User Manual

Prepared by
Rajmund Bocsi
Viktor Erdélyi
Gábor Ferschl
László Gulyás
Márton Iványi
Zsolt Kúti

May 2011, Budapest

Contents

Introduction

Agent-Based Modeling

Agent-based modeling is a branch of computer simulation. It models the individual, together with its imperfections (e.g., limited cognitive or computational abilities), its idiosyncrasies and personal interactions. The approach builds the model from 'the bottom-up', focusing mostly on micro rules and seeking to understand the emergence of macro behavior. Participatory simulation - a branch of agent-based simulation - is a methodology building on the synergy of human actors and artificial agents, excelling in the training and decision-making support areas. In participatory simulations some agents are controlled by users, while others are software governed.

MASS

The Multi-Agent Simulation Suite (MASS) is a software package intended to enable modelers to utilize the tools of agent-based simulation in various fields, without having to develop heavy programming skills.

MASS consists of four applications built around a simulation core. The simulation suite has its own core called the Multi-Agent Core (MAC), but it is also able to run on the popular Repast core. Being multi-core enables modelers to verify that results are core-independent, thus we plan to further develop this option. The Functional Agent-Based Language for Simulation (FABLES) is a programming language and its integrated modeling environment specially designed for creating agent-based simulations. The Model Exploration Module (MEME) is a tool that enables orchestrating experiments, managing results and has support for their analysis. The Participatory Extension (PET) is an optional web-based environment for multi-agent and participatory simulations. The fourth element of MASS, the Visualization Package does not translate into a standalone application. It consists of the various implementations of charts and visualizations used in all the other software.

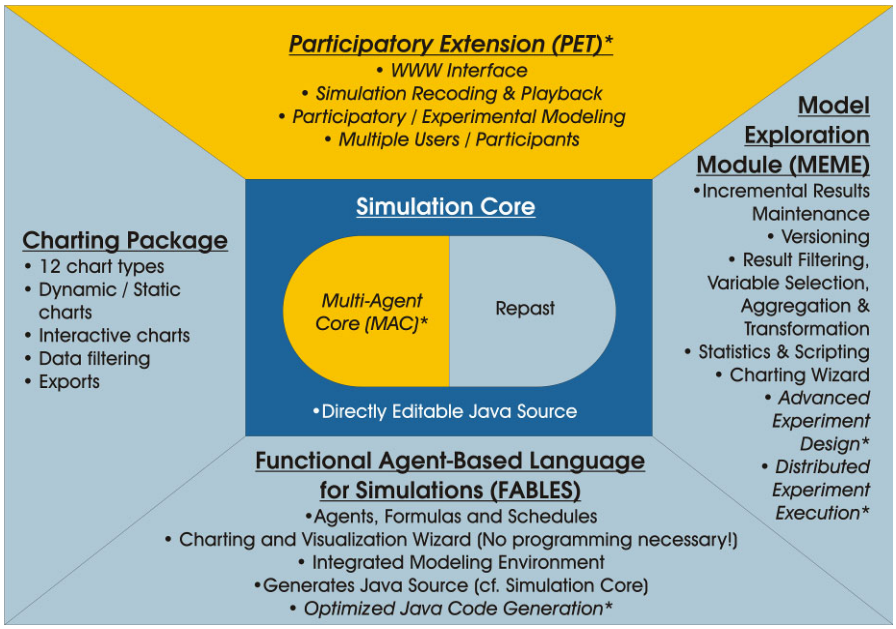


Figure 1 - Multi-Agent Simulation Suite

Model Exploration Module

This is the user manual of MEME, the *Model Exploration Module* of MASS. The word "model" in this name means any agent-based simulation program, written in either FABLES, RepastJ 3.1, NetLogo 4.1.x or pure Java, which attempts to simulate, and what

is more, to *model* a certain real-world phenomena. Such models are generally used for studying, or we can say, *exploring* the behavior of the model in different circumstances. This is done by observing various factors of the model during (or at the end of) unattended simulations, usually long and/or large number of simulations (that are called “batch run of simulations” in the computer jargon). These observations produce a huge amount of raw data, which needs to be analyzed.

MEME is a tool for dealing with such batch runs of simulations and the data produced. It allows the user to store and organize the raw data in database(s), and to create distillated (computed) tables which can be visualized on charts.

MEME can also run simulations directly (there is some support for intelligent parameter space exploration) and collect their results, moreover it interoperates with other existing statistical software like R and Matlab via CSV exporting and external program execution facilities.

These latter developments further simplify the modeling work and allow users to focus on the models themselves.

1 Before Startup

1.1 Installation

MEME is available through an installation exe file (the latest release version can be downloaded from http://mass.aitia.ai/). The installation wizard guides you through the installation process, checking for already installed versions of MEME and the appropriate Java JRE (1.6 or later, installing it if necessary), offers the installation of the RepastJ program, and asks for the folder and shortcut information required to complete the installation.

During the installation MEME is copied to the given program folder, the proper shortcuts are placed in the Start Menu and Desktop (optional) and version information registry notes are made. The database and other program files are created at the first run.

On Windows Vista or Windows 7, if you are not an administrator, it is strongly recommended to change the installation directory of MEME because Vista (or Win7) will not allow creating the database in a subdirectory of *Program Files* without proper rights.

MEME is also available in ZIP archive format for Linux and Mac OS operation systems.

1.2 System Requirements

1.2.1 Hardware

Minimum requirements: 1 GHz CPU, 512 MB RAM

Recommended: Dual-core CPU, 1 GB (or more) RAM

1.2.2 Software

Any system that runs Java JRE 1.6 is able to run MEME. The detailed requirements for each operating system are available at:

http://java.com/en/download/help/6000011000.xml

MEME runs on all systems capable of running Java, although testing on Mac or Linux is still needed.

2 Panels (The Window Menu)

As you may have noticed there are tab buttons under the menu bar. By clicking on these buttons you can activate the different parts (panels) of the user interface.

Figure 2 - Panels

The first one called *Results panel* displays the Results browser (see 4.7 The Results Browser for reference). In this panel you can work with the stored simulation results. The next is called *Views panel* on which you can work with the view tables that are described in 5 Views. The *Charts panel* is only displayable when you are editing a chart. Other panels (called *Monitor, Local Monitor* or *Downloader*) may appear here when you are running a simulation with the application (see 3.3 Local Monitor and 3.5 Monitor) or downloading/importing results from the Model Exploration Server (see 3.6 Run Simulation on the Model Exploration Server).

2.1 Layout

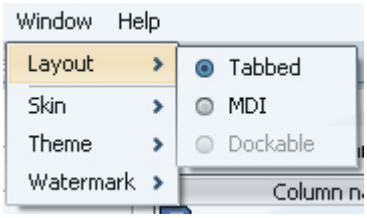


Figure 3 - Window types

The layout of these panels can be changed in the *Window* menu. MDI denotes the so-called multiple document interface, which looks like this:

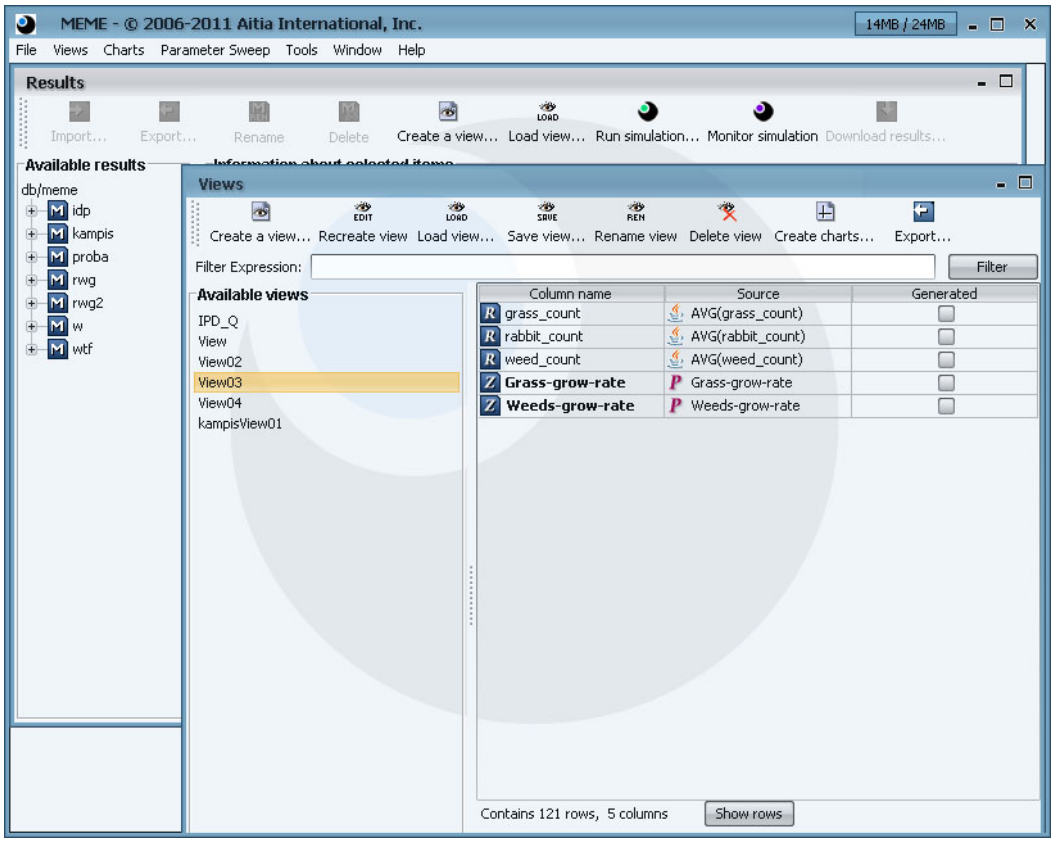


Figure 4 - MDI

2.2 Skins, Themes, Watermarks

To change the skin, theme or watermark used in the program click on the corresponding menu items in the *Window* menu.

Note that you must restart the program for the changes to take effect!

2.3 Local Menus

By right clicking on tables, lists and structure trees local menus with commands related to the given area can be invoked. See Figure 109 - Views for example.

3 Creating and Running Batch Models

The Parameter Sweep Wizard supports RepastJ/FABLES users to perform distributed mass parameter sweep experiments on a cluster or grid of computers using a point-and-click graphical user interface. Users can create batch versions of their models with the appropriate recorders and settings and run them through this application. Performing distributed parameter sweep experiments can be achieved without having to write model versions explicitly for this purpose.

The monitoring tools enable users to follow the progress of current simulations running either on the local computer or on a given cluster or grid of computers and to import the output of finished simulations to the database of MEME. It is also possible to download only these output files to the file system of your computer (in case of cluster or grid simulations).

MEME also supports simulations written in NetLogo 4.1.x or pure Java. These platforms are available as plugins and are not contained by the installer. If you need any of these platforms please contact us at our website: http://mass.aitia.ai/.

The following chapters explain the use of the wizard and the monitoring tool.

3.1 Wizard Preferences

The Parameter Sweep Wizard runs the batch models on the local computer by default. For distributed runs the default settings have to be changed in the *Preferences* dialogue. Other options and settings can also be specified here. To open the *Preferences* dialogue select the *Wizard preferences…* menu item from the *Parameter Sweep* menu. Note that this dialogue can be accessed directly from the Parameter Sweep Wizard too by pressing the *Preferences…* button on the *Model Selection* page.

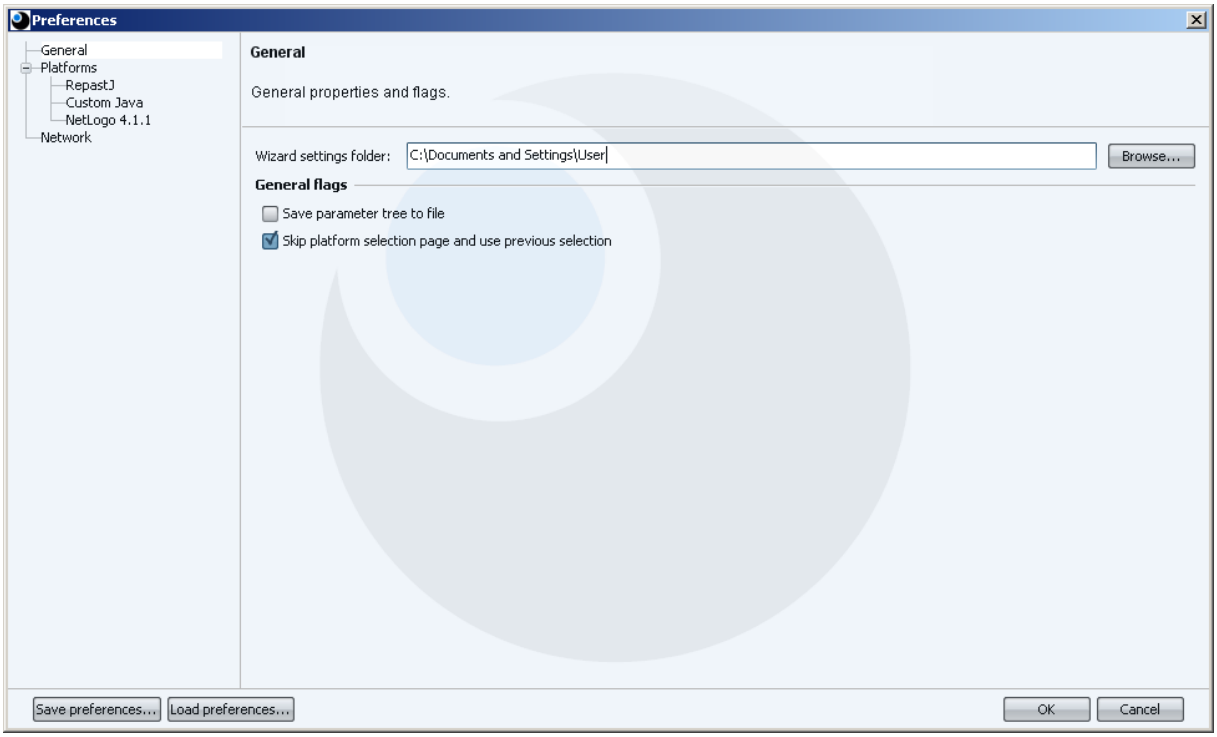


Figure 5 - Preferences dialogue, General settings

The *Preferences* dialogue has three pages (and a subpage for each known platforms): the *Network* page is dealing with distributed run settings; the *Platforms* page is for platform registration and deregistration and the *General* page is for all other common settings. Subpages contain the platform-specific settings. To switch between the pages select their names in the tree on the left side of the dialogue.

Pressing *OK* button applies the changes. In case of an error in the settings a warning message is displayed in the information panel below the title of the page.

When clicking *OK* button the application automatically saves the wizard settings, just like pressing the *Save preferences…* button does after choosing a place and a name for the preferences file. With the *Load preferences…* button you can load previously stored settings.

3.1.1 General Settings

The following settings can be made on the *General* page:

Wizard settings folder: When a model is ran the wizard automatically saves all model settings (i.e. parameter tree, recorders, scripts etc.) which allows you to use these settings later (see details in section 3.2.2.3 Load Wizard Settings). In this field you can define the location of these descriptor files. By default this is the user folder.

Save parameter tree to file: Determines whether the wizard creates a file from the specified parameter tree or not.

Skip platform selection page and use previous selection: If checked, the wizard selects the previously used platform on the *Platform selection* page and starts on the *Model selection* page automatically. However, at the first time it starts with platform selection regardless of this flag.

3.1.2 Platform Settings

This page contains information about the known and registered platforms.

MEME can work with every platforms contained by the *Known platforms* list but you have to register a platform before you can use it. This can be achieved by selecting it from the *Known platforms* list and pressing the *Edit settings* button which switches to the related subpage where you can register or deregister the platform (sometimes you have to provide some information for the registration) and other platform-specific settings can be made.

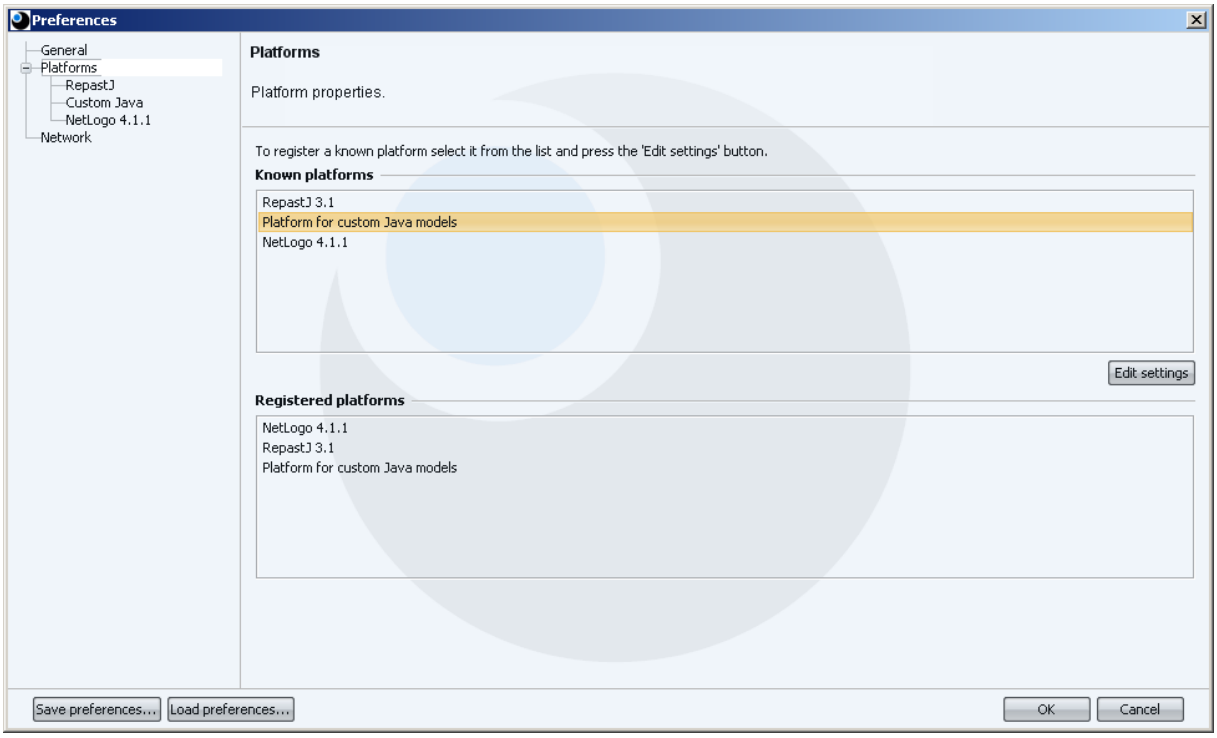


Figure 6 – Preferences dialogue, Platform settings

Each previously registered platform appears in the *Registered platforms* list. Note that RepastJ 3.1 has already registered.

3.1.2.1 RepastJ

The following settings can be made on the *RepastJ* subpage:

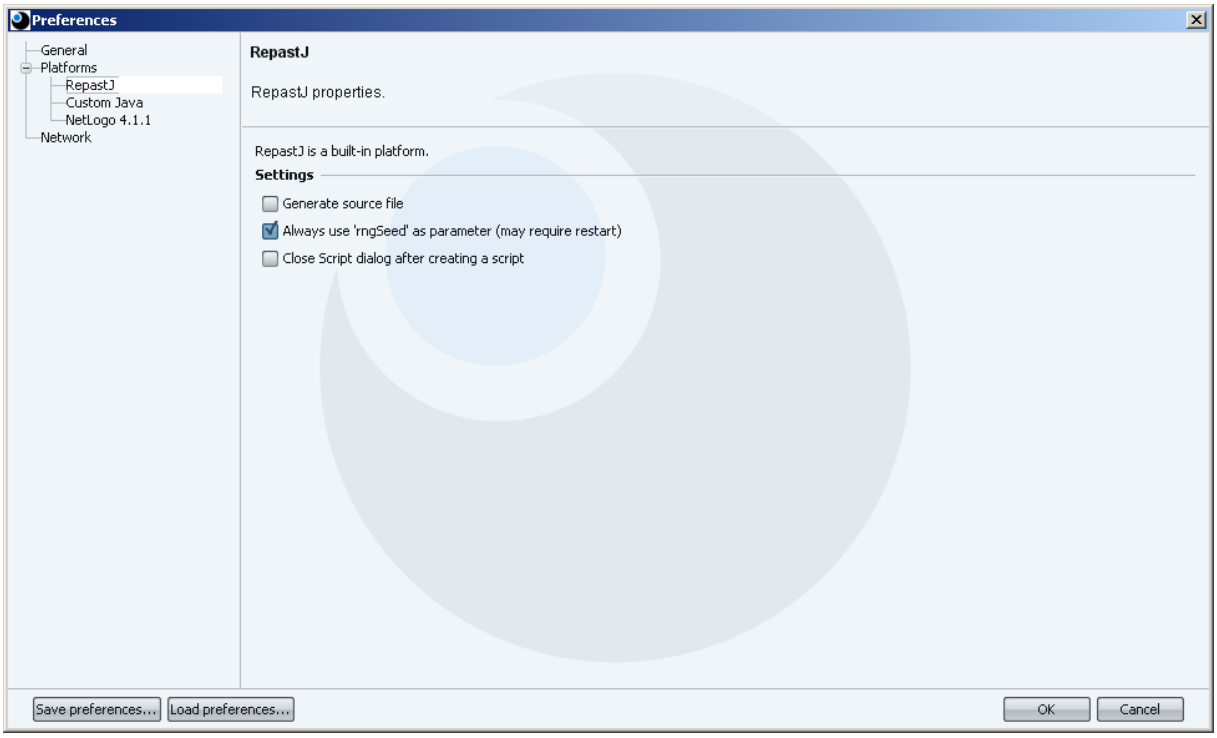


Figure 7 – Preferences dialogue, Repast J settings

Generate source file: If checked the wizard generates the source code of the batch version of the model whenever a model is generated. This source file is placed in the directory where the original model file (the *.class* file) is.

