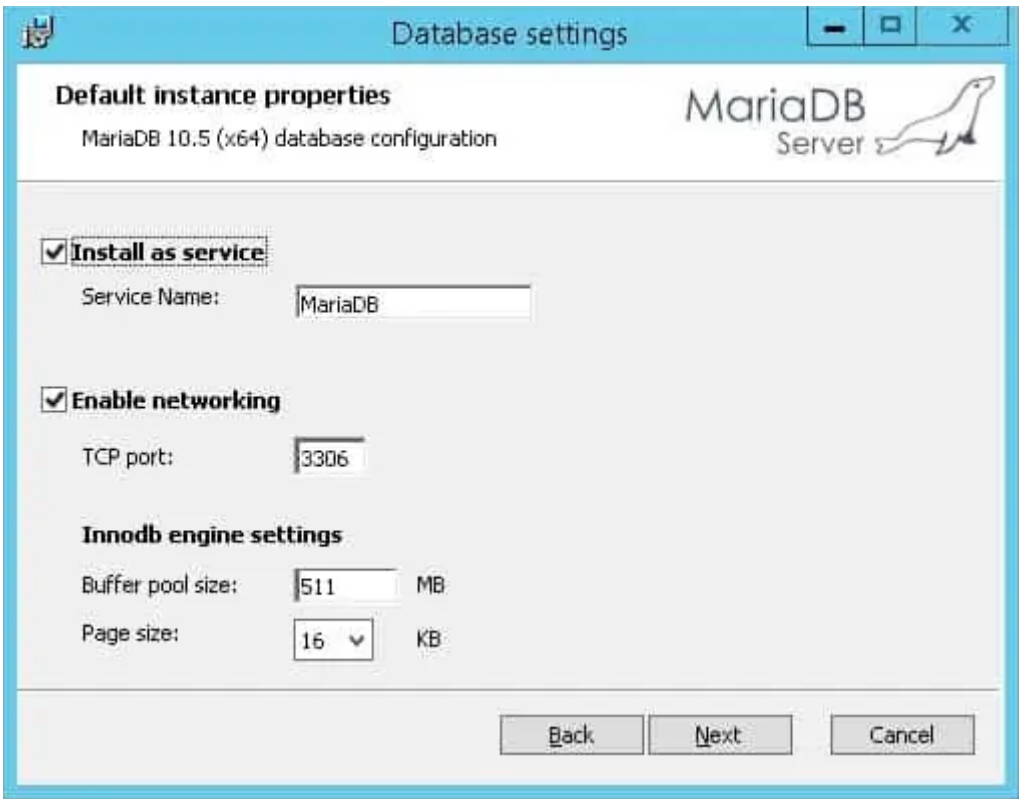




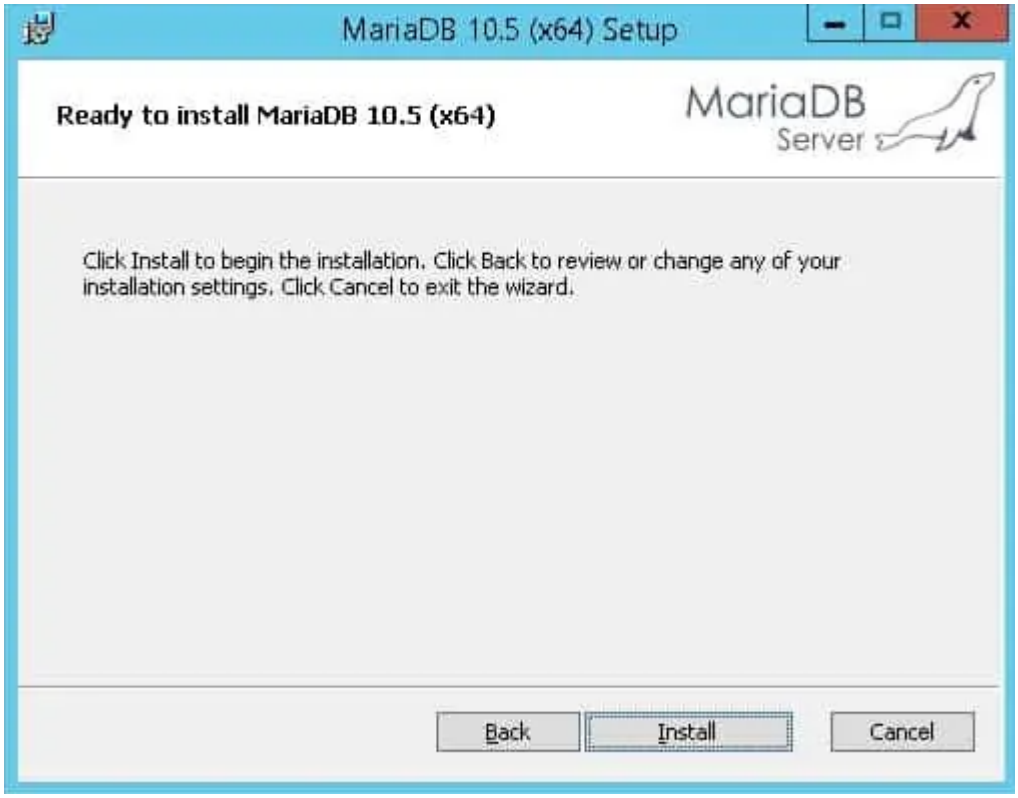
3. Accept all the features and the installation location folder, the default selections are recommended.



4. Please make sure you **unselect** the "**Modify password for database user 'root'**", since Simulator uses the root user's access rights without password set. Also **unselect** the "**Use UTF8 as default server's character set**" since that consumes approximately 10 percent more disk space compared to the default "Latin character set".



5. Please accept “Install as service” with MariaDB as the service name and ”Enable networking” on TCP port 3306 are selected and “Innodb engine settings” with defaults settings.



6. Click **Install** when Ready to install MariaDB.

Please note that you need to have full access rights on the data folder under the MariaDB installation folder. After installing with defaults the data folder path would be C:\Program Files\MariaDB 10.5\data

2.4 Installing the simulator

The Simulator application is delivered as a Zip file with naming convention:

```
BoF-PSS-[domain]-[version label as YYYYMMDD].zip
```

E.g. `BoF-PSS-gen-20201127.zip`

1. Uninstall old PSS2 Simulator `(recommended)`
   1.1 Take a backup of your old simulator projects if needed.
   1.2. Remove the existing \BoF-PSS2 folder structure as obsolete.
   1.3 Drop the `pss2_systemdb` and the old project database created by PSS2. The simulator will create a new upon launch. It might be possible to keep the old `pss2-systemdb` but you might encounter some compatibility issues depending on versions. If the pss2_systemdb is kept it wil continue to point to old projects. Old PSS2 project databases are not supported by PSS3. If you drop the `pss2_systemdb`, and leave old PSS2 project databases, you might also face issues if you create a project with PSS3 with the same name as the old project. To recap the best is you drop the `pss2_systemdb` and the project databases for clan start.

2. Copy the zip file to your C: root folder

3. Extract the zip file using "Extract to here"
   After extracting the zip file the Simulator folder should contain:
   - Simulator application (war file)
   - startup scripts
   - Simulator examples
   - Java virtual machine binaries
   - under the system's C: drive with a folder structure looking like:

```
C:\BoF-PSS
C:\BoF-PSS\EXAMPLES
C:\BoF-PSS\PROGRAM
C:\BoF-PSS\PROGRAM\JDK
```

2.5 Starting the BoF-PSS3 simulator

With your Windows file manager navigate to folder `C:\BoF-PSS` and double-click the file `startMin.cmd` that starts the Simulator application server in minimized command Prompt window. You can verify the Simulator started OK by clicking the black command prompt in Window's Task bar which opens the minimized Simulator command prompt window (the Simulator's primary log window). When the simulator is running OK you should see the following lines as last rows:

```
  .   ____ _                 _      _____
 / \ / ___(_)_ _  _   _| | __|_   _|__  _ _
( ( )\___ | | ' ' \| | | | |/  \ | | /  \| '_|
 \ /  ___)| |      | |_| | | () || || () | |
  '  |____|_|_/|_|_|\___/|_|\_/_||_| \__/|_|
 ==============================================
 SimulatorWeb Version 2020-12-02 gen - Bank of Finland © 2020
 Java version: 11
 Database: MariaDB.10.5 (NOTE! value may vary depending on driver in
  use.)
 Datasource URL: jdbc:mysql://localhost:3306/pss2_systemdb
 Driver: MariaDB Connector/J, version: 2.6.0
```

Open Chrome browser and add the following URL to address bar: `http://localhost:8080/` and press enter. The Simulator main view opens with open Project panel where you can create your first analysis project.

2.6 Run time start-up parameters

The simulator itself is a Java application and it is run in a Java virtual machine. The parameters of Java are defined in the start-up script of simulator in file `C:\BoF-PSS\PROGRAM\startServer.bat`. This file contains a code line starting the Java virtual machine with default memory usage settings.

In case of memory problems the memory parameters can be extended like –Xms***m and –Xmx***m where –Xms sets the amount of memory given for the simulator applications directly at start-up and –Xmx sets the maximum amount of memory that can be given for the application. The asterisks *** represent the amount of memory in megabytes.

Below is an example for reserving memory for Java environment:

```
set jvmParams=-Xms512m -Xmx1g
```

3 Operating the BoF-PSS3 simulator

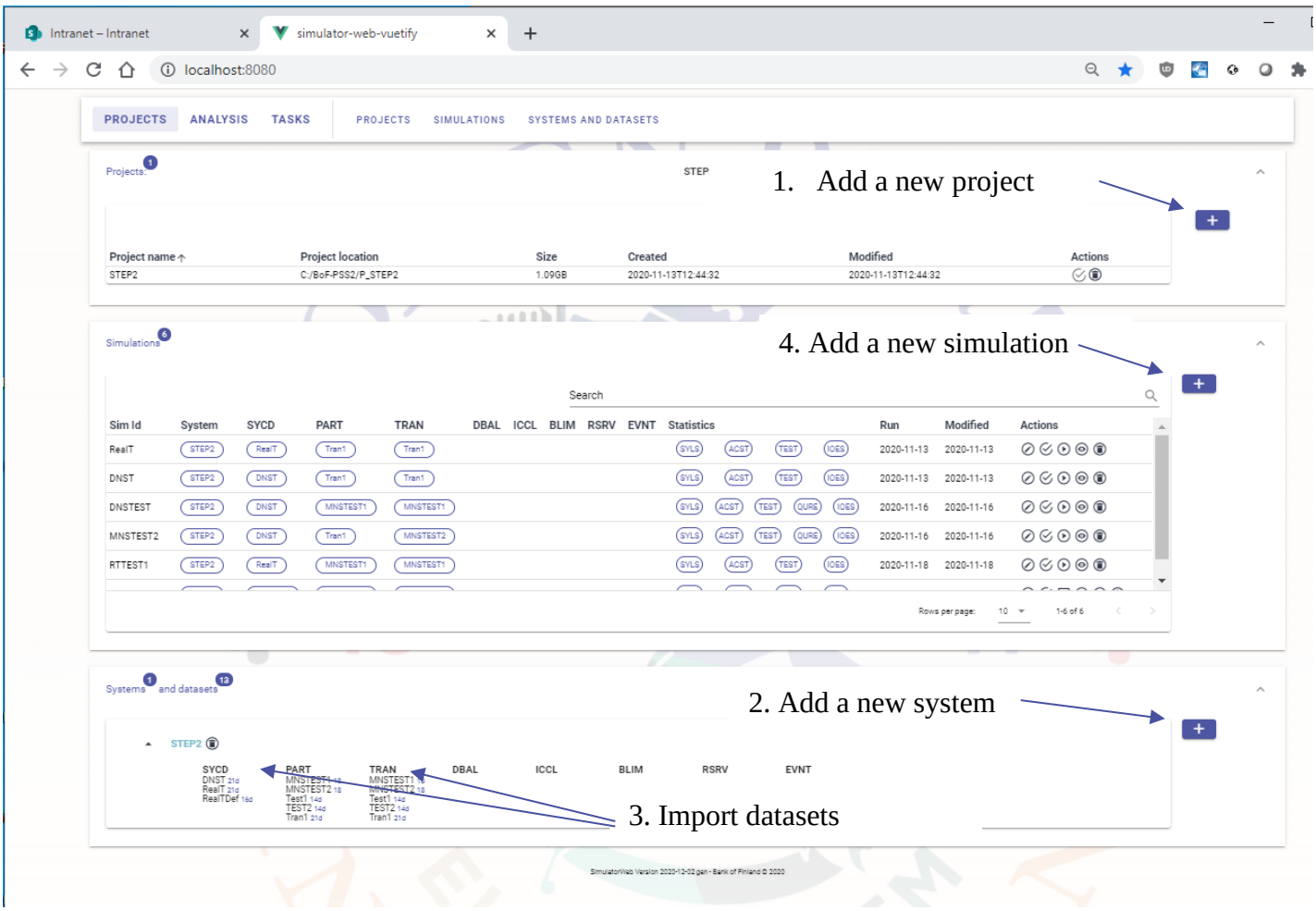
This chapter describes the basic features of **BoF-PSS3**

3.1 Short description of BoF-PSS3 simulator use

Basically the simulator consists of 3 types of functionalities. First it contains features for managing projects, importing datasets, defining systems setups and combining these to simulations. To make the use easier some tools like the automated stress tester helps users to run stress tests automatedly. Under tasks you can find predefined task lists which will allow users to execute consecutive tasks in one go.

A basic simulation process is normally divided into the following distinct phases.

- Creating a project
- Defining the payment and settlement systems
  You begin by specifying the systems you want to simulate. This includes stating the system name, setting the open hours, and selecting the processing logics and algorithms that are used in a system, see 3.3.
- Importing input data
  Next, you import input data for the system just specified into the input database (participant names and transactions and optionally daily opening balances, intraday credit limits and bilateral balances), see 3.4.
- Defining simulations
  Once you have specified the system structure and input data, you configure simulations and cross-check data sets belonging to the simulations.
- Executing simulations
  Simulations are executed by pressing the button at the end of each simulation row. You can run one or more simulations at a time in parallel. Running too many simulations at the same time might crash the simulator due to resource constraints.
- Analysing results
  After running the simulations, you can download ouput tables, browse outputtables with SQL, view some basic graphs. The outputdata can be further analysed and processed outside the simulator.



3.2 Working with projects

Each project has its own directory that carries the project name. Note that the project folder is used by the back end server. This means that in a setup where the back end runs on a separate server, users might not have direct access to these folders. Under this directory following sub-directories are created:

– Default directory where input files are located,
– Default directory where error lists are saved,
– Default directory where output files are saved, and
– Default directory where output reports are saved.

The simulator database files are stored under the `data` folder of the database engine's installation folder.

3.2.1 Project duplicates and backups

All data which is defined or created in a simulation project is stored in project's database. This makes it possible to easily backup and restore or duplicate projects. By copying the involved databases or parts of it.

Similarly backups can be made of simulation projects. The same procedure can also be used for transferring only some parts of the projects such as input database.

NOTE! The simulator application and the databases it uses (projects) must be compatible.

The output database can be updated to a newer version by simply deleting it. When a simulation is run, and the output DB is missing, the system recreates the output DB according to the used version. The same occurs when the system DB is missing.

3.3 Setting up a payment and settlement system

The system data set is referred to with the acronym SYCD. The name is also used in the database. You can add a new system dataset by hoovering over the SYCD title and pressing the add button that will appear.

Opening and closing hours must be between 00:00 and 24:00. The simulator supports business days that can take place on 2 calendar days or start before the weekend. Also default opening hours can be overridden by using an event data set that defines exact opening and closing times and dates for each business day. The use of event files is recommended in cases where business days extend over calendar days.

If an event dataset is not available, the simulator performs Business day deduction from transaction data. This means that the transaction data defines the business days present in a simulation. If transactions of one business day are introduced on 2 calendar days the simulator will deduct that the business day occurs on 2 calendar days and thus the opening and closing hours will be deducted to belong to different days. A business day cannot take place on more than 2 days. Weekends and holidays are supported. If a business day occurs on several calendar days, it is recommended to use event data sets to explicitely define the starts and ends of business days. Especially in cases that transactions do occur only on one calendar day but some other processes can occur on other days for example.

System type (RTGS, CNS or DNS) is mandatory and affects the behaviour of the simulator.

Transfer balances to next day can be used in multi-day simulations for transferring the end-of-day balances to become the beginning-of-day balances for the next day.

Intraday credit availability requires a choice between three options. The selection '**Credits according to limit table'** requires an ICCL dataset containing the intraday credit limits to be defined. '**No credits available'** indicates that only the liquidity on accounts is available. This means that only a DBAL data set is needed. The last option **'credit available without limits'** indicates that overdrafts are freely available. This option can be used to find out the upper bound of liquidity. Note that liquidity has to be provided in some form, otherwise no transactions will settle. **Handling of unsettled transactions** has four options. All unsettled transactions will be kept in a special queue for unsettled transactions until the end-of-day and the processing will be dependent on the selected option. *Transfer unsettled transactions to next day/settlement occasion* will place unsettled transactions back in the transaction queues to be settled later if possible. *Delete unsettled transactions (include in statistics)* will remove the transactions from queue but still include them in output statistics and reports. *Delete unsettled transactions (exclude from statistics)* means that the unsettled transactions will be removed from queue and also from all transaction level statistics and most system and account level statistics. They will only be included in aggregate transaction value and transaction count numbers in system and account level statistics. *Force end-of-day settlement* will result in bookings on the accounts irrespective of any credit limit violations. This can lead to negative account balances at end of day. Forced end-of-day settlement can be used to find out the minimum liquidity needed to settle all transactions at least at the ends of the day. An account violation record (AVST) will be written for every violating transaction.

Selected algorithms define the processing methods. Entry ENT and END end-of-day algorithms are mandatory for all systems. Algorithms are selected by dragging them to the right. Select an algorithm and fill in its parameter values, when required. See 5.1 Algorithms for details.

3.4 Importing data

You can import participant data, daily balances data, intraday credit limits data, transaction data, bilateral credit limit and daily event (EVNT) data. The input file has to be a text file, e.g. .txt or .csv.

The supported default values for the csv data separtor and decimal are ‘;’ and ‘.’. Time is displayed as 'hhmmss.SSSSSS' and dates are displayed as 'yyyymmdd'. The amount of rows to be skipped at the beginning is 2.

Long account names and ID’s are only stored to the PART table with a technical id which numbering starts from 1. This makes referencing of data significantly faster and less storage space consuming. But in order to allow coherent account id indexing, the importing must be done in a more disciplined way. All input data sets have to be associated with a participant dataset explicitely. Before importing any other data set, there must be participant dataset in the system. Or, a participant dataset must be generated from the imported data.

It is also possible to inport the data without PART data. In this case part data will be created by the import process. For this, it is recommendable to create it first with the TRAN data.

For more details on the technical referencing please refer to the chapter 6 on database table descritpions.

The different input data types/data tables are coded as follows:

- **PART** contains participant and account data. This can be defined on participant level only or alternatively on combined participant and account level. In the latter case, the same participant may have multiple accounts, but for each both the participant and account ID should be specified. This feature can be used to define different omnibus accounts for clearing parties in a securities settlement system.
- **DBAL** contains the initial daily balances data of participants or accounts. It is optional. Null values are considered as zero.
- **ICCL** contains intraday credit limit changes of participants. It is also optional. Null values are considered as zeros.
- **TRAN** contains the transactions of a given system. There can also be transactions pointing to other systems. This is done by defining the ‘to-system’ field for transactions. The ‘from-system’ field must always contain the same ID, which is defined as the system ID of the dataset.
- **BLIM** contains the bilateral limits between pairs of participants. It is optional.
- **RSRV** contains information on reservations. Reservations are used to reserve a specific amount of the available liquidity to be used to settle some specific type of transactions. Support for reservations is algorithm specific and for the moment there are no built in algorithms in the generally available version of BoF-PSS3 which support the use of reservations. Reservations data can be used

in own user modules. For the availability RSRV supporting algorithms you should check with the simulator team. There can be many different reservations defined for one account.

– **SYCD** contains system control data. These data must be specified for each system. This specification is done in the System control data specification – screen, not by importing a dataset.
– **EVNT** refers to data sets that contain timing of events such as start and end of days. Can be used to introduce some new tailored eventhandler specific events.

A **system ID** has to be defined for each imported data table. It is used when searching and configuring data that belongs to the same system. System ID is selected from a drop down list, which includes all system IDs that have been defined in the system definition window, see chapter 3.3.

Multiple data sets can be used for running the simulations with varying input data. This is facilitated by a **data set ID** specified for each data table. The input database will thus contain parallel data sets with the same information, e.g. different data sets for intraday credits to simulate a situation with varying liquidity. There may also be different transaction flows depicting e.g. crisis situations. To manage a large number of parallel data sets effectively, it is important to create a consistent naming convention. The data set ID can be up to eight characters long.

It is important to note that the input systems only check the data content at the field level. Due to possibility of multiple parallel data sets, comprehensive cross-checking can only be performed after simulations are configured and parallel data sets selected.

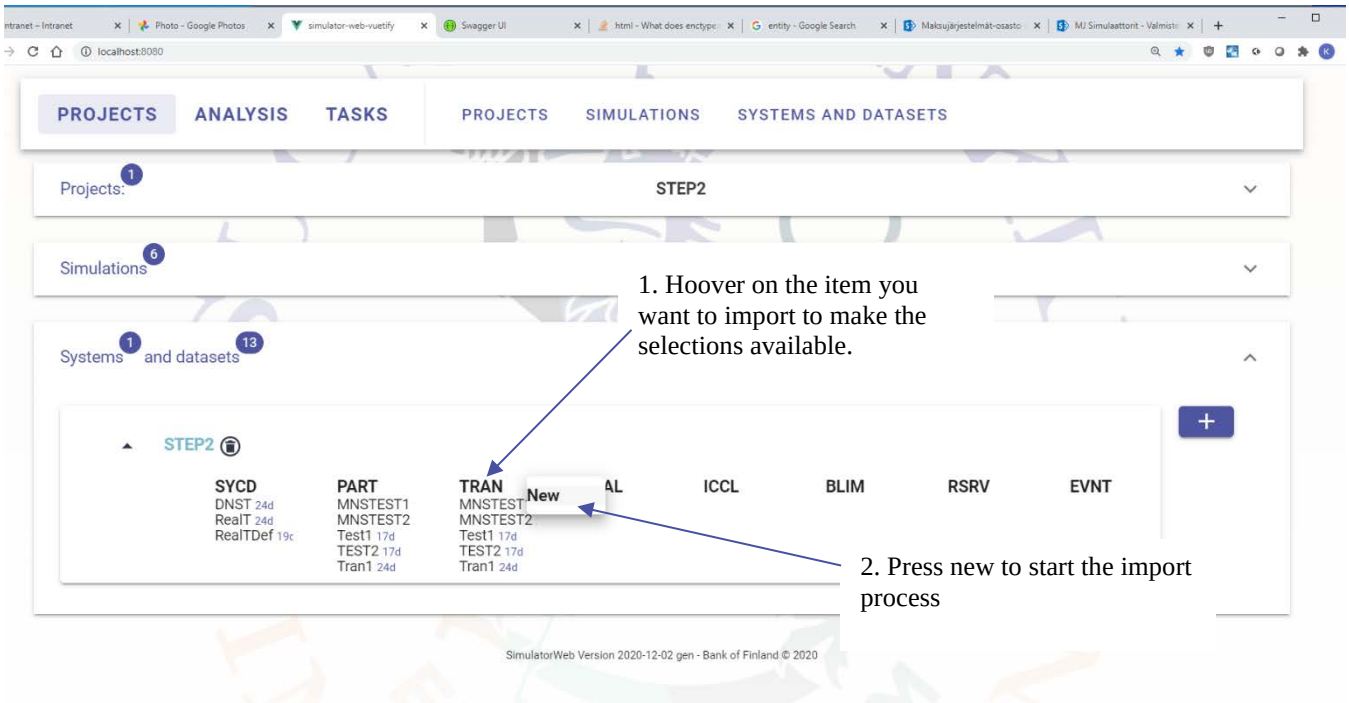
Templates are used for inputting data using CSV files. The templates describe the data field order in the CSV files.

When you create input data in a CSV file, consider the following:

– Make sure that the data and decimal delimiters are specified correctly.
– Values of currency can only be stated to two places after the decimal point.
– All data rows in the CSV-file should have the same number of data fields and the input template defines how these correspond to the input data base of the simulator.
– Transaction ID in TRAN tables can be numeric or alphabetical, they are sorted alphabetically. The transaction ID must be unique as it is a sorting parameter to distinguish between transactions that otherwise would occur in the same order. It is also used as a key when reporting input errors. If you use numeric values,

use a sufficiently large first number (e.g. 10001) for transaction files involving ten thousand transactions to assure successful alphabetic sorting.

– When the simulation contains more than one system and interlinked transactions the TRAN data of a given system must hold all debit transactions (FROM-transactions) of that system. The simulator operates on credit transfer basis so intersystem transactions can only be made as credits to another system (i.e. all direct debit type of transactions in real systems must be converted to credit transfers in the simulator.)
– When DVP/PVP transactions are introduced, a link code is needed to define the linked transactions. A system can have both linked and unlinked transactions. More than 2 trades or payments can be linked together.



3.4.1 Errors in import

The simulator is able to report some errors occurring in the error process or the data. The notification window will shoe an erro status in red if errors are found. Detailed error reports can be found from the tasks window's task run component.

The error list file is located in the ERRORLIST directory of the project. It is named like `ImportInputError_[YYYYMMDD]_[HHMMSS].csv`,
for example, `ImportInputError_090407_121030.csv`. Unnecessary error lists should be deleted.

In the error list the **row** refers to the row number in the input file and the **col** to the column number in the input file. The most common errors found are format errors. For date and time fields the formats used in the imported file should be the same as specified in the data format defaults. Numeric fields should be completely numeric and the decimal sign should be the same as stated in the import screen. The simulator does not support 'thousand' signs. The error 'duplicate entry' indicates that there are duplicated key information in the input file e.g. two rows with transaction data with coinciding transaction IDs. In case of coinciding keys the first data row is imported and the next ones are discarded with an error message.

Different kind of format errors and delimiter changes can arise when the input CSV file has been exported from another program, like Excel. It is advisable to check the content of exported CSV files with a software showing the true content of the CSV file e.g. Notepad. Check delimiters, date and time formats and decimal information, because there seems often to be small differences in these details if the simulator and the CSV data exporting software have not been synchronized earlier.

3.4.2 Cross-checking data sets

Cross-checking verifies that the information in data sets selected for a simulation are coherent, e.g. all accounts needed are available for booking transactions, intraday-credit changes are within transaction dates and opening hours and initial balances are within transaction days. It is good practice to run a cross-check after a simulation is defined.

If the cross-checking finds errors, an error report is created. The error list file is located in the ERRORLIST directory of the project. It is named `SimulationConfigurationError_[YYYYMMDD]_[HHMMSS].csv` for example: `SimulationConfigurationError_090407_121030.csv`.

The error list contains a reference to the incoherent input data record and information about the missing relationship data. The dataset ID and a short description will be shown for each error.
Typical cross-check errors are

- Date errors: the dates in DBAL, BLIM or ICCL tables are outside the simulation dates, i.e. dates for which transactions are specified in the TRAN table. Limits and balances should only be defined for those days which are simulated.
- Time errors: the transaction introduction times specified in the TRAN table are outside the opening hours of the simulated system, as defined in system data or selected EVNT dataset. This might cause confusion to users in cases

transactions are introduced before the opening of the business day. In this case simulations can still be run and the users should disregard this kind of errors. Also in these cases simulations should be run without cross-checks as these cross-checks will stop the simulation execution.

- Participant errors: there are missing participants (or typing errors) in the participant data. TRAN-PART refers to transactions with missing participants in PART data, DBAL-PART refers to missing participant data for DBAL data and ICCL-PART refers to missing participant data for ICCL data. For all participants quoted in TRAN, ICCL, BLIM and DBAL records there must be the corresponding participant in the PART data. If multiple systems or accounts are defined, the participant is checked as a combination of the system ID, participant ID and account ID.
- Erroneous system ID's in TRAN data: if the *from system id* field is explicitly defined, it should be the same as the ID of the system which this data set is attached to.

3.4.3 Creating multi system simulations

Simulations with multiple interacting systems can be created and simulated with BoF-PSS. This enables simulation and analysis of parallel systems with independent processing logics, such as network of several RTGS systems such as TARGET, combination of a RTGS system and an ancillary CNS or DNS system or a RTGS payment system together with a securities settlement system working with DVP processing.

In multi system simulations individual systems are set up independently one by one: system definition and data imports are performed for each simulated system separately. System ID field is used to collect together definitions and input data of individual systems.

For transactions between systems the receiving system name has to be defined in input data. Transactions are always included in transaction data set of that system, where the from-participant of each transaction is located. The system names used in input data (e.g. From-system and To-system in transactions) need to be the same which are used as System IDs in system definition.

The **transaction IDs must be unique simulation-wide**, i.e. the transaction data sets of the different systems cannot use same transaction IDs. Cross-check will display reused IDs as errors of the second transaction data set that uses them.
An example of multi system simulation is provided in example2 material included with the simulator (C:\BoF-PSS\EXAMPLES). It presents a main RTGS system

and an ancillary CNS using the accounts in the RTGS system as the source of liquidity. In the input data the “liquidity injections from system” and “liquidity injections from participant”-fields are defined for each participant. In the input data of this example the RTGS system is referred as “M” and the ancillary system is referred with “K”. These names have to be used also as System IDs in the simulation. In multi system simulations, the cross-check is checking also coherency of multi system transactions and used System IDs.

Multi system simulations are created in **simulation configuration** screen by including all necessary systems one by one on their own rows. Screenshot from example 2 simulation configuration is shown below.

3.4.4 ABM Simulations

Users can instruct the simulator to associate active account management to accounts. This is done by selecting a ABM configuration file on the screen.
See the new field called “Agent file”.

The property file can by edited and copies can be made. The ABM property files are fetched by default from the BOF-PSS\program\ABM directory. The default example property file can also be found from BOF-PSS\program\ABM directory The Browse function allows to store the files also into other places.

The property file contains the information on accounts that have been allocated an account management algorithm with the specific parameters. There can be many account management algorithms and users can meake new one’s based on the ones distributed with the simulator. For more specific description on the ABM algorithms and parameters, please refer to the section 5.5 .

When a simulation is executed as an ABM simulation, additional ABM log files are created besides the basic simulation log. The ABM logs are created by account.

3.5 Executing simulations

You can execute simulations on the Simulation execution screen. The screen opens by clicking the **Simulation execution** button on the Main menu. Simulations can be executed as single runs or in batches consisting of many simulations.

3.5.1 Errors in simulations

On the Simulation execution screen:

View errors arising in the simulations by clicking the **View error report** button. The error list file is located in the error list directory of the project. It is named `SimulationExecutionError_[YYYYMMDD]_[HHMMSS].txt` for example, `SimulationExecutionError_090407_121030.csv`. Unnecessary error lists should be deleted. All errors or relevant simulation related information might not be present in the error file, thus one should always also check the simulation log file.

3.6 Analysing results

Simulation results are stored in the output tables of the project database. The results can be viewed using available reporting features and by accessing the database tables directly with SQL or other tools. All data can also be downloaded as CSVs. The automated stress testing module 3.7 provides some reporting features of its own.

3.7 Automated stress testing module

The Automated stress testing module has been developed to make the running and analysis of the data and stress tests results easier. The version released with 600 has to be considered as a first version and it has been heavily improved since. The idea is to widen the support for different scenario types.

The version version 800 still supports one analysis type: stress testing based on removal and transformation of transactions.. Since version 703 it has been possible to affect the way accounts are treated for scenario creations. Accounts can be affected either individually for each scenario or all selected accounts of a participant can be affected at the same time. As a third option, all selected accounts can be affected in one simulation(scenario).

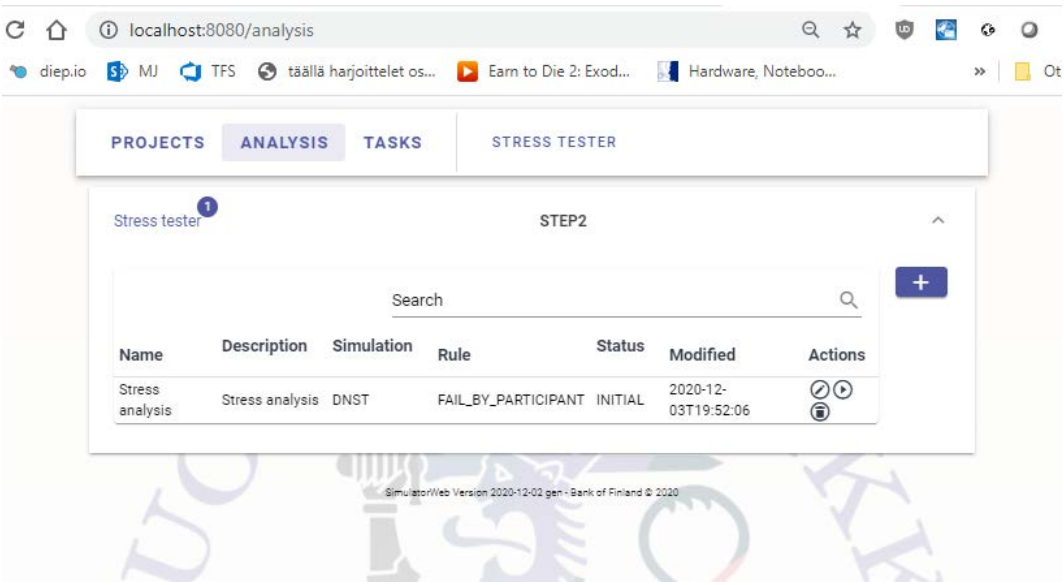
The interface allows to alter the query conditions used to retrieve the data for each scenario. This allows the user to define for example, that only payments not belonging to a certain category or occurring between a specific time period are dropped.

In short, the module works in the following way. First you need to define a benchmark simulation with the data and the system setups you wish to analyse and

put under stress. After that, you can open the stress testing analysis tool and create a new analysis for which you will define the benchmark scenario to be used as a starting point. The user will have to select the accounts to be included in the analysis. Stress scenarios will be run for all the selected accounts. After all the scenarios have been run the user can generate a report tailored for comparative stress testing analysis. The report contains account level information such as upperbounds, maximum upperbounds, liquidity deterioration figures, values of unsettled sent and received transactions and many others. For further details see following chapters.

3.7.1 Creating a new analysis

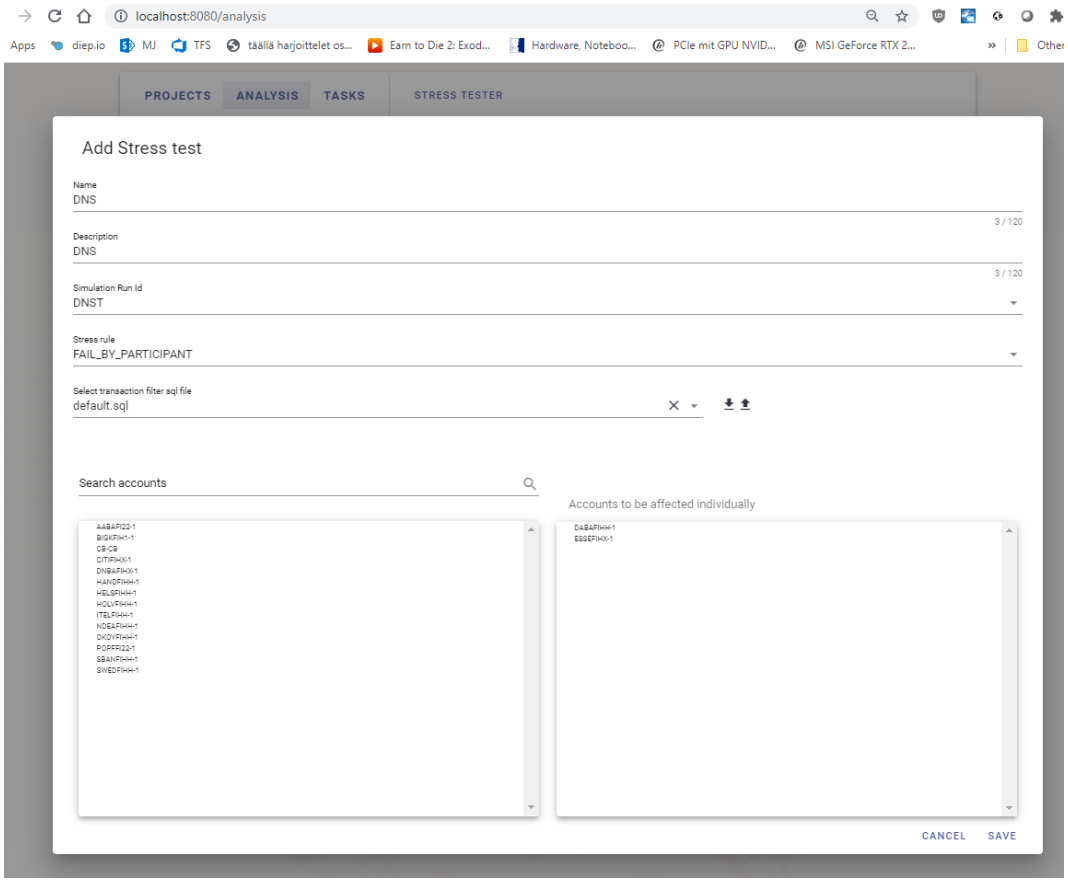
To create a new stress analysis or access an existing one you need to open the stress testing analysis window from the main menu. The Stress testing view will open.



From this view it is possible to view old analysis made with the selected project and create new one's by pressing the + button selecting a benchmark simulation, giving it a name and saving it,.

3.7.2 Selecting the accounts to be affected in a scenario

To select the accounts to be affected in the scenarios, the user needs to open the "setup scenarios and accounts to be failed" view. On the left all the non selected accounts are visible. Accounts can be selected by dragging accounts to the right.



The “Scenario rule” selection affects how the scenarios are generated. “by accounts” means that scenarios will be generated for each account and the scenario definition behind #scenario_id# tag used in the SQL Data filters, will contain only one account id at a time. “by participants” means that scenarios will be generated by participants and the scenario definition behind #scenario_id# tag will contain all the selected accounts of one participant at a time. This allows to have all the accounts of one participant to be affected in the same scenario. “All selected” means that only one scenario will be created and the # scenario_id # tag will refer to all the selected accounts. For more details see below.

The selections are saved only when the save button is pressed. Be aware that saving will discard any priorly run results for this analysis.

3.7.3 SQL-query filters for scenario creation

The Data filter selection allows to select a specific SQL-query to be used in the generation of the scenarios. The queries are used to get the set of transactions to be

used in each scenario. The queries contain tags that are replaced by scenario specific values when the stress tester runs scenarios.

The SQL-queries are stored in text files in folder `C:\BoF-PSS\PROGRAM\filters\stressTester`. The defaultFilter is a query for generating failure scenarios by removing transactions belonging to selected accounts. Users can use the default filter to create customized queries to direct the scenario creation. Users cannot modify the defaultFilter, but can create copies of it.

The following tags are supported since version 7.0.3:

scenario_id #: substitutes the analysis id to the sql query allowing to link to the list of account ids involved with the scenario.
#system_id#: Derived from benchmark definition
#dataset_id#: Derived from benchmark definition
#business_day#: Stress test are run day by day separately.

3.7.4 Running of the analysis

To obtain the analysis report, one needs to run the scenarios first. When pressing the Run scenarios button the system will:

1. run the benchmark if it has not been run yet
2. run the benchmark with illimited intraday credits to calculate figures for the upperbound of liquidity
3. it will generate and run the one day scenarios for each selected account. In order to save space the input data for scenarios is not stored to the input database's tran table. Simulation results are stored only to the tables SYLS, ACST and TEST of the output database.

3.7.5 Working with the results

After you have run the scenarios you can obtain the results by pressing the Run report button. The analyser will collect the account level results for all the scenarios and the benchmark into an excel. The data content is sufficient to obtain basic charts, reports and comparisons related to the benchmark and run scenarios. The report contains all the sufficient data amongst other to:

- obtain rankings for systemical importance of counterparts in terms of casued unsettled payments, intermediated stress
- draw counterparty exposure matrixes in terms of unsettled payments, liquidity deterioration
- observe average sending and receiving times and the differences to the benchmark case

The report will contain account level information rows for each day, scenario and benchmark. The data is partly extracted from the ACST and calculated from the TEST table. The report contains two automatically updated charts and the counterparty risk matrixes.

The analysis report contains the following data fields:

BenchScenario	This field can have 2 values and it indicates whether the row information relates to the benchmark simulation or a scenario. Possible values: Bench, Scenario
SimRunId	Name and ID of the simulation. For scenarios the name is generated by concatenating the name given to the analysis and the accountID
SystemId	System ID of the Benchmark
failingAccountId	Account ID of the failig account
AccountId	Account ID to which the figures belong
BusinessDay	Business day of the figures
BoDBalance	Beginning of day balance for the account as in ACST-table
EoDBalance	End of day balance for the account as in ACST-table
MinBalance	Minimum balance during the day for the account as in ACST-table
EoDCreditLimit	Intraday credit limit at the end of the day for the account as in ACST-table
CreditLimitMaxUsage	If the account has had a negative balance during the day (e.g. has relied on credit limit), the value here is: minimum account balance *-1 / Eod credit limit * 100
SettledCount	Count of the sent and settled transactions
SettledValue	Value of the sent and settled transactions
SentUnstCountDirect	Count of the transactions removed due to the scenario
SentUnstValueDirect	Value of the transactions removed due to the scenario
SentUnstCCPValueDirect	This column is present only if the simulator recognizes, the data to bemorphologically complient with trade data. The field is calculated form usercod_4. The field will contain meaningfull results only if all values entered into usercod4 field are in same currency. The values in the value field used for the other calculations can contain amounts and values in different currencies making them uncomparable.
SentUnstSystemicEffectCount	Count of the unsettled transactions in the simulation or scenario. In scenarios these

	would correspond to the systemic or second round effects due to the altered situation. When calculated for the benchmark, the count is just the count of unsettled transactions. When the value is calculated for a scenario, the benchmark's corresponding value is substracted from the count.
SentUnstSystemicEffectValue	Value of the unsettled transactions in the simulation or scenario. In scenarios these would correspond to the systemic or second round effects due to the altered situation. When calculated for the benchmark, the value is just the sum of unsettled transactions. When the value is calculated for a scenario, the benchmark's corresponding value is substracted from the sum.
ReceivedPaymentsCount	Count of payments received as in the ACST table
ReceivedPaymentsValue	Value of payments received as in the ACST table
ReceivedUnstCountDirect	Count of payments not received because the transactions removed due to the scenario. Calculation is based on TEST data.
ReceivedUnstValueDirect	Value of payments not received because the transactions removed due to the scenario. Calculation is based on TEST data.
ReceivedUnstCCPValueDirect	This column is present only if the simulator recognizes, the data to bemorphologically complient with trade data. The field is calculated form usercod_4. The field will contain meaningfull results only if all values entered into usercod4 field are in same currency. The values in the value field used for the other calculations can contain amounts and values in different currencies making them uncomparable.
ReceivedUnstSystemicEffectCount	Count of payments not received in the simulation. In scenarios these would correspond to the systemic or second round effects due to the altered situation.
ReceivedUnstSystemicEffectValue	Value of payments not received in the simulation. In scenarios these would correspond to the systemic or second round effects due to the altered situation.
ReceivedPaymentsDiffValue	Value of received payments in the benchmark – value of received payments. Positive value indicates a decrease in original value.

LB	Lower bound of liquidity. Net liquidity needed for the day.
LBDiff	LB of benchmark – LB of scenario
UB	Upper bound of liquidity. The amount of initial liquidity needed to settle all payments introduced in the order of the input data. Currently this is only calculated for the benchmark. Calculation occurs by rerunning the benchmark scenario with the selection intraday credits available without limits. UB = beginning of day balance – minimum balance of the simulation with unlimited credits.
MaxUpperBound	Sum of values of all outgoing payments. Total gross outflow.
MinLiquidityDeterioration	Needed extra liquidity to keep end of day(EOD) balance in the scenario, unchanged when other non-failing participants are able to compensate and still send their unsettled payments. MinLiquidityDeterioration = End of day balance in benchmark simulation - End of day balance in scenario + outgoing unsettled in scenario (outgoing systemic) - Unsettled in benchmark (can be used for modell accuracy correction) - incoming unsettled (systemic, not direct) in scenario (it is assumed that others are able to send even if not in the simulation). It reflects the needed extra liquidity to settle all unsettled payments and achieve same level of liquidity as in benchmark. It is assumed that unsettled incoming payments are settled and the buffers of other participants are sufficient and they are able to bring in extra liquidity. If value is negative it is an improvement and it is rounded to 0.
MaxLiquidityDeterioration	Needed extra liquidity to keep end of day(EOD) balance in the scenario, unchanged when other participants are not able to compensate and cannot send all of their payments. Maximum Liquidity Deterioration = End of day balance in benchmark simulation - End of day balance in scenario + outgoing unsettled in scenario (systemic)

	- Unsettled in benchmark (systemic, can be used for model correction) It reflects the needed extra liquidity to settle all unsettled payments and achieve same level of end of day liquidity as in the benchmark. It is assumed that other participants are not able to bring in extra liquidity intra day. If value is negative it is an improvement and it is rounded to 0.
SettlementDelay	Same as a_setdelay from the ACST table.
SettlementDelayDiff	Settlement delay of the scenario – Settlement delay of the benchmark
WeightedAvgReceivingTime	Value weighted average of settlement time of received payments
WeightedAvgReceivingTimeDiff	WeightedAvgReceivingTime of the scenario- respective value of the benchmark
WeightedAvgSendingTime	Value weighted average of settlement time of sent payments
WeightedAvgSendingTimeDiff	WeightedAvgSendingTime of the scenario – respective value of the benchmark.

3.8 Task Automation Tool and Task Sets

The task automation tools functionality is meant to create task lists to allow easier repeatability of specific task sequences. Also new functionalities can be introduced as tasks. The initial versions are still hard coded. In the future releases, more tasks and flexibility are likely to be made available.The idea is that the task automation tool allows the user to define tasks to be performed in one go. The task set for importing and running a benchmark simulation can be found under the Task automation tools. First you need to select the task set called “Import tasks”. Once this is done all the tasks belonging to the task set will be displayed on the right.

In order to instruct the automation tool which tasks to include to the run, the user needs to check the corresponding boxes in the Run column. If the run box of the task is checked, the task will be run. The tasks are run in the top down order.

Many of the tasks need some additional parameter information to function. The parameters can be edited by double clicking the parameter field of a task.

The Import tasks task set has 3 common parameters used by the individual tasks. The system id indicates the system under which the data is imported. The value entered to the field “Data set name of imported datasets”, is used as a dataset name

for all imported datasets. The value is also used by other tasks to indicate the name of the datasets to which a task is targeted.

3.8.1 Import tasks

The import taks set provides the tasks to perform the steps to import and manipulate data and perform a normal day, 1 system (benchmark) simulation. Benchmark simulations are used as reference for scenario simulations like stress tests. Results of scenarios are compared to the results of benchmark simulations. Once set up, the benchmark simulation can be used as a base for stress test or other analysis.

The following Import tasks are available:
- Clear project's old input and output data
- import data
- Exclude transactions
- Account pool data
- Run cross-checks
- Run simulation

The tasks are explained in details here under.

There are 3 common parameters for the tasks. The system, the data set name and the account pooling data set name suffix.

The system selection is actually a combination of a system ID and a system data set.The system is used by the task Import data to indicate the system under which data is imported. The run simulation will use the system dataset information to create the simulation setup.

The dataset name, which default value is Raw, is used to name all the imported datasets. Also the accountpooler if used will name the pooled data sets as a combination of the data set name and account pooler suffix (Ap). This means, if account pooling is used, the final datasets used by the simulation will be named by default RawAp. If the account pooling task is not selected, and there are datasets that have a name that match the combination of the data set name and account pooling suffix, the user will be asked for which data sets the user wants to run the benchmark.

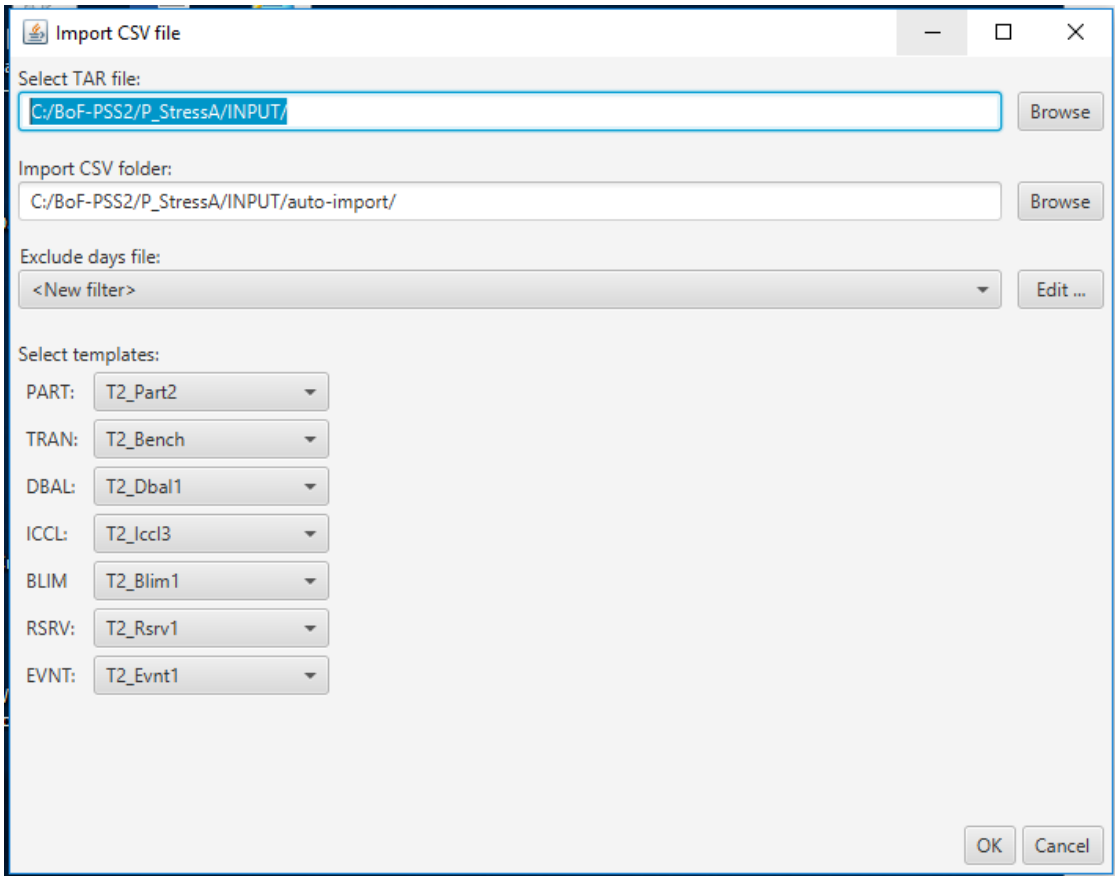
Clear project input and output data
This step is optional. If the project contains old input data and results of obsolete simulations, the databases can be cleared. This step removes the contents of all input and output databases including simulation definitions and stress tester analysis

setups and results. This task does not delete system setups nor templates. The clearing task is fast. Especially it is significantly faster with big datasets comparing to an overwrite operation in the Import data task when reusing an existing dataset name.

It is highly recommended to run this task if there are no reasons opposing it.

Import data task

The Import data task imports all the input files, either from a tar file that respects GZ.tar format or separate csv-files in an indicated directory. First the user needs to double click the parameter field of the import data task. A new window with all the necessary parameter fields will open.



Then the user needs to select either a tar file or indicate the directory from which to import the extracted and unzipped csv files.

The exclude days file allows the user to select a file containing days to be excluded from the import process. The user can edit and create new definition files. The excluded days files are stored in a default folder defined in a property file. The property file is called loadFile.porperties.

It is also necessary to select the correct import templates for the import process. The templates are the same that can be created and edited with the old interface: Import input file. Default templates are defined in the loadFile.properties file. The csv's to be imported either directly or form the tar-files need to follow the formats indicated by the templates. The names of the CSV files need to contain the words corresponding to the dataset type indicators: tran, part, iccl, dbl, blim, evnt, rsrv, or gacc.

When loading data and there already are data sets with the same name, the system prompts the user whether to overwrite the existing data or not. If there are no other input datasets in the database, the system will use the same truncate operations as the "clear project and output data" to remove the old datasets, which is very fast. If there are other datasets in the database with other dataset ids, a slower overwrite operation will take place. If the database is big this might be a very slow (very very slow) operation.

Exclude transactions

The remove transactions task consists simply of the execution of an SQL- sentence. The user needs to select an SQL-file containing an SQL sentence that alters the data somehow. An example file is provided. The file can contain also update queries to change data.

The default file should not be changed and saved with the same name. The file can be edited by users and saved with another name to the same folder.

To select the SQL-sentence to be executed by the task, the user needs to double click the parameter field of the task and select the appropriate file. The edit button opens the text file containing the sql-sentence and allows to create a new file with user made modifications.

This task applies the sql-sentence of the file, to the tran dataset imported before and indicated by the global parameters: system (as selected in the drop down on top of the GUI) and data set name (in the text box at the top as well).

The SQL queries are performed as a SQL prepared statements. This means that the "t_datsetid = ?" condition as such, in the current version, can be present in the sentence only once.

Account pool data (Merge accounts)

This task simply runs the account pooler. It uses the GACC-file present in the input data. Resulting datasets will be renamed by adding the account pooling data set suffix to the dataset name from the top of the GUI. To function this task requires the group of accounts file (GACC) to be present amongs the input files.

Perform cross-checks

This task performs the same cross-checks that can be run from the simulation configuration view.

It is not compulsory to run this task, but if there are doubts on the integrity of the data, it is good to run it. Once the data integrity can be trusted this step can be skipped to speed up the process.

The cross-check report contains results that are categorized as errors and warnings. Errors are to be taken seriously. Warnings are such that the simulator should be able to perform a simulation. Warnings can be errors but aren't necessarily so.

Run benchmark simulation

This task will create and run the benchmark simulation for the datasets named according to the name fields (dataset name and account pooling suffix) on top of the GUI. The system setup used to run the benchmark is the one selected on the top row of the GUI. The benchmark simulation is named "Bench". If the accountpooling task has not been selected but there are datasets with a name corresponfding to the pooled datasets according to the nameming parameters dataset name and accountpooling suffix, RawAp in the default case, the user is asked for which dataversion the benchmark is to be run.

3.8.2 CCP tasks

The version 7.0.3 contains one task set: CCP tasks. The task set has been developed to allow manipulating trade data of a certain form, to allow CSD settlement simulations of CCP's trades. In theory the tool could potentially be used to simulate commodities markets and maybe derivatives to some extent. These would have to be evaluated separately.

The task automation view checks whether there are eligible tran datasets in the tran table that fulfil the requirements of the CCP-tools. In practice this means that there needs to be a tran dataset with the following fields populated:

```
t_assename: ISIN code if available, could be something else too.
t_usercod1 : contains currency code
t_usercod2 : contains CSD id
t_usercod4: contains the money value
```

One CCP template for importing trade data, used in an internal project, is provided with the general version. Other templates could be used too depending on the files

to be imported. This template should be considered as an example, but could be used in other projects too.

The provided tasks are:

Add CCP transactions

This task requires a transaction data set, containing trades, with or without cash legs, to be present in the current project. This task will perform novation of the trades by setting the CCP as a counterparty to each trade. The created datasetid is named “CCP”.

Net CCP data

The task performs a basic CCP netting. The netting for securities is performed by CSD, participant, instrument and currency (CCY). Cash legs are netted according to CSD, participant and currency (CCY). A new transaction dataset is created for all CSDs. The datasets are named according to field t_usercod2 (a maximum of 8 characters still applies!!!). The selection “Generate one CSD” will treat the CSDs as one. The transaction data set will be named: “one”.

Create participant data

Generates participant and account data directly from the selected transaction dataset. One participant dataset is created for each CSD separately. The part dataset is named according to the CSD name.

Create CSD system setup

The task replicates the available system setup to create the CSD specific benchmarks for each CSD indentifyed from the transaction data field t_usercod2. The task will define the CSD specific benchmark simulations.
Note! As there is currently no Benchmark selection, there can be only one system setup defined. The existing system setup will be used in benchmarks.

Create CSD DBAL data

This task will create UB and LB dbal data sets for all CSDs, by running the CSD specific benchmark with the liquidity available without limits setting.

Create and run all CSDs

This task will create the analysis setups for all CSDs and initiate the execution of the automated stress tests. See 3.7. The task assumes the presence of configuration file located in the projects input directory: participantsToBeFailed.csv
The file must contain participant id’s. The first row of the file is a label row, and participant ids are to be defined starting form the second row, one id per row. As the failure scenarios for CCPs are participant and not account level this is justified. The task will run the stress test with the “by participant selection“ and will provide

the stress tester module with the account id's belonging to the participant. The task uses the default.sql filter.

Anyway if the assumptions related to this task are not appropriate, it is always possible to run the scenarios with the stress tester separately.

Create and run cover 2 on all CSD's

Same as above but will use a csv-file named cover2participantsTobeFailed.csv stored in the same input folder.

In addition to traditional transaction data, the following data must be stored according to the following table:

Information	Field in TRAN table
identifier of the asset like ISIN	t_assename
CCY (currency)	t_usercod1
CSD	t_usercod2
Value of the trade in traded CCY	t_usercod4

4 Operating the simulator via HTTP API

The Simulator application provides a HTTP (Hypertext Transfer Protocol) API interface to operate the simulator. The Simulator UI running in a web browser, uses these HTTP API methods to operate the simulator e.g. to create and maintain the user projects and all the data needed to create and run simulations.

The HTTP API can be used in the same way as a CLI allowing users to access all the methods available to the simulator UI. This allows users to integrate the simulator with other tools and programs very efficiently. For example, users can develop even their own GUIs based on these methods.

Users that want to use the Simulator http-API, need to be familiar with the HTTP protocol and the JSON object format that is used to transfer data between clients and Simulator API. The simulator respects REST and MVC design and architectural principles.

To access a HTTP API, utility tools like CURL, provide a way to make HTTP method calls over the network. Many or most analysis software and programming environments provide tools to support http connectivity nowadays.

Annex IV provides the description of some central API methods and a CURL example on how to use the API.

4.1 Used technology

In order to be able to use the http API directly, users would need to be or get familiar with the HTTP protocol, JSON and possibly html techniques e.g. in other terms client side web programming techniques.

4.1.1 HTTP protocol

In short the HTTP protocol lays down rules on how a client and a server communicate between each other. Amongst others, the protocol defines request method types used for communications between clients and servers. The types used by the simulator are mainly: POST, PUT, GET and DELETE.

The simulator API respects the HTTP protocol and provides a standardized and Restfull way to communicate with the simulator over TCP/IP network using HTTP protocol.

In practice the use of HTTP protocol means, that method calls are made with URL calls like http://localhost:8080/ that opens the simulator GUI in your browser.

For more information see https://developer.mozilla.org/en-US/docs/Web/HTTP

4.1.2 JSON notation

JSON notation is a text based format to describe data structures in a simple name value pair manner (see https://www.json.org/json-en.html).
Below is an example of a single data JSON structure:

```
{
"name": "Maria",
"gender": "F"
}
```

Below is an example of data structure of a JSON array (note the square braquets '[' ']'):

```
[
{"name": "JSON", "flexibility": "high"},
{"name": "XML", "flexibility": "low"}
]
```

4.2 Simulator API methods

The Simulator API uses mainly the following HTTP methods types:

GET - is used to query information and it returns typically a JSON object related to given query path
POST - is used to create a data entity
PUT - is used to update a data entity
DELETE - is used to remove data

E.g. HTTP method call DELETE /projects/*myprojectname* removes a project from the Simulator database.

A complete up to date list of API methods can be displayed with Swagger and the springdoc-openapi-ui library. When your simulator application is running, you can obtain the method listing by opening the page `http://localhost:8080/swagger-ui.html` in your web browser. The list is in alphabetical order.

Some of the methods are described in more detail in next chapter. Below some important HTTP API methods are described. The URL used to reach the method is derived by adding http://Domain in front of the "/….." URL path.

In order to interprete the method description under please take note that each method call is an URL call that might have a JSON file passed as form data. Also the method type (DELETE, POST, PUT,…) is passed on the background by the framework. All parameters such as JSON files are not shown on the URL syntax but are passed on, on the background. The CURL example demonstrates this in practice.

For example the GET method /templates in the annex IVwould look like: http://localhost:8080/templates. Inserting a URL to browser's address bar will use the GET method in browser URLs. This means that other methods than GET cannot be made purely as address bar URL calls. They would need to be made through HTML forms, javascript or some framework through which you can set the request method to something else than GET.

5 Algorithms and user modules

5.1 Algorithms

"Algorithms" is a common term applied to the simulator's special settlement functions such as splitting and netting. Common algorithms are provided as part of the software, and users can also develop their own algorithm modules. The interface for BoF-PSS3 algorithms and user-defined modules is the same.

The available algorithms are divided into the following main groups:

- Submission algorithms (SUB) fetch the next transaction to be submitted for processing.
- The system event handler algorithms (I) can be used to bypass the default event handling logics related to events (end of day, introduction of new transaction, limit changes, Transaction expiry, …) occurring during a simulation.
- Entry algorithms (ENT) make the initial processing of each transaction.
- Settlement algorithms (SET) call specified subalgorithms to settle queued transactions. The SET algorithms them selves do not contain any logic to release payments.
- End-of-day algorithms (END) process the final steps during a day or settlement cycle.
- Time estimation algorithms (TEA) are used to estimate the real time used for specific process. For example, a TEA algorithm can be used to induce a more realistic delay due to the processing of a settlement algorithm. TEA is also needed to simulate parallel processing of algorithms. To be able to use a TEA-algorithm, the parent algorithm must support TEA estimation.
- Settlement Confirmation Messenger (SCM) algorithms are used to deliver feedback messages from booking events of eventhandlers. These can be used to activate agents. By modifying them it is theoretically possible to add other functionality to them too. The point is that they are activated when transactions are settled.

The submission algorithm is only available at the simulation level. For every simulation, a submission algorithm must be selected. Its task is to determine which transaction is the next to be processed from all pending transactions in all systems. All other algorithms are specified at system level. The submission algorithm can be thought of as the process in which the bank decides, which is the next transaction to submit for processing to any of the systems in the simulation. This is the algorithm to modify if new behavioural patterns for banks are introduced.

The other main algorithms are assigned on system level. For example, an RTGS and net-settlement system can use different entry-algorithms in the same simulation. For every system, the entry (ENT) and end-of-day (END) algorithms must be specified. The system event handler and settlement algorithm are optional.

The following sub-algorithms can be used with ENT entry algorithms:

- Splitting algorithms (SPL) split a large transaction into sub-transactions according to specific rules.
- Injection algorithms (INJ) transfer liquidity between ancillary and main systems.

The following sub-algorithms can be used with SET settlement algorithms:

- Queue release algorithms (QUE) check and fetch individual transactions for possible settlement from the waiting queue in the order defined in the algorithm. They are useful for settling previously queued transactions once an account or participant has received more liquidity.
- Splitting algorithms (SPL) split transactions into smaller sub-transactions.
- Injection algorithms (INJ) transfer liquidity between ancillary and main systems.
- Bilateral off-setting (BOS) checks and fetches transactions that can be bilaterally off-set from the waiting queues.
- Partial netting algorithms (PNS) seek to settle a group of the queued transactions.
- Multilateral netting algorithms (MNS) attempt to settle all queued transactions in one netting event.