Always use 'rngSeed' as a parameter: If checked *rngSeed* (initializing value of the random number generator) becomes a parameter of the model whether is contained in it originally or not. (See details about the parameters in section 3.2.4 Parameters Page and about the *rngSeed* and the *getInitParam()* in the Repast documentation [2].)

Note: This can only be changed before model selection.

Close script dialogue after creating a script: Determines whether the *Script dialogue* is closed when a single script is created or not. In the latter case multiple scripts can be created and the dialogue must be closed manually by pressing the *Close* button on it (see in section 3.2.6 Creating Data Sources).

3.1.2.2 Custom Java

You can use the parameter sweep capabilities of MEME with models written in pure Java[1] by registering this platform. To achieve this simply click *Register* button. The platform can be removed from the *Registered platforms* list by pressing the *Deregister* button.

The following other settings can be made on the *Custom Java* subpage:

Generate source file: If checked the wizard generates the source code of the batch version of the model whenever a model is generated. This source file is placed in the directory where the original model file (the *.class* file) is.

Close script dialogue after creating a script: Determines whether the *Script dialogue* is closed when a single script is created or not. In the latter case multiple scripts can be created and the dialogue must be closed manually by pressing the *Close* button on it (see in section 3.2.6 Creating Data Sources).

[1] However your model must match some conditions, see in details in section 3.8 Platform for Custom Java Models.

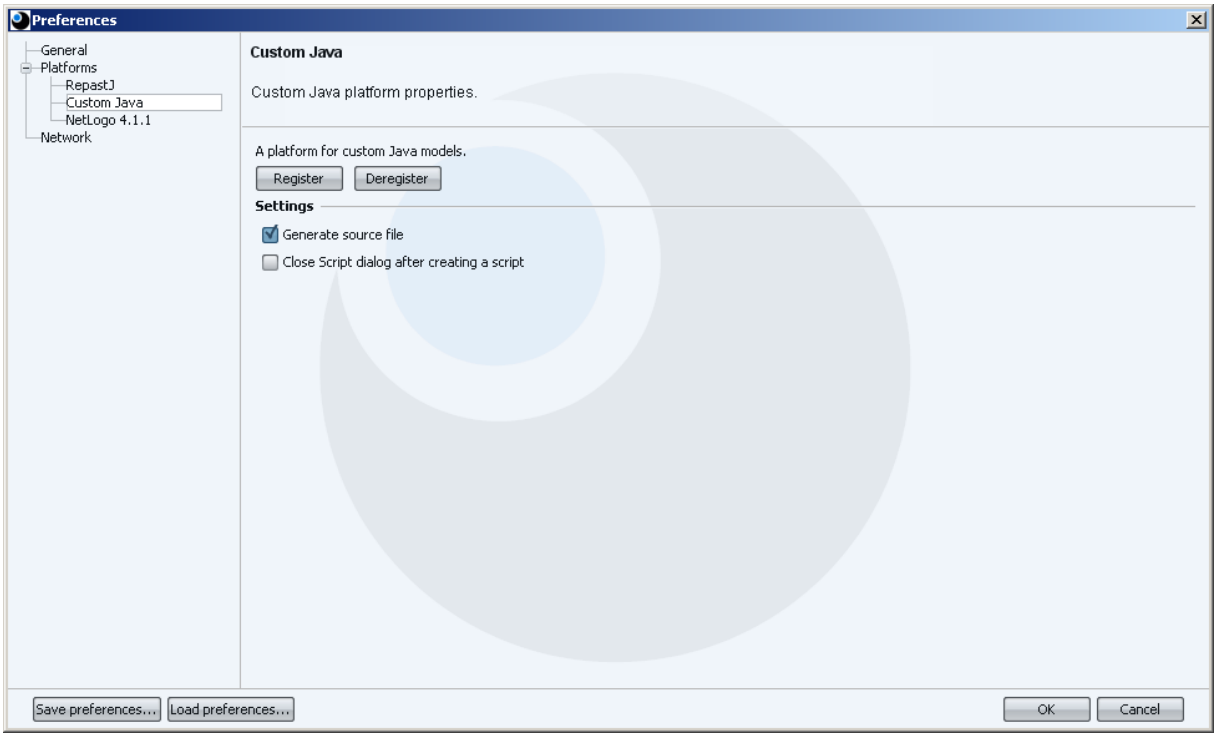


Figure 8 – Preferences dialogue, Custom Java settings

3.1.2.3 NetLogo 4.1.1

You can register NetLogo 4.1.1 platform by specifying the installation directory of NetLogo 4.1.1 and then pressing the *Register* button. Press *Deregister* button to remove the platform from MEME.

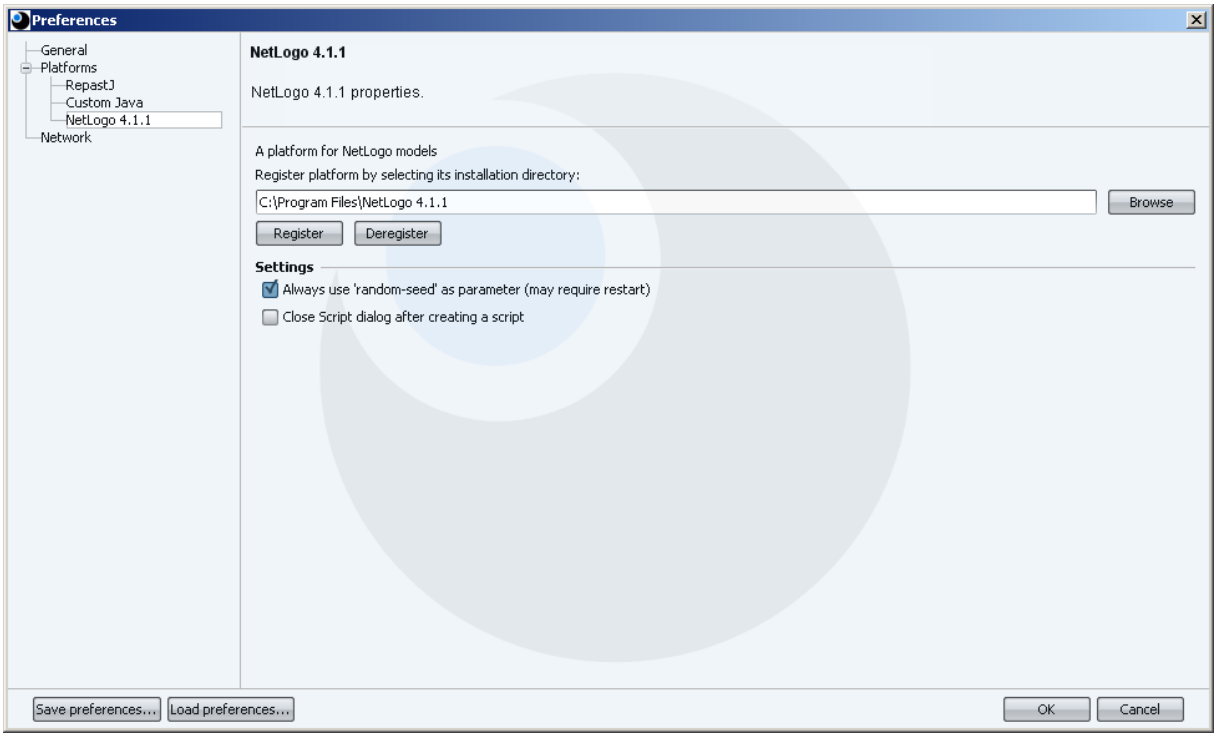


Figure 9 – Preferences dialogue, NetLogo 4.1.1 settings

The following other settings can be made on the *NetLogo 4.1.1* subpage:

Always use 'random-seed' as a parameter: If checked *random-seed* (initializing value of the random number generator) becomes a parameter of the model. (See details about

the parameters in section 3.2.4 Parameters Page and about the *random-seed* in the NetLogo documentation [11].)

Note: This can only be changed before model selection.

Close script dialogue after creating a script: Determines whether the *Script dialogue* is closed when a single script is created or not. In the latter case multiple scripts can be created and the dialogue must be closed manually by pressing the *Close* button on it (see in section 3.2.6 Creating Data Sources).

3.1.3 Network Settings

The *Network* page determines whether the runs are executed locally or remotely, on one or on multiple machines. (See Figure 10 - Preferences dialogue, Network settings)

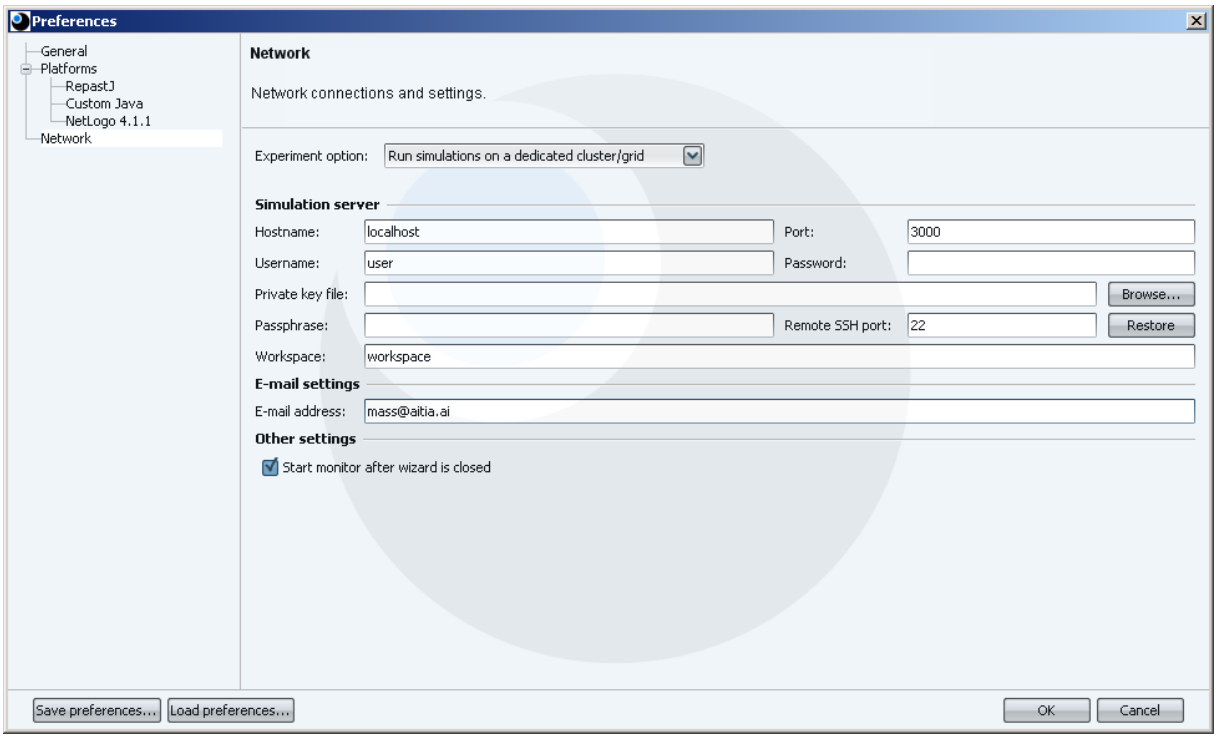


Figure 10 - Preferences dialogue, Network settings

As mentioned before the *Run simulations on local computer* option is the default setting. To switch to distributed mode select *Run simulations on a dedicated cluster/grid* from the *Experiment option* drop-down list.

Simulation server: In distributed mode the wizard runs simulations by connecting to a server application (called the *simulation server*) on a remote host. The details of the server-connection are described in this section.

- *Hostname*: The hostname (or IP-address) of the computer where the simulation server-application is located;
- *Port*: The simulation server port (default: 3000);
- *Username/Password:* The application transfers the files associated with the model via SFTP protocol. Usually this protocol requires a username and a password.
- *Private key:* Some SFTP-servers do not support password identification; they use private key files instead.
- *Passphrase:* Some SFTP-servers require a passphrase in addition to the above mentioned private key.
- *Remote SSH port:* The SSH port of the SFTP-server (default: 22).
- *Workspace*: The working directory on the remote host.

Warning: The workspace must be defined relative to the default user directory of the SFTP-server.

E-mail settings: In distributed mode e-mail notifications can be requested about simulation events (completion, error etc.). The server uses the e-mail address defined in the *E-mail settings* section. This setting is optional.

Start monitor after wizard is closed: If this option checked, the monitor tool is automatically started when running in distributed mode. It allows of the observation of distributed runs. See details about the monitor in 3.5 Monitor.

Note: If you run models on the local host, MEME shows progress automatically.

Alternatively, you can run your simulation on a cloud-based cluster for a fair price. To achieve this, choose the *Run simulations on the Model Exploration Server* experiment option from the drop-down list (see Figure 11).

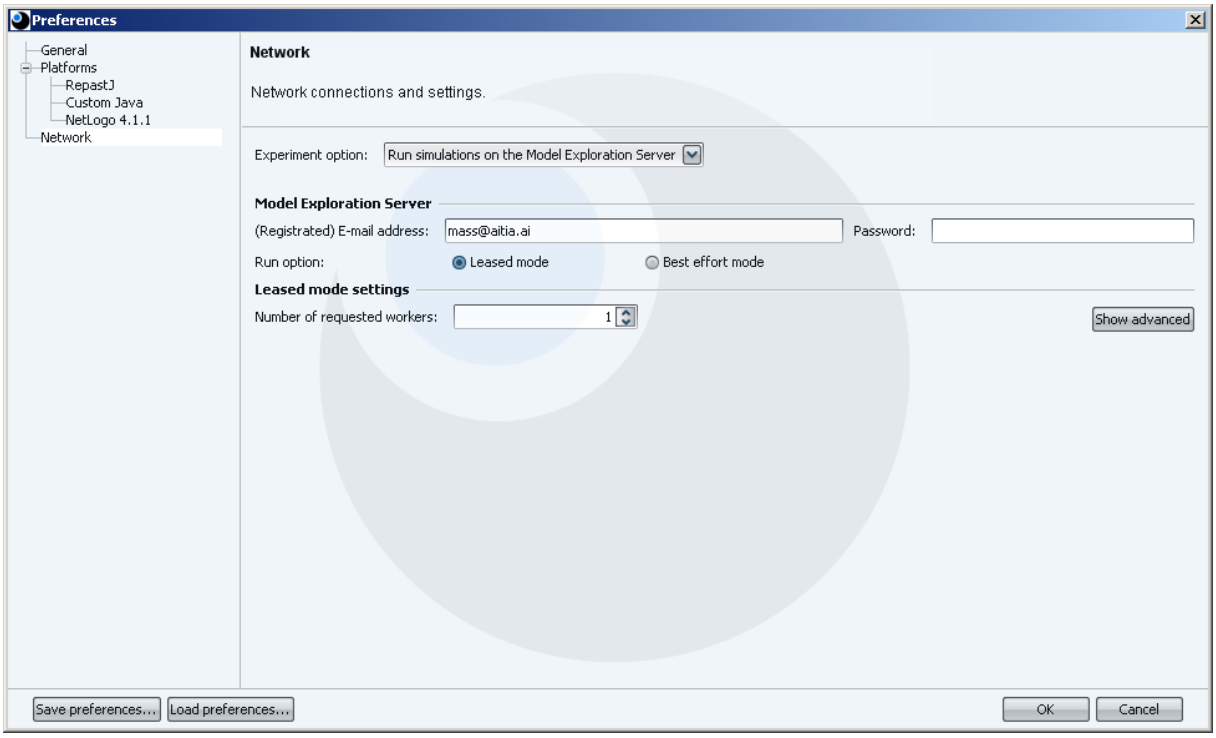


Figure 11 – Preferences dialogue, Model Exploration Service

You might specify an e-mail address (registrated on the Model Exploration Service official website) and password here, or select the required service mode (leased host or best effort, see below). These values are used as defaults on the start experiment dialogue before you send the experiment to the cloud.

The leased-host service provides your simulation with the requested number of dedicated CPU's, so that you can calculate the expected time of completion.

In contrast, the best-effort service helps you to optimize costs. In this scheme, your simulation shares the computational pool with other applications, utilizing as many CPU's as currently available.

The advanced settings section specifies the required connection information to the Model Exploration Server (click the Show advanced button the display the actual settings, see).

Figure 12 – Preferences dialogue, Model Exploration Service, advanced settings

Note: It is not recommended to modify these settings.

See further information about the Model Exploration Service at its official website:

http://modelexploration.aitia.ai

MEME is also able to create a simulation job for a dedicated special-purpose multiprocessor computing system named QosCosGrid[2]. If you choose this option the only thing you have to specify is the number of required workers (see Figure 13).

Figure 13 – Preferences dialogue, QosCosGrid settings

See details about creation simulation jobs for QosCosGrid in 3.7 Create Simulation Job for QosCosGrid.

3.2 Parameter Sweep Wizard

The Parameter Sweep Wizard enables users to create batch versions of their models in five steps and run them. The following sections describe these steps in detail using RepastJ platform. Other platforms are explained after that.

To start the wizard select the *Parameter Sweep/Run simulation…* menu item or press the button in the *Results* panel.

The figure below displays the opening screen (see *Platform selection* page on Figure 14). Note that in certain cases (see in section 3.1.1 General Settings) the wizard skips the *Platform selection* page and starts on the *Model selection* page (see Figure 15).

3.2.1 Platform selection Page

The first step is specifying the platform of the model you want to explore. The *Available platforms* list on the left side contains all registered platforms. You can read a short description about a platform in the description field by selecting its name from the list.

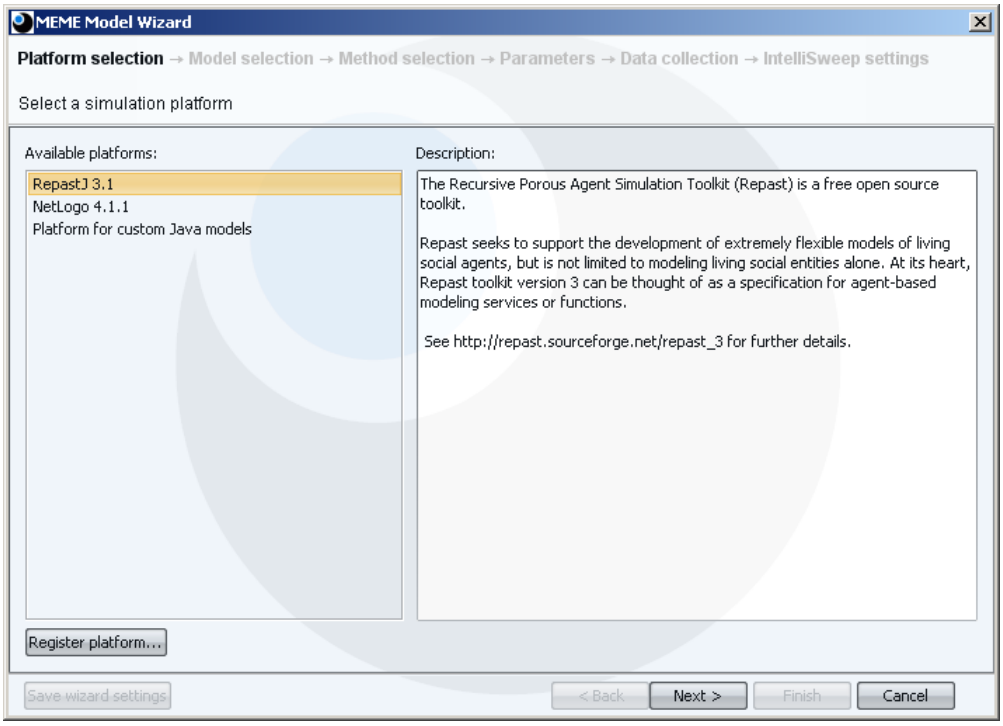


Figure 14 – Platform selection page

[2] http://www.qoscosgrid.eu/

You can extend the list by registering new platforms on the *Preferences* dialogue can be accessed directly by pressing the *Register platform…* button.

If you select the appropriate platform from the list click *Next* to continue.

3.2.2 Model selection Page

Note: The graphical elements of this page may change if you select NetLogo or Custom Java platforms on the first page. See details about these changes in sections 3.8 Platform for Custom Java Models and 3.9 The NetLogo 4.1.1 Platform

The model (its class file at RepastJ) can be selected on this page (see Figure 15). This can be a general model file (without GUI or batch specific code) or a batch model file. If the selected model is not a batch model the wizard will help in creating its batch version. In both cases, the wizard helps generating the parameter tree to be explored and/or running the model on a cluster or grid of computers (or on the local computer).

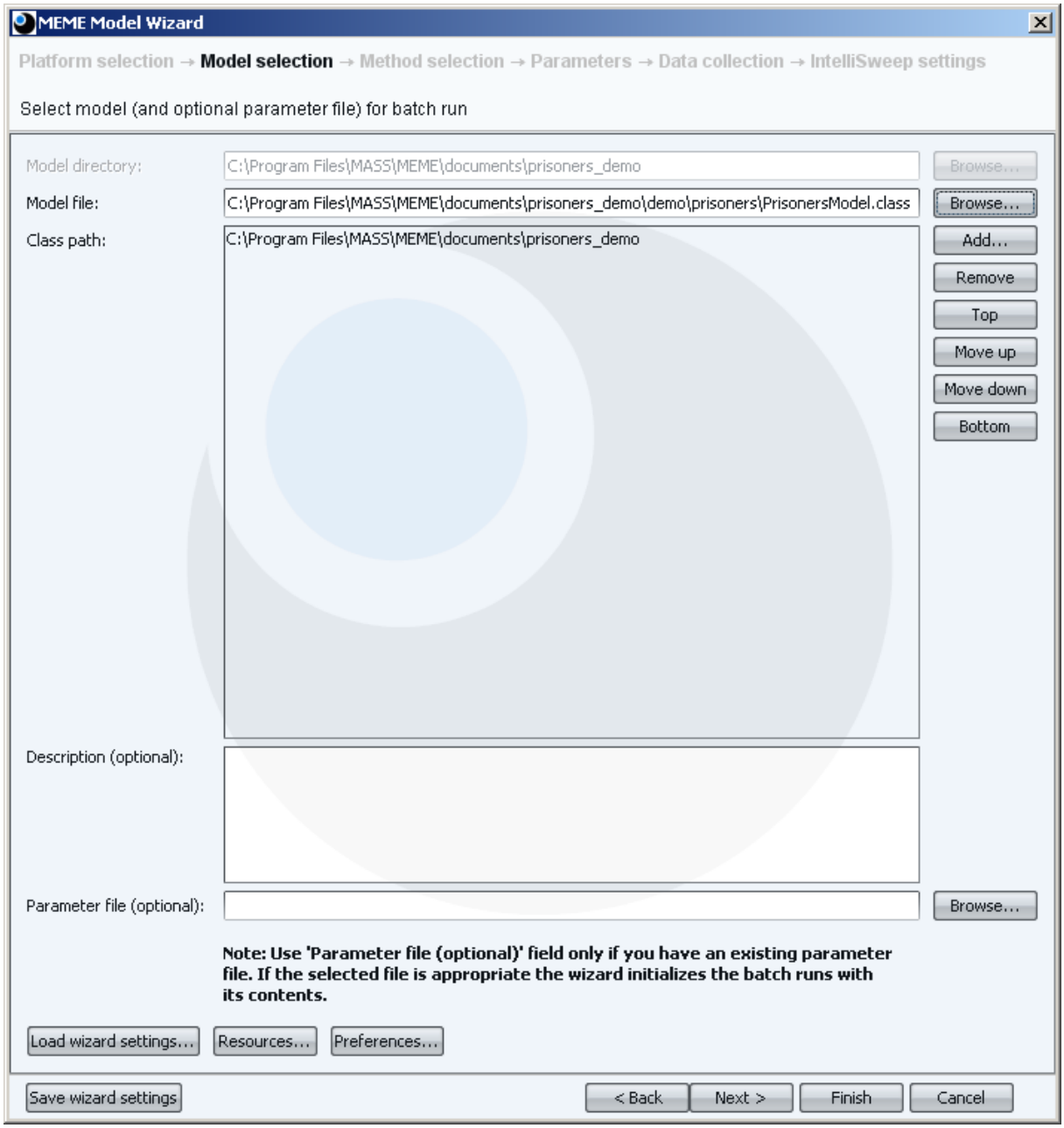


Figure 15 – Model selection page

Note: Please do not try to select GUI-related model (a class file that contains GUI code) because it may cause problems during simulation running, especially if you run the model on a cluster of computers.

Some of the platforms may need information about the model directory from the user. At RepastJ platform this information can be computed from the location of the model class so the model directory field cannot be edited directly.

In order for the wizard to load the complete model, the paths to all related classes must be specified. The directory where the model class can be found (in regard the package structure) is automatically added to the *Class path*. You can extend this list manually by pressing the *Add…* button and selecting one or more directories or JAR files in the file dialogue. To delete elements from the *Class path*, select them and press the *Remove* button. In the case of certain models the order of the *Class path* can be of importance, use the *Move up/Move down/Top/Bottom* buttons to achieve the right order. (See further details in section 3.2.2.1 Extendable Class Path!)

The *Description* and the *Parameter file* fields are optional.

If an existing and appropriate parameter file is selected the wizard initializes the experiment with its content (but it is still modifiable).

3.2.2.1 Extendable Class Path

Upon pressing the *Next* button the wizard tries to load the model. If the *Class path* is defined properly the wizard will move on to the next phase. Otherwise, if the wizard is unable to find a class it will ask for its location in the file dialogue shown in Figure 16.

The name of the missing class is displayed in bold with yellow background on the top of the dialogue window. Select the class file of the class in the file system or the JAR file that contains this class and add it to the class path.

Note that the class path information can be provided in an incremental way.

Note: This dialogue may appear in other situations, whenever the application needs to know the location of a class.

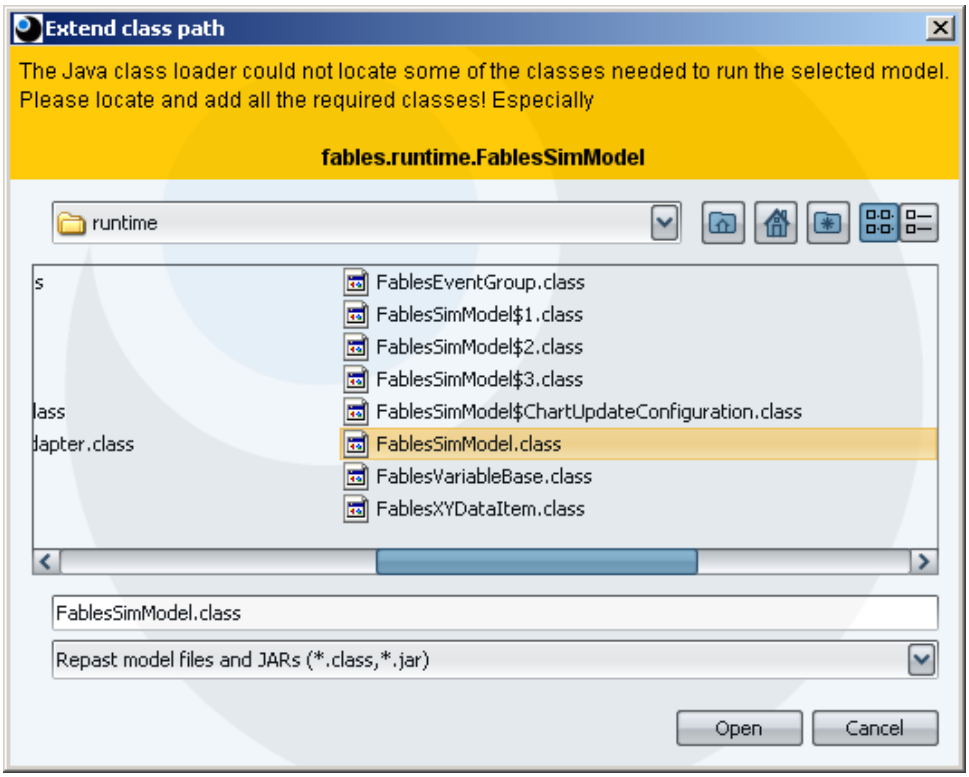


Figure 16 – Extend class path dialogue

3.2.2.2 Resources

Certain models use files (e.g. text files, XML files, database files, etc.) called resources. Running these models on a remote grid or cluster of computers requires the wizard to know the name and location of the resource files. Resources can be defined in the *Resources* dialogue (see Figure 17) which is appears when you press the *Resources…* button on the *Model selection* page of the wizard.

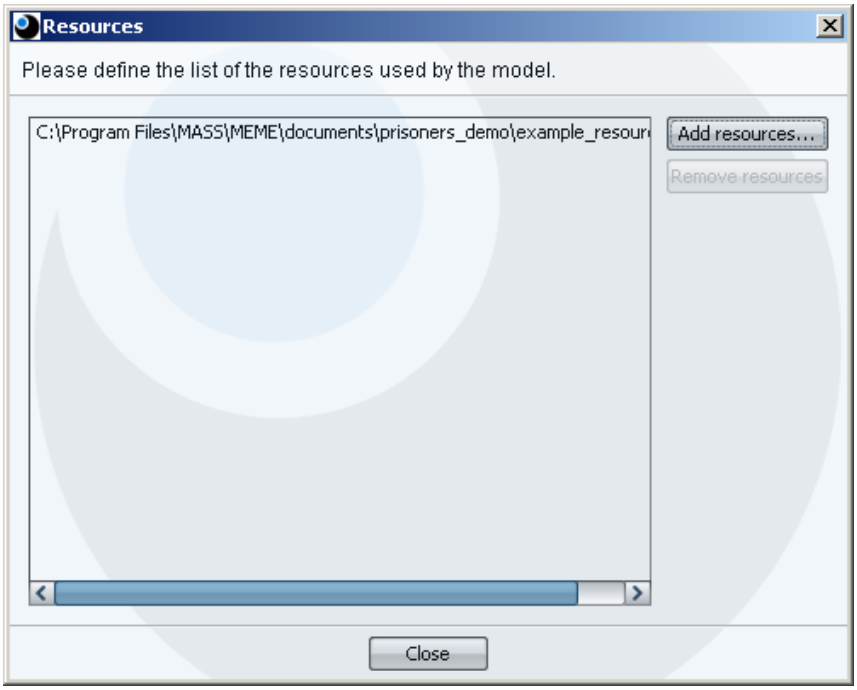


Figure 17 – Resources dialogue

Extend the list of resources by pressing the *Add resources…* button and selecting one or more files in the file dialogue. The *Remove resources* button allows for deleting selected files from the list.

Due to technical reasons the selected resources must be in the model directory or in its subdirectory.

If you choose a file from another location, a warning message is displayed (see Figure 18).

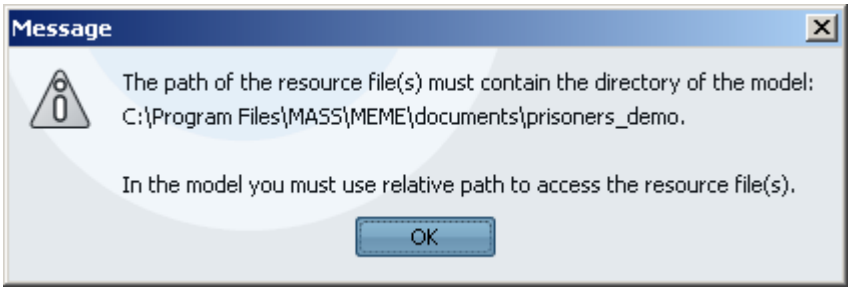


Figure 18 – Invalid resource warning

Note: As the message says above, only those models can be run in distributed mode that accesses their resources in relative way. If a model uses a resource defined by its absolute path, it won't find the file when running on a cluster or grid of computers.

When models are run locally, there is no need for defining resources; hence in local mode the *Resources…* button remains disabled.

3.2.2.3 Load Wizard Settings

When running a model, the wizard automatically saves all model settings (e.g. parameter tree, recorders, scripts etc.). These settings can be loaded by pressing the *Load wizard*

settings… button on the *Model selection* page and selecting a wizard settings descriptor file in the file dialogue shown in Figure 19.

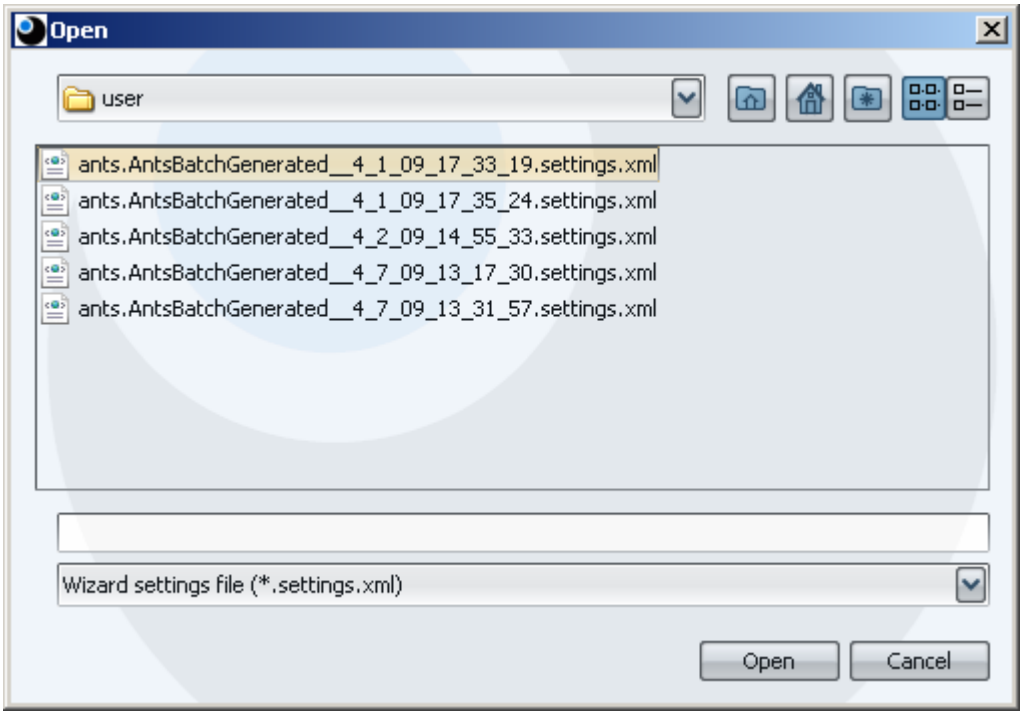


Figure 19 – Load wizard settings… dialogue

This file dialogue shows content of the *Wizard settings folder* specified in the *Preferences* dialogue. A wizard settings descriptor file name always ends with *.settings.xml*. These files have long names, but they hold a lot of information that helps to identify certain wizard settings.

For example, the filename ants.AntsBatchGenerated__4_1_09_17_33_19 indicates that:

- *The original model name is Ants and it is in the ants package.*
- *The model is generated by the wizard (it contains the word „BatchGenerated").*
- *The date of the generation is 1 April, 2009.*
- *The time of the generation is 17:33:19.*

After pressing the *Open* button the wizard initializes with the selected settings. These settings can be later modified or the model can be run with the exact same settings.

Note: When you press the *Finish* button to run a model, the wizard automatically saves the settings even if you didn't do any modifications on previous settings.

3.2.3 Method selection Page

This page lists the available sweeping methods that you can choose from. The default is the *Manual method*, which lets you build a parameter tree described in 3.2.4 Parameters Page. The other choices are Design of Experiments (DoE) methods, which help you setup a simulation design to get specific results from the model. You can choose a method according to your goal (see 3.2.7 IntelliSweep settings Page about the methods).

You can choose the method from the list on the left (seen in Figure 20); you will see a short description of the method on the right. If you press the *Next* button, MEME will show you either the *Parameters page* (if you chose the *Manual method*), or the *Data collection page*.

There are methods that you can use for free, and others which will be available if you purchase them.

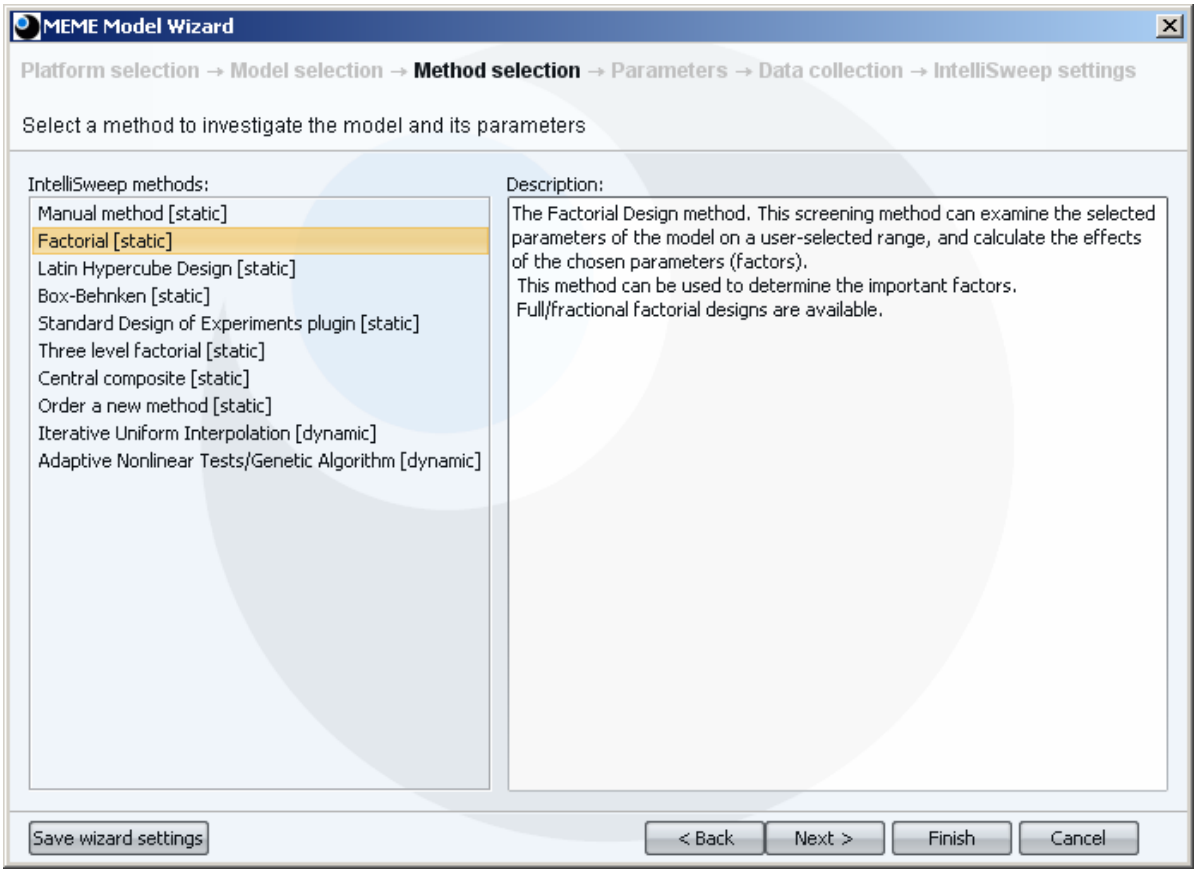


Figure 20 – Method selection Page

3.2.4 Parameters Page

Note: The graphical elements of this page may change if you select NetLogo platform on the first page. See details about these changes in section 3.9 The NetLogo 4.1.1 Platform

The parameter space the model is going to be run on can be defined on this page of the wizard (see Figure 21). The parameter combinations that are going to be explored are defined, or an existing parameter file is modified here.

Note that this page is displayed only if you choose the *Manual method [static]* option on the *Method selection* page. Otherwise the selected method automatically builds the parameter file based on the IntelliSweep plugin settings defined by the user (See 3.2.7 IntelliSweep settings Page).