For special cases following separate algorithm categories are available:

- Queue release algorithm for secondary queue (QU2) is used in special case of receipt reactive gross settlement.
- In simulations with bilateral limits own algorithms are used. See section 5.1.5 for more details. For example, the bilaterally queued payments are released by QUB-algorithms.
- Partial net settlement or bilateral offsetting of bilaterally queued payments is handled by BBS-algorithms in simulations with bilateral limits.

For each payment and settlement system, there can only be one specific sub-algorithm defined of each category in the current ENT and SET algorithms. This means that the main algorithms will use the same splitting and injection algorithms, if these are defined. The order in which the sub-algorithms are set in the simulator

control data specifications is important because **sub-algorithms are called from the main algorithms in the order they were set**.

The specific algorithms are attached to the specified payment and settlement systems on the **System control data specification/modification** screen. The required parameter values are given at the same time as a parameter string. The basic controls are made for the parameters, but it is essential that users are cautious when introducing parameters. Any user-defined modules must be introduced to the system by stating the initial specifications on the **User module definition** screen. Thereafter, it is possible invoke them on the **System control data specification/modification** screen in the same way as originally provided modules and algorithms.

The time estimation algorithms (TEA) are tied to other algorithms and thus defined slightly differently (see chapter 1.1.1 step 0). They provide function to calculate an estimate to the time that would have been used in the real world by a specific algorithm.

The algorithms provided with the simulator are shortly described in the table below. More detailed definitions of the processing logics can be found in separate document, Algorithm descriptions and user module development guide. Descriptions of typical combinations of algorithms depending on the system type can be found in the next chapters. Most of the sub-algorithms are used both for RTGS and CNS systems while DNS systems have a very limited number of specific sub-algorithms.

Type	Name	Parameters	Description
SUB	SUFIFOPR	None	Fetches the next transaction or system event (among all systems) according to simulation time, priority and transaction id.
ENT	ENBASIC1	is entry settlement enabled(true or false; default = true) is FIFO enabled (true or false; default is true)	Performs the basic entry processes on a specified transaction. As default respects FIFO unless the subalgorithms it calls do not or the is FIFO enabled parameter is set to false. Entry settlement can be switched off by setting the “is entry settlement enabled” parameter to false. This is required for DNS setups for example. If the sending participant has no transaction in queue, the algorithm checks the possibilities for booking according to available liquidity (balance + available intraday credit). If booking is not successful,it will run the possibly defined sub algorithms which types are: INJ, SPL, QUE, BOS, BBS, PNS, MNS. The injection(INJ) and splitting(SPL) algorithms are given priority over the others. The rest are executed in the order they have been defined in the system setup. If the

Type	Name	Parameters	Description
			subalgorithms fail too, the payment is put to queue. Supports DVP/PVP settlement with n >=2 linked transactions or trades. ENBASIC1 does not support: Bilateral limits, multilateral limits and reservations are not taken into account.
ENT	ENFORURG	Priority (0-9)	Settles normal transactions according to normal rules i.e. as ENBASIC1 but settles highly urgent payments immediately irrespectively of liquidity constraints on the sending account. When the liquidity constraint is violated a violation entry is written to the AVST-table. The priority code is defining which level of transactions should be treated as highly urgent e.g. a parameter value of 7 indicates that transactions with a priority value equal to 7 or higher will be treated as highly urgent payments. Algorithm is only available in RTGS and CNS systems.
ENT	ENTDUAL1	Limit (0-9) Priority(0-9) Open (hhmmss) Close (hhmmss)	Entry algorithm for RTGS system with secondary receipt reactive queue. (See ch. 5.2.1 for more details). Performs the basic entry process for transactions with equal or higher priority than Limit-parameter and forced immediate settlement regardless of all limits for transactions with higher or equal priority than the Priority-parameter. Transactions with smaller priority than Limit-parameter are placed in secondary queue QU2. Open hours for QU2 are defined with parameters Open and Close.
BNT	ENBILIM1	Priority code (0-9)	Performs the basic entry processes in simulation with bilateral limits in use. See ch. 5.1.5. For each transaction checks the possibilities for booking and passes the transaction or its parts for booking or into the waiting queue. Transactions with a higher priority than the defined will be treated as urgent payments and will be processed irrespectively of bilateral limits, but the bilateral limit balances will still be updated. The other transactions will be checked against the bilateral limits and will only be booked if they pass both the bilateral limit requirement and the overall liquidity/debit cap requirement. Supports Credit cap limits starting from version 3.1.0
END	ENDRTGS1	None	Basic end of day algorithm of RTGS process. Executes the settlement algorithm and specified subalgorithms for one final time and performs end of day procedures for transactions remaining in queues. The settlement algorithm is called for each remaining participant separately passing the participant as parameter.

Type	Name	Parameters	Description
END	ENDCNS01	None	End of day/settlement cycle for CNS system type. When called, the algorithm first triggers the execution of the defined settlement algorithm (type = SET) . If the “Liquidity injections from system” and “Liquidity injections from participant” fields in PART data are defined and the account balance is not null, the algorithm will return balancies back to the main accounts at end of day. The liquidity returns are performed by generating new payments having ID starting with S*. The value is set to be equal to the balance in order to empty the accounts in the CNS system.
END	ENDDNS01	time1, … , time40	End of day/settlement cycle for DNS system type. Time, when the algorithm is executed is defined with the parameters. At least one time has to be given, while maximum number of separate settlement runs is 40. Can be used with sub-algorithms that end with D. The algorithm generates payments for: ID’s start with S This algorithm will become obsolete. DNS can be achieve also with algorithms ending with T in combination with the regular ENDRTGS1 algorithm.
BND	ENDRBIL1	code, starting time	End of day algorithm for systems with bilateral limits. The code parameter defines what should be done with the remaining bilaterally queued transactions (1 = delete, 2 = process as normal payments) and the time parameter defines when the release code is applied.
SET	SEBASIC1	None	Calls specified subalgorithms to settle queued payments. It is invoked each time a new transaction is put into queues or liquidity has been transferred to an account with queued transactions. Calls algorithms of types: QUE, SPL, INJ, PNS, MNS
SET	SETDUAL1	None	Used in simulations with secondary receipt reactive queue (See ch. 5.2.1 for more details) to call sub algorithms for settling payments in queues. It is invoked each time a new transaction is put into queues or liquidity has been transferred to an account with queued transactions. For the normal primary RTGS-queue all normal sub algorithms are available. QU2 algorithm is

Type	Name	Parameters	Description
			used to release payments from the secondary queue.
BET	SEBILIM1	Priority code (0-9)	Calls specified subalgorithms to settle queued payments. It is invoked each time a new transaction is put into queues or liquidity has been transferred to an account with queued transactions. It invokes specified sub-algorithms. Urgent transactions, higher than the specified priority, will be processed without regarding bilateral limits, while the other transactions should fulfil limit requirements. Only bilateral offsetting algorithms for bilateral queues (BBS) are used for settling transactions in bilateral queues. All other algorithms are available for “normal” transactions.
QUE	QUFIFOPR	None	Releases individual transactions from waiting queues upon arrival of additional liquidity in priority and FIFO order.The exact behavior depends on the calling algorithm. QUFIFOPR performs FIFO on the set of transactions it receives as parameter. The 45efault event handler calls it account by account, which means that it acts as an account wise FIFO algorithm. Supports DVP/PVP settlement with n >=2 linked transactions or trades.
QUE	QUUSEDEF	None	Releases transactions from waiting queues upon arrival of additional liquidity in the order defined by user defined fields 1and 2. (T_USERCOD1...2) in ascending order and the first code has the highest priority. This facilitates free definition of queue order by moving the right data to the user code fields e.g. the size of transactions. Note that the User code fields are of type VARCHAR(12) in order to carry all kind of data and thereby sorted in alphabetic ascending order, which means that numeric values need to be same length in order to be sorted correctly. Supports DVP/PVP settlement with n >=2 linked transactions or trades.
QUE	QUUSEDBP	None	Releases transactions from waiting queues upon arrival of additional liquidity in the order defined by user defined fields 1and 2 by using the bypass option. (T_USERCOD1...2) in ascending order and the first code has the highest priority. This facilitates free definition of queue order by moving the right data to the user code fields e.g. the size of transactions. Note that the User code fields are of type VARCHAR(12) in order to carry all kind of data and thereby sorted in alphabetic ascending order, which means that numeric values need to be same length in order to be sorted correctly. In the bypass case transactions are processed in priority and FIFO order with the exception that

Type	Name	Parameters	Description
			if a transaction higher up in the queue cannot be settled, it is bypassed and payments lower in the queue are tested for settlement (in priority or FIFO order) until no more settlable transactions can be found. Supports DVP/PVP settlement with n >=2 linked transactions or trades.
QU2	QURRFIPR	EOD (“gross” or “return”) NewPriority (0-9, optional)	Releases transactions from secondary queue in priority FIFO order in receipt reactive gross settlement simulations. EOD parameter defines the processing logic of secondary queue payments at the end of each period: *gross* means transactions are moved into primary RTGS queue, *return* means transactions are discarded (i.e. returned to original sender). Supports DVP/PVP settlement with n >=2 linked transactions or trades. In the former case transactions are given a new uniform priority if NewPriority has a value. Otherwise they retain their original priority. (See ch. 5.2.1 for more details)
QUB	QBFIFOPR	None	Releases transactions from bilateral waiting queues, when bilateral limits are increased and upon arrival of transactions from the counterparty in priority and FIFO order. Supports Credit cap limits starting from version 3.1.0 Supports DVP/PVP settlement with n >=2 linked transactions or trades.
QUB	QBBYPAFI	None	Releases transactions from bilateral waiting queues, when bilateral limits are increased and upon arrival of transactions from the counterparty in priority and FIFO order with the exception that if a transaction higher up in the queue cannot be settled, it is bypassed and payments lower in the queue are tested for settlement (in priority or FIFO order) until no more settlable transactions can be found. Supports Credit cap limits starting from version 3.1.0 Supports DVP/PVP settlement with n >=2 linked transactions or trades.
QUB	QBUSEDEF	None	Releases transactions from bilateral waiting queues, when bilateral limits are increased and upon arrival of transactions from the counterparty in the order defined by user defined fields 1and 2. (T_USERCOD1...2) in ascending order and the first code has the highest priority. This facilitates free definition of queue order by moving the right data to the user code fields e.g. the size of transactions. Note that the User code fields are of type VARCHAR(12) in order to carry all kind of data and thereby sorted in alphabetic ascending order, which means that numeric values need to be same length in order to be sorted correctly.

Type	Name	Parameters	Description
			Supports Credit cap limits starting from version 3.1.0 Supports DVP/PVP settlement with n >=2 linked transactions or trades.
QUB	QBUSEDBP	None	Releases transactions from bilateral waiting queues, when bilateral limits are increased and upon arrival of transactions from the counterparty in the order defined by user defined fields 1and 2 by using the bypass option. (T_USERCOD1...2) in ascending order and the first code has the highest priority. This facilitates free definition of queue order by moving the right data to the user code fields e.g. the size of transactions. Note that the User code fields are of type VARCHAR(12) in order to carry all kind of data and thereby sorted in alphabetic ascending order, which means that numeric values need to be same length in order to be sorted correctly. In the bypass case transactions are processed in priority and FIFO order with the exception that if a transaction higher up in the queue cannot be settled, it is bypassed and payments lower in the queue are tested for settlement (in priority or FIFO order) until no more settlable transactions can be found. Supports Credit cap limits starting from version 3.1.0 Supports DVP/PVP settlement with n >=2 linked transactions or trades.
SPL	SPMVALU1	Max. transaction value, positive amount with two decimals	Splits transactions into sub-transactions according to specified maximum transaction value. For example, if a max value of 500 is specified, a transaction of 1,350 is split into sub-transactions of 500, 500 and 350, with 350 the last to be processed.
SPL	SPAVLIQ1	None	Splits transactions using available liquidity. For example, when 450 units are available on the account, a transaction of 1,350 is split into 450 and 900 of which the 450 is directly processed and 900 remains in the waiting queue.
INJ	INVALUE1	Positive value with two decimals	Mainly, the INVALUE1 algorithm is called by entry and settlement algorithms such as ENBASIC and SEBASIC1. The algorithm as such does not settle anything. It injects liquidity to a participant/account when required in given amounts defined with the parameter value of the algorithm if called with a transaction as parameter like in entry. The source of liquidity and permission to perform injections are defined by using the “Liquidity injections from system” and “Liquidity injections from participant” fields in PART data. The injection is performed by generating a payment. The payments ID starts with I. The value is set to the amount indicated by the parameter.

Type	Name	Parameters	Description
			When the algorihm is called with a participant as parameter like when it is called by SEBASIC1, if liquidity is sufficient, the injected liquidity is released and returned back to the source participant/account by generating payments. The values are set to be equal with the parameter value defined in system definition for the algorithm. The transaction ID starts with J. If needed the algorithm can generate several transactions. Typically liquidity injections can be used between a main and ancillary payment system.
INJ	INPERCE1	Positive percentage (format 100.00)	Injects an amount that corresponds to a given percentage of the credit limit available in the ancillary system.
INJ	INJEXACT	None	Injects exactly the required value of liquidity from the account defined in PART data. Returns are also performed with exact amounts: either according to what has been injected or how much liquidity is available to return.
BOS	BOBASIC1	None	Performs bilateral off-setting of waiting queues in FIFO and priority order and using available liquidity. The algorithm is performed after each transaction queue entry and liquidity change, so caution is needed with large transaction volumes. Because of the bilateral processing, the priority FIFO rule can become bypassed on system level in bilateral off-setting.
BOS	BOUSEDEF	None	Performs bilateral off-setting of waiting queues using available liquidity in the order defined by user defined fields 1and 2. (T_USERCOD1...2) in ascending order and the first code has the highest priority. This facilitates free definition of queue order by moving the right data to the user code fields e.g. the size of transactions. Note that the User code fields are of type VARCHAR(12) in order to carry all kind of data and thereby sorted in alphabetic ascending order, which means that numeric values need to be same length in order to be sorted correctly. The algorithm is performed after each transaction queue entry and liquidity change, so caution is needed with large transaction volumes. Because of the bilateral processing, the priority FIFO rule can become bypassed on system level in bilateral off-setting.
BBS	BBFIFOPC	None	Performs partial bilateral net offsetting of bilaterally queued transactions in FIFO and priority order by including transactions that can be settled within the available bilateral limit. The algorithm removes transactions one-by-one in priority and time order (starts by removing the most recent submitted transactions with the

Type	Name	Parameters	Description
			lowest priority) for each participant pair. The solution must fulfil the bilateral limit criteria and the overall balance limitations. The algorithm is performed after each transaction queue entry, liquidity transfer and overall credit and bilateral limit change, so caution is needed with large transaction volumes. Supports Credit cap limits starting from version 3.1.0
BBS	BBDEQUEC	None	Performs partial bilateral net offsetting of bilaterally queued transactions in user-defined order (ascending user-defined field 1 and 2) by including transactions that can be settled within the available bilateral limit. The algorithm removes transactions one-by-one in (starts by removing the last based on user defined field 2 first and then field 1) for each participant pair. The solution must fulfil the bilateral limit criteria and the overall balance limitations. The algorithm is performed after each transaction queue entry, liquidity transfer and overall credit and bilateral limit change, so caution is needed with large transaction volumes. Supports Credit cap limits starting from version 3.1.0
BBS	BBFIFOPI	Minutes interval (1-60);starting- time (hhmmss)	Performs partial net offsetting of bilaterally queued transactions in FIFO and priority order at given time intervals during the day (in minutes). A starting-time may be defined (the default starting time is when the system is opened) The algorithm removes transactions one-by-one for participants unable to settle to see if a partial settlement is possible. Supports Credit cap limits starting from version 3.1.0
BBS	BBDEQUEI	Minutes interval (1-60);starting-time (hhmmss)	Performs partial net offsetting of bilaterally queued transactions in user-defined order at given time intervals during the day (in minutes). A starting-time can be defined (the default staring-time is when the system is opened). The algorithm removes transactions one-by-one for participants unable to settle to see if a partial settlement is possible. Supports Credit cap limits starting from version 3.1.0
BBS	BBFIFOPT	Time;time;time;…;time (max 12, 24 h HH:MM format	Performs partial net offsetting of bilaterally queued transactions in FIFO and priority order at the specified times during the day. The algorithm removes transactions one-by-one for participants unable to settle to see if a partial settlement is possible. Supports Credit cap limits starting from version 3.1.0
BBS	BBDEQUET	Time;time;time;…;time (max 12, 24 h HH:MM format	Performs partial net offsetting of bilaterally queued transactions in user-defined order at the specified times during the day. The algorithm removes transactions one-by-one for participants unable to settle to see if a partial settlement is possible. Supports Credit cap limits starting from version 3.1.0

Type	Name	Parameters	Description
PNS	PNFIFOPC	None	Performs partial multilateral net settlement of queued transactions in FIFO and priority order by including transactions that can be settled with available liquidity (the algorithm removes transactions one-by-one for participants unable to settle to see if a partial settlement is possible). The algorithm is performed after each transaction queue entry, so caution is needed with large transaction volumes. Supports DVP/PVP settlement with n >=2 linked transactions or trades.
PNS	PNDEQUEC	None	Performs partial multilateral net settlement of queued transactions in defined order (ascending user-defined field one and two and transaction ID in the TRAN data) by including transactions that can be settled with available liquidity (the algorithm removes transactions one-by-one for participants unable to settle to see if a partial settlement is possible). The algorithm is performed after each transaction queue entry, so caution is needed with large transaction volumes. Supports DVP/PVP settlement with n >=2 linked transactions or trades.
PNS	PNFIFOPI	Minutes interval (1-60); Starting time (hhmmss)	Performs partial net settlement of queued transactions at a given time interval during the day (in minutes) in FIFO and priority order by including transactions that can be settled with available liquidity (the algorithm removes transactions one-by-one for participants unable to settle to see if a partial settlement is possible). The starting-time parameter defines when the first netting occasion occurs. Supports DVP/PVP settlement with n >=2 linked transactions or trades.
PNS	PNDEQUEI	Minutes interval (1-60); Starting time (hhmmss)	Performs partial multilateral net settlement of queued transactions at the given time interval during the day (in minutes) in defined order (ascending user-defined field one and two and transaction identifier in the TRAN data) by including transactions that can be settled with available liquidity (the algorithm removes transactions one-by-one for participants unable to settle to see if a partial settlement is possible). The starting-time parameter defines when the first netting occasion occurs. Supports DVP/PVP settlement with n >=2 linked transactions or trades.
PNS	PNFIFOPT	Time;time;time;…;time (max 40 , 24 h HH:MM format)	Performs partial net settlement of queued transactions at given times in FIFO and priority order by including transactions that are possible to settle with available liquidity (the algorithm removes transactions one-by-one for participants unable to settle to see if a partial settlement is possible). Supports DVP/PVP settlement with n >=2 linked transactions or trades.

Type	Name	Parameters	Description
PNS	PNFIFOPD	None	Performs partial net settlement of queued transactions at given occasions in FIFO and priority order by including transactions that are possible to settle with available liquidity (the algorithm removes transactions one-by-one for participants unable to settle to see if a partial settlement is possible). Supports DVP/PVP settlement with n >=2 linked transactions or trades.
PNS	PNDEQUED	None	Performs partial multilateral net settlement in DNS systems based on ascending order of user defined fields one and two and the transaction identifier. The algorithm removes transactions one-by-one for participants unable to settle to see if a partial settlement is possible. Supports DVP/PVP settlement with n >=2 linked transactions or trades.
PNS	PNDEQUET	Time;time;time;…;time (max 40, 24 h HH:MM format	Performs partial net settlement of queued transactions at defined occasions in defined order (ascending user-defined field one and two and transaction identifier) by including transactions that can be settled with available liquidity (the algorithm removes transactions one-by-one for participants unable to settle to see if a partial settlement is possible). Supports DVP/PVP settlement with n >=2 linked transactions or trades.
MNS	MNSETTLC	None	Performs total net settlement of all queued transactions when sufficient liquidity is available. Total net settlement implies that settlement is only performed in cases where all queues can be emptied (partial multilateral settlement not accepted). The algorithm is performed after each transaction queue entry, so caution is needed with large transaction volumes.
MNS	MNSETTLD	None	Performs total net settlement of all queued transactions in deferred net settlement systems (DNS) when sufficient liquidity is available.
MNS	MNSETTLI	Minutes interval (1-60); Starting time (hhmmss)	Performs total net settlement of all queued transactions when sufficient liquidity is available at the given time interval during the day (in minutes). The starting-time parameter defines when the first netting occasion occurs.
MNS	MNSETTLT	Time;time;time;…;time (max 40, 24 h hhmmss format	Performs total net settlement of all queued transactions when sufficient liquidity is available at the defined times.
SCM	SECOMSGR		Is activated when a settlement system books a transaction. Agent wake up calls are added to the simulation's event queue for the debit and credit accounts of all transactions settled through the booking process. The algorithm can be modified to trigger other sorts of reactions.

5.1.1 Algorithms for RTGS systems

Real-time Gross Settlement (RTGS) systems process transactions one-by-one in a real-time environment using central bank accounts. Each transaction is booked, queued or discarded as defined by the algorithms set for the system. The release of queued transactions is determined using various settlement algorithms. The user specifies the processing patterns by selecting algorithms and assigning necessary parameters.

The main algorithms available for RTGS systems are the ENT algorithms ENBASIC1 and ENFORURG, the END algorithm ENDRTGS1 and the SET algorithm SEBASIC1. ENT and END algorithms should be chosen for all normal RTGS simulations. If a system with a queuing facility is simulated, the SEBASIC1 algorithm should also be selected.

The ENFORURG algorithm enables settlement of highly urgent payments irrespectively of the liquidity constraints. When the balance restriction is violated an account violation statistics entry is written in the AVST-table. The priority parameter defines the urgency level of transaction that will be settled immediately as highly urgent payments. The other transactions are regarded as normal transactions, which follow the general rules for settlement. This makes it possible to find out what the efficient liquidity need (lower bound) would be, if highly urgent payments were settled at once and less urgent payments were settled eg by the end of the day.

RTGS systems may incorporate payment-splitting and liquidity-injection sub-algorithms. These are invoked by the ENT algorithm if they have been specified.

When a queuing facility is included, the SEBASIC1 algorithm invokes the specified sub-algorithms in the order specified on the define system data screen to settle queued transactions. SEBASIC1 will also invoke payment-splitting and liquidity-injection when these have been specified.

Currently, the ENT and SET algorithms can only invoke a single sub-algorithm of the same type (SPL, INJ, QUE, BOS, PNS or MNS), e.g. there can only be one splitting algorithm in use for a given system. In principle, custom user modules with the possibility of handling multiple sub-algorithms can be developed to overcome this limitation. The different algorithm classes can be seen to have partially overlapping functions, particularly related to queue arrangements. This is because gridlock resolution algorithms such as PNS and MNS are independent from QUE algorithms in the way they order the transactions in the queues before processing.

As an example it is possible to create a system with QUE algorithm releasing individual transactions with priority FIFO logic and PNS algorithm which follows user defined queue order in selecting the transactions to be settled e.g. smallest first.

The table below shows examples of possible RTGS configurations using the main and sub-algorithms. In all cases the basic submission algorithm, SUFIFOPR, is used to submit payments in order based on time (earliest first), priority of the transaction (descending order, i.e. highest priority first) and the transaction ID (ascending order). When transactions in the simulation compete for the same time slot, then the transaction with the highest priority is chosen. If there are still several competing transactions, then the transaction ID order becomes the determining factor. This means that in multisystem simulations the transactions in the system containing the smallest transaction IDs will be executed first when time and priority are the same.

Some examples of possible logical algorithm combinations in RTGS systems:

RTGS description	**Main alg.**	**Sub-alg.**	**Comments**
1. No queuing facility available, transactions without liquidity remain unsettled	ENBASIC1 ENDRTGS1		Can be used with “credit without limits” to find the “upper bound” and “lower bound” of liquidity.
2. FIFO queuing	ENBASIC1 ENDRTGS1 SEBASIC1	QUFIFOPR	Transactions are queued when liquidity is insufficient for settlement; they are released in FIFO and priority order as liquidity becomes available from incoming payments or extended credits.
3. Bypass FIFO queuing	ENBASIC1 ENDRTGS1 SEBASIC1	QUBYPAFI	Same as 2, except that the FIFO order is bypassed to settle payments lower in the queue for which sufficient liquidity is available.
4. FIFO queuing and bilateral offsetting	ENBASIC1 ENDRTGS1 SEBASIC1	QUFIFOPR BOBASIC1	Transactions are queued in FIFO order and settled also with bilateral offsetting when sufficient liquidity is available. Note: Bilateral offsetting can cause bypasses in strict system level priority FIFO order of transactions.
5. FIFO queuing, bilateral offsetting and full multilateral netting at a given interval	ENBASIC1 ENDRTGS1 SEBASIC1	QUFIFOPR BOBASIC1 MNSETTLI	As in 4, plus full multilateral netting is attempted at a given interval (e.g. every 10 minutes).
6. Queuing and splitting according to maximum value	ENBASIC1 ENDRTGS1 SEBASIC1	QUFIFOPR SPMVALU1	As in 2, but large transactions are split into smaller ones to better circulate liquidity.

RTGS description	Main alg.	Sub-alg.	Comments
7. Queuing and splitting according to available liquidity	ENBASIC1 ENDRTGS1 SEBASIC1	QUFIFOPR SPAVLIQ1	As in 2, but all unsettlable transactions are split based on available liquidity. This algorithm gives the benchmark for maximal liquidity employment (difficult to implement in practice).

The desired RTGS system structure is defined by combining the various available standard algorithms or user-developed modules. In most cases, the SEBASIC1 will be invoked in addition to the mandatory ENDRTGS1 and ENBASIC1 or ENFORURG. A queue release algorithm, e.g. QUFIFOPR, is also often used in RTGS simulations because most RTGS systems contain queuing possibilities.

Only those algorithms that are strictly necessary to describe the desired processing logic should be included in system definition. Before performing large scale simulations it is wise to validate the created model by testing the process with simple examples with only few transactions so that the correct outcome for the input can be verified from the output.

Note that there is no checking logic in the simulator to assess whether the selected algorithm combination is rational. The user is responsible for selecting appropriate algorithms among those applicable for RTGS simulations.

5.1.2 Algorithms for CNS systems

Continuous Net Settlement (CNS) systems are private systems that process transactions one-by-one in a real-time environment. Transactions are booked on private settlement accounts during the day. In most cases, these are settled with central bank accounts at the end of the day and possibly more often. From the simulator's standpoint, RTGS and CNS systems are quite similar. However, CNS systems offer the possibility for intraday liquidity swaps with an RTGS system and the end-of-day settlement in an RTGS system.

Processing (booking, queuing and discarding) of transactions takes place as in an RTGS system. The user specifies the processing patterns by selecting algorithms and assigning necessary parameters. The algorithms available for CNS systems are also the same as for RTGS systems.

When CNS systems are simulated separately (not as part of a multisystem environment with a main RTGS system), the initial liquidity needed for the processing can be introduced as:

– initial balances representing liquidity reservations made on RTGS accounts,
– credit limits representing the credit risk of the private settlement bank or the system itself, and
– liquidity transactions made from the special account of the settlement bank.

When the CNS system operates based on credit risks, the simulator calculates the open positions for each participant.

The table below shows examples of possible CNS configurations using the various main and sub-algorithms. In all cases, the submission algorithm, SUFIFOPR, submits payments in time, priority and transaction ID order. If there are transactions in single or several systems competing for the same time slot, then the transaction with the highest priority is chosen. If there are still several competing transactions, then the transaction ID order becomes the determining factor. All systems are treated equally, so the priorities used in different systems should be on the same scale if submission order is critical. Because transaction ID's must be unique simulation-wide, you should use smaller IDs for "more important" systems, as these IDs will be the final determining factor.

Some examples of possible logical algorithm combinations in CNS systems:

CNS description	**Main alg.**	**Sub-alg.**	**Comments**
1. No queuing facility available, transactions without liquidity will remain unsettled	ENBASIC1 ENDCNS01		Can be used with "credit without limits" to find the "upper bound" and "lower bound" of liquidity.
2. FIFO queuing	ENBASIC1 ENDCNS01 SEBASIC1	QUFIFOPR	Transactions are queued when liquidity is insufficient for settlement; they are released in FIFO and priority order as liquidity becomes available from incoming payments or extended credits.
3. Bypass FIFO queuing	ENBASIC1 ENDCNS01 SEBASIC1	QUBYPAFI	Same as 2, except that the FIFO order is bypassed to settle payments lower in the queue for which enough liquidity is available.
4. FIFO queuing and bilateral offsetting	ENBASIC1 ENDCNS01 SEBASIC1	QUFIFOPR BOBASIC1	Transactions are queued in FIFO order and settled also with bilateral offsetting when

CNS description	Main alg.	Sub-alg.	Comments
			sufficient liquidity is available Note: Bilateral offsetting can cause bypasses in strict system level priority FIFO order of transactions.
5. FIFO queuing, bilateral offsetting and full multilateral netting at given intervals	ENBASIC1 ENDCNS01 SEBASIC1	QUFIFOPR BOBASIC1 MNSETTLI	As in 4, plus full multilateral netting is attempted at a given interval (e.g. every 10 minutes).
6. FIFO queuing and splitting according to maximum value	ENBASIC1 ENDCNS01 SEBASIC1	QUFIFOPR SPMVALU1	As in 2, but large transactions are split into smaller ones to better circulate liquidity.
7. FIFO queuing and splitting accord available liquidity	ENBASIC1 ENDCNS01 SEBASIC1	QUFIFOPR SPAVLIQ1	As in 2, but all unsettlable transactions are split based on available liquidity. This algorithm gives the benchmark for maximal liquidity employment (difficult to implement in practice).
8. FIFO queuing with liquidity injections	ENBASIC1 ENDCNS01 SEBASIC1	QUFIFOPR INVALUE1	Transactions are queued and liquidity swaps are executed in both directions with an RTGS system, when liquidity is needed or superfluous.

Define the desired CNS system structure by combining the available algorithms and user-developed modules. In most case, SEBASIC1 is also invoked along with the mandatory ENBASIC1 (or ENFORURG) and ENDCNS01. A queue release algorithm (e.g. QUFIFOPR) is also used in most CNS simulations because most CNS systems contain a queuing facility.

When simulations involve a CNS system interacting with an RTGS system, it is important to provide in PART data the correct account information for end-of-day settlement and possible intraday liquidity swaps. For each account in CNS system having possibility of liquidity swaps, the source system and source account of liquidity must be defined. Similarly account on which the end of day settlement is performed has to be defined for each CNS account with this feature. The ENDCNS01 algorithm performs the end-of-day settlement bookings. These may violate the liquidity restrictions in the RTGS system, so the user can select an option that writes violations to an output table. One way of avoiding violations is to use debit caps and reservations to keep the positions within acceptable limits. If there is a partly or complete net settlement during or at the end of the settlement cycle, the type of net settlement algorithm should be specified.

Only those algorithms that are strictly necessary to describe the desired processing logic should be included in system definition. Before performing large scale simulations it is wise to validate the created model by testing the process with simple examples with only few transactions so that the correct outcome for the input can be verified from the output.

Note that there is no checking logic in the simulator to assess whether the selected algorithm combination is rational. The user is responsible for selecting appropriate algorithms among those applicable for CNS simulations.

5.1.3 Algorithms in DNS systems

Deferred Net Settlement (DNS) systems settle transactions in batches at given settlement occasions. Participants hold settlement accounts. Although the DNS system uses batch settlement, the simulator process resembles the RTGS and CNS processes to the extent that each transaction is processed in real-time according to its submission time and is queued until the next settlement occasion. This approach makes it possible to track the accumulation of credit risks when the transactions are processed by participant and delivered as final to end customers. The DNS processing also reports the queuing time.

Transactions are booked in DNS systems when sufficient liquidity is available, i.e. the settlement accounts have a positive value or the debit cap (credit limit) for the accounts has not been exceeded. Otherwise, transactions are transferred to the next settlement occasion or discarded. No queue release algorithms should be assigned for DNS systems as this makes the system work in continuous mode. DNS systems use time-of-netting algorithms (PNS or MNS). The user specifies the processing patterns by selecting algorithms and assigning necessary parameters.