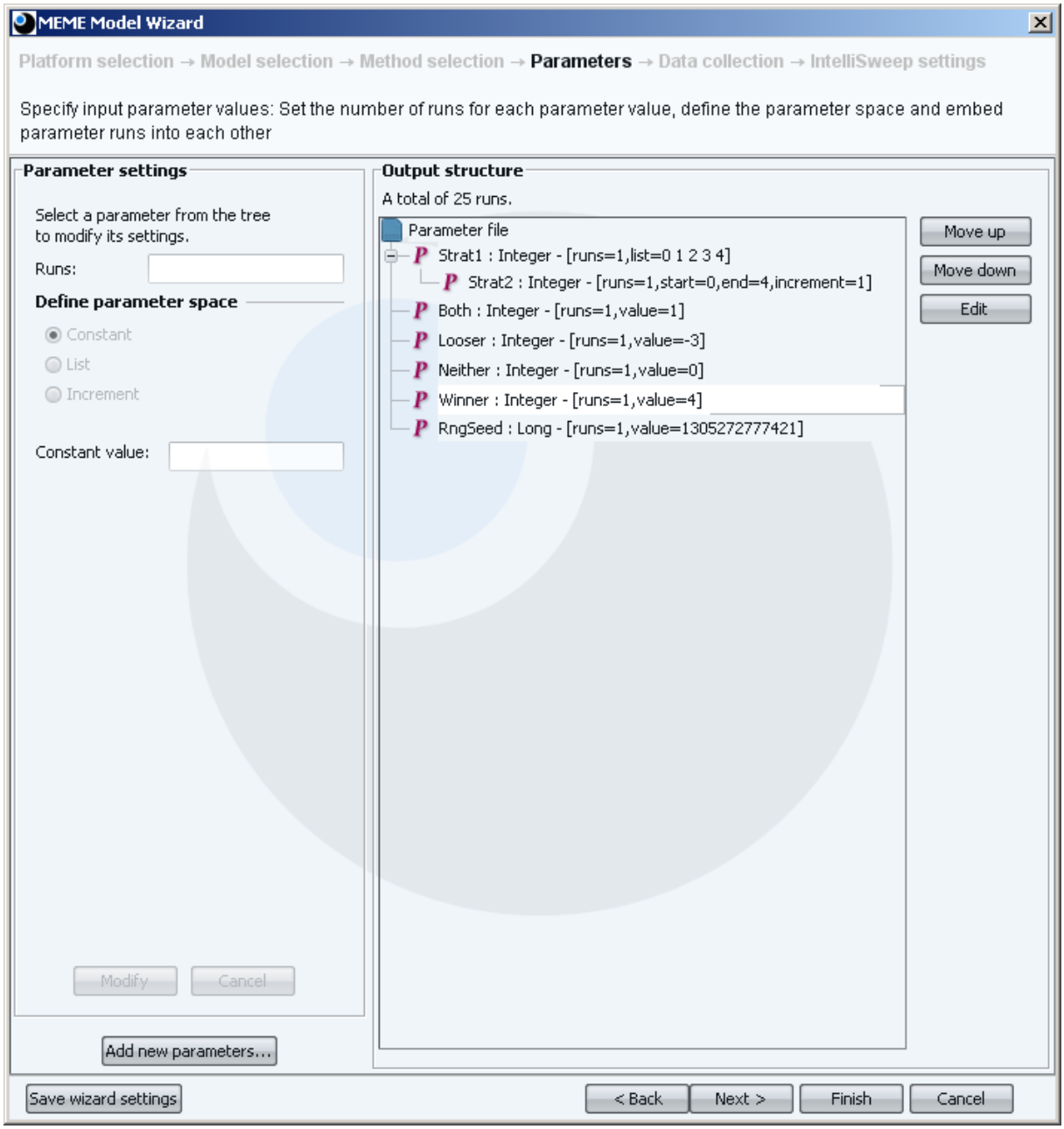


Figure 21 – Parameters page

3.2.4.1 The Parameter Tree

The parameter combinations are represented by a tree. If a parameter is "under" another, it means that its values will be set of *all* values taken by the "parent" parameter. For example, the *Strat1* and *Strat2* parameters on the figure above will generate the following lines in the parameter file (if *Save parameter tree to file* flag is checked on the *Preferences* dialogue):

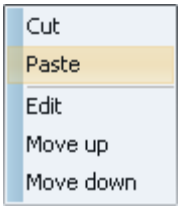
```
runs: 1
Strat1 {
set_list: 0 1 2 3 4
                {
                        runs: 1
                        Strat2 {
                                start: 0
                                end: 4
                                incr: 1
                        }
                }
```

```
}
```

According to the RepastJ parameter file syntax (see details in the RepastJ documentation [2]), this means that the *Strat1* parameter will take five values, 0, 1, 2, 3 and 4, and for both of these values the *Strat2* parameter will iterate over the values of 0, 1, … up to 4.

Note: Other platforms may use another parameter file format. See details in sections 3.8 Platform for Custom Java Models and 3.9 The NetLogo 4.1.1 Platform

Parameters can be moved "under" one another in the tree by dragging and dropping them. To change the order of the parameters, select one and use the move buttons.



For editing the settings of a parameter, select it in the tree and press the *Edit* button or double click on it in the tree.

All these functions are available in the context menu (right click) of the parameter tree too.

This menu also contains *Cut* and *Paste* commands as an alternative of the drag and drop technique in parameter embedding.

Note that if there are two or more non-constant parameters on the top level of the parameter tree, MEME interprets this the following way: it creates a list for all non-constant branches and then gets the first element from each list and runs the model using them. It then gets the second element from each list and so on. If any of the lists depletes the batch run stops. This may happen before all parameter values are assigned. To explore all possible combinations you must create only one branch from the non-constant parameters in the tree.

The total number of parameter combinations (computed from the current state of the parameter tree) is shown above the tree.

Note: The parameter tree syntax doesn't allow parameters to be "under" a constant parameter. The wizard checks this requirement after pressing the *Finish* button.

3.2.4.2 Modify Parameter Settings

After pressing the *Edit* button (or double click on the parameter in the tree) the settings of the selected parameter can be defined in the left panel. The values can be specified in three ways (according to general conventions):

- by specifying a constant value (see below),
- by specifying a list of values (see below), or
- by specifying an iteration by providing a start value, an end value and a specific increment (see below).

Runs settings describe how many times a parameter is going to takes a value before it goes on to the next value.

Note: Other platforms (i. e. NetLogo 4.1.1) supports only a global *Runs* value which means all defined parameter combinations will be are going to be explored *Runs*-times. See details in section 3.9 The NetLogo 4.1.1 Platform.

Pressing the *Modify* button commits the changes.

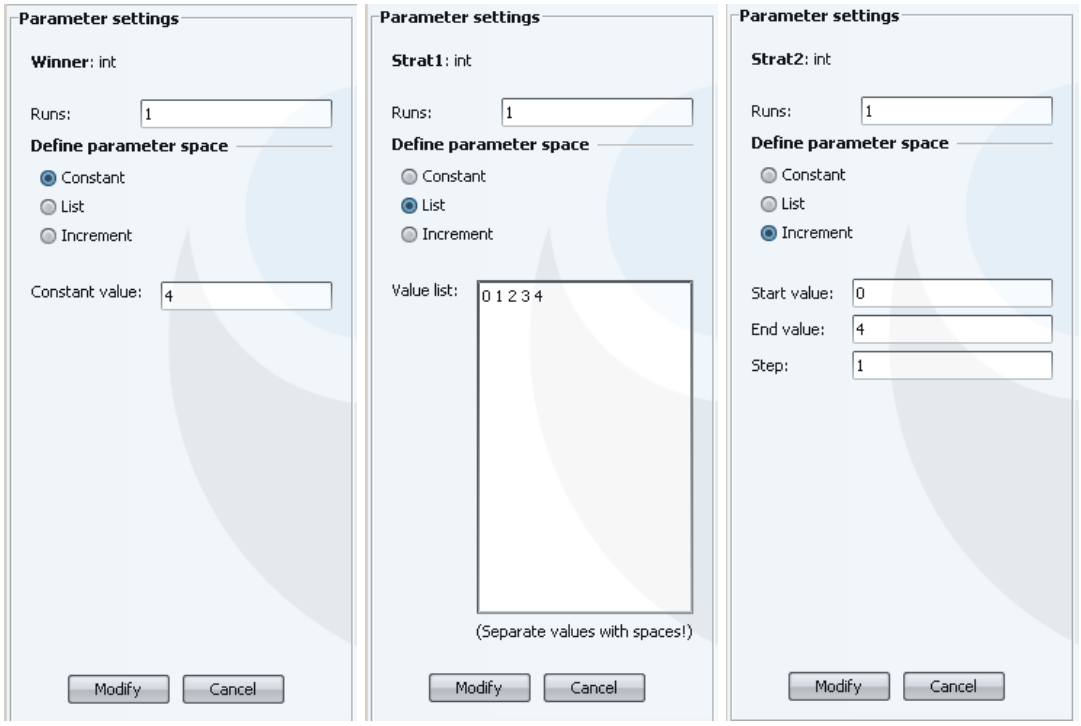


Figure 22 – Modify Parameter Settings

3.2.4.3 New Parameters

It is also possible to extend the group of available parameters (that is normally specified in the *getInitParam()* method of a RepastJ model) by declaring a non-parameter variable to a parameter. For this, use the *Add new parameters…* button at the bottom of the left panel (see Figure 23).

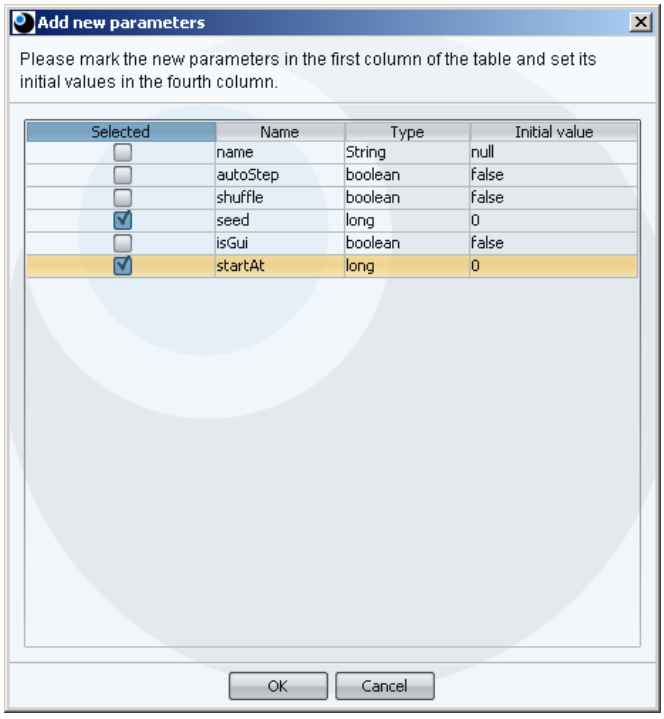


Figure 23 – Add new parameters dialogue

Select the variables by checking the box before their name in the table. To define initial values, simply double click in the *Initial value* field. Press the *OK* button to create parameters from the selected variables.

Note: Use this option before any other operation related to the parameters or the parameter tree because all previous user settings will be lost.

Note that when you add new parameters to the tree for the second time, your selection of variables will override the previous one. For example, if you select **seed** first, then when defining new parameters again you do not select **seed**, it will not be a parameter. You can use this behavior to delete a previously selected variable from the parameter tree. However, parameters defined by the model can not be deleted.

The wizard may modify the model upon adding new parameters. At RepastJ, for example, it will create the appropriate getter/setter methods for the new parameter variables and add them to the parameter list.

3.2.5 Data collection Page

Note: The graphical elements of this page may change if you select NetLogo platform on the first page. See details about these changes in section 3.9 The NetLogo 4.1.1 Platform.

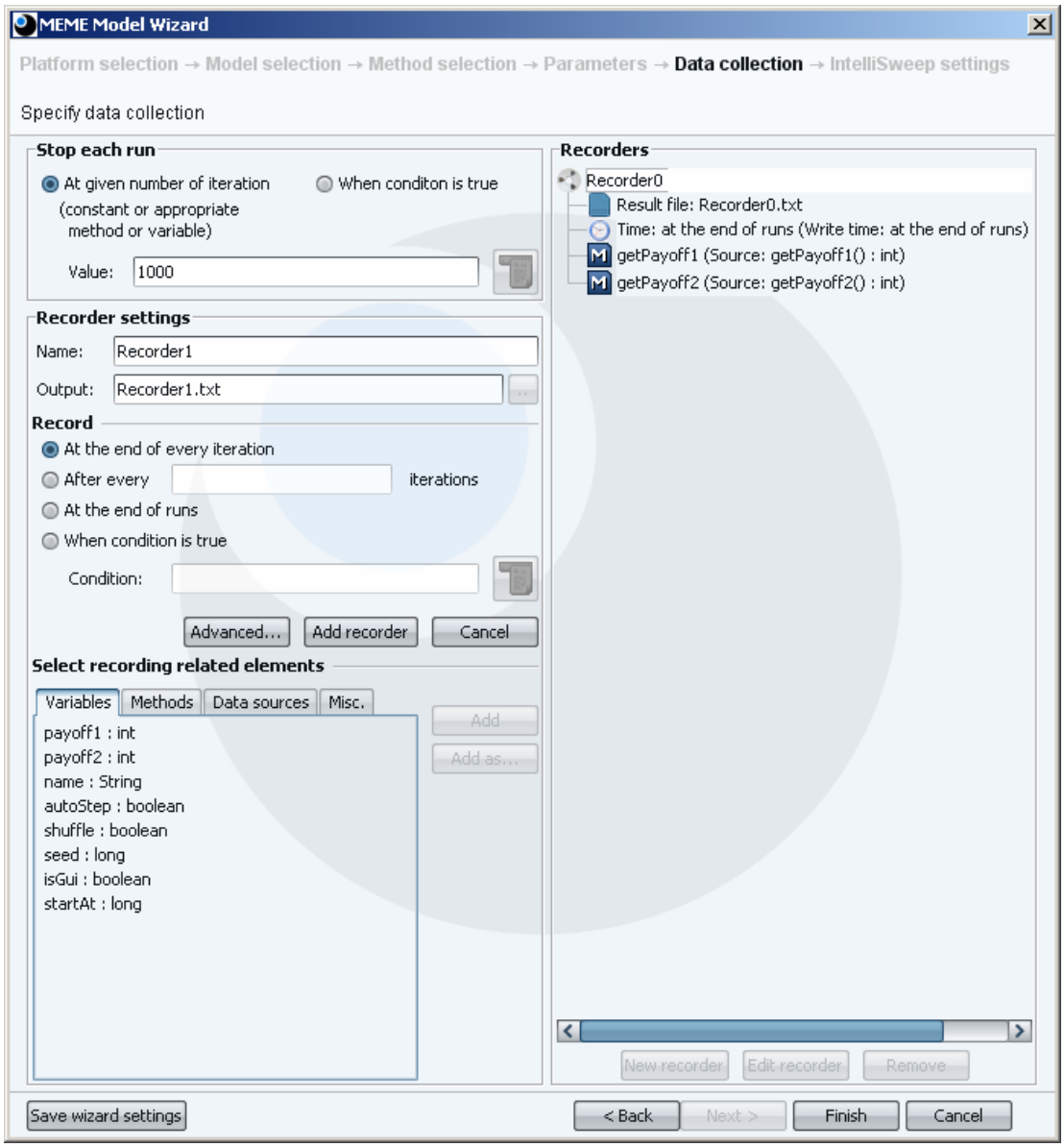


Figure 24 – Data collection page

The *Data collection* page of the wizard is shown on Figure 24. This page helps specifying what kind of information is to be collected during the experiment and the stop condition for each simulation can be defined here.

Pressing *Finish* button will generate the model (if necessary), parameter file (if *Save parameter tree to file* flag is checked on the *Preferences* dialogue) etc. corresponding with the settings just provided and will start running the model (except if you choose the *Create simulation job for QosCosGrid* option on the *Preferences* dialogue, see the details in 3.7 Create Simulation Job for QosCosGrid).

Note that at this point you are expected to press the *Finish* button only if you have chosen the *Manual method [static]* option on the *Method selection* page. In other cases an additional page is available where you can define method specific settings. See in details in 3.2.7 IntelliSweep settings Page.

3.2.5.1 Stop Condition

The stop condition can be a number (i.e. the specific time step at which runs will be stopped), it can be a variable with appropriate type (i.e. integer type), a method with the appropriate return type, or a logical expression built from the model's variables and methods (i.e. in the time step when this condition becomes true the simulation will be stopped).

Note: The wizard checks the stop condition when you press the *Finish* button, but, at RepastJ only if there is a new model generated (you defined any recorders and/or new parameters).

3.2.5.2 Recorders

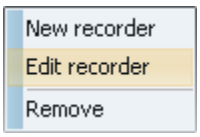
The tree on the right panel (see Figure 24) lists the specified recorders. It shows all relevant information concerning the recorders:

- name;
- name of the output file (one per recorder);
- time/frequency of recording (and time of writing data to file in brackets, see details below);
- what to record (names of variables and/or methods returning numeric, textual or logical values) etc.

The recorder collects the values of the specified variables and methods during the simulation. The collected values are then eventually saved into the specified output file. In order to speed up simulations, recorders store collected information in memory first and save them to files only at a given frequency.

In the current version of MEME, beside variables or methods already existing in the model, derivative values can also be specified as the source of information to be collected (see section 3.2.6 for details!).

A new data recorder can be added to the tree using the *New recorder* button. An existing recorder can be modified by selecting any element (e.g. recorder's name, a recordable element etc.) of the recorder in the tree and pressing the *Edit recorder* button or double clicking on it. To remove a recordable element from a recorder simply select it and then press the *Remove* button. This button deletes the whole recorder if the selected element of the recorder is other than a recordable element, such as its name or the name of the output file etc. You can also access these functions through the context menu (right click) of the tree.



Note that these buttons (menu items) are not enabled when you create/edit a recorder (when the *Recorder settings* section is active).

The settings of the new recorder can be defined in the *Recorder settings* section on the left panel. The wizard offers a default name and a default output name that can be changed. There are four time/frequency options for recording:

- Record at the end of each iteration (i.e. every time step), or
- record after every *x* iterations (e.g. every 10th time step), or

- record at the end of each run (i.e. one recording for every defined parameter combinations), or
- record, when a condition is true.

The abovementioned condition must be a logical expression built from the model's variables and methods. The wizard checks it when it tries to build/modify the recorder.

Note: Some of these options may be disabled if the selected platform doesn't support them.

You can specify the time of actual writing of the data in to a file on the *Advanced recorder settings* dialogue. This dialogue appears if you press the *Advanced…* button. There are three options here:

- Write to file after every recording, or
- write to file at the end of the runs (this is the default option), or
- write to file after *x* iterations (e.g. every 20th time step).

Note: Some of these options may be disabled corresponding with the time/frequency of the recording. For example, the *After x iterations* option is disabled if the recording option of *At the end of the runs* is selected. Furthermore the selected platform may disable the unsupported options too.

Note: The *Write to file* option may be changed if you change the recording time/frequency option *LATER*. The general rule is the following: There is no writing before recording.

Recorder settings can be added to the recorder only after pressing *Add recorder* button, which results in the data recorder appearing in the tree on the right. In case of editing recorder settings you must press the *Modify recorder* button to commit the changes.

Adding recordable elements to a recorder is done by selecting any elements of the recorder in the tree and selecting the elements from the tabs at the bottom of the left panel. Selected values can be added by pressing the *Add* button.

You can also add a recordable element to a recorder with a new name. In order to do this select one of the recorders in the tree, select an element from any tab and then press the *Add as…* button. A new dialog appears on the screen where you may specify the new name. Press *OK* button to finish the operation. In the output file of the selected recorder the new name will appear as the name of the recordable element.

Note: Some of the platforms don't support this functionality.

Note that adding a method (or a data source) to a recorder may cause problems when you want to use its output later in a Beanshell script, because the name of the recordable element (hence the name of the output column) is not a valid Beanshell identifier (for example, if it contains brackets). For avoiding this use the *Add as…* button instead of *Add* and create an appropriate alias name. The other option is to use `get("name")` method in the Beanshell scripts whenever you want to reference the column. See in details in section 7.4 Beanshell Scripting.

3.2.6 Creating Data Sources

As it is mentioned above the wizard allows the user to calculate derivative values (e.g. statistics) from "raw" information (e.g. already existing variables and methods) using a simple point-and-click interface and/or simple scripting.

To create a data source, press the *Create* button on the *Data Collection* page. The *Create data source* dialogue is shown on Figure 25. The *Create* button appears below the *Add as…* button if the *Data sources* or the *Misc.* tab is the active one.

On the *Create data source* dialogue you can create data sources that cannot be recorded because their types are not numerical, textual or logical. However, you can use these data sources as an actual parameter of other data sources. The non-recordable data sources appear on the *Misc*. tab while the recordable ones on the *Data sources* tab.

To remove an existing data source from the lists, select it and then press the *Remove* button. Note that you cannot delete a data source if it is referenced by other data source(s). In this case a warning message appears on the screen.

You can also edit an existing data source by selecting it from one of the lists and pressing the *Edit* button. The same dialogue that is used for creating data sources shows up here as well.

There are two kinds of data sources: the statistics and scripts (i.e. where you can actually write code). In fact, there is a third type called operators. They are similar to the statistics but their result is often not a number, a string or a logical value that you can use as a recordable element but a list or other type. You can use the result of operators as an actual parameter of a statistic and/or other operators.

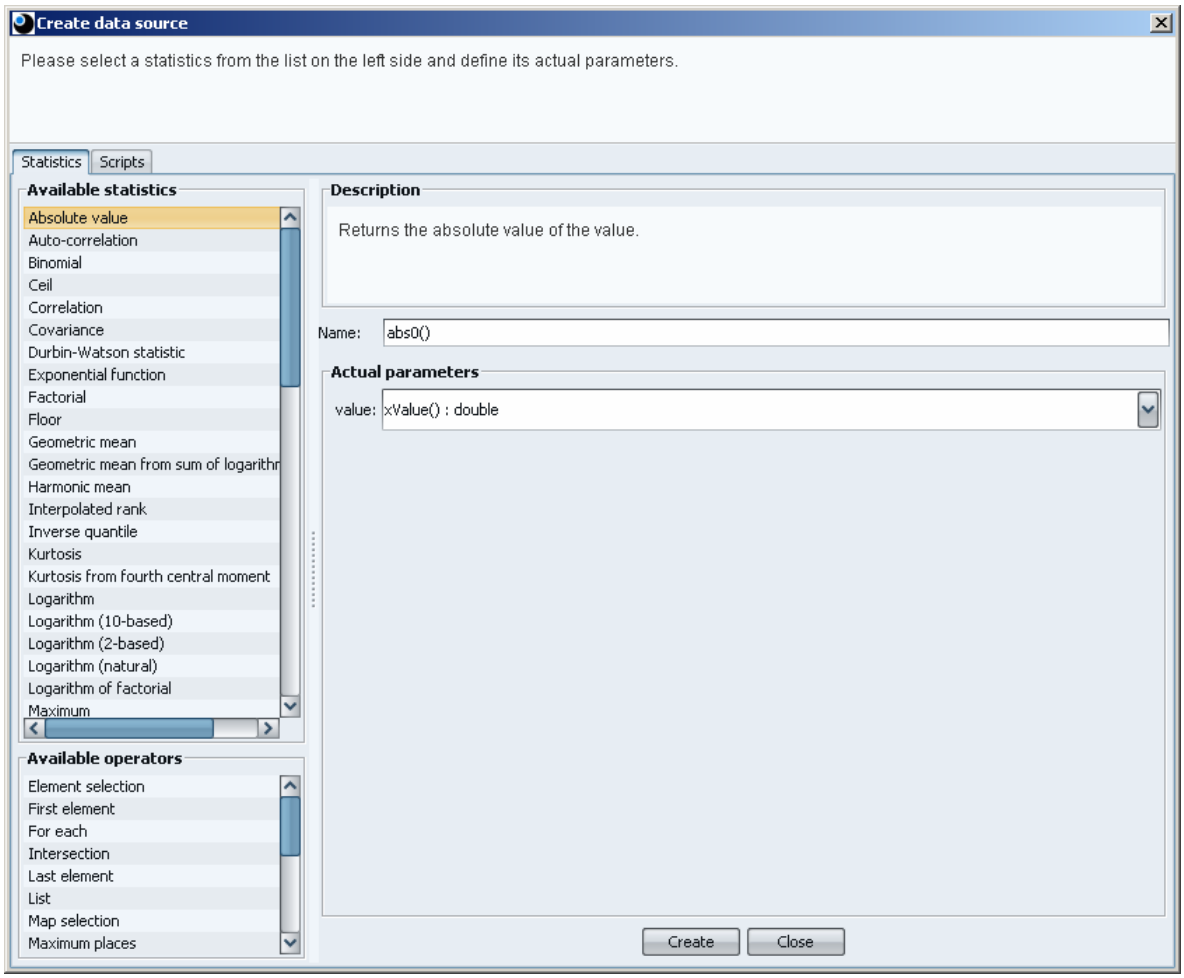


Figure 25 – Create data source dialogue, Statistics tab

3.2.6.1 Operators and Statistics

The *Statistics* tab of the *Create data source* dialogue (where you can define operator instances as well) is shown on Figure 25.

On the left side of the dialogue there is a list of available statistics (in alphabetic order). In the current version there are more than 70 of these. Below the statistics panel is the 21-item sorted list of available operators.

When you select a statistic or an operator a short description appears on the right panel. The statistics are based on the Colt library, for further details see the Colt documentation ([7]).

Below the description field there is a name field. A name must be unique (not used in the model yet). The wizard offers a default one that can be changed. If the specified name is already taken the wizard returns an error when it checks the statistic/operator.

Below the name field the parameters of the selected statistic/operators can be given.

Note: The graphical elements of the parameter specifying interface may change if you select NetLogo platform on the *Platform Selection* page. See details about these changes in section 3.9 The NetLogo 4.1.1 Platform.

Operator Parameters

Note: Some of these operators are not available at NetLogo platform. See details about these changes in section 3.9 The NetLogo 4.1.1 Platform.

- ***Element selection, First element, Last element***: Returns an element of a collection or array.

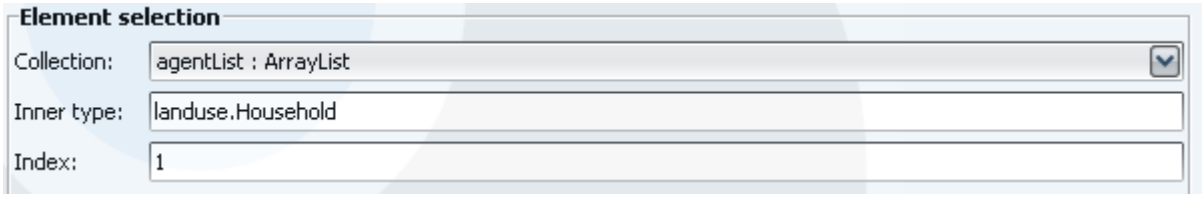


Figure 26 – Element selection

The parameters are the source collection/array, the type of the elements and the index that specifies the position of the element you want to select. For example, the parameters on Figure 26 mean the second element of the `agentList` which stores `landuse.Household` instances.

Note: when the data source dialogue needs any type information, it first tries to deduce it by investigating the model. In case of success, it automatically fills the type field what you can't modify. Otherwise you must fill the field manually with the fully qualified name of the appropriate type (e. g. `landuse.Household` on the figure above).

According to the element type, the result of the operator may be a recordable element (for example an element of an array that contains numbers). The *First element* and *Last element* operators fall into this category, with no need, however, to define an index parameter.

- ***Member selection***: Returns member (a variable or method) of an object.

Figure 27 – Member selection

To define the parameter, click the + sign next to an object in the top level (the tree expands, the appropriate members appear in the tree as children's of the selected object) and select one of its children. For example, on Figure 27 the

`size()` method of the `agentList` variable is selected which provides the number of agents.

According to the type of the selected member the result of the operator may be a recordable element (on Figure 27, `agentList.size()` is a recordable element).

- ***Map selection***: Selects an element from a map. It is very similar to the *Element selection*, except that a map uses keys instead of indices so you must select the appropriate key object from the drop-down list. If the selected map uses string or number keys you can also define the key by editing the drop-down list. According to the value type of the map, the result of the operator may be a recordable element.
- ***List***: Creates a list from a collection. Gets selected member from each element of the collection and builds a list using those.

Figure 28 – List

The parameter selection method is the same as at the *Member selection* operator but the meaning is different. For example, on Figure 28, the *List* operator results in a list of integers which contain the value of the `choice` variable for each agent stored in the `agentList` list.

In the tree there can appear lists with unknown element type. In this case a question mark (?) is shown as type name in the tree (e.g. *agentList: ArrayList<?>*) and there is no "+" sign before the name of the list. To use this list as a source of the operator you must define the element type by clicking with the right mouse button on the name of the list and type in the fully qualified name on the appearing dialogue (see Figure 29).

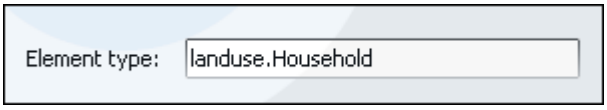


Figure 29 – Defining element type of a list

- ***Filter***: Returns a list containing only those items of the source list/array for which filter condition is true.

Figure 30 – Filter

The parameters are the source list/array, the type of the elements and a filter (logical) condition. As you can see on Figure 30 in the filter condition you can use the `%value%` expression to reference to the current element of the list/array. For example, the operator on Figure 30 results in a list containing only those agents from `agentList` whose `choice` value is not zero.

- ***Timeseries***: Creates a list for the specified parameter (can be any non-collection and non-array variable or method) and in every time step of the simulation appends the current value of the parameter to the list. To define the parameter simply select an object or a primitive value from the drop-down list.

- ***Sort, Tail, Maximum places, Minimum places***: The *Sort* operator sorts the specified list/array into ascending order, according to the natural ordering of its elements. The *Tail* operator returns the selected list/array without its first elements. The *Maximum places/Minimum places* operator provides the list of indices of the maximum/minimum values in the specified list/array. The parameters are the same in all four cases of operators: the source list/array and the element type.
- ***Union, Intersection***: Returns the union/intersection list of the parameters (lists and/or arrays). The user interface of the parameter selection is shown on Figure 31 (there is a union definition on it, but the interface of the intersection is the same). The parameter list has to be specified in the list on the right (*Selected elements*). In order to do so, select elements from the left list (*Available elements*) then press *Add* button. Remove elements from the right list by selecting them and then pressing the *Remove* button. Note that lists with unknown element type may appear on the left side (e.g. *getMediaProducers(): Vector<?>*) as at the *List* operator. You can define the type by clicking the right mouse button on the name.

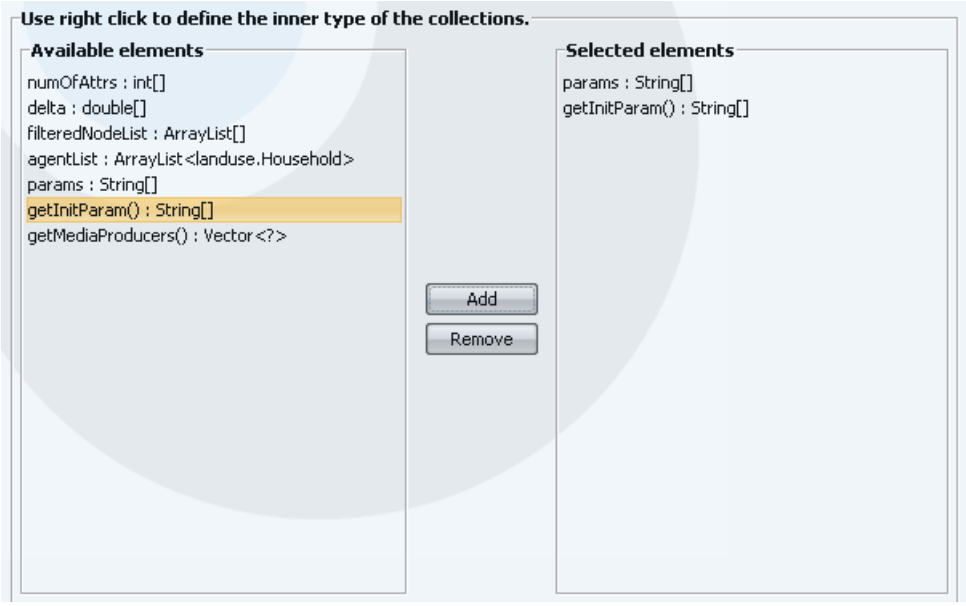


Figure 31 – Union

- ***Subtract***: Returns the elements contained in the first list/array but not in the second. To define the parameters you must simply select two lists/arrays from the two drop-down lists.
- ***Remove element, Remove element (all instances)***: Returns a list that contains the same elements than the selected list/array except the first instance/all instances of the given element. To do this, you must select a list, define the type of the elements and select the removable element from a drop-down list. If the selected list/array contains strings or numbers you can also define the removable element by editing the drop-down list.
- ***Random permutation***: Returns the permutation of the selected list/array in a list. To define the parameter simply select a list or an array from the drop-down list.
- ***Size***: Returns the size of an arbitrary collection/array/object. To define the parameter simply select an object from the drop-down list. The result of this operator is always a recordable element.
- ***For each***: Executes the selected function with each element of the specified list/array and creates a list from the results.

Figure 32 – For each

The source list/array can be selected from the first drop-down list. You must define the element type of the source list/array (most cases the wizard can automatically do this). The function can also be selected from a drop-down list which contains the appropriate methods of the model and some built-in methods from the `Math` class (like `abs()` on the figure above).

- ***Count element***: Returns the number of elements equals to the specified one in the given collection. To do this, you must select a collection, define the type of the elements and select the element you want to count from a drop-down list. If the selected list/array contains strings or numbers you can also define the last parameter by editing the drop-down list.

To create an operator instance press the *Create* button. Error message (for example when the given fully qualified name is unknown for the wizard) are shown on the information panel at the top of the dialogue.

Statistic Parameters

Generally, there are two kinds of parameters:

- Where a single number is specified (e.g. in the case of the *Absolute value* statistic on Figure 25),
- Where a number collection is specified (e.g. in the case of the *Maximum* statistic on the Figure 33.)

In the first case select the actual parameter from an editable drop down list. This list contains all numeric variables and appropriate methods (whose return value is a number) from the model (existed and previously generated too). Constant parameters can also be defined by editing the drop down list.

In the latter case a number collection has to be specified in the list on the right (*Selected elements*). In order to do so, select elements from the left tree (*Available elements*) then press *Add* button. Remove elements from the right list by selecting them and then pressing the *Remove* button. The tree contains all number and number collection (e.g. `int[]`, `double[]`, etc.) variables and appropriate methods (whose return value is a number or number collection) from the model (existed and previously generated too).

The tree also contains other non-numeric objects and collections that cannot directly be added to the list on the right. Instead you can use the built-in *Member selection* (for non-numeric objects, see details above) and *List* (for the non-numeric collections, see details above) operators to create a numeric element/collection. For example, on Figure 33 a new list is created from `choice` values of the agents contained by `agentList` list, and the statistic will return the greatest `choice` value.

Extend this list with constant values by editing the field below the list and then pressing the *Add constant* button.

To create a statistic instance press the *Create* button. Error message (for example if the *Selected elements* list is empty) are shown on the information panel at the top of the dialogue.

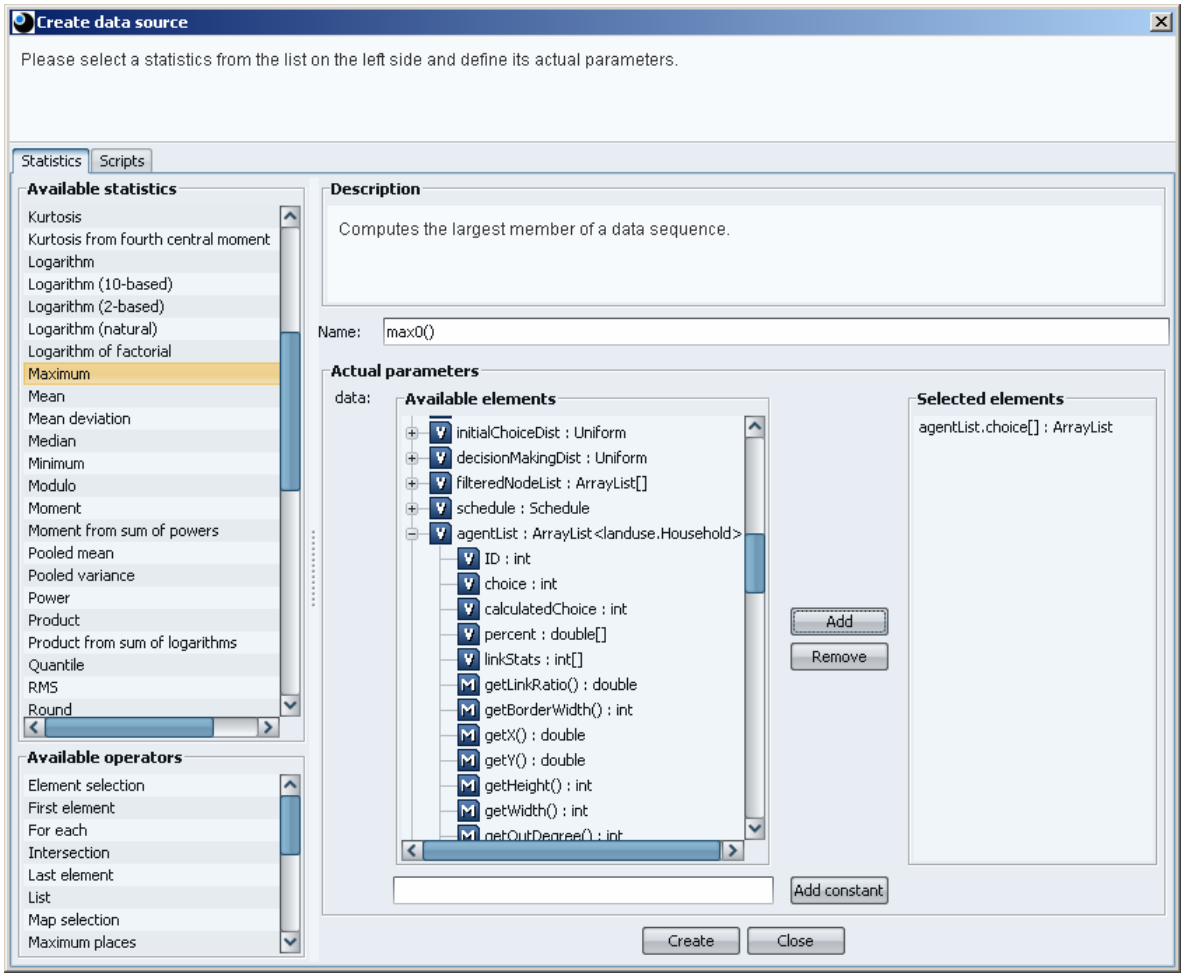


Figure 33 – Create data source dialogue, Statistics tab, Maximum

3.2.6.2 Scripting

Note: Scripting is for advanced users. It requires some programming skills in Java and/or NetLogo 4.1.1.

The *Scripts* tab of the *Create data source* dialogue is shown on Figure 34.

To create a script you define the following:

- *Script name*: Must be a unique name (not used in the model yet). The wizard offers a default name, but it can be changed. If the specified name is already taken the wizard returns an error when it checks the script.
- *Return type*: The return type of the script, which can be selected from the drop-down list or type a new one.

Note: If the return type is not a numeric type, logical type or string, the created data source can't be used as a recordable element, only in other data sources.

- *Body*: the source code of the script. See the *Limitations* section below.

In the body (and in the return type) the fully qualified names of the Java classes have to be used, unless they are in one the following packages:

- `java.lang`
- `java.util`
- `uchicago.src.sim.analysis` (only at RepastJ platform)
- `uchicago.src.sim.engine` (only at RepastJ platform)

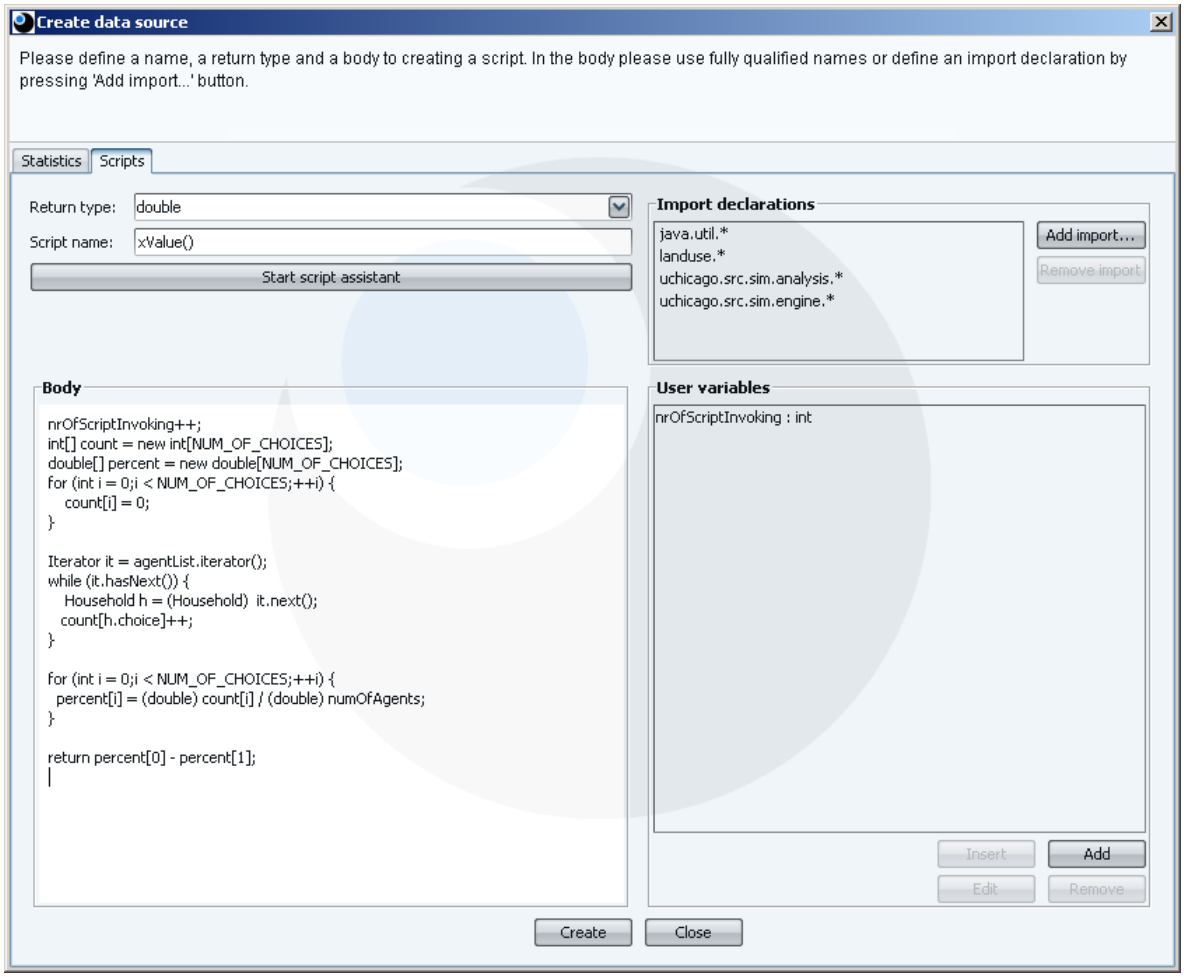


Figure 34 – Create data source dialogue, Scripts tab

Instead of using fully qualified names, import declarations can be defined by pressing the *Add import…* button and specifying an import declaration (such as in Java, except the usage of keyword `import`) on the *Import declaration* dialogue (see Figure 35). To remove an import declaration from the list select it, then press the *Remove import* button. Note that the predefined import declarations can't be removed.