When DNS systems are simulated separately (not as a part of a multisystem environment with a main RTGS system), the initial liquidity needed for the processing can be introduced as initial balances or credit caps for each participant. “Credit without limits” assigns open-ended credit caps.

DNS systems use the same entry algorithm ENBASIC1 as other systems, but have a special END algorithm ENDDNS01 for defining the end-of-day occasion and any intraday settlement cycles. The ENBASIC1 parameter “is entry settlement enabled” must be set to false. DNS systems use very few sub-algorithms. A set of deferred PNS algorithms (PNFIFOPD and PNEQUED) and a deferred MNS algorithm (MNSETTLD) are available. These are performed at the end of each specified settlement cycle. It is also possible to use MNSETTLT (and (PNFIFOPT and

PNEQUET) algorithms with the regular end algorithm ENDRTGS. The algorithm ending T refers to time schedule which is the same as deferred. The algorithms with the D ending are likely to become deprecated. Payments can also be split according to value.

The table below shows examples of possible DNS configurations using the different main and sub-algorithms. The submission algorithm, SUFIFOPR, submits payments in time, priority and transaction ID order. If there are transactions in single or several systems competing for the same time slot, then the transaction with the highest priority is chosen. If there are still several competing transactions, then the transaction ID order becomes the determining factor. All systems are treated equally, so the priorities used in different systems should be on the same scale if submission order is critical. Because transaction ID's must be unique simulation-wide, you should use smaller IDs for "more important" systems, as these IDs will be the final determining factor.

Examples of algorithm combinations in DNS systems:

DNS description	Main alg.	Sub-alg.	Comments
1. Full multilateral netting at given occasions	ENBASIC1 ENDRTGS1 or *ENBASIC1* *ENDDNS01*	MNSETTLT *MNSETTLD*	Transactions are netted with full ("all or nothing") multilateral netting at end of the settlement cycle. The second option with ENDDNS01 and MNSETTLD will become obsolete.
2. Partial netting at given occasions	ENBASIC1 ENDRTGS1 or *ENBASIC1* *ENDDNS01*	PNFIFOPT *PNFIFOPD*	Transactions are settled in FIFO order within the given debit caps/liquidity using partial net settlement. The second option with ENDDNS01 and *PNFIFOPD* will become obsolete.

Define the desired DNS system structure by combining the available algorithms or user-developed modules. These are basically net settlement algorithms in DNS systems. DNS systems can also include splitting and injection features. Queue release algorithms should not be invoked.

Typical simulations regarding DNS are dividing the process into more settlement cycles and checking the effects on liquidity and speed. This is preferably done by dividing the transaction input data in separate slots e.g. by using the export input data by selecting the time slots according to settlement cycles. Each settlement cycle can then be run separately resulting in complete statistics for each cycle.

In simulations where a DNS system is cooperating with an RTGS system it is important to provide the correct account information for end-of-day settlement and possible intraday liquidity swaps. The ENDDNS01 algorithm will perform the end-of-day settlement bookings. Note that these may violate the liquidity restrictions in the RTGS system. Violations are written to AVST output table, if this output form is selected in simulation configuration phase.

Note that there is no checking logic in the simulator to assess whether the selected algorithm combination is rational. The user is responsible for selecting appropriate algorithms among those applicable for DNS simulations.

5.1.4 Algorithms in DVP/PVP processing systems

In delivery-versus-payment and payment-versus-payment (DVP/PVP) processing, two or more trades or transactions defined by the T_LINKCODE and T_LINKSYST are linked together. Booking of linked transactions can only be done when all linked legs are settlable. DVP/PVP transactions can be processed in separate dedicated DVP/PVP systems or among normal one-legged transactions in traditional systems. DVP/PVP transactions can also have their “legs” in different systems as defined by the T_LINKSYST field. See also chapter 5.2.2 Group codes for DVP linking multiple transactions.

Note that you always need to include a QUE-algorithm for DVP-processing, because even if introduced at the same time the first transaction leg has always to wait in queue for the submission of the other transaction leg even if it might be the next transaction to be processed.

DVP/PVP limits the use of the available netting algorithms. Available settlement algorithms function only on a single-system level and cannot be used for linked transactions between different systems (e.g. netting can only be done within one system at a time). There are also limitations for the algorithms applied within one system. The user can design own user modules with complex DVP-algorithms to overcome these limitations.

For DVP/PVP within the same system, full multilateral settlement is available. Partial net settlement algorithms are provided in such form that when a leg is unsettlable, the other legs are also discarded. Otherwise, DVP/PVP functions like PNS algorithms for normal one-legged transactions. However, the “optimality” of such algorithm for DVP/PVP transactions has not been verified and the results can be inconsistent (caution is needed; it is more a generalization of the algorithm possibilities).

Splitting and bilateral offsetting is not possible for DVP/PVP transactions as this would be difficult to introduce for the other leg. If defined, splitting and offsetting are only performed for normal one-legged transactions.

Examples of algorithm combinations in DVP/PVP processing in a single system:

DVP/PVP processing	Main alg.	Sub-alg.	Comments
1. Straight-forward RTGS processing	ENBASIC1 ENDRTGS1 SEBASIC1	QUEFIFOPR	Transactions are settled in FIFO order when sufficient liquidity available.
2. DNS based on full multilateral netting	ENBASIC1 ENDDNS01	MNSETTLD	Transactions are netted with full ("all or nothing") multilateral netting at given occasions.
3. CNS system with queuing and partial netting at given intervals	ENBASIC1 ENDDNS01 SEBASIC1	QUEFIFOPR MNSETTLI	Transactions are queued and settled in FIFO order and multilateral netting is performed at given time intervals.

DVP/PVP transactions can be defined for all types of systems (RTGS, CNS and DNS). These can be defined for processing within one system or as intersystem transactions.

Note that there is no checking logic in the simulator to assess whether the selected algorithm combination is rational. The user is responsible for selecting appropriate algorithms for the system type (RTGS, CNS or DNS).

5.1.5 Algorithms for systems with bilateral limits

Bilateral limits can be used to describe debit caps, credit caps (since version 3.1.0) and similar participant level bilateral or multilateral restrictions for payment clearing and settlement. The functioning of these limits is described below first in bilateral level. Definition of multilateral level is explained separately at the end.

If a bilateral limit (debit cap) is set from participant *A* to participant *B*, the cumulative net value of payments settled between these participants – called bilateral balance – must remain within the given limit. The debit cap defines the smallest allowed value for this bilateral balance. A 'sending surplus' (i.e. when A has sent a greater value of payments to B than it has received) is equivalent to a negative value of the bilateral balance. Thus a negative limit value defines a situation, where a higher value of payments is allowed to be sent to a given participant than is received. The opposite case, a positive bilateral limit, means that participant *A* requires that a certain value of payments has arrived from participant

B before it settles any outgoing payments to B.The constraint can be formulated the following way for any given moment *T*:

$$bilateral\ balance(T) = S_{BA}(T) - S_{AB}(T) \geq debit\ cap(T)$$

where $S_{XY}(t)$ denotes the cumulative value of settled payments from participant *X* toward participant *Y* starting from beginning of day until the time *t*.

A bilateral credit cap defines similarly the upper limit for the bilateral balance between A and B. Thus if A sets a credit cap against B, the incoming payments from B to A are blocked if they lead to an increase of A's bilateral balance over the given credit cap. The constraint takes the following form:

$$bilateral\ balance(T) = S_{BA}(T) - S_{AB}(T) \leq credit\ cap(T)$$

The calculation of bilateral balance starts from zero from the beginning of each simulated day, or from the moment when first bilateral limit is defined to the given pair of participants. Thus bilateral positions are followed only for those participant pairs where some constraints are also in place. No balances are transferred to next day in multiday simulations.

Bilateral limits can be viewed from two perspectives depending on the type of system under study:

- In liquidity based systems (typically RTGS) the limits are defined by the sender of payments. Here the sender can restrict the value of outgoing payments in order to save liquidity. Limit can be set individually for each bilateral pair of participants or multilaterally to limit the total net value of liquidity flowing out from a given participant.
- In credit based systems (typically CNS), the participant receiving payments is actually setting the limits. These can be viewed as restrictions for each sender's ability to send payments towards the receiver –limits for the credit exposure the receiver is accepting. In this case, for example, credit limit granted from participant A to B has to be implemented in the simulator as bilateral debit cap limit in the opposite direction, from B to A. This way it is correctly limiting the payments sent from B to A.

All bilateral limitsare set within the BLIM input table. The limits do not need to be symmetric, i.e., limit from *A* to *B* does not need to be same as from *B* to *A*. Bilateral limits are only in force for those participant pairs for which they are explicitly specified. Debit caps are defined by setting a value to the L_NEWVALUE field and credit caps are defined by setting a value in the L_DBCVALUE field of the BLIM data table. either one or both values can be set with one row of input data file. In

such simulator projects, which are created with older version than 3.1.0, such debit and credit cap values, which have same participants and same time label, are necessary to be imported in one and same row of data file. For more details see Descriptions of Databases and files.

Specified bilateral limits are valid until the simulation ends or the limit is changed. Individual bilateral limits can be altered during the day and also completely removed by placing a special value (0.99) in the input.

If a new value of bilateral limit is defined, which causes the already existing bilateral balance to be infeasible, the limit change will still take place. The new limit will block all payments, which would cause the situation to become still worse. Thus violated debit cap does not prevent inflow of payments even if bilateral position would remain below the defined limit still after the payment is received. Similarly violated credit cap position does not prevent outgoing payments.

In addition to bilateral limits, also multilateral intraday limits can be defined in BLIM table by stating the receiving participant as *MULTILIMIT. This means that net value of payments sent and received between specified participant/account and all others has to remain within the given multilimit. The difference between multilimit functionality and normal intraday credit limit is that ICCL restriction can be affected by beginning of day balance of the participant while multilimit only records and limits the total change of balance during the day. Values for multilimit are given similarly as other bilateral limits i.e. negative value indicates that sending surplus is allowed. Another way to describe the difference is to note that ICCL data can be used to define actual liquidity for participants, while BLIM data only defines limitations for the flow of existing liquidity. Thus such simulation would not be able to settle any payments regardless of the BLIM values, where only bilateral limits would be defined but no liquidity would be given with DBAL or ICCL data or by granting of unlimited credit in the system setup.

The bilateral limits can be applied to one part of the transaction flow. This is carried out by giving a high priority for those transactions, which need to be settled regardless of bilateral limits. All transactions that pass the bilateral limit control have also to pass the general liquidity availability control i.e. general credit limits may not be violated. High priority transactions, and those which have no bilateral limits affecting them, are processed normally by the simulator.

Simulated systems with bilateral limits require their own bilateral algorithms. These are filtered to be visible in the **system control data specification window** when **bilateral limits in use** is selected. Bilateral algorithms are available for RTGS and CNS systems. The behavioural rules related to bilateral limits are algorithm specific

and thus the exact specific behavioural details for each algorithm must be checked from the algorithm specific descriptions. One example of such rules is the special treatment of high priority payments. Some examples of bilateral processing alternatives and algorithm combinations are given below.

Description of setup	Main alg.	Sub-alg.	Comments
1. Straight-forward RTGS processing with bilateral limits. Bilateral limits given in BLIM dataset.	ENBILIM ENDRBIL1 SEBILIM1	QBFIFOPR	Transactions are settled in FIFO order when sufficient liquidity is available. Besides normal tests for available liquidity, transactions have to pass bilateral limit tests.
2. CNS-processing with user defined queue order and bilateral limits. High priority transactions bypass bilateral checks.	ENBILIM(5) ENDRBIL1 SEBILIM1	QBUSEDEF	Transactions are settled in user defined order (i.e. ascending by user defined input fields 1 and 2). Transactions with high priority (i.e. smaller than given parameter 5) are settled whenever liquidity is sufficient. Others have to also pass the bilateral limit tests. Note: same algorithms apply in CNS as in RTGS.

5.2 Algorithms for special cases

This chapter contains descriptions of different sets of algorithms that have been developed for special cases. Often the settlement convention in question requires a special set of algorithms to be selected, and these algorithms will probably not function as intended in other combinations. A given set of algorithms need to be used as described and if used in other combination great caution is needed.

Following special algorithm sets are described here:
- receipt-reactive RTGS in chapter 5.2.1
- DVP linking of multiple transactions i.e. Group code algorithms in chapter 5.2.2

5.2.1 Receipt-reactive RTGS

The general idea of the receipt-reactive RTGS convention is that the participants can divert some part of the outgoing payment flow to a secondary queue. Using a predetermined time period (e.g. one minute, one hour) to cumulate the amount of incoming funds, this secondary queue releases payments whose amounts aggregate up to, but do not exceed, this total amount of incoming funds. Figure 1 contains a visual presentation of the dynamics in the receipt reactive model for two time periods.

This settlement convention is a type of liquidity management convention that functions independently from a participant's total liquidity balance. The secondary receipt-reactive queue complements the higher priority RTGS payment flow so that a participant's total liquidity balance never goes down under the starting balance because of the receipt-reactive queue's handling of its lower priority payments.

The receipt reactive model requires a set of three special algorithms: entry algorithm ENT/ENTDUAL1, settlement algorithm SET/SETDUAL1 and queue release algorithm for secondary queues QU2/QURRFIPR. A time period parameter is introduced in the first of the user-defined fields in the participant table (PART). This predetermines the amount of time during which incoming funds will be cumulated for the purpose of allowing payment release from the receipt-reactive queue. Note that there need to be a time period value for each participant using this feature. If this value is zero or there is no period value given, transactions for such participant stay in the secondary queue until it is closed.

The processing steps in the receipt reactive model can be described as follows

- in the entry phase the transactions can be divided into three streams:
  - immediately forced settlement of payments with highly urgent priority (i.e. settled even if these violate all limits),
  - normal transactions for RTGS processing including the normal primary queue and
  - low priority payments for the secondary queue during the open hours of the secondary queue
- in the settlement phase all normal RTGS algorithms can be called and in addition a QU2 algorithm for releasing transactions from the secondary queue
- in the queue release phase of the secondary queue the transactions which fit the positive net balance of that given period will be settled in priority and FIFO order,. The period in minutes can be defined separately for each participant. At the end of the period the remaining transactions in the secondary queue of that period can be moved up to the RTGS queue with their original priority or by giving them a new uniform priority or alternatively they can be discarded.

The entry algorithm ENTDUAL1 takes four parameters:

- Limit (0-9), which defines the value of priority required for entering the normal RTGS process. Transactions with a lower priority than the limit value are placed into the secondary queue.
- Priority (0-9) defines the minimum value of priority for highly urgent payments. Transactions with equal or higher priority are settled immediately even if they would violate any limit.

- Open (hhmmss) defines when the secondary queue is opened. All transactions entered before this point in time are treated as normal RTGS transactions.
- Close (hhmmss) defines when the secondary queue is closed. All transactions entered after this point in time are treated as normal RTGS transactions.

The SET algorithm SETDUAL1 takes no parameters, but can include a set of the normal RTGS sub-algorithms such as QUBYPAFI (see for example SEBASIC1). In addition, one QU2 algorithm can be specified for releasing transactions from the secondary queue.

The QU2 queue releasing algorithm, QURRFIPR, releases transactions from the secondary queue based on the period information provided in the P_USERCOD1 field of the PART table and it employs the following parameters:

- EOD ("gross" or "return"), which defines if unsettled secondary queue transactions are moved to the RTGS queue – the "gross" case – or if unsettled transactions are discarded i.e. returned to sender – the "return" case.
- NewPriority (0:9) is optional. It defines the value of priority given to the transactions moved via "gross" to the normal RTGS queue. If the parameter has not any value, transactions are moved with their original priorities.

The period parameter in the PART-table has an important function as it defines how the open hours of the secondary queue can be divided into sub-periods. The format for this parameter is hh:mm, for example 01:30. If incorrect format is used, period of 60 minutes is assumed and an error message is shown in the console. Within each sub-period the net received balance starts from zero and is calculated such that outgoing payments released by the secondary receipt-reactive queue are netted against all incoming payments. This net received balance must always be greater than or equal to zero. Payments sent via the normal RTGS process by certain participant do not affect the processing of payments in the secondary queue of this particular participant: only amount of funds received and payments sent from the secondary queue are relevant. For example if the period is defined to be 30 minutes, a participant can settle as many secondary queue transactions in FIFO order up to the aggregate amount of surplus received during the given 30 minute interval. This is not an end-of-period settlement, but payments can be released from the receipt-reactive queue all time during the period so long as the net received balance is non-negative. After the end of the period in our example 30 minutes, the net received balance is set to zero and the process begins again.

When the secondary queue closes by the end of the period, the EOD process will determine how unsettled transactions are treated. When value of EOD parameter is “gross”, the transactions from secondary queue are moved to primary queue with the NewPriority value and parameter value “return” will discard the transactions. For example if OPEN is defined as 10:15:00, CLOSE as 15:30:00 and the period as 30 minutes, the receipt-reactive calculation periods will be 10:15-10:44:59, 10:45:00-11:14:59,…,15:15:00-15:30:00.

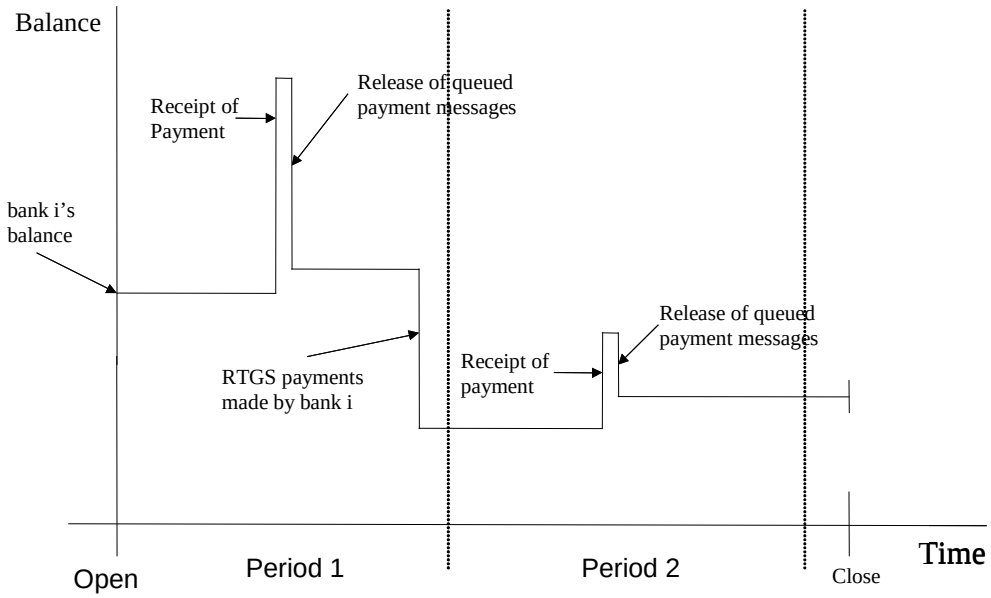


Figure 1 Dynamics of participant balance under receipt-reactive settlement

5.2.2 Group codes for DVP linking multiple transactions

Group code algorithms allow arbitrary many transactions to be tied together in a group, where none of the transactions in the group is settled unless all the transactions in that group can be settled simultaneously the same way as the algorithms supporting DVP and PVP.

Group codes for transactions are imported in the first user defined field of transaction data. For each group the number of transactions in that specific group is also required. This is imported also in transaction data, in second user defined field of each transaction with a group code.

Group code algorithms were created as user modules and are included in the simulator as normal algorithm modules. For linking transactions in a group

efficiently together they maintain their own data structure. This data structure is used by all group code algorithms and thus it is stored in the simulator engine, in first of the user object data handles reserved for user module data. Because of this, other user modules which utilise the user defined fields 1 or 2 of transaction data or the userObjectData1 handle in the engine should not be used together with group code algorithms to avoid data conflicts. User ObjectData handles are described in more detail in the algorithm descriptions and user module development guide document.

Following algorithms are included in the group code modules:

Type	Name	Parameters	Description
SUB	GCSUB	None	Performs normal submission algorithms tasks in priority FIFO order and in addition creates the data structure linking grouped transactions together and includes all transactions with group code in it.
ENT	GCENT1	None	Performs the basic entry processes of a specified transaction. Group codes are taken into account when direct settlement of transactions at the entry is tried.
QUE	GCQUBF1	None	Performs queue release continuously in bypass FIFO-logic taking into account the group codes.
PNS	GCQUBFI2	Interval (0-60) Starting time (hhmmss, optional) maxRound	This user module is implemented as PNS algorithm but it acts like a QUE algorithm: it releases from queue only individual transactions or groups tied together with one group code. Algorithm is executed for the first time at moment given with the optional second parameter or at beginning of day and after that at intervals given with the first parameter. Processing of the queue is performed in bypass FIFO mode: queue is ordered in Priority FIFO order, but if the first transaction cannot be settled, next one is tried and so on. When a settlable transaction is found, the search loop is continued instead of restarting from the first transaction. If settlable transactions are found during one loop, new round is started. During one execution of the algorithm the whole queue is worked through at most as many times as indicated with the third parameter maxRound.
PNS or MNS	GCPNST	Time;time;time;…;time (max 40 , 24 h HH:MM format)	This algorithm is not yet included in version 2.4.0 delivery. Performs batch runs on designated moments to settle a group of the queued payments with group codes. Implementation of this algorithm will be provided, where total value of transactions to be included in the group of settled transactions is maximized.

Type	Name	Parameters	Description
			Algorithm can be implemented as MNS so that it can be used together with PNS-queue release presented above.

SET algorithms and END algorithm from the normal RTGS algorithms should be used together with GC-modules. Other sub algorithms should not be mixed with GC-algorithms since these do not follow the conditions for settlement stated in the group codes.

5.3 System event handler algorithms (SEH)

This is a special type of algorithms that are used to adjust the basic rules of a simulated system. Rules here mean the way a system reacts to specific events occurring during a simulation. Each payment system can have its own set of rules according to which it reacts to different events occurring in a settlement process.

The events recognized by the simulator's default event handler are the following:
- Introduction of a new transaction
- Bilateral limit change
- Intraday Credit limit change
- Receipt reactive period start
- Receipt reactive period end
- Reaching the from time of a payment (PROCTYPE)
- Expiry of till time (PROCTYP2)
-End of day

The event handler contains also some common routines like the booking routine. Thus it contains the logics that follow "booking events".

The I algorithm is optional and if one is defined it will override the default behaviour.

Algorithms may also introduce new types of events, which will require new processing logics which can be introduced using I algorithms.

5.4 Time estimation algorithms (TEA)

Time estimation algorithms are used to estimate the time, a specific process or algorithm would have used in the real world. To do so, the time estimation algorithms can use variables such as the amount of transactions, amount of

iterations, used CPU time and lot of other possible variables related to a specific process for which time estimation is required.

The variables and parameters according to which a TEA –algorithm can calculate a time estimate will only depend on the properties and the implementation of the algorithm. Also the calling parent algorithm must be able to provide the required dynamic parameters to the TEA-algorithm. The parameters used to estimate time usage are the following:

- Dynamic parameters are the parameters the parent algorithm will provide to the TEA-algorithm. Dynamic parameter values are defined during simulation runs.

- Fixed parameters are defined in the system definition before the simulation, and they are coefficients of the estimation function.

To be able to attach a time estimation algorithm to a parent algorithm, all the dynamic parameters of the TEA-algorithm must be supported by the parent algorithm. The parent algorithm can support more dynamic parameters than the TEA-algorithm.

The following time estimation algorithmis are provided:

Type	Name	Parameters	Description
TEA	TEALGO1	Dynamic parameters: **x** = transaction count in the parent algorithm **y** = account (participant) count in the parent algorithm (e.g. in netting solution) **z** = actually used CPU time in milliseconds (varies, set relevant coefficients as 0 for environment independent results) Fixed parameters: a9, a8, a7, a6, a5, a4, a3, a2, a1, a0, b1	Time estimation function: $b1\ (a0 + a1\ x + a2\ y + a3\ z + a4\ x^2 + a5\ y^2 + a6\ z^2 + a7\ x\ y + a8\ y\ z + a9\ x\ z)$

New TEA algorithms can be introduced as user modules similarly as any other algorithms.

5.5 Agent based modelling (ABM) algorithms

Note!
The GUI of the Beta version of PSS3 does not support importing or definening ABM simulations well. Users would need to define setups directly into the DB. Even then the functioning is only partly tested with promising results though.

The ABM algorithms contain proactive decision making rules for actors or agents included in a simulation. Their main function is to incorporate behavioural rules for entities involved in the simulations.

We understand ABM as a computer modelling or simulation technique in which we replicate algorithmically the behaviour of some actors and allow them to interact in computer simulations as independent agents. ABM is essentially a computer simulation modelling technique which affects the architecture and structuring of the simulation software and code allowing AI logics to interact in the simulation world.

ABM's can be seen as micromodels in which macro level inference is made out of the outcome of micro agent's interactions. The fact that ABMs are implemented at microlevel, ABMs are also likely to bring in more realism.

For more on Agent based modelling in general, please refer to other sources, there are many of them. As a hint be carefull with the sources related to economics and favour articles of som other fields. It seems that in some economic papers ABM is slightly misunderstood to simply mean the incorporation of behavioural rules into traditional economic analytical models. The application of ABM into economics is still at its dawn.

5.5.1 Basic functioning

With the ABM configuration file, accounts are associated with a specific bank logic (or agent implementation). These agent logics, in programming terms, extend a class called Bank which contains some basic implementations of some basic features. The available basic implementations are called `CautiousBank` and `CustomerDiscriminatingBank` in addition to the `ExampleBank`. Human readable code for all of these can be found in the modules directory and it includes extensive

comments that explains the functioning and the structure. Studying the code is highly recommended before using the ABM algorithms.

When an agent implementation is configured and connected to an account in a simulation, this is done by giving the agent implementation full control of the payments where this account is the sender. The payments are no longer submitted to the simulations' payment system directly but instead they are moved into a separate transaction queue owned and handled by the ABM agent implementation.

The agent implementation can decide whether and when to send payments to the payment system just like in the real world and also create new events or transactions in the simulation based on the triggers in the data and status of the simulation. How complex the AI inside the agent is depends on the algorithm implementation.

5.5.2 When and how are agents activated?

Agents are activated with agent wakeup events. These wake up events are stored in the simulations basic event queue with all other events like introduction of (non ABM) transactions, credit limit change, end of day etc. Wakeup events are used in different situations as described below.

When the agents are activated, they can schedule and create wake up events for themselves for any future time. If the agent is aware of the account of another agent, it is able to schedule a wake up event also to another agent. The bank interface includes methods for placing the created wakeup events to the systems event queue.

5.5.2.1 Simulation initialisation

When the transactions of an account with agent code are separated into the agents own queue, a wakeup event is generated for each transaction to the simulation's general event queue. These wakeups include a reference to the related transaction and have the same time label as the given transaction.

In the initialization in the beginning of the simulation, the possibility to create new events can be used to establish fixed times known in advance when the agent needs to be active.

5.5.2.2 Processing of a wake up event

A wake up event triggers a call of the agent instance, which is related to that given event and allows it to perform actions or decisions based on the situation in the simulation. The processing of the wake up event can be separated based on system event types. Default options available for the ABM related event types include AGENT_WAKEUP, AGENT_ALERT_END_OF_DAY, AGENT_END_OF_DAY and AGENT_TRANSCATION_HAS_SETTLED. Users can define also own event types if needed. These should not overlap with the values already defined in the SystemEvent class.

If the wake up event is related to a transaction, this is the moment when the transaction was originally send to the settlement system in production. The agent can decide what action it performs on the given transaction. Note however, that ABM code can decide to submit transactions which are mandated to it also during any other wake up call. Thus transactions can be submitted later or earlier than what was their original submission time in the input data. For transaction submission see ch 5.5.2.3.

Agents can also schedule them selves new wakeup events. The agent can create a wake up event for itself with method `addToQueue` e.g.:

```
SystemEvent event = createWakeupEvent(long currentSimulationDate, long time);
addToQueue(event);
```

5.5.2.3 Sending a payment for settlement

When an agent is activated it can decide to send payments for settlement. For this purpose 3 methods are available:

```
sendPaymentForSettlement(transaction)
```

This method removes the transaction from the agents own queue and puts it on the top of the simulations submission queue from where the transaction will be sent to the payment system. The system sets the submission time of the transaction to be equal to the current simulation time. If there is a linked wakeup event for that transaction, it is removed from the simulations event queue.

```
sendPaymentForSettlement(transaction, boolean)
```

Other wise same as above with the exception that by setting the Boolean parameter to false, the method will leave a possible linked weakup event in the simulation's event queue.

```
sendPaymentsForSettlement(SystemEvent, String)
```

This method is used when the agent wants to send all the rest of it's payments from its local transaction queue for settlement. A typical time to invoke this method would be when the agent receives the `SystemEvent.AGENT_ALERT_END_OF_DAY` or `SystemEvent.AGENT_END_OF_DAY` system event. The SystemEvent parameter is only used for logging purposes.The String parameter defines the submission origin of the payment that is stored in `TEST` statistics.

5.5.2.4 Transaction booking

When transactions are booked and a SCM (Settlement Confirmation Messenger) algorithm is defined in the system setup, agent wakeup calls will be added to the simulation's event queue according to the algorithmic rules of the selected SCM algorithm.

The basic implementation, called SECOMSGR, will add agent wake up events to the simulations's event queue for both the credit and debit accounts of each settled transaction in case both credit and debit accounts have a agent configured. The wake up calls are scheduled for the settlement time of the settled transaction.

The bank agent will receive the settlement wake up events in it's process method's `switch case` structure hooked for system event type `SystemEvent.AGENT_TRANSACTION_HAS_SETTLED`.

See the example ABM code in the annex.

5.5.2.5 End of day calls

Separate wake up event is sent to the agents when the simulated day ends. In addition the bank agent can invoke a system event to it self as an alert when the business day is about to end. The timing of these is controlled by system definition and the agent parameter values.

It is noted that as a default, the simulated days are treated completely separately in the ABM thus an agent cannot send a system event to some other business day.
If the agent needs to pass on any information from one day to another, then the user must implement this transfer e.g. via file and make sure that the simulation is executed in a sequential mode for the simulated days.

5.5.3 Account management AI algorithms

Below is listed the parameters of `CautiousBank` and `CustomerDiscriminatingBank` agents:

Name	Parameters	Description
CautiousBank	**alertsSecondsBeforeEndOfDayEvent:** in seconds **onPercentage:** decimal **offPercentage:** decimal **delayTypes:** list of transaction types separated with a blank **logEvents:** true or false	Agent switches from normal state to cautious if the balance falls below "onPercentage" parameter value. There is possibility to define hysteresis, where returning to normal requires higher balance (offPercentage) than the triggering of cautious mode. When the agent is cautious, transaction classes listed in the "delayTypes" parameter are postponed and not submitted to the system until the agent returns to normal state. Postponed transactions are stored in a separate internal queue, which has priority FIFO order. When the end of day approaches (alertSeconds…) and when it arrives all payments from the internal queue are submitted.
CustomerDiscriminatingBank	**latestTimeToPostponePayments**: in microseconds **alertsSecondsBeforeEndOfDayEvent**: in seconds **delayMaxNormal**: in microseconds **delayMinNormal**: in microseconds **liqLowNormal:** decimal number **liqHighNormal:** decimal number	The agent represents direct participant which may postpone the pass through of payments coming from its indirect counterparties. These payments are identified based on Usercode_1 field. It should contain a value where the first 8

Name	Parameters	Description
	priorityThresholdUrgent: urgency values like 1 to 9 depending on the data used **delayMaxUrgent**: in microseconds **delayMinUrgent:** in microseconds **liqLowUrgent:** in microseconds **liqHighUrgent:** decimal number **priorityThresholdHighlyUrgent:** decimal number **delayMaxHighlyUrgent:** in microseconds **delayMinHighlyUrgent:** in microseconds **liqLowHighlyUrgent:** decimal number **liqHighHighlyUrgent:** decimal number **defaultDelay:** in microseconds **createWakeUpEventsOnMaxDelay:** true, false **logEvents:** true**,** false;	characters differ from the direct participants ID for a payment to become delayed. This serves as a proxy for implying the behaviour only on a counterparties which are not in the same banking group. Minimum and maximum delay can be defined separately for payments above different priority thresholds. Also the liquidity levels, which trigger delaying are defined separately for different priority groups. The delay is varied between the extreme values in inverse linear manner based on the liquidity position of the account.
MSC: SettlementConfirmationMessenger		Is activated when a settlement system books a transaction. Agent wake up calls are added to the simulation's event queue for the debit and credit accounts of all transactions settled through the booking process. The algorithm can be modified to trigger other sorts of reactions.