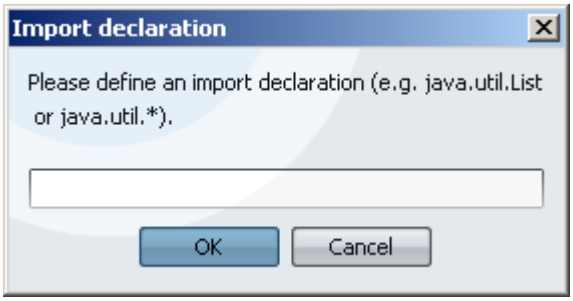


Figure 35 – Import declaration dialogue

In the body of a script you can't use all features of the Java language. There are some limitations.

Limitations:

- The new syntax introduced by J2SE 5.0 (including enums and generics) is not supported.
- Array initializers, comma-separated lists of expressions enclosed by braces { and }, are not available unless the array dimension is one.
- Inner classes or anonymous classes are not supported.

- Labeled `continue` and `break` statements are not supported.

User Variables

You can define new variables to your model and use them in your scripts. These variables can be use to store data between script invoking. To create a new variable press the *Add…* button on the *User Variables* panel. The *Defining Variables* dialogue is shown on Figure 36.

You have to define

- a unique name (not used in the model yet). If the specified name is already taken the dialogue returns an error when it checks the variable.
- the type of the variable, which can be selected from the drop-down list or type a new one.

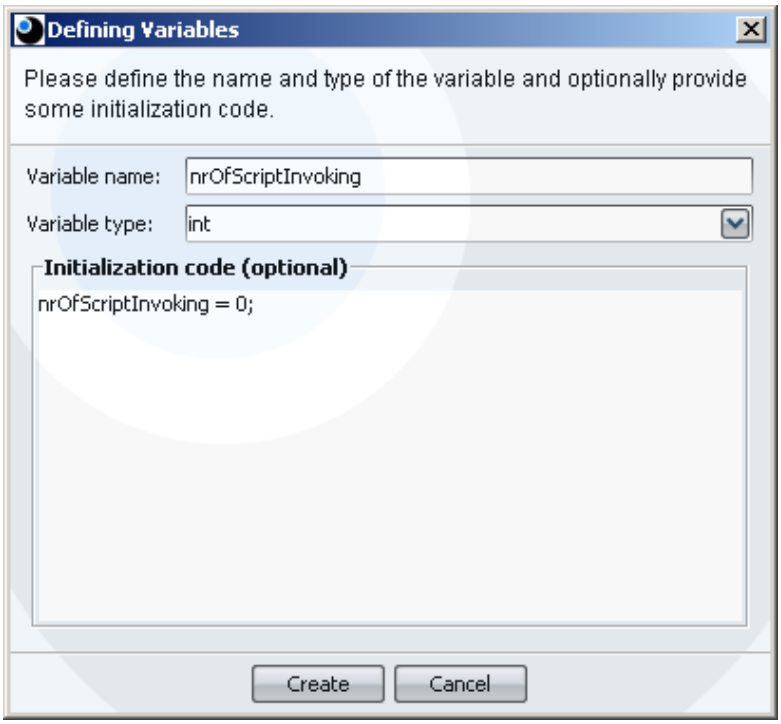


Figure 36 – Creating a new user variable

Optionally, you can provide some initializing code for the new variable. If you don't use this option the application generates a default initialization code according to the type of the variables (e.g. 0 for numerical variables). Press the *Create* button to finish the variable creation or the *Cancel* to undo it. The specified user variable appears in the *User Variables* list on the *Scripts* tab of the *Create data source* dialogue (Figure 34).

The initialization code of an existing variable can be modified by selecting the variable in the list and pressing the *Edit* button. However, you can't modify its name and type. To remove a variable simply select it and then press the *Remove* button. Please note that you cannot delete a variable if it is referenced by any script (a warning message appears in this case). Finally, you can insert the name of the selected user variable into the actual cursor position of the *Body* area by pressing the *Insert* button or double clicking on the variable in the list.

Script Assistant

To start the Script Assistant, press the *Start script assistant* button (below the *Script name* field). The dialogue is shown on Figure 37.

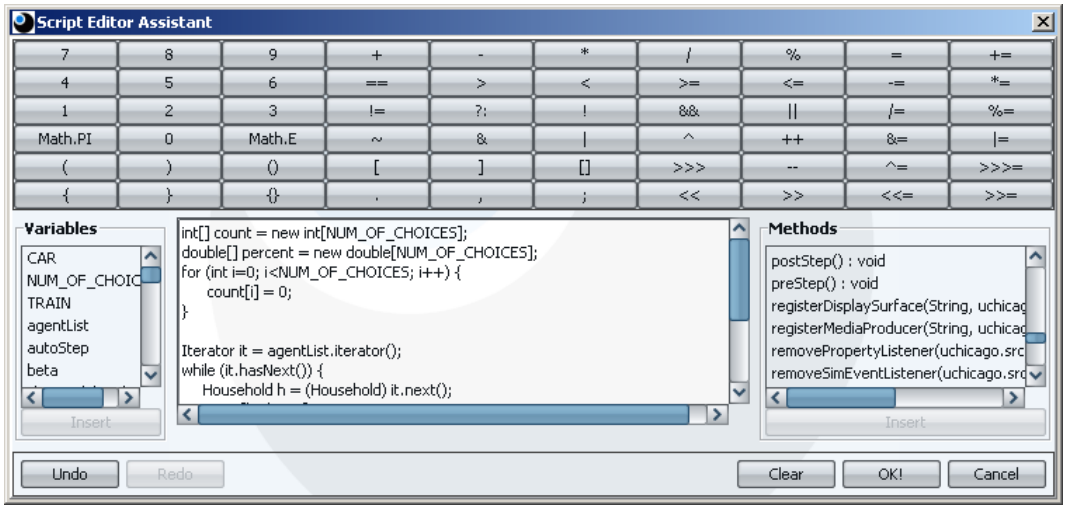


Figure 37 – Script Assistant dialogue

On the top of the window there are buttons for the most used symbols (numbers, arithmetic, relational and logical operators, brackets etc.). Press any button to insert the appropriate symbol to the actual cursor position of the editing area which is in the center of the dialogue. The variables of the model (including all user variables) are in a list on the left part of the window. Select one then press the *Insert* button below the list to insert the name of the selected variable to the editing area. There is a similar list for the methods of the model on the right side of the dialogue.

You can undo any change by pressing the *Undo* button. Use *Redo* button to cancel an undo event. The *Clear* button deletes the content of the editing area.

To finish editing press the *OK!* button. The content of the editing area appears in the *Body* area of the script dialogue.

To create the defined script, press the *Create* button. It is then checked and error messages (if any) are shown on the information panel at the top of the dialogue.

3.2.7 IntelliSweep settings Page

This is the last page of the wizard if you chose a different method from *Manual method* on the *Method Selection* page. What you can see here depends on the chosen method. You usually have to select one or more parameters from the model, set some values for them and the method builds the parameter tree from the settings in a specific way.

In almost every method you are able to set up the random number generator seeds, select the parameters on which you want to study the natural variation, select the parameters, that can be 'blocked', i.e. you assume that those parameters do not have a significant effect on the outcome of the experiment, and you want to test that in the experiment.

3.2.7.1 Factorial Method

The Factorial method produces designs where each parameter has two discrete levels. The method combines the levels in many (if not all) combinations. You can choose the parameters you are interested in, and set the levels for them in the setup screen seen in Figure 38.

In the upper part of the setup screen you can choose the parameters you are interested in from the *Parameters* list, and make them 'Factors', by pressing the *Add factor>* button. The chosen factors will be in the *Factors* list, from where you can remove them with the *<Remove factor* button.

You can change the default value of a parameter in the *Default value* textbox, by entering the new value and pressing the *Set* button. You can change the factor levels similarly at the right side of the *Factors* list, but you have to specify two values here, the low and the high ones.

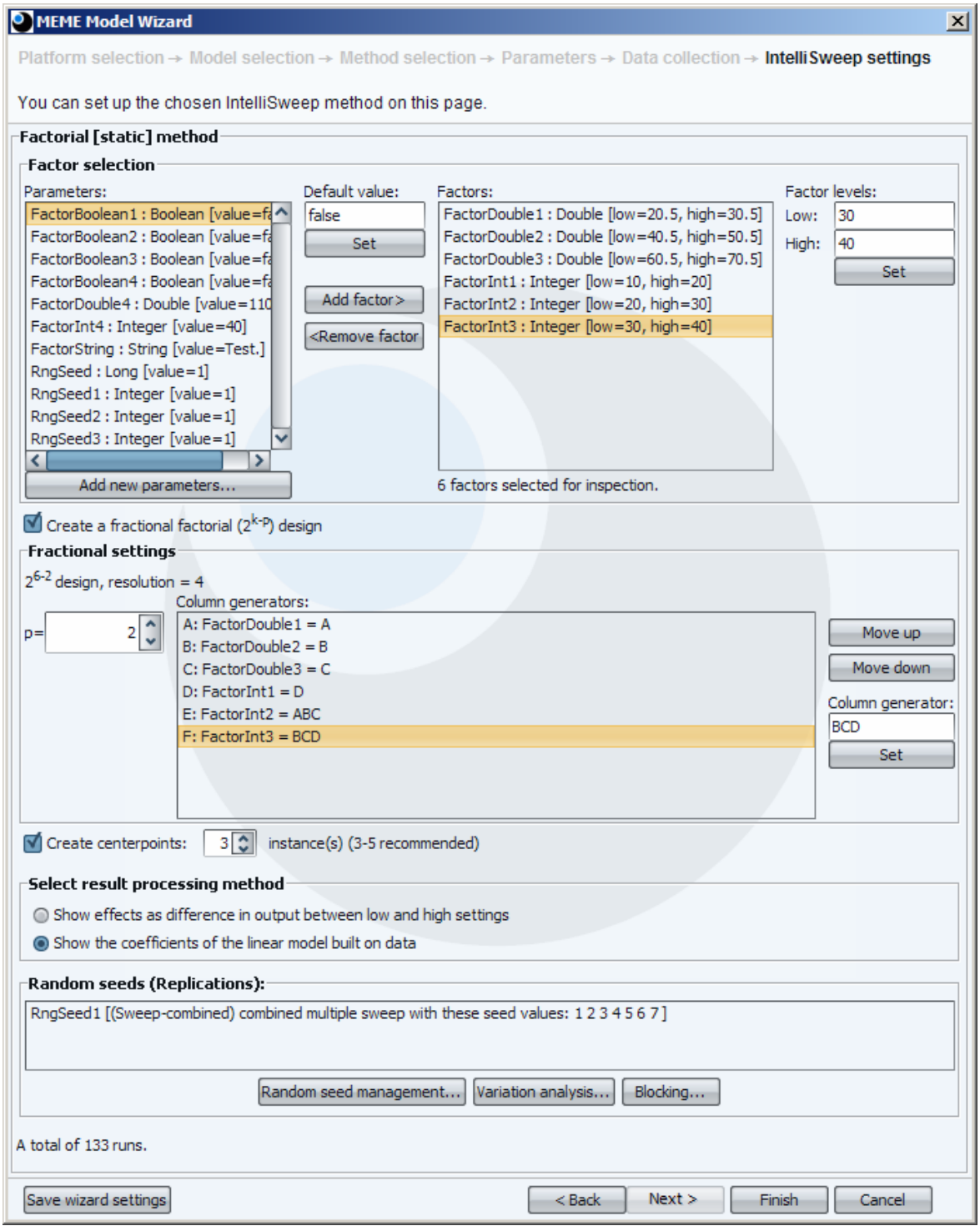


Figure 38 – Factorial method setup screen

There is an *Add new parameters…* button at the bottom of the *Parameters* list. This is the same one, with the same functionality as described in 3.2.4.3 New Parameters.

If you check the *Create a fractional factorial (2^{k-p}) design* checkbox, the *Fractional settings* area will be enabled. The 'k' parameter equals the number of factors, here you can set the 'p' parameter.

With this option, the design will only use the ½, ¼ etc. of the 2^k runs (hence the 2^{k-p} in the name). Then this design will be called a *Fractional Factorial Design*. The *Column generators* list will contain the fractional design generators by default, you only have to use the controls at the right, if you want to change the default settings, or the particular 'k-p' setting is not supported by default.

The resolution of the built-in fractional design generators is displayed after the 2^{k-p} *design* text. The resolutions of user-built generators are not calculated, a warning message appears instead.

Fractional designs confound factor effects with higher order interactions. Resolution is a number that describes the confounding: in a resolution R design the k-level interaction effects are confounded with the (R-k)-level interaction effects. E.g. in a resolution 4 design the main effects are confounded with 3-level interactions, and the 2-level interactions are confounded with each other.

There is a *Create a centerpoints* checkbox under the *Fractional settings* part. You can check it only if you do not have Boolean or String factors. If you check it, you can choose how many centerpoints you want to add, and your design will have extra runs with all the factors with values at the center. You must have at least one sweep random seed to add more than one centerpoint, because after the first centerpoint run the design will need another random seed values to be able to create new, non-repeating centerpoints.

If you have at least one centerpoint, the result processing of the factorial method will calculate the curvature of the output at the centerpoint. This is a deviation of the model prediction from the sample average measured in percentage. The results can be seen in a version of the original results in Figure 39.

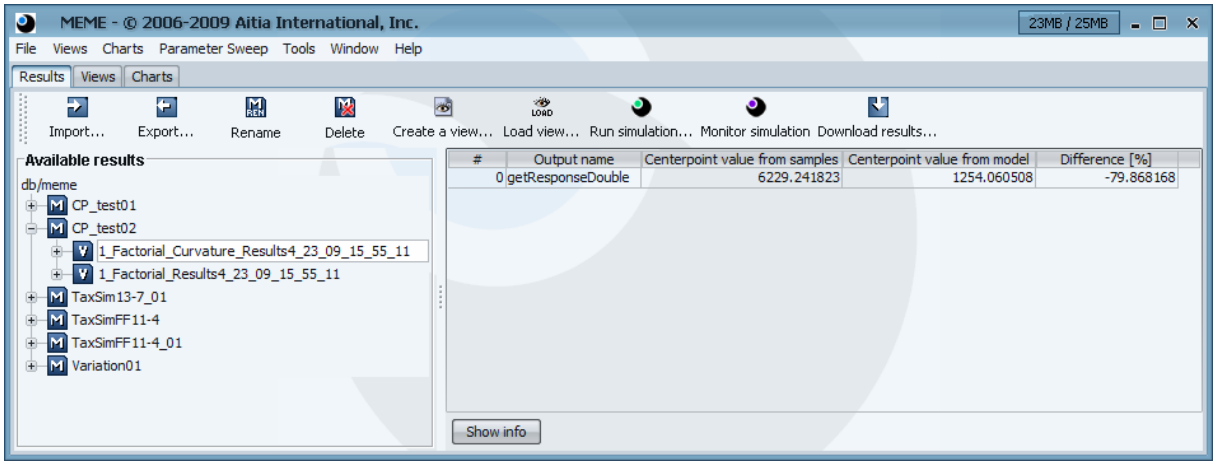


Figure 39 - Factorial curvature results

Under the *Create centerpoints* checkbox there is the *Select result processing method* part. You can select which method you want to use to process the results. The first setting (*Show effect as difference in output between low and high settings*) will show you how much the outputs changed when the input parameters were set from low to high. The other setting (*Show the coefficients of the linear model built on data*) will create a linear model, and you will have its coefficients from which you know how much the output changed when the input changed 1 unit.

Under the *Select result processing method* part there is the *Random seeds* part, where you can manage the random number generator seeds, handle the natural variation measurement, and the blocking. A special dialogue will appear if you press the *Random seed management…* button, where you can handle the seeds. You can read more about this in 3.2.7.8 Random Seed Manipulator Dialog.

The text at the bottom informs you about the total number of runs with the settings you specified. In the case in Figure 38, the design is a 2^{6-2} design that makes 16 runs, plus 3 centerpoints, with that it is 19, and it is multiplied by 7, because of the seven random seed values listed with the *RngSeed* parameter in the *Random seeds* list. It is 133 runs altogether.

3.2.7.2 Latin Hypercube Method

The Latin Hypercube method produces designs where all chosen parameters have the same number of levels, and every parameter level is used only once in the design. So the number of runs is defined by the number of parameter levels in the basic case.

The coordinates of the points in the design were obtained from Space-filling designs [10]. These are Maximin designs, which means the minimum distance between the points in the parameter space is maximal.

The setup screen of this method can be seen in Figure 40.

The upper part of the screen is for selecting the factors that we want to examine. In the *Factors* list on the left, there are parameters with the constant values, which can be changed in the *Default value* textbox by pressing the *Set* button.

On the right there is the *Factors to study* list, which contains the observed parameters. You can move the parameters between the lists with the '->' and '<-' buttons.

You can specify how many levels do you want for every chosen factor. This can be set under the *Factors to study* list with the spinner. You can add new parameters to the model with the *Add new parameters…* button under the *Factors* list. This is described in 3.2.4.3 New Parameters.

Under the lists you can set the low and high values of the selected factor from the list on the right. The method will make equidistant points in the range bounded by the low and high values of the parameter. First, click on a factor on the right, enter the low and high values in the textboxes and click the *Set levels* button. Or, if you use the keyboard, the Enter key in the *Low value* textbox will jump to the *High value* textbox, and the *Enter* in the *High value* textbox will change the values and advance the selection in the *Factors to study* list.

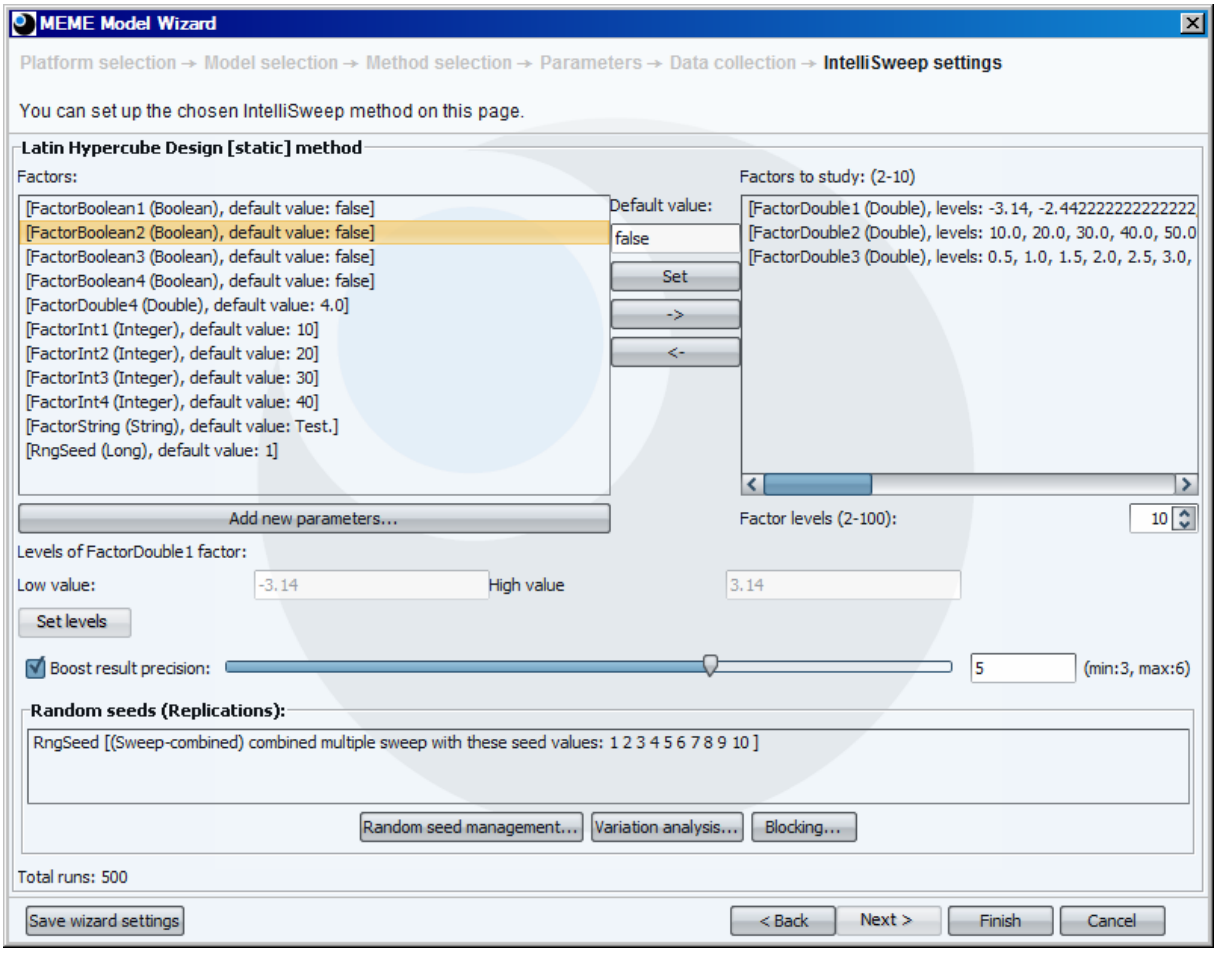


Figure 40 – Latin Hypercube method setup screen

The *Boost result precision* part is an extra feature, which tries to make the results of the simulation more accurate. It permutes the design dimension-parameter associations to create new designs from the original. If k factors are selected to study, then there are k! possible permutations. You can set the boosting factor with the slider, but if k! is larger than 100 (k>4), then that is the maximum number you can set with the slider. You can

however write a larger number in the textbox if you want, but remind that doing so will multiply the number of runs with the chosen boost factor, which could result in a long simulation run.