The name of the algorithms in the ABM simulation configuration file relect the actual Java implementation class names. The above parameter names are defined and used inside the ABM algorithms. Users can modify the set of parameters available in a given ABM algorithm by introducing new parameters or discarding the ones which are unnecessary for their implementation.

The configuration file used to set up ABM simulations follow the syntax presented in the ANNEX 2. The property file allows the user to define which ABM algorithms are in use by which participant and with which parameters.

With the property file tag `agents=` it is possible to declare which ABM algorithms are in use. The syntax is the following for declaring the algorithms i.e. your ABM agent modules.

In below example seting we'll setup the CautiousBank and CustomerDiscriminatingBank algorithms:

```
agents=CautiousBank;CustomerDiscriminatingBank
```

The semicolon is used to separate the algorithms.

To declare common global parameters for participants using the same ABM logics the following syntax is used:

```
[agent class name]Settings= \
[parameter 1]=[parameter value]; \
[parameter 2]=[parameter value]; \
…
[parameter 3]=[parameter value];
```

Here the first row indicates for wich ABM algorithm the common parameters are defined.

Please notice that the characters `\` at end of a line means that the line is continuing to the next line.

Example of an algorithm specific setting:

```
CautiousBankSettings= \
alertsSecondsBeforeEndOfDayEvent=1800; \
onPercentage=0.2; \
offPercentage=0.4; \
delayTypes=1.1 1.2; \
logEvents=true;
```

To associate participants to use a specific ABM logic, the following syntax is used:

```
ABMclassfilenamexParticipants= \
[system id]-[participant id]-[account id]; \
[system id]-[participant id]-[account id]
```

The `system id` is defined in your simulator database `part` table.

You can use the below query to help finding out the your system-participat-account candidates you can use as values in above setting.

```
SELECT DISTINCT
  CONCAT(t1.system_id,
  CONCAT('-' ,
  CONCAT(t1.from_particid,
  CONCAT('-' , t1.from_accounid))))
```

```
FROM tran_view_all t1;
```

Below is an example of a algorithm specific participants setting:

```
CautiousBankParticipants= \
1-NDEAFIHH-NDEAFIHH00000000000000001; \
1-DABAFIHH-DABAFIHH00000000000000001;
```

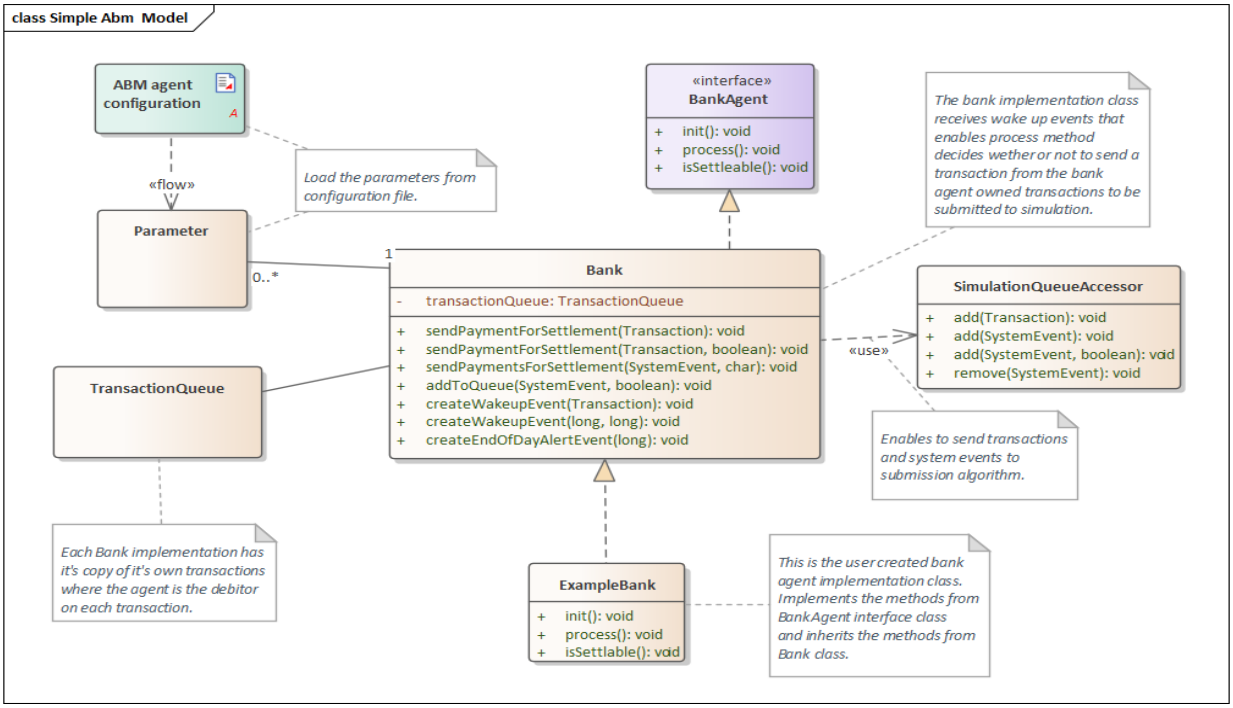
Participant specific agent paramaters are defined by first declaring the algorithm and account identifiers and then after the paramaters on their own rows in the following way:

```
ABMclassfilenamexSystemID-participant-id-account_id= \
parametername1= wanted value; \
…
```

5.5.4 ABM Object model

In below diagram the `ExampleBank` class is an implementation of a bank agent which you can use as a refenrence as you start to create your own ABM bank agent implementation.

ABM bank agent implementations are created as java classes that extend the Bank class which in turn implements the BankAgent interface.



Please notice that all the other classes are existing simulator classes . Atleast in simple ABM implementations all the logic can be written to the new Bank child class. The methods listed for the `Bank` class are intended to be used when the derived bank agent class needs to communicate with the simulator's settlement process.

The `sendPayment[s]ForSettlement` methods are used as the agent decides to release one or more transactions from its own queue of transactions.
The `addToQueue(SystemEvent)` method enables the agent to send a system event to simulator's event queue to be invoked at given time of business day.
You can create wake up system events with the `createWakeupEvent` and `createEndOfDayAlertEvent` methods.

For more details see the `ExampleBank` implementation in annex.

5.6 User module interface

Disclamer!
The PSS3 Beta version GUI does not support importing of user modules. It is possible to define user modules by entering appropriate information into the database.

Adding user modules gives you the possibility to create your own settlement algorithms and processing conventions. See **User module development guide** for details. The easiest way to develop user modules is by copying relevant parts from an existing algorithm and inserting the desired modifications.

5.6.1 Adding a user module

Before you can add a user module to the simulator, you first have to compile it from the Java code file (.java) with a Java compiler to a class file (.class). You can accomplish this with Sun's javac. The simulator's main JAR-file BoF-PSS2.jar must be in included class path while compiling. For detailed instructions, see the document *BoF-PSS2 Algorithm Descriptions And User Module Development Guide*.

On the **User module definition** screen:

1. Type in the name of the user module (max. 8 characters).
2. Click the **Browse** button and select the Java class file of the module.
3. Select a user module type from the drop-down list.
4. Select the system types this algorithm is available for (RTGS, CNS, DNS).

5. If the module can accept parameters in the system data definition stage, you have to define the parameter names and types. Type in the name of the in the edit box, and select a checking rule from the drop-down list. Add the parameter and the checking rule for the user module by pressing the **Add parameter** button. Repeat the procedure for each parameter required.

   If you want to delete a parameter from the user module, select it from the table and click the **Delete parameter** button.
6. Click the Save definition button. The algorithm should now be selectable in the system data definition screen.

6 Data content and databases

6.1 File directory structure

The simulator has a file directory structure that is partly built by the setup program and partly by the application based on users' project specifications.

The **setup program creates** the following directories:

- MODULES (contains built-in modules' / algorithms' code files),
- PROGRAM (contains sub-directory JDK for Java Run Environment, necessary script files and the simulator `simulator-api.war` file),
- USERMODULES (for user defined modules), and
- EXAMPLES (The file ex#_description.txt contains information about the specific simulation example).

The **application creates** the directory pss2_systemdb (for the system database) when the first project is defined. It also creates a directory in the BoF-PSS directory for each project using its name preceded by "`P_`" and the following sub-directories:

- ERRORLIST (for error lists)
- INPUT (for input files)
- OUTPUT (for output files)
- OUTPUT_REPORTS (for output reports)
- NETWORKS (for generated networks)
- NETWORK_REPORTS (for network analysis results)
- TEMP for temporary storage (all files in TEMP can be destroyed when the simulator is not in use)

If you are going to make a large number of simulations analysing different aspects, it can be good to organise them in separate projects

Here is an example of a directory structure:

```
C:\BoF-PSS\EXAMPLES
C:\BoF-PSS\EXAMPLES\DECIMAL_POINT
C:\BoF-PSS\PROGRAM
C:\BoF-PSS\PROGRAM\filters
C:\BoF-PSS\PROGRAM\JDK
C:\BoF-PSS\PROGRAM\log
C:\BoF-PSS\P_proj1\ERRORLIST
C:\BoF-PSS\P_proj1\INPUT
C:\BoF-PSS\P_proj1\NETWORKS
C:\BoF-PSS\P_proj1\NETWORK_REPORTS
C:\BoF-PSS\P_proj1\OUTPUT
C:\BoF-PSS\P_proj1\OUTPUT_REPORTS
```

6.2 Database files and locations

There are 2 types of databases: one system database and project specific databases. The simulator has a common system database, which is created during the first session with the program. With some database versions the database files have to be saved into the database's data folder. For MariaDB versions the simulator doesn't allow the user to define freely the saving location. Otherwise, the program creates the directory `pss2_systemdb` in the installation directory. Project databases are created automatically for each project defined by the user.

`C:\BoF-PSS` is the default directory for the simulator, see Figure 2.

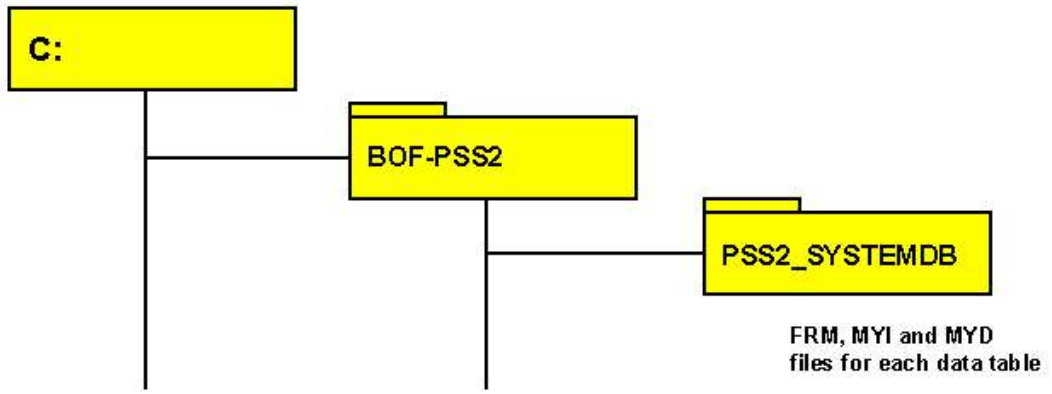


Figure 2.

The simulator uses the MyISAM storage engine. With this setup, each database has three separate files for each database table. They are:

– data dictionary information (.FRM files)

– index information (.MYI files), and
– data files (.MYD files).

MariaDB 10 stores all databases into the same data folder which is `[database installation folder]\data` folder of the database engine.

The name of the project directory by default is identical with the project name with `P_` prefix.

6.3 Data sets

One input database can store many data sets for each type of input data. The different data sets are stored in the same physical database table and are distinguished by their data set ID. The user defines the data set ID separately for each data table; it has no internal database relation with any other data set ID of other database tables.

In the simulation execution phase, the user defines which specific data sets are to be used in a specific simulation as described in the Figure 3. These are cross-checked to see that the information is coherent, e.g. all accounts or participants can be found for the transactions, and all systems are specified the account or participant to which the transaction data refer.

To manage a large number of parallel data sets, a consistent naming convention is a good idea.

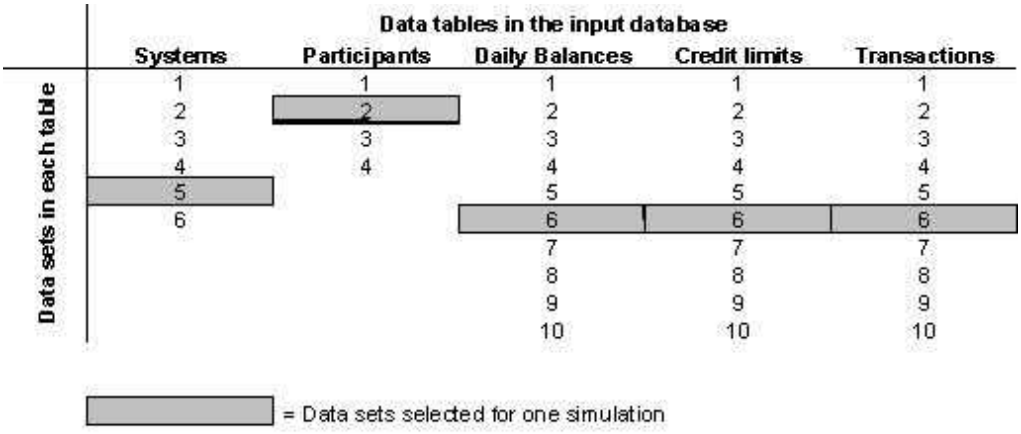


Figure 3.

6.4 About MariaDB

MariaDB is a popular and efficient open source database product with good documentation and a good reputation. Information about MariaDB can be found on website https://mariadb.org. An online reference manual is available and it can also be downloaded from the site. Advanced simulator users can make their own database retrieval procedures directly to the databases as SQL queries. Also Java and C++ connectors are available, as well as a general ODBC connector that can be used with e.g. MS Access or Visual Basic.

There are free and easy to use tools available with graphical user interface for browsing and monitoring the database structure and viewing data contents of the tables with simple queries. These tools can also be used to make small manual editions or deletions in the database that can be helpful in advanced use of the simulator. Examples of such operations include deletions of unnecessary templates, user modules or projects. Because user friendly tools are available for this, no special user interfaces have been included in the simulator.

Below two practical tools for direct use of database are presented: HeidiSQL database browser (6.4.1) and MyODBC (6.4.2). Under separate topic (6.4.3) there are instructions how these tools can be utilized.

6.4.1 HeidiSQL database browser

HeidiSQL database browser is a free visual tool, which can be used to browse the structure of a database and its data contents, and build and execute SQL queries. Queries can also be generated graphically and contents of the databases can be modified manually if necessary.

Video tutorials are recommended as a quick start reference. For example the Edit queries –tutorial shows how to make manual changes in table contents, which can be necessary as simulator database maintenance work. Instructions for these are given in chapter 6.4.3.

The first time HeidiSQL is used you need to identify the database you are connecting to:

- In the start-up window of database Browser, use
  - “localhost” as server host value
  - 3306 as port value
  - “root” as user name

- Or other settings according your own hardware setup (contact your local IT personnel if proposed settings don't work)

Database browser can be used to build and execute SQL queries e.g. to export data from simulator databases into files. Below some example queries are given.

Exporting database tables are executed by the **SELECT INTO OUTFILE** command.

For example, to export a database table **to a CSV file**:

SELECT * INTO OUTFILE 'c/temp/partfile.csv' FIELDS TERMINATED BY ';' FROM PART WHERE P_DATSETID='ds1';

Similarly, a query can be built in database Browser and the result set can be exported with menu functions after right clicking the result set.

6.4.2 ODBC interface

ODBC is a standard interface, which enables connections to a database from an analysis software working on Microsoft Windows platform and using large data sets. Examples of applications that can be connected are Access, Excel and SAS.

After installing an ODBC driver for MariaDB, new data sources are defined from Windows control panel / Administrative tools / Data sources. Own data source name needs to be defined for each database, which is to be accessed from the third party software.

https://mariadb.com/kb/en/about-mariadb-connector-odbc/#connection-parameters,

6.4.3 Direct modifications of simulator database

All features that a simulator user might want to have are not included in the simulators graphical user interface. Some of these tasks can be performed by directly accessing the database. These are mainly deletions of instances that can be created in the simulator but not removed if they turn out to be useless such as user modules or import templates. Below are listed some possible maintenance tasks and how to perform them with database browser tool.

Caution is always needed when direct modifications are made in the database. Backup copies and use of graphical tool, such as database browser, which shows visually the changes and allows undoing are recommended.

Task	How to do in Query Browser	Notes
Delete unused import or output **templates**	- Open system database and TEMP table - Click “edit” from bottom toolbar - Right click the row with the template to be deleted. First column contains the template names. - Select “delete row” from the menu - Click “Apply changes” from bottom toolbar	Example of Query browser view is shown below. Table contains also the built in ALL- and EMPTY-templates
Delete **user modules**	- Open system db and ALDE table - Select and delete the line with name of the unused module in the first column as was explained above.	User modules are in the end of the table. Don’t delete the references to built in algorithms.
Delete an unnecessary **project**	- Open System database and PROJ table. - Select and delete the line with name of the unnecessary project in the first column as was explained above. -The project folder has to be deleted manually.	Do not delete project that was active when simulator was last time running. If this is done, SD_PROJEID field in DEFA-table of system database has to be also manually altered before simulator can start again.

7 Description of database tables

The simulator uses to types of databases. A simulator installation has one system database and one separate project database for each project. All the database tables are listed under with descriptions of all of their fields. The first column indicates the variable name of the field. The second column indicates the type of field. The third column indicates whether the field is a foreign (F) key, a primary (P) key or

just a key (K). The fourth column gives the detailed name of the field. The fifth column describes the field. The last column indicates if the data is optional (O) or mandatory (M).

As the database has been heavily rationalised for the version 800, there are a lot of changes compared to old database tables. To help old and new users, views have been created to combine information that used to be in one table before. An example is the test_view_all that combines information form TRAN, tran_generated_by_simulation and the result table TEST. Prio to version 800, the test table used to have all the same information as the TRAN table as duplicate. Now same information is not any more copied to output tables, but they need to be combined in the SQL-queries. The new view tables named like xxx_view_all do that.

Another example is the account and participant data. Basic participant and account information is not anymore duplicated to all tables but is stores only to the PART table. Other tables refer to account information only with technical id's anymore.

7.1 System database

7.1.1 Defaults [DEFA]

Contains default information for projects.

Field name	Data type	Key	Description	
SD_DEFAUID	CHAR(1)	P	Value = 1, just to introduce the mandatory key.	M
SD_PROJEID	CHAR(8)		Current project ID.	M
SD_SEPARAT	CHAR(1)		Separator used between data fields in CSV files.	M
SD_DECIMAL	CHAR(1)		Decimal point format.	M
SD_TIMEFOR	CHAR(15)		Time format.	M
SD_DATEFOR	CHAR(10)		Date format.	M
SD_TRAVALU	CHAR(5)		+/-HH:MM value to change input time	

7.1.2 Project [PROJ]

Contains data of all projects.

Field name	Data type	Key	Description	
id	INT	P	Primary id	M
SP_PROJEID	CHAR(30)	P	Identifier for the project.	M
SP_DATABASE	VARCHAR(255)		Directory of the project's database.	M
SP_ICSVDIR	VARCHAR(255)		Directory for input file.	M
SP_OCSVDIR	VARCHAR(255)		Directory for output file.	M

SP_ERRORDIR	VARCHAR(255)		Directory for error list.	M
SP_OREPDIR	VARCHAR(255)		Directory for output reports.	M
SP_NETDIR	VARCHAR(255		Default location for created networks	M
SP_NETREPORTS	VARCHAR(255		Default location for generated network reports	M
created	TIMESTAMP			
modified	TIMESTAMP			

7.1.3 Algorithm definition [ALDE]

Contains information of user modules and algorithms.

Field name	Data type	Key	Description	
SA_ALGORID	CHAR(8)	P	Unique identifier of the algorithm.	M
SA_MODFILE	VARCHAR(255)		Name of module file.	M
SA_MODTYPE	CHAR(3)		Type of module.	M
SA_PARAMET	TEXT		Enumerated list of parameters and checking rules used by algorithm.	O
SA_SYSCODE	TINYINT(4)		Describes in which system the algorithm is available. Values are additive 8=RTGS, 4= CNS and 2=DNS e.g. 12 indicates availability in RTGS and CNS systems.	M

7.1.4 Template [TEMP]

Contains template data of input files.

Field name	Data type	Key	Description	
ST_TEMPLID	CHAR(8)	P	Unique identifier of the template.	M
ST_TABTYPE	CHAR(4)	P	Refers to specific data set table.	M
ST_SKPROWB	SMALLINT (3)		Number of rows to skip in the beginning.	O
ST_SKPROWE	SMALLINT (3)		Number of rows to skip in the end.	O
ST_TEMPLAT	TEXT		Enumerated list of column numbers, which describes the column structure of the CSV file. Value in form: DB-Column1 match column in CSV file, DB-Column2 match column in CSV file ,…	M

7.1.5 Database version [db_version]

The table contains information on the simulator version by whitch it has been created.

Field name	Data type	Key	Description	
db_name	VARCHAR(64)		Unique identifier of the project.	M
version	VARCHAR(4)		Unique identifier of acceptable system.	M
detected	DATETIME		Date and time, when the version of the databse has been identified	

7.1.6 Acceptable system Ids [ASID]

Note! Not in use.

Field name	Data type	Key	Description	
SY_PROJEID	CHAR(8)	P	Unique identifier of the project.	M
SY_SYSIDTB	CHAR(8)	P	Unique identifier of acceptable system.	M

7.2 Project's input data tables

To minimize the size of the database, the storing of duplicate information is avoided. The information used in results already present in the input data is no longer copied to the output tables.

In practice participant's account information is written only once in to the participant table. Also transactions basic information is only written once in the tran table. Only new information created during simulations are written during simulations.

When data is imported, it is anchored to a selected participant dataset. This dataset acts as a reference dataset. During the import technical id's are generated to make the linking of data tables more efficient. More precisely, the technical account reference is stored to the *part_id* field of the PART table. All the other input and output data use this id to reference the account information stored only in to the PART table. The *part_id* is not unique and must be referenced together with the corresponding part dataset id stored either with the *part_id* in the input tables like TRAN, ICCL and DBAL or in the SIRI table containing the simulation definitions for the simulation results. Take note that the the technical autoincrement *id* of the PART table is not used for the referencing.

The dataset table contains also the information on which datasets use the same or compatible technical account referencing. Compatibility is indicated by the field *account_id_reference_group*.

7.2.1 System [system]

Contains information on systems belonging to the project.

Field name	Data type	Key	Description	
id	INT Autoincrement	P	Technical id of a system	M
name	TINYINT(3)		Unique name of a system	M
modified	Timestamp		Timing of last modification	M

7.2.2 Dataset [dataset]

This table contains basic information of all imported and created datasets. This table also defines which input dataset use the same technical account references and are compatible from this perspective.

Field name	Data type	Key	Description	
id	INT Autoincrement	P	Technical id of a dataset. In version 921 not yet used for referencing	M
system_id	TINYINT(3)		Reference to id in SYSTEM table	M
name	VARCHAR(50)	K	The user defined data set name. In the version 921, this field is still used for referencing	M
entity	VARCHAR(50)		Type of the dataset	O
account_id_reference_group	TINYINT(4)		An Id of a PART table dataset. This field tells which participant data set the data refers to. This means that the account id's used in the dataset are the same as in the here defined participant dataset. When datasets have the same account_id_reference_group value, they are compatible between each other and use the same account id references. Omitted when the data does not refer to participant info	O
description	VARCHAR(120)		Description of dataset.	O
created	Timestamp		Time of creation	M
modified	Timestamp		Timing of last modification	M

7.2.3 System setupdataset [SYCD]

Contains system specificationsfor a specific system, for example system ID, name, type, and open hours. There can be many system datasets for one system.

Field name	Data type	K	Description	
id	INT Autoincrement	P	Technical id of a system.	M
system_id	TINYINT(3)		Reference to id in SYSTEM table	M
S_DATSETID	CHAR(8)	K	Unique identifier of the data set to distinguish the data set from other parallel data sets used in simulations.	M
S_FULLNAME	VARCHAR(20)		Full name of system.	O
S_SHORTACR	CHAR(5)		Short acronym for system.	O
S_DESCRIPT	VARCHAR(120)		Description of system.	O
S_SYSTTYPE	CHAR(4)		Possible system types are RTGS, CNS and DNS. CNS and DNS systems typically settle their end-of day net-positions. RTGS systems and sometimes CNS systems may have intraday liquidity injections from an RTGS system. The DNS system settles transactions at specified settlement occasions on a batch net basis, while CNS systems settle continuously at the transaction level. This selection only affects the list of algorithms made available in the GUI for algorithm selection. This does not affect the simulations it selves…	M

			Possible values: RTGS DNS CNS	
S_NOCRELIM	TINYINT(4)		Gives the opportunity to specify that all participants or accounts in a system have infinite credit limits. Mainly used for DNS systems, but may also be used for RTGS and CNS systems to determine maximum liquidity requirements. Possible values: 1: Credits according to limit table 2: No Credits available 3: Credits available without limits	O
S_TRANSBAL	TINYINT(1)		Transfer of end-of-day balance to the next day. Mainly used in RTGS systems. Possible values: 0: Balaces are not transferred 1: Balances are transferred	O
S_TRANSTRA	TINYINT(4)		At the end of the day, there may be unprocessed transactions in the RTGS and CNS queues. These can be eliminated or transferred to the next day. In the DNS system, the choice is to eliminate or transfer unprocessed transactions to the next settlement occasion. Possible values: 0: unsettled payments are not transferred 1: unsettled payments are transferred	O
S_OPENTIME	INT		Defines the time from which transactions will be submitted to the system in the beginning of the day. Transactions that have an earlier submission time will wait until the open time point is reached.	M
S_CLOSETIM	INT		Transactions with submission times after the closing time of the system will be submitted at the beginning of the following day.	M
S_BILIMUSE	TINYINT(4)		Values: 0 denotes that bilateral llimits are not in use 1 denotes that bilateral limits are in use	
modified	TIMESTAMP		Last modified time stamp	M

7.2.4 Participant data table [PART]

Contains participant data for a given system. Participants can be distinguished at two levels. The participant ID can be 11 characters long and can contain, e.g. a SWIFT BIC address. The account ID can be 34 characters long and can contain an IBAN. Both fields can also be used for other identifiers, e.g. in securities settlement systems, the account ID could be the ISIN code. The participant and account IDs are mandatory.

Field name	**Data type**	**Key**	**Description**	
id	INT auto increment	P	Primary id identifying uniquely p_datsetid, system_id, p_particid, p_accounid	M
part_id	INT(10)	K	Technical account reference used from other tables to refer to part table's accounts. 2 part datasets can have the same references. This id is generated during import automatically. It is possible to use an existing indexing with the selection box in the import GUI. The import facilities	

			can create a part dataset from other input datasets if an existing part dataset is not available.	
system_id	TINYINT(3)	K	Reference to id in SYSTEM table	
P_DATSETID	CHAR(8)	K	Unique data set identifier distinguishes this data set from other parallel data sets used in simulations. This is a reference to the name field in the dataset table	M
P_PARTICID	CHAR(11)		Textual identifier for the participant used in the initial input data.	M
P_ACCOUNID	VARCHAR(34)		Textual identifier of account in which credits and debits are made. The value is imported from input data files.	M
P_ACCOTYPE	VARCHAR(1)		Used to distinguish different types of accounts.	O
P_FULLNAME	VARCHAR(35)		Full name of participant.	O
P_SHORTACR	CHAR(5)		Acronym for full name of participant. The acronym is used in the run-time view of the simulator, if available.	O
P_ACCONAME	CHAR(10)		Name of account, e.g. “Euro RTGS account.”	O
P_SETINSYS	CHAR(8)		For DNS or CNS systems, the ID of the system where proceedings are booked. May also be used in RTGS systems for transferring end-of-day positions from sub-systems or accounts to main systems or accounts.	O
P_SETONPAR	CHAR(11)		For DNS or CNS systems, the ID of the participant to whom the end-of-day proceedings are booked. May also be used in RTGS systems for transferring end-of-day positions from sub-systems or accounts to main systems or accounts.	O
P_SETONACC	CHAR(34)		For DNS or CNS systems, the ID of the account in which the end-of-day proceedings are booked. May also be used in RTGS systems for transferring end-of-day positions from sub-systems or accounts to main systems or accounts.	O
P_LIQFRSYS	CHAR(8)		For CNS systems, the ID of the system to and from which liquidity injections are booked. May also be used in RTGS systems for transferring liquidity to and from sub-systems or accounts from and to main systems or accounts.	O
P_LIQFRPAR	CHAR(11)		For CNS systems, the ID of the participant to and from which liquidity injections are booked. May also be used in RTGS systems for transferring liquidity to and from sub-systems or accounts from and to main systems or accounts.	O
P_LIQFRACC	CHAR(34)		For CNS systems, the ID of the account to and from which liquidity injections are booked. May also be used in RTGS systems for transferring liquidity to and from sub-systems or accounts from and to main systems or accounts.	O
P_LIQINJVA	DECIMAL (20,2)		When specified, the injection value overrides any system-level value.	O
P_USERCOD1...5	VARCHAR(16)		Five optional fields where user-defined information can be stored for use by user-defined algorithms during simulations or in analysis of simulation output.	O
modified	TIMESTAMP		Last modified time stamp	

7.2.5 Daily balances table [DBAL]

Contains daily opening balances for the participants in the PART table.

Field name	**Data type**	**Key**	**Description**	
id	INT Auto increment	P	Primary id	M
B_DATSETID	CHAR(8)	K	Unique identifier of data set to distinguish this data set from other parallel data sets used in simulations.	M
SYSTEMID	TINYTINT	K	Reference to id in SYSTEM table	M
part _dataset_id	CHAR(8)	K	Reference to the datset id of the correspondgin part dataset	
PART_ID	INT	K	Reference to PART table field part_id. To be used togehther with the part_dataset_id	M

B_DATEEFFE	INT	K	Date opening balance is effective.	M
B_NEWVALUE	DECIMAL (20,2)		Value of opening balance.	M
B_USERCOD1 …5	VARCHAR(16)		Five optional fields where user-defined information can be stored for use by user-defined algorithms during simulations or in analysis of simulation output.	O
modified	TIMESTAMP		Last modified time stamp	

7.2.6 Intraday changes in credit limit [ICCL]

ICCL Intraday credit limits data
Contains information of original values and changes in intraday credit limits for participants specified in the PART table.