Under the *Boost result precision* part there is the *Random seeds* part, where you can manage the random number generator seeds. A special dialogue will appear if you press the *Random seed management…* button, where you can handle the seeds. You can read more about this in 3.2.7.8 Random Seed Manipulator Dialogue.

The text at the bottom informs you about the total number of runs with the settings you specified. In the case in Figure 40 there are 10 levels of the parameters, that makes 10 runs, but the boost factor multiplies it with 5, that makes 50, and the random seed settings multiplies this with 10.

3.2.7.3 Iterative Uniform Interpolation Method

Iterative Uniform Interpolation (IUI) was designed to reveal parameter-output correlations applying some rules recursively. In other words the method interpolates the 'curve' of the output by sampling the parameter space more densely where the curve is more 'interesting'.

The measure of interest is defined by a *Gradient* and a *Deviation* value of the expected curve. In the first iteration IUI checks the measured values at the endpoints of each interval (it runs the model with the proper parameters) and when the actual gradient value differs more from the user-defined value than *Deviation*, the method applies itself recursively. From the second iteration the user-defined gradient is replaced with the one measured on the parent interval.

IUI interpolates an *n*-variant, *n*-valued function. The method has the following parameters (see Figure 41):

- *Dimension*: The value of *Dimension* is the upper limit of the number of parameters changing during the experiment. Note: in the current implementation *Dimension* is unalterably 1.
- *Parameters list*: IUI expects at least one, at the most *Dimension* number of model parameters set to *Increment*. Increment type parameters belong to the IUI method's settings. Choose a parameter and select the *Increment* radio button on the *Values* panel to specify the upper and lower bounds of the analyzed parameter values. Finally click the *Modify* button to save changes. The exact of analyzed parameter values are calculated using the *Number of Intervals* setting by dividing the defined interval to the given number of subintervals: the border points are to be analyzed. Constant model parameters can be set in similar way, leaving the *Constant* radio button selected. Random seeds can be set on the *Random seeds* panel.
- *Values*: Set the upper and lower bounds of the inspected interval on the *Values* panel, or assign a constant value to the selected parameter.
- *Add new parameters*: see 3.2.4.3 New Parameters.
- *Output*: The IUI method's output variable can be any model variable, function value or data source value which was added to a recorder previously.
- *Number of Iterations*: The maximal number of iterations during the method (the method terminates when no new *interesting* intervals found).
- *Number of Intervals*: The number of examined intervals in the first iteration. All further examined intervals will be divided into the given number of sub-intervals recursively.
- *Gradient*: Expected gradient of the resulting curve.
- *Deviation*: The tolerated deviation.
- *Random seeds*: See 3.2.7.8 Random Seed Manipulator Dialog.
- *Aggregate result*: Using random seeds the results will be aggregated by the selected function.

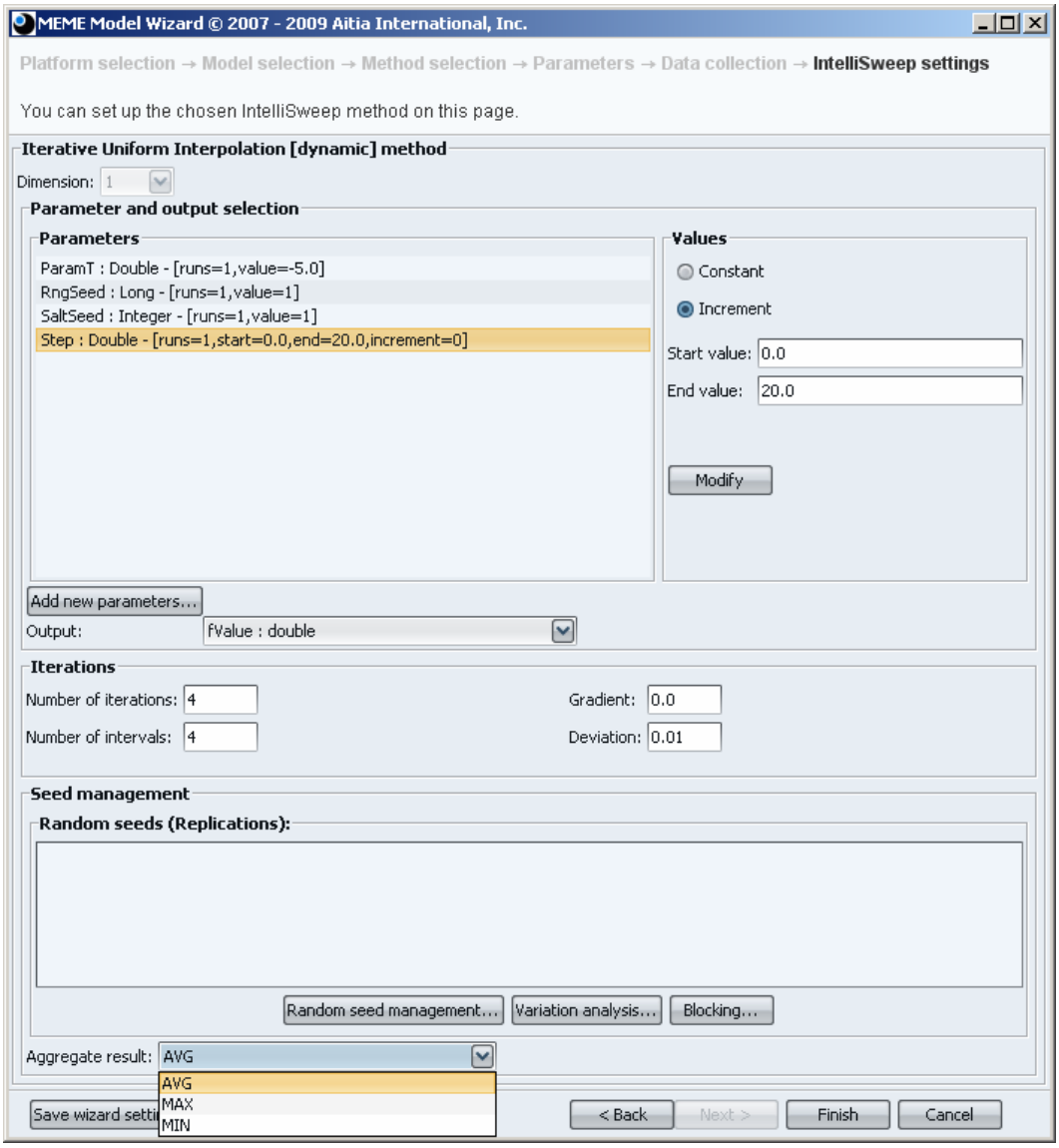


Figure 41 - Settings of the Iterative Uniform Interpolation plugin

When no output variable selected, clicking on the *Finish* button will pop up the following error message:

Figure 42 – Error: no recorded values were added on the recorders page

3.2.7.4 Three-level Response Surface Methodology Designs

The following three methods share a common base: they are response surface designs. This means that they provide designed experiments with results on which one can build a higher order model. This is usually a quadratic model that can approximate the curvature in the data, which the factorial method is not capable of. This is achieved by using more than two values per parameter. All the current response surface methods require three values: low, center and high (center can be automatically calculated). From these the methods create a design in their unique way.

The settings dialogue for these methods are similar. The upper part seen in Figure 43, where you can select the parameters you are interested in, and set the constant values for the rest are the same for every response surface method.

The rest of the dialogue is more or less the same, you can set the low and high values of the chosen parameters, and set the center value manually if you want. You can also handle the random seeds the same way as in other methods, and view the runs the design generated, and also modify them. Note that if you manually modify the runs, the design is no longer an IntelliSweep design.

3.2.7.5 Central Composite Method

This method is an extension of the factorial method. It contains a two-level full factorial design with the low and high values of the parameters, plus a centerpoint, and 'star points' that are on the axes of the parameters.

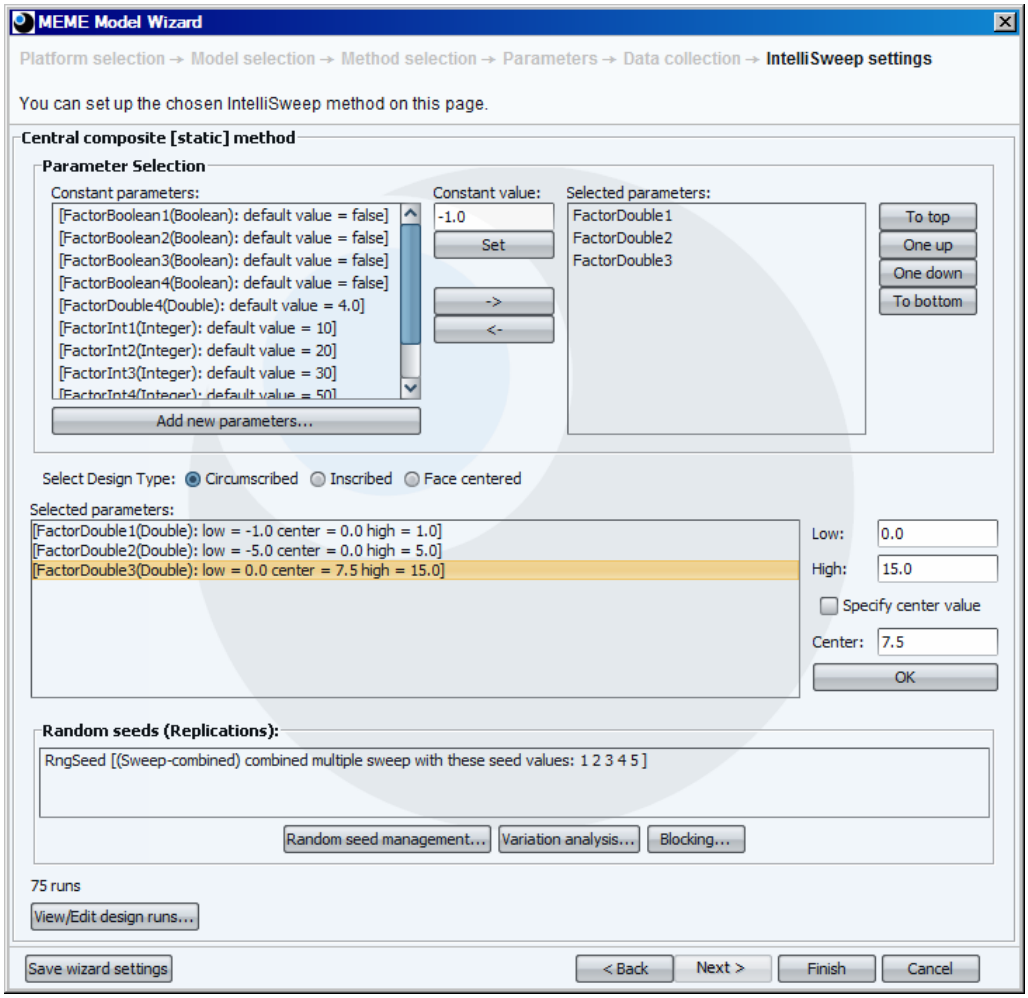


Figure 43 - Central composite method setup screen

There are three different approaches to create designs:

- Circumscribed: The factorial part of the design is at the original low/high values, the additional star points are on a hyper sphere around the hypercube of the factorial.
- Inscribed: This setting is the same as the circumscribed, but it is scaled down to fit into the factorial hypercube. This is useful if the low and high values are true limits and cannot be exceeded.
- Face centered: This setting can be used if the only possible choices for parameter values are the low, center and high ones. This is the only choice if there is at least one String parameter.

You can select which option you want to use under the 'Parameter selection' part of the dialogue.

3.2.7.6 Box-Behnken Method

This is an independent design in the sense that it does not contain a factorial design. The design points are at the center of the edges of the factorial hypercube, and in the centerpoint as seen in Figure 44.

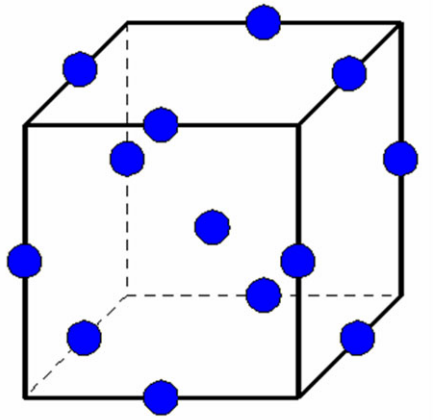


Figure 44 – Box-Behnken design points in 3 dimensions

You can see the Box-Behnken method setup screen in Figure 45, which is very much the same as the central composite setup dialogue.

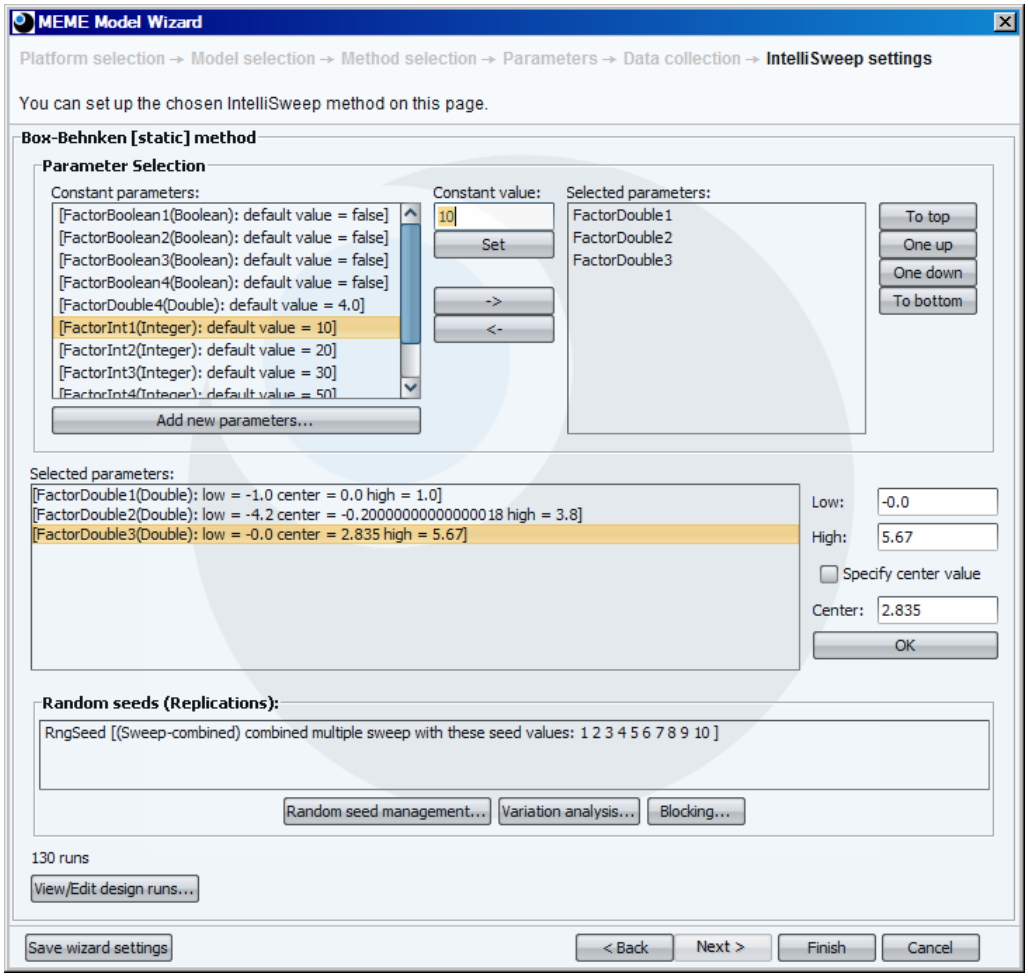


Figure 45 – Box-Behnken method setup screen

3.2.7.7 Three-level Factorial Method

This method uses every possible combination of the three values of every parameter: low, center and high. If there are *k* parameters, this design creates 3^k runs. The setting dialogue is the same as the Box-Behnken dialogue.

3.2.7.8 Random Seed Manipulator Dialogue

This special component cares about the random number generator seeds. There are models, which have many, and they often cannot be fit into the chosen IntelliSweep method's design as a parameter. What this component does, is that it receives a design with no random seed settings, and modifies it according to the seed settings defined on its GUI. The GUI is shown in Figure 46.

There is a simplified GUI that appears on the settings page of every IntelliSweep method. You can double-click the appearing seeds there to bring up the *Random Seed Manipulator* dialogue and modify that seed.

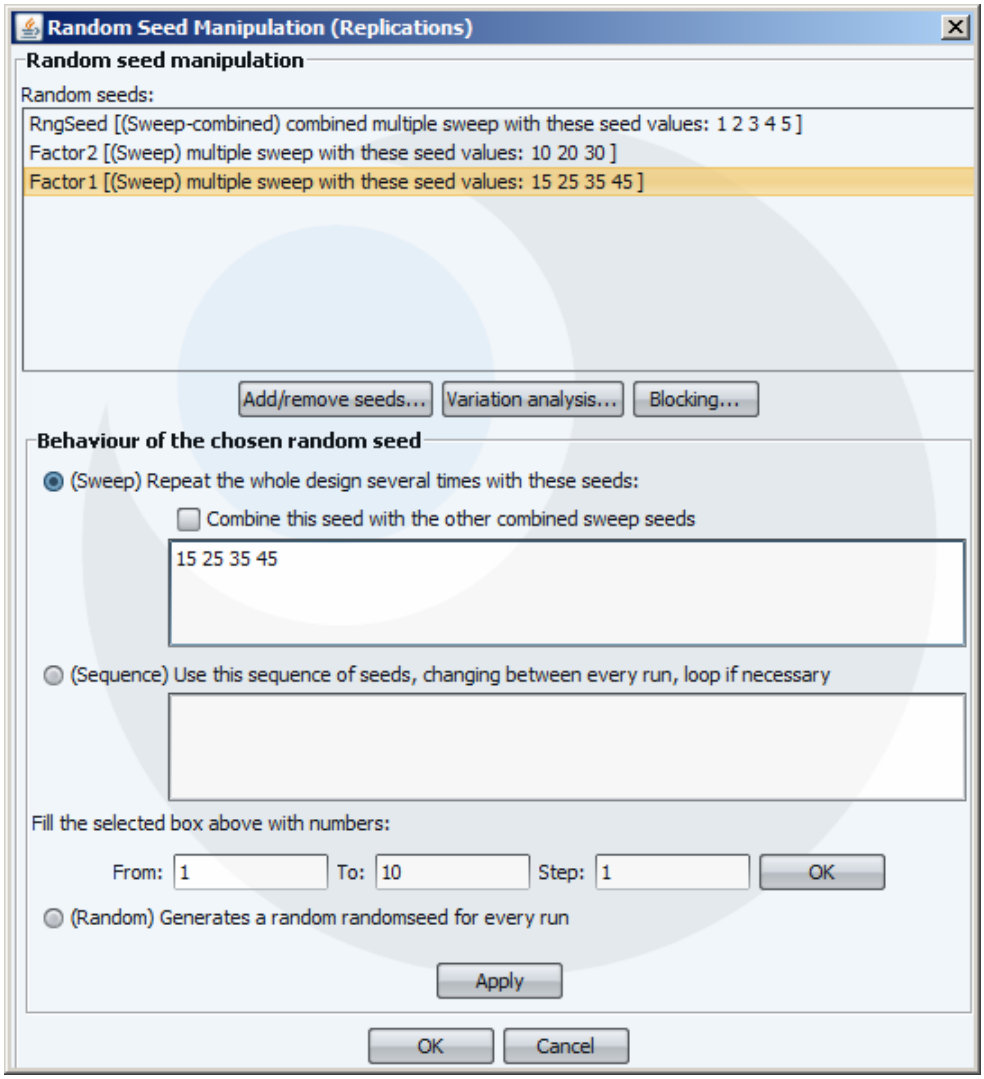


Figure 46 – Random Seed Manipulation dialogue

On the GUI you can choose which parameters you want to use as random seeds. The *Add/remove seeds…* button pops up a dialogue, where you can select the seeds. You can only have numeric seeds.