Field name	Data type	Key	Description	
id	INT(10)	P	Primary id	M
I_DATSETID	CHAR(8)	K	Unique identifier of data set to distinguish this data set from other parallel data sets used in simulations.	M
system_id	TINYINT(3)	K	Identifier of system.	M
part_dataset_id	CHAR(8)	K	Reference to the datset id of the correspondgin part dataset	
part_id	INT	F	Reference to PART table field part_id. To be used togehther with the part_dataset_id	M
I_DATEEFFE	INT (11)		Date from which new credit limit is effective.	O
I_TIMEEFFE	BIGINT (20,2)		Time from which new credit limit is effective.	O
I_NEWVALUE	DECIMAL (20,2)		Value of new credit limit.	M
I_USERCOD1 …5	VARCHAR(16)		Five optional fields where user-defined information can be stored for use by user-defined algorithms during simulations or in analysis of simulation output.	O
business_day	CHAR(8)		Business day to which the limit value belongs to. This value is used to determine the ICCL events to be included to the corresponding simulation day having the same business day.	M
Transaction_link_id	VARCHAR(30)		Id of the transaction linked to this credit limit order.	
order_type	VARCHAR(10)			O
entry_date	INTEGER		Date when the order becomes visible to the system	
entry_time	BIGINT(12)		Time when the order becomes visible to the system	
revocation_date	INTEGER		Day when the iccl order is revoked	
revocation_time	BIGINT(12)		Time when the iccl order is revoked	
id	BIGINT(20)		Unique identifier of the iccl order. It acts as a link to the output table.	
modified	TIMESTAMP		Last modified time stamp	

7.2.7 Bilateral limit table [BLIM]

Contains information of the original value and changes of bilateral limit values for given pairs of accounts specified in PART table. BLIM table can also be used to define multilateral limits i.e. limits for transactions between one participant and all the others in simulated system.

Field name	Data type	Key	Description	

id	ITN(10)	P	Primary id	
L_DATSETID	CHAR(8)	K	Unique identifier of data set to distinguish this data set from other parallel data sets used in simulations.	M
L_system_id	CHAR(8)	K	Reference to id in SYSTEM table. Identifier of the system this dataset belongs to.	
from_part_dataset_id	CHAR(8)	K	Reference to the datset id of the correspondgin part dataset	
from_part_id	INT(11)	F	Reference to PART table field part_id. To be used togehther with the part_dataset_id. Identifier for the account. This field is mandatory because it is a primary key, but it has a default value of space character when the Account ID level is not in use.	M
to_part_dataset_id	CHAR(8)		Reference to the datset id of the correspondgin part dataset	
to_part_id	INT(11)	F	Reference to PART table field part_id. To be used togehther with the part_dataset_id. Identifier for the receiving account. To define a multilimit this field is given value 0.	M
L_DATEEFFE	INT(11)		Date from which the new credit limit is effective.	M
L_TIMEEFFE	BIGINT(12)		Time from which the new credit limit is effective.	M
L_NEWVALUE	DECIMAL (20,2)		Value of the new lower limit for the bilateral balance (or multilateral balance if counterparty is *MULTILIMIT). The debit cap will constrain outgoing payments, if resulting bilateral position would go below the limit. The value can be positive or negative. A negative value is the most common case. It indicates that net outflow of liquidity is allowed while positive value indicates a request for a reception surplus. A value of .99 indicates that no limit is in force. It can be used to remove limits that have been assigned earlier during the day. Defining the debit cap will start the recording of bilateral position if no limits defined in BLIM data were in place previously for the given pair of participants.	M
L_DBCVALUE	DECIMAL (20,2)		Value of the new upper limit for the bilateral balance (or multilateral balance if counterparty is *MULTILIMIT). The credit cap will constrain incoming payments if the resulting bilateral balance would go above the limit. The value can be positive or negative. A positive value is the most common case. It indicates that inflow of liquidity is allowed while negative value would be a request for a sending surplus. A value of .99 indicates that no limit is in force. It can be used to remove limits that have been assigned earlier during the day. Defining the credit cap will start the recording of bilateral position if no limits defined in BLIM data were in place previously for the given pair of participants. In projects, which are created with version 3.1.0 or later credit cap value can be imported either separately on an own row of BLIM data or together in a row which also has value for the debit cap with same time label. In older database versions, values which have same time label need to be always imported in one row.	O
L-USERCOD1…5	VARCHAR(16)		Five optional fields where user-defined information can be stored for user-defined algorithms.	O
modified	TIMESTAMP		Last modified time stamp	

7.2.8 Reservations table [RSRV]

Reservations are only supported by specific non-public algorithms

Field name	Data type	Key	Description	
id	INT	P	Primary id	M
R_DATSETID	CHAR(8)	K	Unique identifier of data set to distinguish this data set from other parallel data sets used in simulations.	M
system_id	TINYINT(3)	K	Identifier of system.	M
part_dataset_id	CHAR(8)	K	Reference to the datset id of the correspondgin part dataset	
part_id	CHAR(11)	F	Reference to PART table field part_id. To be used togehther with the part_dataset_id.	M
R_DATEEFFE	INT(11)		Date from which new reservation is effective.	M
R_TIMEEFFE	BIGINT(12)		Time from which new reservation is effective.	M
R_NEWVALUE	DECIMAL (20,2)		Value of the new reservation	M
R_RESRVTYP	CHAR(1)		Type of the new reservation (H= highly urgent, U=urgent). If both reservation types are changed at the same time two update records are needed. Only positive values or zero are accepted.	M
R_USERCOD1…5	VARCHAR(16)		Five optional fields where user-defined information can be stored for user-defined algorithms.	O
modified	TIMESTAMP		Last modified time stamp	

7.2.9 Transaction data table [TRAN]

Contains transactiondata sets. Accounts stored in the PART table are referenced with technical id's since version 800. the dataset table contains information on compatibility between datasets in relation to the technical account ids. Transactions generated by the simulator are stored in the tran_generated_by_simulation table. To help usage a view called tran_view_all combines the input transaction information from the different tables.

Field name	Data type	Key	Description	
id	INT(10)	P	Primary id	
T_DATSETID	CHAR(8)	K	Unique identifier of data set to distinguish this data set from other parallel data sets used in simulations.	M
system_id	TINYINT	F	Identifier of the system this dataset belongs to.	M
T_TRANSAID	CHAR(20)	K	Unique identifier of transaction. From imported data.	M
T_INTRDATE	INT(11)	K	Day of transaction.	M
T_INTRTIME	BIGINT(12)	K	Time of transaction.	M
T_TRANVALU	DECIMAL (20,2)		Value of transaction.	M
from_part_dataset_id	CHAR(8)	K	Reference to the datset id of the correspondgin part dataset	
from_part_id	INT(11)	F	Account id from which payment is debited. Reference to PART table field part_id. To be used togehther with the part_dataset_id	M
from_part_dataset_id	CHAR(8)	K	Reference to the datset id of the correspondgin part dataset	
to_part_id	INT(11)	F	Account to which payment is credited. Reference to PART table field part_id. To be used togehther with the part_dataset_id	M
T_TRANCLAS	VARCHAR(8)		Transaction class is used to categorize payments eg. interbank payments, customer payments,… . This categorization can be used for variable purposes in specific	O

			algorithms and some parts of the processes. The main available algorithms do not use this information.	
T_TRANCLA2	VARCHAR(8)		Transaction class 2 is used to categorize the transactions same way as T_TRANCLAS. For example it can be used to direct payments to different queues or to be settled by different algorithms. The main available algorithms do not use this information.	
T_LINKCODE	VARCHAR(30)		A code used recognize all transaction belonging to a group. Linkcode can be used to link the different legs of e.g . DVP or PVP transactions.	O
T_LINKSYST	CHAR(8)		ID of system in which the other leg of the transaction is settled.	O
T_LINKTRNN	INT(11)		Count of the transaction linked by the same T_LINKCODE. Not used.	O
T_USERDEID	VARCHAR (50)		User-defined transaction ID that allows transaction to be compared in internal system runs.	O
T_DESCRIPT	VARCHAR(255)		Text description of transaction.	O
T_ASSENAME	VARCHAR(20)		Name of transaction asset.	O
T_USERCOD1 …5	VARCHAR(16)		Five optional fields where user-defined information can be stored for use by user-defined algorithms during simulations or in analysis of simulation output.	O
T_PRIORITY	TINYINT(4)		Value indicating importance of payment from 0-9, with 9 the highest priority. Used to order transactions in payment queues.	O
T_PROCTYPE	TINYINT(4)		Gives the opportunity to introduce various delayed processing options for transactions at a reference time. Possible values: 0 – Not defined. Is set to 0 automatically during import if null or empty. 1 – This transaction is settled exactly at the time described in T-PROCTIME and T-PROCDATE attributes. (Not in use) 2 – This transaction is not settled before the time described in T-PROCTIME and T-PROCDATE attributes.	O
T_PROCDATE	INT(11)		Day processing takes place as defined in T_PROCTYPE. Is set to -1(not defined) during the import process if the T_PROCTYPE is set to 0.	O
T_PROCTIME	BIGINT (12)		Time processing takes place as defined in T_PROCTYPE. Is set to -1(not defined) during the import process if the T_PROCTYPE is set to 0.	O
T_PROCTYP2	TINYINT(4)		Gives the opportunity to introduce second set of control variables to affect the settlement of the transaction. Feature is not yet in use in general version 3.0.0. Default value: 0 – Not defined. Is set to 0 automatically during import if null or empty.	O
T_PROCDAT2	INT(11)		Day processing takes place as defined in T-PROCTYPE. Is set to -1(not defined) during the import process if the T_PROCTYP2 is set to 0.	O
T_PROCTIM2	BIGINT (12)		Time processing takes place as defined in T-PROCTYPE. Is set to -1(not defined) during the import process if the T_PROCTYP2 is set to 0.	O
T_ASBIC	CHAR(11)		Bic code of the ancillary system	
business_day	CHAR(8)		Business day of the transaction. The field content is used to deduct the business days of a simulation if the selection: Business day deducted from transaction data is in force.	M
Iccl_link_id	VARCHAR(30)		Link to the Transaction_link_id of the iccl table.	O
modified	TIMESTAMP		Last modified time stamp	

7.2.10 Transactions generated by simulations [tran_generated_by_simulation]

The content of this table is very similar to the tran table. The content is only generated by simulations. The table contains all the automatically generated transactions.

Field name	Data type	Key	Description	
id	INT(10)	P	Primary id	
…			… Same fields as in Tran table…	M
t_submorig	CHAR (8)		Submission origin of the transaction. This means the name of the algorithm that created the transaction.	M
t_subevent	SMALLINT (8)		Number of sub-event	M
modified	TIMESTAMP		Last modified time stamp	

7.2.11 Simulation events [business_day_event]

The table is used to store the date and time of business day events. This table has been available since version 4.0.0. The table is used to define individual sart and end of day events for each business day. The table can be used to store other similar timetable material but support for these will be dependent on the algorithms used.

Field name	Data type	Key	Description	
id	INT(10)	P	Primary id	
data_set_id	CHAR(8)	k	Unique identifier of data set to distinguish this data set from other parallel data sets used in simulations.	M
system_id	TINYINT(3)	k	Identifier of system.	M
business_day	CHAR(8)	P	Business day to which the event belongs to. This value is used to determine the events to be included to the corresponding simulation day having the same business day.	M
event_id	SMALLINT(6)	P	Identifier for the event. The identifier is added during the import process according to the mapping defined in the PSS.properties files located within the BOF-PSS2.jar package. The supported default values for the general part are the following: - 0 = start_day=0 - 4 = end_day Note that the input file must have the readable string value. The corresponding integer code will be stored to the database.	
name	VARCHAR(20)		name of the event	M
date	INT (11)		Date when the event takes place	M
time	BIGINT (20,2)		Time when the event takes place	M
modified	TIMESTAMP		Last modified time stamp	

7.2.12 System algorithms [SALG]

This table contains the algorithm selected for the different system setups. Also parameter values of algorithms are stored in this table.

Field name	Data type	Key	Description	
A_DATSETID	CHAR(8)	K, F	Same data set ID as in the SYCD table.	M
system_id	TINYINT	K, F	Identifier of system.	M
A_ALGORIID	CHAR(8)	K, F	Unique identifier of algorithm.	M
A_SYALGOID	INT	K	Unique identifier of system algorithm in a system definition. Defines the order in which algorithms are displayed on the system definition view's algorithm table.	M
A_ALGOTYPE	CHAR(3)		Type of algorithm.	M
A_PARVALUE	TEXT		Enumerated list of parameters values used by algorithm. Parameters in form: parameter1?value][parameter2?value2][…[parameterlast?value	O
A_TEALGOID	CHAR(8)		ID of the time estimation algorithm (TEA) algorithm	O
A_TEALGPAR	TEXT		parameters defined in the system definition for the selected T–A -algorithm	O
A_PARALPRO	TINYINT(4)		Optional code for indicating whether the algorithm is processed in parallel. Possible values: 0 or empty: not processed in parallel 1: is processed in parallel	O
A_ALGDEFID	INT(11) auto increment	P, F	Unique identifier for algorithm definition	M
modified	TIMESTAMP		Last modified time stamp	

7.2.13 Analysis [analysis]

The analysis table is used by the automated stress testing tool to store general information regarding an analysis. Stores information is name of an analysis and the benchmark simulation the analysis uses as source data.

Field name	Data type	Key	Description	
id	INT(10)	P	Primary id	M
sim_id	INT(11))		Simulation ID of the simulation setup selected as benchmark or bases for the analysis. Reference to id in SIRI table	M
simulation_run_id	VARCHAR(50)		Name of the benchmark simulation defined by sim_id	O
name	VARCHAR(50)		Name of the analysis	M
type	SMALLINT(6)		Not in use yet	
status	VARCHAR(20)		Not in use yet	
description	VARCHAR (250)		Description of the analysis	
iccl_screen	SMALLINT(6)		Not in use yet	
dbal_screen	SMALLINT(6)		Not in use yet	
modified	TIMESTAMP		Date and time when has been changed the last time	

7.2.14 Analysis accounts [analysis_account]

This table serves the automated stress tester and contains the selected accounts for each analysis.The table is populated when the analysis is saved.

Field name	Data type	Key	Description	
analysis_id	BIGINT(20)	K	Unique identifier for an analysis. Key to analysis table	M
participant_id	VARCHAR(11)			
account_id	VARCHAR(34)	K	Account ids present in the benchmark simulation's participant data. Key to participant table's id	M

7.2.15 Failing accounts [failing_account]

Contains all the account id's that are affected in each scenario run under the automated stress tester. All the accounts in the table will be affected according to the account selections made in the stress tester. See the query base used for the scenario generation for more details. The table is updated when an analysis is saved.

Field name	Data type	Key	Description	
scenario_id	BIGINT(20)	P	Unique identifier for an analysis. Key to scenario table	M
account_id	VARCHAR(34)	P	Account ids present in the benchmark simulation's participant data. Reference to PART table field P_ACCOUNID. Used in the query filters of the stress tester.	M

7.2.16 Scenario data[scenario]

The table is updated when an analysis is saved under the automated stress tester. Contains scenario specific data generated according to the selected accounts and treatment rule (all selected/by participant/by account).

Field name	Data type	Key	Description	
id	BIGINT(20)	P	Unique identifier for an analysis scenario.	M
analysis_id	BIGINT(20))	F	Analysis ID of the analysis to which the scenario belongs.	M
simulation_run_id	VARCHAR(30)	F	Simulation id of the scenario	M
first_round_effect_number	BIGINT(20))		NOT in USE	
first_round_effect_value	Decimal(20,2)		NOT in USE	
iccl_cutter	BIGINT(20)		Not in use yet	
dbal_cutter	BIGINT(20)		Not in use yet	
modified	TIMESTAMP		Date and time when has been changed the last time	

7.3 Project's output tables

After each simulation, the result data is stored in dedicated tables.

Output tables contain simulation results and additional technical information such as simulation logs and batch run information. A unique simulation run identifier identifies the information belonging to the same simulation run. All tables are optional, i.e. the user must define the necessary output to be recorded for each simulation run. Sometimes only a small output sample is is sufficient.

7.3.1 System level statistics [SYLS]

Field name	Data type	Key	Description
id	INT(10)	P	Primary id
sim_id	INT(11)	F	ID of associated simulation
system_id	TINYINT(3)	F	ID of system.
business_day	CHAR(8)		Date of the Business day(format YYYYMMDD).
Y_SYSTNAME	CHAR(20)		Name of system.
Y_VALUDATA	DECIMAL (22,2)		Total value of transactions in day's transaction data.
Y_VALUCARR	DECIMAL (22,2)		Total value of transactions carried over from previous day(s).
Y_VALUSUBM	DECIMAL (22,2)		Total value of transactions submitted to system by submission algorithm.
Y_VALUSETT	DECIMAL (22,2)		Total value of transactions settled by settlement algorithms.
Y_VALUUNST	DECIMAL (22,2)		Total value of transactions remaining unsettled during the day [Y-VALUDATA] + {VALUCARR] – [Y-VALUESETT].
Y_NUMBDATA	INT(11)		Total number of transactions in day's transaction data.
Y_NUMBCARR	INT(11)		Total number of transactions carried over from previous day(s).
Y_NUMBSUBM	INT(11)		Total number of payments submitted to the system by submission algorithm.
Y_NUMBSETT	INT(11)		Total number of transactions settled by settlement algorithms.
Y_NUMBUNST	INT(11)		Total number of transactions remaining unsettled during the day [Y-NUMBDATA] + {NUMBCARR] – [Y-NUMBSETT].
Y_BODBALAN	DECIMAL (22,2)		The sum of the day's initial balances of the participants/accounts.
Y_EODBALAN	DECIMAL (22,2)		The sum of the day's ending balances of the participants/accounts.
Y_AVGCRLIM	DECIMAL (22,2)		The time weighted average of the available credit limits of the participants/accounts at system level.
Y_LIQAVAIL	DECIMAL (22,2)		The sum of the beginning of day balances and the time weighted average intraday credit available to the participants/accounts during the day, i.e. Y_BODBALAN + Y_AVGCRLIM.
Y_ABSCLUSA	DECIMAL (22,2)		The sum of average overdrafts (negative balances) for the participants/accounts during the day.
Y_RELCLUSA	DECIMAL (22,2)		The average overdraft divided by the average credit limit for the participants/accounts during the day.
Y_TOTLIQAV	DECIMAL (22,2)		Total liquidity available across all participants during the day.
Y_LOWBOUND	DECIMAL (22,2)		The sum of net liquidity requirement for the participants/accounts in the system (see Annex 1).
Y_MAXQUEVA	DECIMAL (22,2)		Maximum (peak) queue value during the day.

Y_AVEQUEVA	DECIMAL (22,2)		Average queue value during the day (the average time weighted value of queue balance).
Y_AVEQUELE	BIGINT(20)		Average queue duration for queued payments i.e. the sum of queuing time of queued payments divided by the total number of queued payments. Directly settled payments are not taken into account. With the system setup "Delete unsettled transactions (exclude from statistics), Unsettled transactions are not included in the average. (format hhhhmmss000, where 000 denotes milliseconds).
Y_QUENUMBE	INT(11)		Number of queued transactions per day.
Y_QUETOTVA	DECIMAL (22,2)		Total value of queued transactions per day.
Y_QUESTTIM	BIGINT(20)		Total time during the day that outgoing transactions were queued and the process was blocked due to insufficient liquidity for the participants i.e. the sum of the individual participant level queue stop times (format hhhhmmss000 where 000 denotes milliseconds). If many participants have long queues this value can be longer that the open hours.
Y_AVERTISE	BIGINT(20)		Simple average of queuing times of all payments. Note that also payments that are always settled directly by definition and that cannot be queued will also affect the average. With the system setup "Delete unsettled transactions (exclude from statistics), Unsettled transactions are not included in the average. (The database storing and export format is hhhhmmss000, the value is calculated with hhhmmss precision)
Y_LIQUSAGC	DECIMAL (22,2)		Liquidity usage indicator based on consumed liquidity i.e. consumed overdrafts and reserve deposits compared with submitted volume. Calculation explained in document Annex 1.
Y_LIQUSAGR	DECIMAL (22,2)		Liquidity usage indicator based on available liquidity (rigid credit limits) i.e. total credit limits compared with submitted volumes. Calculation explained in Annex 1.
Y_SETDELAY	DECIMAL (22,2)		Indicator of settlement delay i.e. actual delay compared to theoretic maximum delay at end of day. Calculation explained in Annex 1.
Y_SETTINGS	TEXT		Reserved for future needs.
Y_MAXCRUSG	DECIMAL (22,2)		Peak value of credit used during the simulation.
modified	TIMESTAMP		Date and time when has been changed the last time

7.3.2 Account statistics [ACST]

Field name	Data type	Key	Description
id		P	Primary id
sim_id	INT(11)	F	ID of simulation
part_id	INT(11)	K	ID of account. Reference to part table's field "part_id". Also the SIRI table's part datset id is used in combination to link the account and participant information form the PART table.
business_day	CHAR(8)		Date of the business day (format YYYYMMDD)
A_LIQINJVA	DECIMAL (22,2)		When specified, the injection value overrides any system-level value
A_VALUDATA	DECIMAL (22,2)		
A_VALUCARR	DECIMAL (22,2)		
A_VALUSUBM	DECIMAL (22,2)		
A_VALUSETT	DECIMAL (22,2)		

A_VALUUNST	DECIMAL (22,2)		
A_VALURECE	DECIMAL (22,2)		
A_NUMBDATA	INT(11)		
A_NUMBCARR	INT(11)		
A_NUMBSUBM	INT(11)		
A_NUMBSETT	INT(11)		
A_NUMBUNST	INT(11)		
A_NUMBRECE	INT(11)		
A_BODBALAN	DECIMAL (22,2)		The day's initial balance.
A_EODBALAN	DECIMAL (22,2)		The day's ending balance.
A_AVEBALAN	DECIMAL (22,2)		Average balance during the day.
A_MINBALAN	DECIMAL (22,2)		Minimum balance during the day.
A_MAXBALAN	DECIMAL (22,2)		Maximum balance during the day.
A_AVGCRLIM	DECIMAL (22,2)		Weighted (time) average credit limits. In case of extending credits without restrictions, the automatically granted limit is assumed to be in force until the end-of-day or until more credit is extended.
A_AVELIQAV	DECIMAL (22,2)		Average liquidity available during the day, i.e. average balance plus relevant credit limit.
A_CREDUSAG	DECIMAL (22,2)		Average overdraft during the day, i.e. average of the negative balances of the day.
A_CREDUSAP	DECIMAL (22,2)		Average overdraft percentage during the day, i.e. average of the negative balances of the day compared to relevant total credit limit.
A_LOWBOUND	DECIMAL (22,2)		See Annex 1.
A_UPPBOUND	DECIMAL (22,2)		Upper bound of liquidity is defined as the amount of liquidity need for immediate settlement of all transactions (i.e. no queues). This is not calculated in the simulation, because it requires a special simulation run in which there are no limits on intraday credit. This field is reserved if the user wants to include this information in the table.
A_MAXQUEVA	DECIMAL (22,2)		Maximum queue value during the day.
A_AVEQUEVA	DECIMAL (22,2)		Average queue value during the day (average time weighted value of queue balance).
A_AVEQUELE	BIGINT(20)		Average queue duration for queued payments i.e. the sum of queuing time of queued payments divided by the total number of queued payments. Directly settled payments are not taken into account. With the system setup "Delete unsettled transactions (exclude from statistics), Unsettled transactions are not included in the average. (format hhhhmmss000000, where 000000 denotes microseconds).
A_QUENUMBE	INT(11)		Number of queued transactions per day.
A_QUETOTVA	DECIMAL (22,2)		Total value of queued transactions per day.
A_QUESTTIM	BIGINT(20)		Total time during the day that outgoing transactions were queued and the process was blocked due to insufficient liquidity for this account (format hhhhmmss000, where 000 denotes milliseconds).
A_AVERTISE	BIGINT(20)		Simple average of queuing times of all payments. Note that also such payments that are settled directly by definition and that cannot be queued will also affect the average. With the system setup "Delete unsettled transactions (exclude from statistics), Unsettled transactions are not included in the average.

			(format hhhhmmss000000, where 000000 denotes microseconds. The calculation precision is in seconds)
A_LIQUSAGC	DECIMAL (22,2)		Liquidity usage indicator based on consumed liquidity i.e. consumed overdrafts and reserve deposits compared with submitted volume. Calculation explained in document Annex 1.
A_LIQUSAGR	DECIMAL (22,2)		Liquidity usage indicator based on available liquidity (rigid credit limits) i.e. total credit limits compared with submitted volumes. Calculation explained in Annex 1.
A_SETDELAY	DECIMAL (22,2)		Indicator of settlement delay i.e. actual delay compared to theoretic maximum delay at end of day. Calculation explained in Annex 1.
eod_credit_limit	DECIMAL (22,2)		End-of-day credit limit
modified	TIMESTAMP		Date and time when has been changed the last time

7.3.3 Bilateral statistics table [BIST]

Field name	Data type	Key	Description
id	**INT(10)**	P	Primary id
sim_id	INT(11)	F	ID of simulation run
from_part_id	INT(11))	F	ID of account
business_day	CHAR(8)		Date of business day (format YYYYMMDD)
to_part_id	INT(11)	F	ID of receiving account
D_EODBALAN	DECIMAL (22,2)		The day's ending bilateral balance (a sending surplus is a negative balance)
modified	TIMESTAMP		Date and time when has been changed the last time

7.3.4 Transaction event statistics [TEST]

This table contains additional simulation specific information related to transactions. The basic non-variable information can be retrieved from the tran and tran_generated_by_simulation tables. To help usage, the view called test_view_all combines the transaction information from the table test to the views TRAN_view_all and TRAN_GENERATED_BY_SIMULATION_view_all. The us eof the view makes it easier to access the data.

If performance becomes an issue it might be needed to acces the data directly from the tables with tailored queries without redundant information.

Field name	Data type	Key	Description
tran_id	INT(11)	P	Reference to id in TRAN OR in TRAN_GENERATED_BY_SIMULATION table. This field is not necessarily unique.
sim_id	INT(11)	F	ID of simulation
E_SUBMDATE	INT(11)		Date transaction was submitted for settlement.
E_SUBMTIME	INT(12)		Time transaction was submitted for settlement.
E_SETTDATE	INT(11)		Date transaction was settled.
E_SETTTIME	INT(12)		Time transaction was settled.

E_SUBMORIG	CHAR(8)		ID of algorithm generating an internal transaction, or 0 if from the transaction data.
E_SETTALGO	CHAR(8)		The ID of algorithm that settled the transaction.
E_SENDACBA	DECIMAL (22,2)		Sending account balance after settlement.
E_RECEACBA	DECIMAL (22,2)		Receiving account balance after settlement.
E_SETTSTAT	TINYINT(4)		Value indicating if transaction was settled: -3 = unsettled because introduction after end of day -2 = unsettled directly at entry -1= unsettled because of defined latest debit time, 0=unsettled 1=settled directly 2=settled via queue 3=forced end of day settlement). 4=payment replaced due to process reasons and recorded for reference. 5= technical payment excluded from statistics
E_BILABALA	DECIMAL (22,2)		Bilateral balance seen from the sending account after the transaction has been settled
E_ENTRDATE	INT(11)		Date when transaction is finally entered into clearing process in the simulated system. The time label can be different from submission time due to delays caused by simulated parallel processing or TEA time estimation. All statistics are based on submission time, not on entry time.
E_ENTRTIME	BIGINT(12)		Time when transaction is finally entered into clearing process in the simulated system. The time label can be different from submission time due to delays caused by simulated parallel processing or TEA time estimation. All statistics are based on submission time, not on entry time.
generated	TINYINT(4)		If value is 1 the transaction's basic information can be found in tran_generated_by_simulation table.

7.3.5 Intraday credit limit order execution statistics [iccl_order_execution_statistics]

Field name	**Data type**	**Key**	**Description**
id	BIGINT(20)	P	Id of the icl order acting as a link to the input database
Iccl_id	BIGINT(20)		Link to the original icl order in the iccl table of the input db
simulation_id	VARCHAR(20)		Link to the simulation
dataset_id	VARCHAR(20)		Dataset id to which the order belongs
system_id	CHAR(8)		ID of associated system
business_day	CHAR(8)		Business day the order belongs to
execution_status	INTEGER		0 = rejected, 1 = executed directly, 2 = executed after being queued, 650 = pending credit line decrease replaced by new order, -3 = removed at cut off bank
entry_date	INTEGER		
entry_time	BIGINT(12)		
Resolution_date	INTEGER		Date when order was either removed from queue or executed
Resolution_time	BIGINT(12)		Time when order was either removed from queue or executed
modified	TIMESTAMP		Date and time when has been changed the last time

7.3.6 Netting event statistics [NEST]

Field name	Data type	Key	Description
id	**INT(10)**	P	Primary id
sim_id	INT(11)	F	Reference to id in SIRI table
system_id	TINYINT(3)	K	ID of associated system
N_ALGORIID	CHAR(8)	K	ID of netting algorithm
N_NETTDATE	INTEGER		Date netting (e.g. gridlock resolution) was executed
N_NETTTIME	BIGINT(12)		Time netting (e.g. gridlock resolution) was executed
N_NETTINID	CHAR(8)		ID of associated net settlement
N_TRANSVAL	DECIMAL (22,2)		Value of additional transactions generated by the netting algorithm
N_TRANSVOL	INT		Number of additional transactions generated by the netting algorithm
N_SETTLVAL	DECIMAL (22,2)		Value of original transactions settled by the netting algorithm
N_SETTLVOL	DECIMAL (22,2)		Number of original transactions settled by the netting algorithm
N_TOTALVAL	DECIMAL (22,2)		Value of all transactions subject to netting
N_TOTALVOL	DECIMAL (22,2)		Volume of all transactions subject to netting
modified	TIMESTAMP		Date and time when has been changed the last time

7.3.7 Account violation statistics [AVST]

Field name	Data type	Key	Description
id	**INT(10)**	P	Primary id
sim_id	INT(11)	F	Reference to id in SIRI table
part_id	INT(11)	F	ID of account in which violation occurred. Reference to part_id field in the PART table. Also the SIRI table's part datset id is used in combination to link the account and participant information form the PART table.
V_EVENDATE	INT(11)		Date violation occurred
V_EVENTIME	BIGINT(12)		Time violation occurred
V_VIOLCAUS	VARCHAR (12)		Reason for violation. Typically, forced end-of-day settlement or credit limit reduction. Value ICCL when depends on new lower credit limit, ANCSETTL when depends on ancillary system settlements and the value equal a transaction ID when the violation is caused by a forced end-of-day settlement.
V_VIOLVALU	DECIMAL (22,2)		Value of violation
business_day	CHAR(8)		Date of the business day
modified	TIMESTAMP		Date and time when has been changed the last time

7.3.8 Queue reason information [QURE]

Contains the reasons why a payment has been put to queue. The table also contains an entry for the removal time of the transaction. This means every transaction can have 2 rows in this table.

Field name	Data type	Key	Description
id	**INT(10)**	P	Primary id
sim_id	INT(11)	F	Reference to id in SIRI table
system_id	TINYIN	F	Reference to id in SYSTEM table
K_TRANSAID	CHAR(20)	P	Identifier of transaction
R_DATEMODI	INT(11)	P	Date when change in queuing reason
R_TIMEMODI	BIGINT(12)	P	Time when change in queuing reason
R_QURECODE	TINYINT(4)		The queue reason code can take following values: 0 = queued due to process reasons. e.g. deferred system or payment is always settled via queue 1 = not enough liquidity on the account, 2 = bilateral limit exhausted and 3 = multilateral limit exhausted, when transactions are placed in queue. 4 = bilateral credit cap is limiting 5= multilateral credit cap is limiting 9 = Transaction is cleared or removed from queue 10 = bilateral credit cap exhausted 11 = multilateral credit cap exhausted 12 = FIFO, blocking payment in queue 100 = Removed Because Settled 101 = Removed because of end of day 103 = Removed because of forced end of day settlement 111 = Removed because of end of day trade phase 1 112 = Removed because of end of day trade phase 2 113 = Removed because of end of AS6 cycle 114 = Removed because of Latest Debit Time 115 = Removed because replaced by new ICCL order
modified	TIMESTAMP		Date and time when has been changed the last time

7.3.9 Analysis indicators [analysis_indicator]

The table is only populated by the stress tester. It is recalculated each time the “Run report” button is pressed.

Field name	Data type	Key	Description
id	BIGINT(20)	P	Technical id
simrunid	VARCHAR(30)	F	key to the simulation run id
systemid	CHAR(8)	*	Id of the system
failing_accounid	VARCHAR(34)	*	Affected account id or participant of the scenario
business_day	CHAR(8)	*	Business day
is_bench	TINYINT(1)	*	Identifier for whether the row belongs to the benchmark of the analysis
bench_eod_balance	DECIMAL(22,2)		end of day balance of the benchmark simulation
bench_unst_value	DECIMAL(22,2)		Value of unsettled transactions in the benchmark.
sent_unst_value	DECIMAL(22,2)		Value of unsettled transactions
sent_unst_count	BIGINT(20)		Count of unsettled transactions
sent_unst_value_direct	DECIMAL(22,2)		Value of transactions removed from the input data for the scenario
sent_unst_count_direct	BIGINT(20)		Count of transactions removed from the input data for the scenario
received_payments	DECIMAL(22,2)		Sum of the value of received transactions
received_unst_value	DECIMAL(22,2)		Value of transactions not received in scenario simulation
received_unst_count	BIGINT(20)		Count of transactions not received in scenario simulation

received_unst_value_direct	DECIMAL(22,2)		Value of transactions not received due to input data manipulations. Transactions removed due to scenario
received_unst_count_direct	BIGINT(20)		Count of transactions not received due to input data manipulations. Transactions removed due to scenario
lower_bound	DECIMAL(22,2)		Lower bound of liquidity. Net liquidity need to allow full settlement.
max_upper_bound	DECIMAL(22,2)		Maximum value of the upper bound of initial liquidity to allow direct settlement at entry of all transactions. Same as gross outflow.
bench_setdelay	DECIMAL(22,2)		Settlement delay observed in the benchmark
weighted_avg_receiving_time	BIGINT(20)		Value weighted average receiving time in the scenario
weighted_avg_receiving_time_bench	BIGINT(20)		Value weighted average receiving time in the benchmark
weighted_avg_sending_time	BIGINT(20)		Value weighted average sending time in the scenario
weighted_avg_sending_time_bench	BIGINT(20)		Value weighted average sending time in the benchmark

7.4 Technical tables

There are some technical tables that relat to the definition of simulations and logging of events.

7.4.1 Batch run information [BARI]

This table is used to store information on which simulations are run in a batch. This information is used by the simulation execution user interface.

Field name	Data type	Key	Description
R_SIMBATID	CHAR(8)	P	Name of simulation batch.
R_NROFRUNS	SMALLINT (6)		Number of simulation runs in batch.
R_PROCTIME	BIGINT(12)		Total processing time for simulation batch.
R_SIMRUNID	TEXT		The IDs of simulation included in batch.

7.4.2 Simulation run information [SIRI]

This table contains 2 types of information related to a simulation. When defining a simulation, the definition and configuration information of a simulation is stored into this table. When running a simulation also the information of selected output tables and some basic information related to the running os a simulation such as time and duration are stored here. Note when a simulation is rerun, the old information is overwiten.

Field name	Data type	Key	Description
id	INT(10)	P	Primary id
parent_id	INT(11)		Reference to id in SIRI table
M_SIMRUNID	CHAR(8)		Textual simulation id

M_SIMUNAME	VARCHAR(20)		Name of the simulation
M_SIMDESCR	VARCHAR(120)		Description of the simulation
M_PROCDATE	INT(11)		Processing date for simulation run (format YYYYMMDD).
M_PROCTIME	BIGINT(12)		Processing time for simulation run (format hhmmss000 where 000 stands for milliseconds).
M_DURATION	INT(11)		Duration of simulation run (format hhmmsss000 where 000 stands for milliseconds, where the milliseconds can also have specific values).
M_SYSTEMID	TEXT		System IDs belonging to the simulation. Values in form: system1,system2,…
M_OUTPTABL	TEXT		Output data selected for the simulation. Values in form: SYLS,ACST,…
M_SYCDDSID	TEXT		Data set IDs of systems belonging to the simulation. Values in form: dataset1,dataset2,…
M_PARTDSID	TEXT		Data set IDs of participants belonging to the simulation. Values in form: dataset1,dataset2,…
M_DBALDSID	TEXT		Data set IDs of balances belonging to the simulation. Values in form: dataset1,dataset2,…
M_ICCLDSID	TEXT		Data set IDs of credit limits belonging to the simulation. Values in form: dataset1,dataset2,…
M_TRANDSID	TEXT		Data set IDs of transactions belonging to the simulation. Values in form: dataset1,dataset2,…
M_BUSDESID	TEXT		Data set Ids indicating the events data set to be used in the simulation.
M_NUMBSYST	SMALLINT(6)		Number of systems belonging to the simulation.
M_NUMBPART	MEDIUMINT(9)		Number of participants belonging to the simulation.
M_NUMBTRAN	INT(11)		Number of transactions belonging to the simulation.
M_SUBALGID	CHAR(8)		Identifier of submission algorithm.
M_ALGOTYPE	CHAR(8)		Type of algorithm, value ‘SUB’.
M_SUBPARAM	TEXT		Enumerated list of parameters.
M_BLIMDSID	TEXT		Data set IDs of bilateral limits belonging to the simulation. Values in form: dataset1,dataset2,…
M_RSRVDSID	TEXT		Data set Ids of reservation data used in the simulation. Values in form: dataset1,dataset2,…
modified	TIMESTAMP		Date and time when has been changed the last time

7.4.3 Applicationruns [Applicationruns] (Not in use)

Field name	**Data type**	**Key**	**Description**	
ID	INT(10)	P		M
SetupID	CHAR(1)			M
SetupName	CHAR(1)			M
RunInfo	TEXT			M
StartTime	TEXT			M
EndTime	TEXT			M
USERCOD 1...5				

7.4.4 Process log [Processlog]

This table is used to log information on algorithm runs during simulations. All algorithms do not support this. also this feature needs to be enabled from property files and is mainly used for testing purposes.

Field name	Data type	Key	Description	
ID	INT(10)	P	Technical ID for entry used as unique key	
ApplicationRunID	VARCHAR(8)	F	Link to Application run ID	
ProcessRunID	BIGINT(12)		All log rows associated to one algorithm run are stored with the same ProcessRunID	
ProcessName	VARCHAR (30)		Name of Algorithm or process	
Event	VARCHAR (30)		Start of algorithm Start of algorithm postponed End of algorithm execution	
Description	VARCHAR (255)		Field for additional information. Content can be dynamically formed by the algorithm running to mediate information to the user. This can be used for debugging and validating the functioning of some features..	
Date	INT(11)		Current date when the row is logged	
Time	BIGINT(12)		Time when the row is logged	
SimDate	INT(11)		Day in simulation	
SimTime	BIGINT(12)		Time in simulation	
Info	VARCHAR(512)			
USERCOD 1...5	VARCHAR (16)			

8 Miscellaneous

8.1 Date format

The supported date format is: `yyyymmdd`

8.2 Time format

The supported time format is: `hhmmss.ssssss`

8.3 File template

A file template describes which columns in the CSV file correspond to particular fields in the database table. For example, if you want to import a CSV file with participant data to the PART table and the CSV file's first column contains participant ID and the second column the name of the participant, you define in the

import template "1" in the first row (P_PARTICID) and "2" in the third row (P_FULLNAME). The other rows stay empty, if these are the only fields to import.

Templates are saved in the TEMP (template) table in the SYSTEM database. The input data tables PART (participant), DBAL (daily balances), ICCL (intraday credit limits) and TRAN (transaction) have all their own templates.

Ready-made templates are provided in the simulator for all output database tables for exporting all data fields. The names of these templates are the table name followed by –ALL e.g. TEST-ALL.

Templates are updated when you change the information in them. If you want to remove templates, you have to modify the database directly. For instructions see 6.4.3.

8.4 About using Microsoft Excel with the simulator

Microsoft Excel is a handy tool for editing simulator data, analysing simulation output and creating reports and graphs.

The following facts are worth noting if you plan to use Excel with the **BoF-PSS2** simulator:

- Old Excel versions have a limit of 65,536 rows per worksheet. Excel 2007 can handle 1,048,576 rows.
- Excel may produce additional rows and columns when saving a table as CSV file (all rows and columns that have been active in the table during Excel calculations will be saved in the CSV file, even though they are empty at the time of saving).
- Large values may be distorted (less accuracy).
- Check that delimiters (decimal and data separators) and presentation formats (date and time) are identical with simulator specifications.
- The actual content of CSV files stored by Excel can be checked with Notepad or some other text editor.

The output reports and output CSV-files have not been edited. The idea is that everyone can edit them according to own desires using Excel or other reporting tools. When some reports are used frequently it is a good idea to read the output CSV-files into a predefined Excel table.

8.5 Error list

When errors are found, an error list is generated. The name of the file is errorlist_date_time and the file type is plain text / comma separated values (.csv). The list contains:

Row	Description
1	The heading “Errors.”
2	Informs where errors have arisen. Values “simulation execution”, “import input data” or “cross-check input data.”
3	Empty row.
4	In this underlined row is stated - in case of import input date the faulty data type (corresponds to input data tables), the data set ID and the system ID - in case of cross-check input data simulation ID, system ID, data set ID and data type - in case of simulation execution simulation batch ID, simulation ID, system ID, data set ID
5	Empty row.
6	Faulty row in input file or in database table.
7	Error code and description of the error.
8-…	If the same data type, simulation ID, system ID and data set ID have more errors, they are listed as rows 5, 6 and 7. If the data type, simulation ID, system ID or data set ID change, rows 3 and 4 are written before rows 5, 6 and 7. These rows are written until all errors have been listed.

8.6 CSV and Excel files

The program creates CSV files from reports and export data. Data are moved to files from the saved database tables. Data in the CSV files are separated from each other by a separator, which the user can select. The extension is .csv for the CSV file. If the user has defined it, the first row in the file consists of field headings.

9 Technical documentation

The following documents on **BoF-PSS3** are available via the internet-site https://www.suomenpankki.fi/simulator:

– This user manual
– Simulator presentation and basic information

10 Troubleshooting guide

This chapter is designed to help users to find and eliminate problems when employing the simulator.

It is a list of frequently encountered problems by the users. This list will be updated based on user experiences. Please send you experiences to the email address:

bof-pss@bof.fi

The updated guide will be posted on the web-site https://www.suomenpankki.fi/simulator and will be distributed with any new version of the simulator. If you have used your version already for some time, it might be good to check for the updated trouble shooting guide on the internet.

No data sets to configure in the simulation configuration screen

You must Define system data in the Input Generation Subsystem before the configuration screen can offer you data sets for the systems to simulate.

No settled transactions although simulations was run successfully

The system could be lacking liquidity. Check that you have granted enough liquidity via initial balances or intraday credit limits (tables of free usage).

The liquidity could also be lacking due to date and/or time errors. Check via View data sets that the date and time data for transactions, initial balances and intraday credit limits are correct. Check also that the open hours of the system is correctly specified (hhmm in 24 hour format) in the Define system data screen. Be especially cautious if you have been using Excel for editing the data, because Excel is often changing the date and time formats when writing to CSV files. Check for instance with Notepad that the formats are in the correct format in the input CSV files.

No transactions found and simulation terminated/done immediately

The simulator and MySQL perform well with most regional settings. However, with some special regional settings control characters seem to be converted and thereby corrupted. Please, try using some common regional setting alternative (e.g. English UK or USA). Regional settings are changed in the control panel section of Windows.

10.1 Database table repairs

If the Simulator and MySQL are closed by the user while the software is writing into some database table, the database can become corrupted. Typically this can happen if the simulator seems to be stuck and the user closes it with "end task" in Windows task manager.

As a result of corrupted database table the simulator won't start and in the start up window, e.g. following error message can be presented "*Can't open file: 'tablename.MYI'. <errno:144>*"

This can be fixed by repair table command using e.g. database browser or in command-line console if the previous is not available. For more information see https://mariadb.com/kb/en/repair-table/ For console view, run `[MariaDB installation folder]\bin\mysql.exe`. This will open the database server console view.

Assuming that input database table "TRAN" is corrupted in project "example1", following commands are required.
use i_example1; (+ Enter)
repair table tran; (+ Enter)
For more details see the manual matching the installed database product.

11 Acknowledgements

We would like to acknowledge the contribution made by the following developers and contributors who have assisted in creating the BoF-PSS2 Payment and Settlement Simulator

Developers/Contributors at Bank of Finland

The BoF-PSS2 simulator is the second payment and settlement simulator built by the Bank of Finland. The first one was originally developed for internal use only, but it expanded well outside the Finnish borders. The new version has been built based on experience gathered from the first, but designed for international and independent usage and includes more features than its predecessor did. BoF-PSS2 also has a technical design that is more efficient for large simulations. Acknowledgement is also given to the persons from the Bank of Finland who were involved in the first version, because without the first version there would not have been a second one.

Currently involved:

Kasperi Korpinen

Project manager/designer/specifyer/tester/support of the BoF-PSS2 simulator from version 2.4.0 co-author of the reference manual and other user documents

Tatu Laine

Project manager/designer/specifyer/tester/support of the BoF-PSS2 simulator from version 3.0.0 co-author of the reference manual and other user documents

Previously involved:

Virpi Andersson

Tester of user interfaces and simulations of BoF-PSS2

Matti Hellqvist

Designer/developer/support of the BoF-PSS2 simulator from version 1.0.0 till 3.1.0

Co-author of the reference manual and other user documents

Co-athor of first ABM algorithms for the simulator, later incorporated to the version 9.2.1 by the simulator development team.

Risto Koponen

Project manager for the BoF-PSS1 simulation project

Hannu Lampela

Technical advisor for BoF-PSS2

Harry Leinonen

Adviser/designer of the BoF-PSS1 simulator

Project manager for the BoF-PSS2 project

Designer/developer of the BoF-PSS2 simulator

Co-author of the reference manual and other user documents

Markus Penttilä

Designer / developer of the BoF-PSS2 simulator for the version 2.4.0

Co-author of the reference manual and other user documents

Kati Salminen

Testing and distribution of BoF-PSS2

Kimmo Soramäki
Designer/developer/programmer of the BoF-PSS1 simulator

Kirsti Tanila
Tester and user of BoF-PSS1 simulator

Eero Tölö
Designer / developer of the BoF-PSS2 simulator for the version 3.2.0
Co-author of the reference manual and other user documents

Petri Uusitalo
Distribution design and organisation

Contributors at Fujitsu Finland

Fujitsu was choosen as new development partner starting from Autumn 2012 replacing MSG Oy.

Harri Engblom
Lead developer / designer

Kalle Saarela
Developer / designer

Maarit Aalto
Documenting / Tester

Cotributors at MSG Software Oy

The development of the BoF-PSS2 simulator was contracted to MSG Software Oy based on the specifications developed by the Bank of Finland until mid 2013.
The last version MSG contributed was 4.0.0.

Maritta Halonen
User interface design
Co-author of reference manual and data dictionary
Testing

Markku Kilvio
Project manager, developer

Timo Koistinen
Developer data import, export and statistical analysis/reports

Riku Peltokorpi
Developer user interfaces and system data imports

Kai Rauha
Developer output reports

Ville Ruoppi
Lead developer, simulator engine and algorithms
System design
Technical and algorithm documents

Leena Tyni
Project manager
System/user interface design
Testing
Co-author of user documents

Developers/Contributors at the European Central Bank (ECB)

Special acknowledgement is given to the ECB for assigning resources for the development of BoF-PSS2.

Argyris Kahros
Co-author of first ABM liquidity management algorithms (Version 9.2.1)

Kimmo Soramäki (early versions)
Specifications and design
Testing of BoF-PSS2 simulator with the BoF-PSS1 simulator
Co-author of user documents
Alpha and beta testing

Sponsorship contributors

Special acknowledgement is given to Bank of Canada, the Bank of England and the Federal Reserve Bank of New York for their sponsorship in developing version 2.0.0 of the BoF-PSS2 simulator, which include such- new features as bilateral limits, improved efficiency, enhanced database options and time transposition possibility.

For version 2.1.0 of the simulator, special acknowledgement is given to Federal Reserve Bank of New York. New features introduced in this version include the RRGS algorithms.

For version 3.1.0 of the simulator, special acknowledgement is given to EBA Clearing S.A.S. New features developed in co-operation with them include credit cap functionalities of BoF-PSS2.

The sponsorship and cooperation of these contributors has made it possible to distribute these features to the whole user community.

We are indebted to the following persons in the above mentioned organisations

Bank of Canada: Neville Arjani, Devin Ball, Lorraine Charbonneau, Allan Crawford, Alejandro Garcia, Dinah Maclean, Darcey McVanel and Jeffrey Smith.

The Bank of England: Stephen Millard and George Speight

The Federal Reserve Bank (NY): Morten Bech, Kurt Johnson, James J. McAndrews

Alpha/Beta testing and development contributors

The early BoF-PSS2 simulator version was distributed to other central banks as an alpha and beta version. Important contributions in the form of new ideas, testing, bug-finding etc have been received from following persons involved in alpha and beta testing. Contribution to further development and bug fixes of the production version is also acknowledged.

Bank of England: Paul Bedford, Stephen Millard and Jing Yang

Bank of Slovenia: Simon Anko

Bank of Thailand: Tanai Khiaonarong

Central Bank of Iceland: Rafn Arnason

Central Bank of the Republic of Turkey: Pinar Akan

European Central Bank: Peter Galos

Nationalbanken, Danmark: Kasper Sylvest Olsen

Singapore Monetary Authority: Wai Leong Lee

Sveriges Riksbank: Johan Pettersson

Bank of Canada: Darcey McVanel, Alejandro Garcia, Neville Arjani, Devin Ball, Jeffrey Smith

De Nederlandsche Bank: Elisabeth Ledrut, Ronald Heijmans

ANNEXES

I. Calculation of specific indicators

Trough this annex text following notations are used.

$n \in N$ Number of participants (or accounts) in the system.

$i \in \{1,...,n\}$ Index number pointing to one particular participant.

$d \in N$ Total number of payments send in the system over the course of business day.

d_i Number of payments sent by bank *i*.

i, k Pair of index numbers pointing to one particular payment *k* of participant *i*. Here $k \in \{1,...,d_i\}$

Lower bound liquidity demand Y_LOWBOUND and A_LOWBOUND

On the low extreme all banks might have just enough liquidity to settle all the day's payments before the end of the day by using multilateral net settlement to solve gridlock situations. We shall refer to this amount of liquidity as the lower bound of liquidity *LB*. The lower bound of liquidity [A_LOWBOUND] for the *i*th participant/account LB_i can be written as

$$LB_i = \max\left(0, \sum_{k=1}^{d_i} a_{i,k} - \sum_{j=1}^{n}\sum_{k=1}^{d_j} a_{j,k}\Big|_{(r_{j,k}=i)}\right), \text{ where}$$

$a_{j,k} \in \Re_+$ = the value of payment *k* of participant *j*.

$r_{j,k} \in \{1,..., j-1, j+1,...,n\}$ = the receiver of payment.

The first sum is the value of payments send and the second sum is the value of the payments received over the course of the business day by bank *i*.

If the value of payments received during the day is larger than the value of payments sent, a participant/account only needs to use the liquidity it receives in the form of incoming payments for settling its own payments and thus the lower bound equals zero. If the value of payments sent exceeds the value of payments received, the difference has to be available at least at the end of the day.

Lower bound of liquidity in the system level [Y_LOWBOUND] is simply the sum of lower bounds of individual participants/accounts.

Settlement delay Y_SETDELAY and A_SETDELAY

The delay indicator is a relative indicator ranging from 0 to 1. If not transactions are queued the value is 0 if all transactions are queued the maximum time ie to the

end of the day the value is 1. The value is calculated as the time weight queuing value for each queued transaction (transaction value times the time in queue) divided by the time weighted value if all payments were delayed to the end of the day (the transaction value times the time from submission to the end of the day). The values are calculated for each participant/account.

$$Settlementdelay = \frac{\sum_{i=1}^{n}\sum_{k=1}^{d} q_{i,k} * a_{i,k}}{\sum_{i=1}^{n}\sum_{k=1}^{d} s_{i,k} * a_{i,k}} \text{ where}$$

q = queing time for each payment
s = maximum settlement delay ie time difference between submission and end-of-day.
The values of unsettled transactions are included in both factors.

Consumed liquidity Y_LIQUSAGC and A_LIQUSAGC

The consumed liquidity indicator measures to which extent overdrafts (ie negative balances) and reserve deposits have been used for settling payments ie the difference between the beginning of day and minimum balance during the day divided by volume of submitted transactions. It measures the consumed liquidity compared with the throughput volume or inversely to which extent the liquidity of received payments have not been able to cover the liquidity needs of outgoing payments.

$$\text{Consumed liquidity} = \frac{\sum_{t=0}^{T} L_t^d}{\sum_{t=0}^{T}\sum_{i=0}^{t} V_i^O} \text{ where}$$

L^d = the difference between daily opening and minimum balance

V^o = the average transaction volume

Rigid liquidity indicator Y_LIQUSAGR and A_LIQUSAGR

The rigid liquidity indicator gives the relation between the total available credit limit compared to the transaction volume to be settled ie the sum of transactions to be sent. It measures the credits allocated compared to the throughput volume.

$$Rigid\ liquidity\ indicator = \frac{\sum_{t=0}^{T} L_t^a}{\sum_{t=0}^{T}\sum_{i=0}^{t} V_t^O} \text{ where}$$

L^a = the average credit limit available during the day

II. ABM property file example

```
##
# Simulator agent configuration property file
#
# NOTE!
# With any property setting you can split it to multiple rows
# by appending a blank following a backslash [ \] to the end of each row
# excluding the last row of the setting

##
# Common properties to all agents
# List of transaction class types that are processed from transaction queue
# and not sent to bank agent queues
nonProcessableTranClassTypes=
nonProcessableTranClass2Types=
nonProcessableTranPriorities=

##
# List of all agent implementations
# NOTE!
# The name must match with the Java class name
agents=CautiousBank;CustomerDiscriminatingBank

##
# Common CautiousBank agent specific parameters.
# This setting contains the full list of parameters that can be applied to the
agent
# delimited by semicolon
# NOTE!
# The property name follows the agent's Java class name having string
# [Settings] appended to the end
# delayTypes:
#   may contain zero or multiple type values separated by blank and
#   it refers to value in TRAN table t_tranclas column
CautiousBankSettings= \
alertsSecondsBeforeEndOfDayEvent=1800; \
onPercentage=0.2; \
offPercentage=0.4; \
delayTypes=1.1 1.2; \
logEvents=true;

##
# List below the participants having an agent as a list of participant keys.
# Each participant key is in format
# [system id]-[participant id]-[account id]
# delimited by semicolon
# NOTE!
# The property name follows the agent's Java class name having string
# [Participants] appended to the end
CautiousBankParticipants= \
1-FRSDFR2S001-FRFRSDFR2S00100000000000000001;
#1-SDMFFAPA055-FASDMFFAPA055;

##
# Participant specific agent parameters.
# If provided it overloads the common agent parameters
# thus you can specify below just the ones that differ from common agent
parameters
# NOTE!
# The property name follows the agent's Java class name having string
# [Settings][participant key] appended to the end
CautiousBankSettings1-FRSDFR2S001-FRFRSDFR2S00100000000000000001= \
onPercentage=0.1; \
```

```
offPercentage=0.5;


##
# Common CustomerDiscriminatingBank agent specific parameters.
# This setting contains the full list of parameters that can be applied to the
agent
# NOTE!
# The property name follows the agent's Java class name having string
# [Settings] appended to the end
CustomerDiscriminatingBankSettings= \
latestTimeToPostponePayments=163000000000; \
alertsSecondsBeforeEndOfDayEvent=1800; \
delayMaxNormal=1000000000; \
delayMinNormal=1000000; \
liqLowNormal=100; \
liqHighNormal=1000; \
priorityThresholdUrgent=4; \
delayMaxUrgent=200000000; \
delayMinUrgent=0; \
liqLowUrgent=0; \
liqHighUrgent=500; \
priorityThresholdHighlyUrgent=7; \
delayMaxHighlyUrgent=0; \
delayMinHighlyUrgent=0; \
liqLowHighlyUrgent=0; \
liqHighHighlyUrgent=0; \
defaultDelay=1000000000; \
createWakeUpEventsOnMaxDelay=true; \
logEvents=false;

##
# List below the participants having an agent as a list of participant keys.
# [system id]-[participant id]-[account id] delimited by semicolon
# NOTE!
# The property name follows the agent's Java class name having string
# [Participants] appended to the end
CustomerDiscriminatingBankParticipants=

##
# Participant specific agent parameters.
# The ones provided below overload the common agent parameters
# thus you can specify below just the ones that differ from common agent
parameters
# NOTE!
# The property name follows the agent's Java class name having string
# [Settings][participant key] appended to the end
CustomerDiscriminatingBankSettings= \
#[key1];[key2];...


##
# You can use the below query to help setting up the list of agent participant
keys
# SELECT
# DISTINCT CONCAT(t1.system_id, CONCAT('-' , CONCAT(t1.from_particid, CONCAT('-'
, t1.from_accounid))))
# FROM tran_view_all t1
# WHERE t_datsetid = 'RawAP' AND t1.from_accounid IN
('FRFRSDFR2S00100000000000000001', 'FASDMFFAPA055')
# ;
```

III. Example ABM bank agent implementation

Below is a simplified example bank agent implementation that highlights the methods it must contain, how it gains it's turn to execute its own logic and the methods it can use to send it's own transactions to the settlement process.

```
package modules;

import java.math.BigDecimal;
import java.util.ArrayList;
import java.util.List;

import com.bof.pss2.common.AccountData;
import com.bof.pss2.common.SystemEvent;
import com.bof.pss2.common.Transaction;
import com.bof.pss2.common.TransactionQueue;
import com.bof.simulator.util.DateTime;

/**
 * This algorithm is allowed to be modified by holders of the BOF-PSS2 Simulator
 * license. For further details please refer to the BOF-PSS2 User license
 * conditions.
 *
 * This is a simplified example implementation of a bank agent to clarify the
 * basic behaviour and the methods to use in communicating with the simulation
 * process.
 *
 * In case you need a parameter(s) that controls you logic e.g.:
 * beActivaAfterTime that keeps the bank agent passive until the time set in
 * parameter is passed. What to do:
 *
 * <pre>
 * 1. define the parameter in configuaration file
 *    <code>beActivaAfterTime=120000</code>
 * 2. define a member variable to this class
 *    <code>private long beActivaAfterTime;</code>
 * 3. introduce the parameter with default value in class construtor
 *    <code>addParameter("beActivaAfterTime", "120000");</code>
 * 4. in init() set the parameter value to the member variable
 *    <code>beActivaAfterTime =
     getParameterLongValue("beActivaAfterTime");</code>
 *    NOTE! the default value is set
 *    if the value is not defined or set in configuration file
 * </pre>
 */
public class ExampleBank extends Bank {
    // Introduce parameters that controls you logic e.g.:
    // beActivaAfterTime that keeps the bank agent passive until the time set in
    // parameter is passed.
    private long beActivaAfterTime;

    public ExampleBank() {
        // You may introduce you own parameter in the ABM configuration file
        // and setup the default values for those using appParameter() methods.

        // Set up parameters with default values
        addParameter("beActivaAfterTime", "120000");
        addParameter("alertsSecondsBeforeEndOfDayEvent",
            String.valueOf(alertsSecondsBeforeEndOfDayEventDefault), true);
        addParameter("logEvents", String.valueOf(isLogEvents()), true);
    }
```

```
    /**
     * In init you can initialize variable(s) required by your bank agent.
     */
    @Override
    public void init() {
        // logEvents parameter value defines wether or not to log the events
        // to log file.
        setLogEvents(getParameterBooleanValue("logEvents"));

        beActivaAfterTime = getParameterLongValue("beActivaAfterTime");

        String info = String.format("Part %s parameters: %s=%s, %s=%s, ",
            getAccount().getParticipantID(),
            "beActivaAfterTime", getParameterLongValue("beActivaAfterTime"),
            "logEvents", getParameterStringValue("logEvents"));
        logEvent("init", info);

        // Create an AGENT_ALERT_END_OF_DAY system event based on given
        // alertsSecondsBeforeEndOfDayEvent parameter
        Long value = getParameterLongValue("alertsSecondsBeforeEndOfDayEvent");
        getSimulationQueueAccessor().add(createEndOfDayAlertEvent(value), true);
    }

    /**
     * In process method's switch case structure you can write your logic how
     * this bank agent reacts on different wakeup calls. In each of them
     * you can study the bank agent's (account's) position at the moment
     * by inspecting
     *
     * <pre>
     * 1. the time of the simulation
     *    <code>DateTime simulationDateTime = systemEvent.getDateTime();</code>
     * 2. the bank agent's account information
     *    <code>AccountData myAccount = getAccount();</code>
     * 3. the list of non-sent transactions owned by bank agent
     *    <code>TransactionQueue myTransactions = getTransactionQueue();</code>
     * </pre>
     *
     * and decide wether or not to send agent's transaction(s)
     * to the settlement process.
     */
    @Override
    public void process(SystemEvent systemEvent) {
        // System event typically has a transaction attached
        Transaction transaction = getTransaction(systemEvent);
        logEvent(systemEvent, transaction);

        if (systemEvent.getDateTime().getTime() < beActivaAfterTime) {
            // Be passive, do nothing yet
            return;
        }

        switch (systemEvent.getType()) {
        case SystemEvent.AGENT_WAKEUP:
            // The system sends a wake up call based on bank agents
            // transactions' when those are supposed to be processed thus
            // this agent has the opportunity to react or do nothing on
            // that moment.
            // Other wake up calls may occur in case created by this
            // or other agents but the name of the game is the same:
            // react or not.
            if (getTransactionQueue().isEmpty()) {
                // Nothing to process
                break;
            }
```

```
            if (transaction == null) {
                // NOTE!
                // No transaction attached, this might be user defined
                // wake up event.
                // For now just log a warning:
                logger.warn(String.format("Received agent wake up event at"
                + " %s-%s with no transaction attached.",
                    systemEvent.getDate(), systemEvent.getTime()));
            } else {
                //
                if (isSettleable(systemEvent, transaction)) {
                    // Log release
                    logEvent(systemEvent, "Release",
                        String.format("Release tranId %d for settlement",
                        transaction.getId()));
                    sendPaymentForSettlement(transaction);
                }
            }
            break;
        case SystemEvent.AGENT_TRANSACTION_HAS_SETTLED:
            // The settlement confirmation messenger algorithm
            // sends these system events after a transaction has been booked
            // for the account having the settled transaction attached
            // i.e. the attribute transaction is set.
            // At this point the account's position has changed and again
            // the agent may or may not react.
            break;
        case SystemEvent.AGENT_ALERT_END_OF_DAY:
            // This is an alert message on approaching end-of-day event
            // upon which the agent may or may not react.
            // This implementation decides to send all its non-sent transactions
            // to the settlement process.
            sendPaymentsForSettlement(systemEvent,
                String.valueOf(systemEvent.getType()));
            break;
        case SystemEvent.AGENT_END_OF_DAY:
            // On end-of-day event the agent may send its' transactions
            // to system which will depending on system setup
            // 1. add the transactions to statistics or
            // 2. move the transaction to next day for further processing.
            break;
        default:
            // The execution should never fall here.
            break;
        }
    }

    /**
     * Determine here whether the transaction is to be released or
     * suspended by the bank agent
     */
    @Override
    public boolean isSettleable(SystemEvent systemEvent,
        Transaction transaction) {
        // This is a simple example of logics to determine
        // wether or not to send a transaction to be settled.
        if (getAccount().getBalance() > transaction.getValue()) {
            return true;
        }
        return false;
    }
}
```

IV. HTTP API examples

In below HTTP API examples the commands contain the following parts:

Part	**Value**	**Mandatory / Optional**
HTTP method	GET, POST PUT DELETE	M
Request path	Example: /projects/{name}/simulations where path parameters are identified with surrounding curly braces {}	M
Request body		O
HTTP response	200 - OK 201 - Created 202 - Accepted 400 - Bad Request	M
Application response	Typically a JSON object	M

1. Template methods

Templates are used in importing data from CSV files to Simulator database. The possible template types for data input are: PART, TRAN, DBAL, ICCL, BLIM, RSRV, EVNT. After running the Simulator the first time Simulator creates a set of default Templates to the database table "pss2_systemdb.temp".

GET /templates
List all existing templates.

Path parameters: NA
Request body: NA
HTTP response: `200`
Response: JSON object
Example:


```
[
    {
        "id": 0,
        "created": null,
        "modified": null,
        "templateId": "ACST-ALL",
        "type": "ACST",
        "skipFirstNRows": 0,
        "skipLastNRows": 0,
        "templateSetting": " , 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11,
        12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26,
        27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41,
        42, 43, 44, 45, 46, 47, 48, 49",
        "templateFields": null
    },
    {
        "id": 0,
        "created": null,
```

```
                    "modified": null,
                    "templateId": "ACST-EMP",
                    "type": "ACST",
                    "skipFirstNRows": 0,
                    "skipLastNRows": 0,
                    "templateSetting": " , , , , , , , , , , , , , , , , , , , ,
                    , , , , , , , , , , , , , , , , , , , , , , , , , , , , , ",
                    "templateFields": null
                },
                {
                    "id": 0,
                    "created": null,
                    "modified": null,
                    "templateId": "AVST-ALL",
                    "type": "AVST",
                    "skipFirstNRows": 0,
                    "skipLastNRows": 0,
                    "templateSetting": " , 1, 2, 3, 4, 5, 6, 7, 8",
                    "templateFields": null
                }, ...
                ]
```


GET …/templates/type/TRAN

List all existing templates of type TRAN.

Parameters: NA
Request body: NA
HTTP response: `200`
Response: JSON object
Example:


```
                [
                {
                    "id": 0,
                    "created": null,
                    "modified": null,
                    "templateId": "ccpData",
                    "type": "TRAN",
                    "skipFirstNRows": 2,
                    "skipLastNRows": 0,
                    "templateSetting": " , , 1, 2, 3, 14, , 4, 5, , 6, 7, , , ,
, , , , 9, 10, 11, 8, 12, , , , , , , , , , 13, ",
                    "templateFields": null
                },
                {
                    "id": 0,
                    "created": null,
                    "modified": null,
                    "templateId": "example",
                    "type": "TRAN",
                    "skipFirstNRows": 2,
                    "skipLastNRows": 0,
                    "templateSetting": " , , 1, 2, 3, 4, , 5, 6, , 7, 8, 9, , ,
, , , , , , , , , , , , , , , , , , 10, ",
                    "templateFields": null
                }, ...
                ]
```


POST /templates

Adds a template to the system database. The example under is for adding a PART template.

Path parameters: NA
Request body: JSON

```
{
    "templateId": "src-1",
    "type": "PART",
    "skipFirstNRows": 2,
    "skipLastNRows": 0,
    "templateSetting": " , , 1, 2, , 3, , , , , , , , , , , , , , ",
}
```

HTTP response: `201`
Response: JSON object
Example:

```
{
"id": 0,
"created": null,
"modified": null,
"templateId": "src-1",
"type": "PART",
"skipFirstNRows": 2,
"skipLastNRows": 0,
"templateSetting": " , , 1, 2, , 3, , , , , , , , , , , , , , ",
"templateFields": [
    {
    "name": "P_PARTICID",
    "description": "Participant ID",
    "inputFieldIndex": 1,
    "mandatory": true
    },
    {
    "name": "P_ACCOUNID",
    "description": "Account ID",
    "inputFieldIndex": 2,
    "mandatory": true
    },
    {
    "name": "P_ACCOTYPE",
    "description": "Account type",
    "inputFieldIndex": 0,
    "mandatory": false
    },
    {
    "name": "P_FULLNAME",
    "description": "Full name",
    "inputFieldIndex": 3,
    "mandatory": false
    },
    {
    "name": "P_SHORTACR",
    "description": "Short acronym",
    "inputFieldIndex": 0,
    "mandatory": false
    },
    {
    "name": "P_ACCONAME",
    "description": "Account name",
    "inputFieldIndex": 0,
    "mandatory": false
    },
    {
```

```
                    "name": "P_SETINSYS",
                    "description": "Settles in system",
                    "inputFieldIndex": 0,
                    "mandatory": false
                    },
                    {
                    "name": "P_SETONPAR",
                    "description": "Settles on participant",
                    "inputFieldIndex": 0,
                    "mandatory": false
                    },
                    {
                    "name": "P_SETONACC",
                    "description": "Settles on account",
                    "inputFieldIndex": 0,
                    "mandatory": false
                    },
                    {
                    "name": "P_LIQFRSYS",
                    "description": "Liquidity injection from system",
                    "inputFieldIndex": 0,
                    "mandatory": false
                    },
                    {
                    "name": "P_LIQFRPAR",
                    "description": "Liquidity injection from participant",
                    "inputFieldIndex": 0,
                    "mandatory": false
                    },
                    {
                    "name": "P_LIQFRACC",
                    "description": "Liquidity injection from account",
                    "inputFieldIndex": 0,
                    "mandatory": false
                    },
                    {
                    "name": "P_LIQINJVA",
                    "description": "Participant/account specific liquidity
injection value",
                    "inputFieldIndex": 0,
                    "mandatory": false
                    },
                    {
                    "name": "P_USERCOD1",
                    "description": "User defined code 1",
                    "inputFieldIndex": 0,
                    "mandatory": false
                    },
                    {
                    "name": "P_USERCOD2",
                    "description": "User defined code 2",
                    "inputFieldIndex": 0,
                    "mandatory": false
                    },
                    {
                    "name": "P_USERCOD3",
                    "description": "User defined code 3",
                    "inputFieldIndex": 0,
                    "mandatory": false
                    },
                    {
                    "name": "P_USERCOD4",
                    "description": "User defined code 4",
                    "inputFieldIndex": 0,
                    "mandatory": false
```

```
    },
    {
    "name": "P_USERCOD5",
    "description": "User defined code 5",
    "inputFieldIndex": 0,
    "mandatory": false
    }
]
}
```

2. Project methods

After installation the Simulator database has no content and the first step is to create a project. Basically a project is container that holds data for a specific study. A project is a nice way to isolate different studies in that the data of a project is independent from another.

POST /projects

Creates a new project to the Simulator. It adds a new project schema to the database. In other terms it creates a new project specific database. The method returns the created project data. Please notice that to create a project, it is enough to provide only the project name attribute in the JSON object.

Path parameters: NA

Request body: JSON object

Example:

```
{
"name": "myproj"
}
```

HTTP response: `200`

Response: JSON object

Example:

```
{
"id": 0,
"created": "2020-12-02T14:36:05",
"modified": "2020-12-02T14:36:05",
"name": "myproj",
"databaseLocation": "C:/Program Files/MariaDB 10.2/data/myproj",
"inputFolder": "C:/BoF-PSS/P_myproj/INPUT/",
"outputFolder": "C:/BoF-PSS/P_myproj/OUTPUT/",
"errorFolder": "C:/BoF-PSS/P_myproj/ERRORLIST/",
"outputReportFolder": "C:/BoF-PSS2/P_myproj/OUTPUT_REPORTS/",
"networkFolder": "C:/BoF-PSS/P_myproj/NETWORKS/",
"networkReportFolder": "C:/BoF-PSS2/P_myproj/NETWORK_REPORTS/",
"location": "C:/BoF-PSS/P_myproj",
"size": null,
"default": true
}
```

GET /projects

Returns a list of all projects found in the Simulator database in JSON format.

Path parameters: NA
Request body: NA
HTTP response `200`
Response: JSON object
Example:


```
[
{
"id": 0,
"created": "2020-12-02T14:36:05",
"modified": "2020-12-02T14:36:05",
"name": "myproj",
"databaseLocation": "C:/Program Files/MariaDB 10.2/data/myproj",
"inputFolder": "C:/BoF-PSS/P_myproj/INPUT/",
"outputFolder": "C:/BoF-PSS/P_myproj/OUTPUT/",
"errorFolder": "C:/BoF-PSS/P_myproj/ERRORLIST/",
"outputReportFolder": "C:/BoF-PSS2/P_myproj/OUTPUT_REPORTS/",
"networkFolder": "C:/BoF-PSS/P_myproj/NETWORKS/",
"networkReportFolder": "C:/BoF-PSS2/P_myproj/NETWORK_REPORTS/",
"location": "C:/BoF-PSS/P_myproj",
"size": 255634,
"default": true
}
]
```


DELETE /projects/{name}

Removes the named project and all its contents from the database.

Path parameters:
name: Project name (like"myproject”)
Request body: NA
HTTP response: `200`
Response: text
Example:
"SUCCESS"

3. System methods

Systems belong to projects. These operations will require a project to exist.

POST /projects/{name}/systems

Path parameters:
name: Project name
Example: "mysystem" or "sys1"
Request body: JSON object
Example:


```
{
"name": "sys1"
}
```


HTTP response: `201`
Response: JSON object

Example:

```
{"id":1,
 "created":null,
 "modified":"2020-12-03T10:39:16",
 "name":"sys1",
 "datasets":null}
```

4. System dataset methods

A system in a project must have at least one system data set.

POST /projects/{name}/systemdatasets

This method creates a system dataset for the system.

Path parameters:

name: Project name

Example: "myproj"

Request body: JSON object

Example:

```
{
"systemId": 1,
"datasetId": "ds",
"name": "sysDs",
"description": "System data set",
"creditAvailability": "ACCORDING_TABLE",
"transferBalances": false,
"bilateralLimitUse": false,
"transferTransactions": "TRANSFER_TO_NEXTDAY",
"openingTime": 70000,
"closingTime": 180000,
"systemAlgorithms": [
    {
    "algorithmName": "ENBASIC1",
    "type": "ENT",
    "parameters": "is entry settlement enabled?41][is FIFO
enabled?41"
    },
    {
    "algorithmName": "SEBASIC1",
    "type": "SET",
    "parameters": ""
    },
    {
    "algorithmName": "ENDRTGS1",
    "type": "END",
    "parameters": ""
    }
],
"editable": true
}
```

HTTP response: `201`

Response: JSON object

Example:

```
{
"id": 1,
"created": null,
"modified": "2020-12-03T11:41:48",
"systemId": 1,
```

```
                "datasetId": "ds",
                "name": "sysDs",
                "systemName": "sys1",
                "description": "System data set",
                "type": "RTGS",
                "creditAvailability": "ACCORDING_TABLE",
                "transferBalances": false,
                "bilateralLimitUse": false,
                "transferTransactions": "TRANSFER_TO_NEXTDAY",
                "openingDate": 0,
                "openingTime": 70000,
                "closingDate": 0,
                "closingTime": 180000,
                "systemAlgorithms": [
                    {
                    "id": 9,
                    "created": null,
                    "modified": "2020-12-03T11:41:48",
                    "systemId": 1,
                    "datasetId": "ds",
                    "algorithmName": "ENDRTGS1",
                    "systemAlgorithmId": 0,
                    "type": "END",
                    "parameters": null,
                    "teaAlgorithmId": null,
                    "teaAlgorithmParameters": null,
                    "parallelProcessingIndicator": 0,
                    "algorithmParameters": null,
                    "selected": true
                    },
                    {
                    "id": 8,
                    "created": null,
                    "modified": "2020-12-03T11:41:48",
                    "systemId": 1,
                    "datasetId": "ds",
                    "algorithmName": "SEBASIC1",
                    "systemAlgorithmId": 0,
                    "type": "SET",
                    "parameters": null,
                    "teaAlgorithmId": null,
                    "teaAlgorithmParameters": null,
                    "parallelProcessingIndicator": 0,
                    "algorithmParameters": null,
                    "selected": true
                    },
                    {
                    "id": 7,
                    "created": null,
                    "modified": "2020-12-03T11:41:48",
                    "systemId": 1,
                    "datasetId": "ds",
                    "algorithmName": "ENBASIC1",
                    "systemAlgorithmId": 0,
                    "type": "ENT",
                    "parameters": "is entry settlement enabled?41][is FIFO
enabled?41",
                    "teaAlgorithmId": null,
                    "teaAlgorithmParameters": null,
                    "parallelProcessingIndicator": 0,
                    "algorithmParameters": null,
                    "selected": true
                    }
                ],
                "editable": true,
```

```
                "transferTransactionsTypes": [
                    "TRANSFER_TO_NEXTDAY",
                    "DELETE_INCLUDE_STATS",
                    "DELETE_EXCLUDE_STATS",
                    "FORCE_SETTLEMENT_ON_EOD"
                ],
                "creditAvailabilityTypes": [
                    "ACCORDING_TABLE",
                    "NO_AVAILABLE",
                    "AVAILABLE_WITHOUT_LIMITS"
                ],
                "systemTypes": [
                    "DNS",
                    "CNS",
                    "RTGS"
                ]
                }
```


5. File methods

Next the input data CSV files are uploaded to the project folder on application server before importing the file content to database.

POST /projects/{name}/files/upload
This method creates a system dataset for the system.

Path parameters:
name: project name
Example: "myproj"
Request body: multipart/form-data

HTTP response: `200`
Response: JSON object
Example:


```
                {
                "fileName": "part.csv",
                "fileDownloadUri":
"http://localhost:8080/downloadFile/part.csv",
                "fileType": "application/octet-stream",
                "size": 212365,
                "message": null,
                "status": null
                }
```


NOTE!
In chapter 4.3, among examples there is a example how to upload files using CURL command.

6. Dataset methods

Next import the necessary input data sets to the corresponding database table.

Please notice that in this phase the file upload commads are expected beeing run thus the files are waiting to be imported in project's folder on application server side.

POST /projects/{name}/datasets/import/{isWaitToComplete}

This method creates a system dataset for the system.
This command can be run for each required input type:
PART, TRAN, DBAL, ICCL, BLIM, RSRV, EVNT

Parameters:
name: Project name. Example: "myproj"
isWaitToComplete: Use value true to wait and receive the result of the method With value false, the methods returns without waiting the import method to complete. Example: "true"
Request body: JSON object
Example:

```
{
"systemId": 1,
"entityType": "PART",
"templateName": "example",
"filename": "part.csv",
"datasetName": "ds",
"mapToPartDataset": null
}
```

HTTP response:
`200` - isWaitToComplete parameter is set to true
`202` - isWaitToComplete parameter is set to false
Response:
text
Example:
true

7. Simulation methods

After all needed data sets are in place in database a simulation can be created.

POST /projects/{name}/simulations
This method creates a simulation.

Parameters:
name: Project name
Example: "myproj"
Request body: JSON object
Example:

```
{
"parentId": 0,
"simRunId": "sim1",
"name": "simulation one",
"description": "Enter description...",
```

```
"outputTables": "true,true,true,false,false,false,false,false",
"systemIDs": "t2",
"systemDatasetSelection": "ds",
"partDatasetSelection": "ds",
"tranDatasetSelection": "ds",
"dbalDatasetSelection": "null",
"icclDatasetSelection": "null",
"blimDatasetSelection": "null",
"rsrvDatasetSelection": "null",
"evntDatasetSelection": "null",
"agentConfigurationFilePath": "",
"submissionAlgorithmId": "SUFIFOPR",
"algorithmType": "SUB",
"parametersValues": null,
}
```


HTTP response: `201`

Response: JSON object

Example:


```
{
"id": 1,
"created": null,
"modified": "2020-12-03T15:53:13",
"runningMode": "EXECUTE",
"parentId": 0,
"simRunId": "sim1",
"name": "simulation one",
"description": "Enter description...",
"processDate": 0,
"processTime": 0,
"duration": 0,
"outputTables": "true,true,false,false,false,false,false,false",
"systemIDs": "sys1",
"systemDatasetSelection": "ds",
"partDatasetSelection": "ds",
"tranDatasetSelection": "ds",
"dbalDatasetSelection": "null",
"icclDatasetSelection": "null",
"blimDatasetSelection": "null",
"rsrvDatasetSelection": "null",
"evntDatasetSelection": "null",
"agentConfigurationFilePath": null,
"numberOfSystems": 0,
"numberOfAccounts": 0,
"numberOfTransactions": 0,
"submissionAlgorithmId": "SUFIFOPR",
"algorithmType": "SUB",
"parametersValues": null,
"run": null,
"selectedOutTables": [
    "true",
    "true",
    "true",
    "false",
    "false",
    "false",
    "false",
    "false"
],
"selectedSystemIds": [
    "sys1"
],
"selectedSystemDatasets": [
    [
    "ds",
```

```
                    "ds",
                    "ds",
                    "",
                    "",
                    "",
                    "",
                    ""
                    ]
                ],
                "crossCheckDone": false
                }
```


GET /projects/{name}/simulations/run/{id}/{isWaitToComplete}

This method runs the simulation indicated by the id.

Parameters:
name: Project name
Example: "myproj"
id: simulation id
Example: "1"
isWaitToComplete: Use value true to wait and receive the result of the method With value false then methods return without waiting the method to complete. Example: "true"
Request body: NA
HTTP response:
`200` - isWaitToComplete parameter is set to true
`202` - isWaitToComplete parameter is set to false
Response:
JSON object
Example:


```
                {
                "id": 1,
                "created": null,
                "modified": "2020-12-04T12:18:33",
                "runningMode": "EXECUTE",
                "parentId": 0,
                "simRunId": "sim1",
                "name": "simulation one",
                "description": "Enter description...",
                "processDate": 20201204,
                "processTime": 121833430000,
                "duration": 443,
                "outputTables": "true,true,true,false,false,false,false,true",
                "systemIDs": "sys1",
                "systemDatasetSelection": "ds",
                "partDatasetSelection": "ds",
                "tranDatasetSelection": "ds",
                "dbalDatasetSelection": "null",
                "icclDatasetSelection": "null",
                "blimDatasetSelection": "null",
                "rsrvDatasetSelection": "null",
                "evntDatasetSelection": "null",
                "agentConfigurationFilePath": null,
                "numberOfSystems": 0,
                "numberOfAccounts": 0,
                "numberOfTransactions": 0,
                "submissionAlgorithmId": "SUFIFOPR",
```

```
"algorithmType": "SUB",
"parametersValues": null,
"run": "2020-12-04T12:18:33",
"selectedOutTables": [
    "true",
    "true",
    "true",
    "false",
    "false",
    "false",
    "false",
    "true"
],
"selectedSystemIds": [
    "sys1"
],
"selectedSystemDatasets": [
    [
    "ds",
    "ds",
    "ds",
    "",
    "",
    "",
    "",
    ""
    ]
],
"crossCheckDone": false
}
```


GET /projects/{name}/simulations/data/syls/{simId}

The following method call lists the SYLS output data for the given simulation id. The output data for ACST, TEST, QURE, AVST and BIST tables can be retrieved in similar manner.

Path parameters:
name: Project name
Example: "myproj"
simId: simulation id
Example: "1"
Request body:
HTTP response: `200`
Response: JSON object
Example:


```
[
{
    "id": 0,
    "systemId": 1,
    "simulationId": 1,
    "modified": null,
    "key": null,
    "businessDay": "20030512",
    "systemName": "dsEx",
    "valueInData": 7.7626218987E9,
    "valueCarriedOver": 0.0,
    "valueSubmitted": 7.7626218987E9,
    "valueSettled": 4.6236442517E8,
    "valueUnsettled": 7.30025747353E9,
```

```
                    "numberInData": 778,
                    "numberCarriedOver": 0,
                    "numberSubmitted": 778,
                    "numberSettled": 122,
                    "numberUnsettled": 656,
                    "beginningOfDayBalance": 0.0,
                    "endOfDayBalance": 0.0,
                    "averageCreditLimit": 5.453449056E7,
                    "liquidityAvailable": 5.453449056E7,
                    "absoluteCreditLimitUsage": 1.702447774E7,
                    "relativeCreditLimitUsage": 0.02,
                    "totalLiquidityAvailable": 9.8162083003E8,
                    "lowerBoundOfLiquidity": 4.110843018E8,
                    "maxQueueValue": 7.30025747353E9,
                    "avgQueueValue": 2.0759429298E8,
                    "avgQueueLength": 0,
                    "numberOfQueuedTransactions": 656,
                    "totValOfQueuedTransactions": 7.30025747353E9,
                    "queueStopTime": 0,
                    "avgTimeOfSettlement": 0,
                    "liqUsageIndicCollateral": 0.06,
                    "liqUsageIndicRepo": 0.13,
                    "settlementDelay": 0.0,
                    "settings": "",
                    "maxCreditUsage": 0.0
                }
                ]
```


8. Analysis methods

Analysis requires a simulation that will be used as a benchmark to be defined.

POST /projects/{name}/analysis

This method creates an analysis.

Parameters:

name: Project name

Example: "myproj"

Request body: JSON object

Example:


```
                {
                "name": "a1",
                "simId": 1,
                "simRunId": "sim1",
                "description": "a1",
                "rule": "FAIL_BY_PARTICIPANT",
                "accounts": [
                    {
                    "participantId": "13",
                    "accountId": "13"
                    },
                    {
                    "participantId": "17",
                    "accountId": "17"
                    }
                ]
                }
```

HTTP response: `201`
Response: JSON object
Example:


```
{
"id": 1,
"created": null,
"modified": "2020-12-04T14:57:04",
"name": "a1",
"simId": 1,
"simRunId": "sim1",
"type": 0,
"status": "INITIAL",
"description": "a1",
"rule": "FAIL_BY_PARTICIPANT",
"fileFilterName": null,
"icclScreen": 0,
"dbalScreen": 0,
"accounts": [
    {
    "id": 0,
    "created": null,
    "modified": null,
    "analysisId": 1,
    "participantId": "13",
    "accountId": "13"
    },
    {
    "id": 0,
    "created": null,
    "modified": null,
    "analysisId": 1,
    "participantId": "17",
    "accountId": "17"
    }
]
}
```


GET /projects/{name}/analysis/run/{id}/{isWaitToComplete}

This method runs the analysis of provided id.

Parameters:
name: Project name
Example: "myproj"
id: analysis id
Example: "1"
isWaitToComplete: Use value true to wait and receive the result of the method. With value false then methods return without waiting the method to complete. Example: "true".
Request body: NA
HTTP response:
`200` - isWaitToComplete parameter is set to true
`202` - isWaitToComplete parameter is set to false
Response: JSON object
Example:


```
{
```

```
"id": 1,
"created": null,
"modified": "2020-12-04T14:57:04",
"name": "a1",
"simId": 1,
"simRunId": "sim1",
"type": 0,
"status": "RUN",
"description": "a1",
"rule": "FAIL_BY_PARTICIPANT",
"fileFilterName": null,
"icclScreen": 0,
"dbalScreen": 0,
"accounts": [
    {
    "id": 0,
    "created": null,
    "modified": null,
    "analysisId": 1,
    "participantId": "13",
    "accountId": "13"
    },
    {
    "id": 0,
    "created": null,
    "modified": null,
    "analysisId": 1,
    "participantId": "17",
    "accountId": "17"
    }
]
}
```


GET /projects/{name}/analysis/runReport/{id}/{isWaitToComplete}

This method runs the analysis report for the provided analysis id. The report is stored as two files:

a. in CSV format that contains the report data,

b. in XLSM format that contains the Excel macro that on file open phase reads the CSV data and loads it to a pivot table and thus it's capable to show Excel diagrams.

Parameters:

name: Project name

Example: "myproj"

id: analysis id

Example: "1"

isWaitToComplete: Use value true to wait and receive the result of the method. With value false then methods return without waiting the method to complete. Example: "true"

Request body:

HTTP response:

`200` - isWaitToComplete parameter is set to true

`202` - isWaitToComplete parameter is set to false

Response: JSON object

Example:

```
                {
                "id": 1,
                "created": null,
                "modified": "2020-12-04T15:43:02",
                "name": "a1",
                "simId": 1,
                "simRunId": "sim1",
                "type": 0,
                "status": "REPORTED",
                "description": "a1",
                "rule": "FAIL_BY_PARTICIPANT",
                "fileFilterName": null,
                "icclScreen": 0,
                "dbalScreen": 0,
                "accounts": [
                    {
                    "id": 0,
                    "created": null,
                    "modified": null,
                    "analysisId": 1,
                    "participantId": "13",
                    "accountId": "13"
                    },
                    {
                    "id": 0,
                    "created": null,
                    "modified": null,
                    "analysisId": 1,
                    "participantId": "17",
                    "accountId": "17"
                    }
                ]
                }
```


GET /projects/{name}/analysis/download/{id}/{type}

This method is used to download the analysis report files. Download the CSV file first and then only the macro file. After both files have been downloaded you can open the .xlsm file.

Parameters:
- name: Project name
  - Example: "myproj"
- id: analysis id
  - Example: "1"
- type: Must be either csv or xlsm
  - Example: "csv"

Request body:
HTTP response: NA
Response: Textual CSV data or Excel xlsm format (XML)
Example:


```
BenchScenario;SimId;SimRunId;SystemId;failingParty;ParticipantId;AccountId;Busin
essDay;BoDBalance;EoDBalance;MinBalance;EoDCreditLimit;CreditLimitMaxUsage;Settl
edCount;SettledValue;SentUnstCountDirect;SentUnstValueDirect;SentUnstSystemicEff
ectCount;SentUnstSystemicEffectValue;ReceivedPaymentsCount;ReceivedPaymentsValue
;ReceivedUnstCountDirect;ReceivedUnstValueDirect;ReceivedUnstSystemicEffectCount
;ReceivedUnstSystemicEffectValue;ReceivedPaymentsDiffValue;LB;LBDiff;UB;MaxUpper
```

```
Bound;MinLiquidityDeterioration;MaxLiquidityDeterioration;SettlementDelay;Settle
mentDelayDiff;WeightedAvgReceivingTime;WeightedAvgReceivingTimeDiff;WeightedAvgS
endingTime;WeightedAvgSendingTimeDiff

Bench;1;sim1;1;;1;1;20030512;0;370379.4;0;0;0;5;444037;;;47;75145110.7;5;814416.
4;;;29;64854605.8;0;0;0;62487362.4;75589147.7;;;0;0;00:05:13;00:00:00;00:02:28;0
0:00:00
                Bench;1;sim1;1;;10;10;20030512;0;264063.2;-
105983.4;201802;52.5;3;105983.4;;;13;15034115.6;3;370046.6;;;12;14971854.3;0;0;0
;12356069.4;15140098.9;;;0;0;00:10:09;00:00:00;00:02:57;00:00:00
                ...
```

9. CURL API example

Here is a CURL example with which everything is performed from project creation to the running of an analysis. The commands of the example can be used with Windows command prompt using the CURL utility.

Performed operations in the CURL example:

1. create a project
2. create a system to the project
3. create a system data set
4. upload part and tran CSV files
5. Import the files to the project as data sets
6. create a simulation based on above data
7. run the simulation
8. creates an analysis
9. run the analysis

The contents of the needed JSON files for each CURL command are displayed after the CURL commands.
Please note that in order to perform the example:

a. The CURL executable is set to your Windows path.
b. The below example lines expect that the json files referred as `@[file name].[file extension]` are found on current run folder.
c. Similarly the below part.csv and tran.cvs files are found on current run folder.

The used CSV files are from the example 1 distributed and included within the Simulator installation.

```
:: --- Start of Windows CMD script >>> ---
set pssDomain=localhost:8080
set pssProj=myProj

:: Delete project in case it exists
curl -X DELETE -H "Content-Type: application/json"
 http://%pssDomain%/projects/%pssProj%
:: Create project
curl -X POST -H "Content-Type: application/json" http://%pssDomain%/projects
 -d @project.json
```

```
:: Create system
curl -X POST -H "Content-Type: application/json" http://%pssDomain%/projects
 -d @system.json

:: Create system data set
curl -X POST -H "Content-Type: application/json"
 http://%pssDomain%/projects/%pssProj%/systemdatasets -d @systemDataset.json
:: Upload file
curl -i -X POST -H "Content-Type: multipart/form-data"
 http://%pssDomain%/projects/%pssProj%/files/upload -F "file=@part.csv"
curl -i -X POST -H "Content-Type: multipart/form-data"
 http://%pssDomain%/projects/%pssProj%/files/upload -F "file=@tran.csv"
:: Import uploaded file's data to database with wait for completion set to
 true
curl -i -X POST -H "Content-Type: application/json"
 http://%pssDomain%/projects/%pssProj%/datasets/import/true -d
 @importPartData.json
curl -i -X POST -H "Content-Type: application/json"
 http://%pssDomain%/projects/%pssProj%/datasets/import/true -d
 @importTranData.json

:: Create simulation
curl -X POST -H "Content-Type: application/json"
 http://%pssDomain%/projects/%pssProj%/simulations -d @simulation.json
:: Run simulation
curl http://%pssDomain%/projects/%pssProj%/simulations/run/1/true
:: Get simulation SYLS output data
curl http://%pssDomain%/projects/%pssProj%/simulations/data/syls/1

:: Create analysis
curl -i -X POST -H "Content-Type: application/json"
 http://%pssDomain%/projects/%pssProj%/analysis -d @data/analysis.json
:: Run analysis
curl http://%pssDomain%/projects/%pssProj%/analysis/run/1/true
:: Run analysis report
curl http://%pssDomain%/projects/%pssProj%/analysis/runReport/1/true
:: Download the report in CSV format
curl http://%pssDomain%/projects/%pssProj%/analysis/download/1/csv >
 analysis.csv
curl http://%pssDomain%/projects/%pssProj%/analysis/download/1/xlsm >
 analysis.xlsm
:: --- <<< Windows CMD script end ---
```

CURL example JSON files

Here are the JSON file contents used in the example above.

project.json

```
{
"name": "myProj",
}
```

system.json

```
{
"name": "sys1"
}
```

systemDataset.json

Below the system id must correspond to existing system.

```
{
"systemId": 1,
"datasetId": "dsEx",
"name": "dsEx",
"description": "System data set",
"creditAvailability": "ACCORDING_TABLE",
"transferBalances": false,
"bilateralLimitUse": false,
"transferTransactions": "DELETE_INCLUDE_STATS",
"openingTime": 70000,
"closingTime": 190000,
"systemAlgorithms": [
    {
    "algorithmName": "ENBASIC1",
    "type": "ENT",
    "parameters": "is entry settlement enabled?41][is FIFO enabled?41"
    },
    {
    "algorithmName": "SEBASIC1",
    "type": "SET",
    "parameters": ""
    },
    {
    "algorithmName": "ENDRTGS1",
    "type": "END",
    "parameters": ""
    }
],
"editable": true
}
```

importPartData.json

Below the system id must correspond to existing system and the template name must correspond the input CSV file data structure of given file name.

```
{
"systemId": 1,
"entityType": "PART",
"templateName": "example",
"filename": "part.csv",
"datasetName": "dsEx",
"mapToPartDataset": null
}
```

importTranData.json

Below the system id must correspond to existing system and the template name must correspond the input CSV file data structure of given file name. The mapToPartDataset refers above part data set.

```
{
"systemId": 1,
"entityType": "TRAN",
"templateName": "example",
"filename": "tran.csv",
"datasetName": "dsEx",
"mapToPartDataset": "dsEx"
}
```

importIcclData.json

Below the system id must correspond to existing system and the template name must correspond the input CSV file data structure of given file name. The mapToPartDataset refers above part data set.

```
{
"systemId": 1,
"entityType": "ICCL",
"templateName": "example",
"filename": "iccl.csv",
"datasetName": "dsEx",
"mapToPartDataset": "dsEx"
}
```

simulation.json

Below the system ids and data set names must correspond to existing entities in database. If a data set selection has no existing data set the data set selection is set with value "null".

```
{
"parentId": 0,
"simRunId": "sim1",
"name": "simulation one",
"description": "Enter description...",
"outputTables": "true,true,true,false,false,false,false,false",
"systemIDs": "sys1",
"systemDatasetSelection": "dsEx",
"partDatasetSelection": "dsEx",
"tranDatasetSelection": "dsEx",
"dbalDatasetSelection": "null",
"icclDatasetSelection": "dsEx",
"blimDatasetSelection": "null",
"rsrvDatasetSelection": "null",
"evntDatasetSelection": "null",
"agentConfigurationFilePath": "",
"submissionAlgorithmId": "SUFIFOPR",
"algorithmType": "SUB",
"parametersValues": null
}
```

analysis.json

Below the simulation ids must correspond to existing entities in database. If a data set selection has no existing data set

the data set selection is set with value "null".

```
{
"name": "a1",
"simId": 1,
"simRunId": "sim1",
"description": "a1",
"rule": "FAIL_BY_PARTICIPANT",
"accounts": [
    {
    "participantId": "13",
    "accountId": "13"
    },
    {
    "participantId": "17",
    "accountId": "17"
    }
]
}
```