In the *Behaviour of the chosen random seed* part you can choose a setting for the selected seed. The default setting is the *Sweep-combined* setting that can be seen in Figure 46 as the setting for RngSeed. This means that the whole design will be run multiple times with every combination of the random seed values with this setting. You

can choose not to combine sweep seeds, the result is the *Sweep* setting. All sweep seeds will step their value simultaneously after a whole design run, but stop as the shortest list ends. The *Sweep-combined* seeds will be combined in every possible combination.

The settings in Figure 46 result in 15 whole design replications (the shorter Factor2 have 3 values that will be combined with the 5 values from RngSeed).

If Factor1 was *Sweep-combined*, there would be 60 design replications.

The *Sequence* setting takes the seed list, and assigns the next number in it for every run. If it reaches the end of the list, it goes back to the start.

The last one is the *Random* setting, it assigns a random seed value for every run.

You can use the *Fill the selected box above with numbers* part to easily fill the chosen textbox with the number sequence.

You have to click the *Apply* or *OK* button to make the settings change. The *OK* and *Cancel* buttons will close the dialogue.

The two unmentioned buttons under the seed list are the *Variation analysis…* and *Blocking…* buttons. These bring up features that are discussed in the following chapters.

3.2.7.9 Variation and random seed analysis

This feature computes the standard deviation of every recorded output value of the model for a subset of the input parameters. You have to specify which input parameters are you interested in on a dialogue like in Figure 47. You must have at least one sweep random seed to do this.

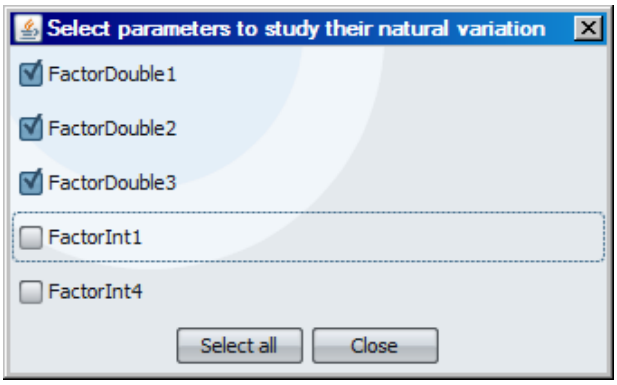


Figure 47 – Natural variation input parameter selection

With this feature you can examine the natural variation of the model's outputs for every input parameter value combinations of the selected parameters. MEME calculates standard deviation and coefficient of variation for every input parameter value combinations (not just the selected ones), then combines the results of the partial results. Combination means weighed arithmetic mean in the case of standard deviation, and weighed harmonic mean in the case of coefficient of variation. Weighs are the sizes of groups created by grouping by every input parameter. After the simulation has run and MEME imported the results, there will appear a version of the model with the name ending with 'Natural_VariationXXXXX' (XXXXX is a timestamp). You can find the standard deviations in it, like in Figure 48.

#	FactorBoolean1	FactorBoolean2	FactorBoolean3	FactorBoolean4	Standard deviation of getResponse	Coefficient of variation for getResponse
0	☐	☐	☐	☐	470.532799	0.165178
1	☑	☐	☐	☐	523.271743	0.194525
2	☐	☑	☐	☐	643.171158	0.239593
3	☑	☑	☐	☐	808.732804	0.285473
4	☐	☐	☑	☐	332.128569	0.123044
5	☑	☐	☑	☐	405.901531	0.142333
6	☐	☑	☑	☐	491.065759	0.172446
7	☑	☑	☑	☐	547.064232	0.203454
8	☐	☐	☐	☑	969.38848	0.36312
9	☑	☐	☐	☑	1225.542744	0.435495
10	☐	☑	☐	☑	1527.381606	0.545374
11	☑	☑	☐	☑	1730.522277	0.656618
12	☐	☐	☑	☑	175.482975	0.061182
13	☑	☐	☑	☑	165.936169	0.061181
14	☐	☑	☑	☑	165.932117	0.06118
15	☑	☑	☑	☑	175.471664	0.061179

Contains 16 rows, 6 columns Show info

Figure 48 – Processed natural variation data

A version of the model with the name ending 'Seed_StatisticsXXXXX' will be also generated, that lists the random seeds used in the simulation, the size of their values, and the combined standard deviation and coefficient of variation on the whole dataset. You can see an example of this in Figure 49.

#	Random Seed	Seed values size	Combined standard deviation of getResponse	Combined coefficient of variation for getResponse	Combined standard deviation of getResponseDouble	Combined coeffici...
0	RngSeed1	5	653.118988	0.001421	72733.75291	0.001421
1	RngSeed2	6	248.259651	0.081901	27644.629445	0.081901
2	RngSeed3	7	60.029632	0.019548	6684.567861	0.019548

Figure 49 – Seed Statistics

3.2.7.10 Blocking Handling

This feature is a special one. In computer simulations one usually does not need the concept of blocking, because the simulations are deterministic. At this point MEME does not help you with blocking in non-deterministic simulations, because this feature is about something else.

Blocking in the Design of Experiments is about eliminating the effects of unwanted and uncontrollable variables in the process. In computer simulations, usually everything is controllable. If you select parameters for blocking in MEME, those parameters will be created as if they were blocking parameters, and the result processing will calculate the standard deviation of the response variables for every values of every blocking variable. The result of this calculation will appear as a version of the results with the 'Blocking' string and a timestamp in it. You can see one result table in Figure 51.

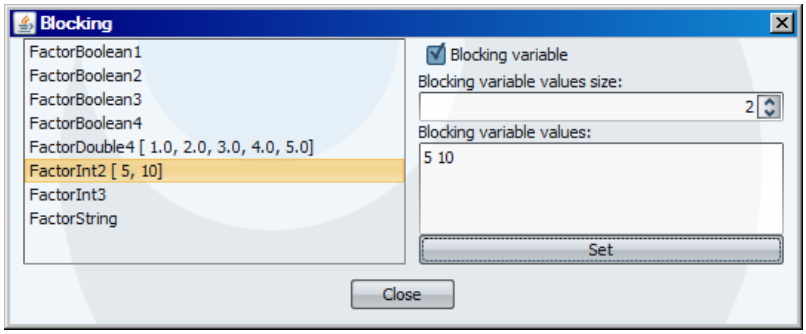


Figure 50 – Blocking setup dialogue

With this 'blocking behavior' MEME will turn the selected blocking variables into 'nuisance' variables, which are blocked, and one can measure whether that variable had any impact on the result variables.

The blocking dialogue seen in Figure 50 will appear if you press the *Blocking…* button in the *Random Seed Manipulator*. You will be offered all the parameters not selected in the design, and you can make blocking variables from them by setting how many values they will take and what those values are.

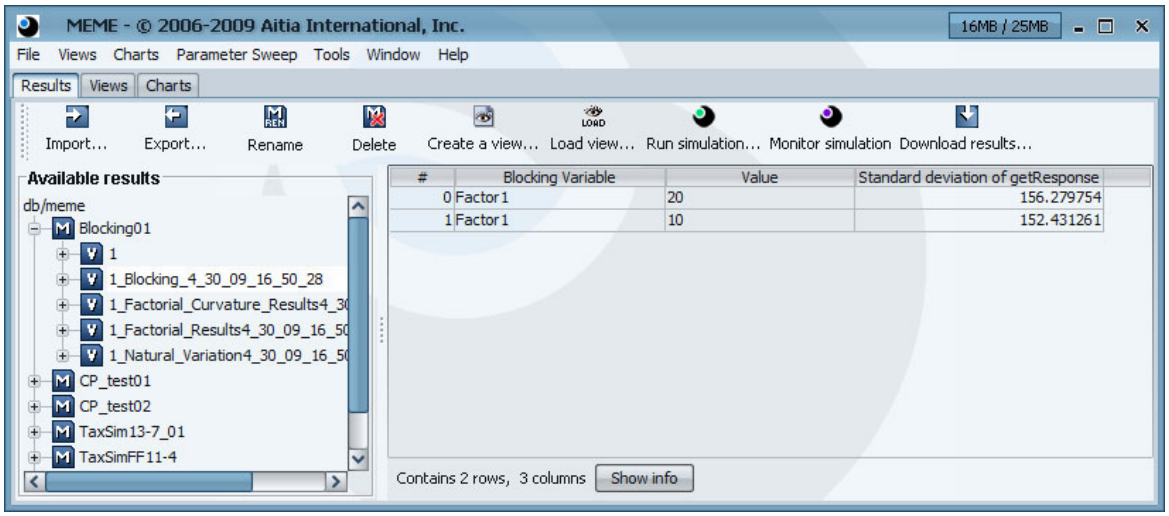


Figure 51 – Processed blocking data

3.3 Local Monitor

When you run (by pressing the *Finish* button of the Parameter Sweep Wizard) a model on the local computer, a new panel appears on the main window of MEME called *Local Monitor*. The *Local Monitor panel* is shown on Figure 52.

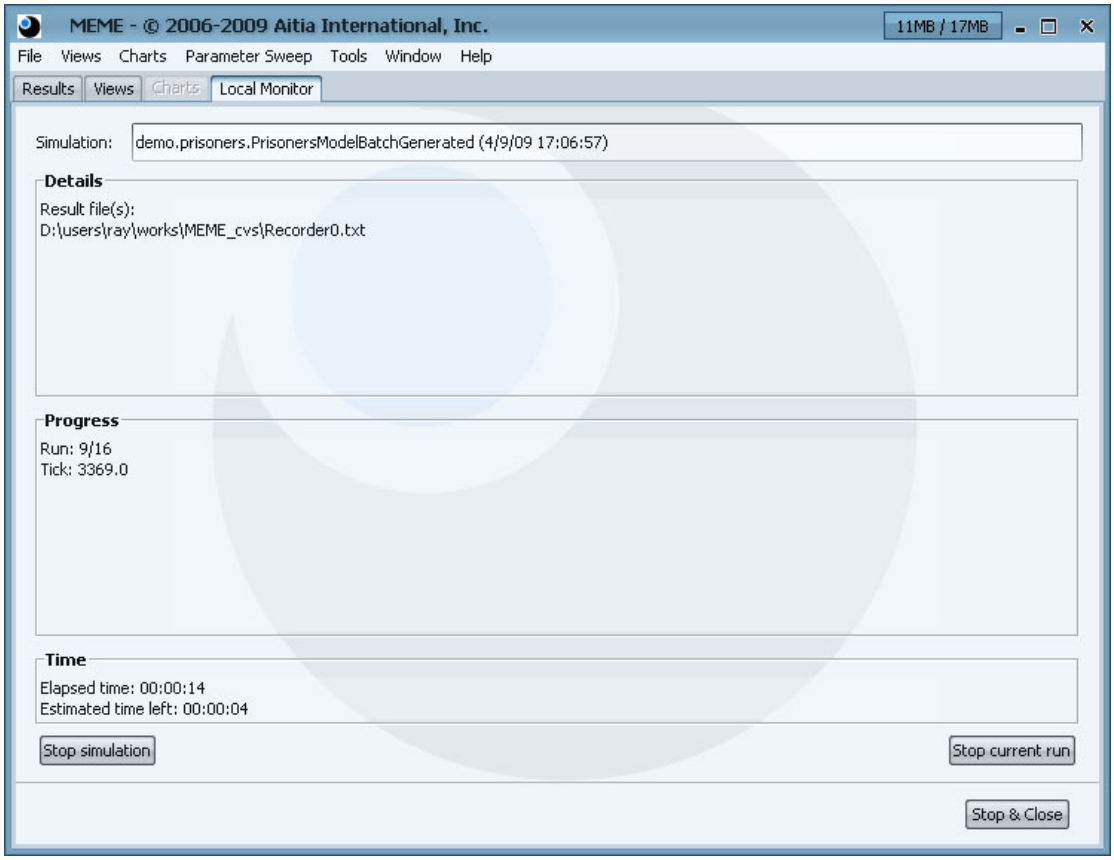


Figure 52 - Local Monitor panel

The monitor displays the following information about the running model:

- Full name of the model (with the date and time of the generation/start);
- Name of the output file(s);
- Elapsed time of running;
- Remaining time of running (Note that this is only an ***estimated*** value);
- Number of the current and the total runs, and
- Number of the current time step in the current run (usually called *tick*).

If you select a dynamic method on the *Method selection* page the number of the current iteration is also shown.

The *Stop current run* button stops the current run and starts the next one (if any). The *Stop simulation* button stops the whole simulation.

When simulation is done an import dialogue appears on which you can import the results to the database of MEME. See in details in section 4.5 File Import.

3.4 Monitor Preferences

The monitor tool enables users to follow the progress of simulations running on clusters or grids of computers and to import the output of finished simulations to the database of MEME (or downloading it to the local file system).

If it is started automatically by the Parameter Sweep Wizard (see section 3.1.3), the monitor inherits the network settings from the wizard so there is no need for manual configuration. If started separately though, the details of the network connection have to be specified in the *Monitor configuration* dialogue (see Figure 53).

To open the dialogue select the *Monitor preferences…* menu item from the *Parameter Sweep* menu. Note that it can be accessed directly from the *Monitor* panel, too, by pressing the *Configure…* button.

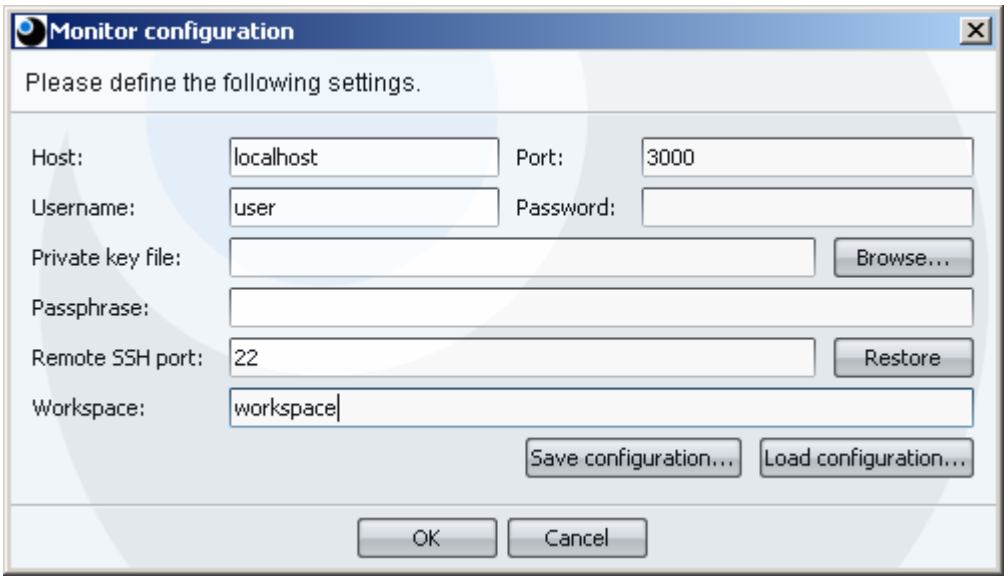


Figure 53 – Monitor configuration dialogue

As you can see, the settings on this dialogue are very similar to settings on the *Preferences* dialogue.

- *Host*: Hostname (or IP-address) of the simulation server;
- *Port*: Port of the simulation server (default: 3000);
- *Remote SSH port:* The SSH port of the SFTP-server (default: 22).
- *Workspace*: workspace on the server (where they are on the remote host).

A username/password pair also needs to be given, because the monitor uses the SFTP protocol to download the output of simulations. Some SFTP-servers doesn't support this

kind of authentication, they use private keys instead (with or without a password called passphrase). The *Configuration* dialogue allows specifying both types of authentication methods.

Warning: You must define the workspace relative to the default user directory on the SFTP-server.

When clicking *OK* button the application automatically saves the wizard settings, just like pressing the *Save configuration…* button does after choosing a place and a name for the preferences file. With the *Load configuration…* button you can load previously stored settings.

3.5 Monitor

The monitor tool enables users to follow the progress of simulations running on clusters or grids of computers and to import the output of finished simulations to the database of MEME (or downloading it to the local file system).

To start the monitor select the *Parameter Sweep/Monitor simulation* menu item or press the button in the *Results* panel. The monitor may start automatically after the Parameter Sweep Wizard is closed if the appropriate option on the wizard's *Preferences* dialogue is checked (see 3.1.3 Network Settings).

3.5.1 Current simulation Tab

The monitor collects and displays information about the model currently run by the simulation server (see Figure 54).

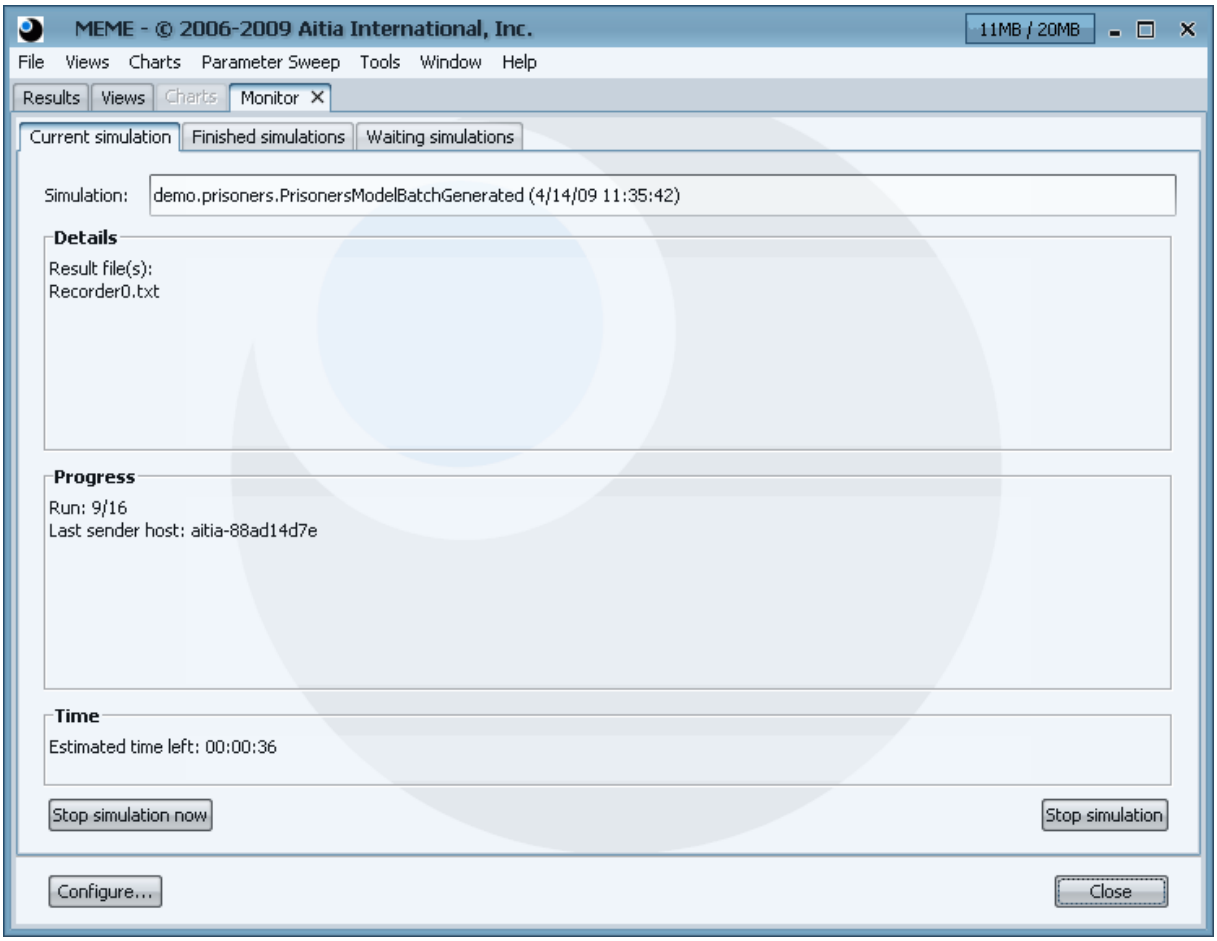


Figure 54 – Current simulation

The full name and description (if any) of the models is shown, together with the time of uploading and name of the output files. Naturally, the progress of the given model is also displayed: the number of the current run and the number of the maximal run are shown

at the middle section, together with the name of the *Last sender host* the computer that performs the current run. The ***estimated*** remaining time is displayed at the bottom section.

By pressing the *Stop simulation* button you can order the simulation server to finish the model that is currently run and start the next one. Note that the simulation server finishes the current runs before it stops the running of a model. If you wish to finish the simulation immediately use the *Stop simulation now* button.

3.5.2 Finished simulations Tab

The second tab (Figure 55) of the monitor displays information about and gives access to *Finished simulations*. Results of completed experiments can be downloaded from here or imported directly to the database of MEME.

The table on the left displays finished simulations, identified by the full name, the platform of the model and the time of upload. This table is automatically updated whenever a simulation experiment is completed on the server. (The table can also be updated manually by pressing the *Update list* button.) You can sort the content of the table by name, by date or by platform by clicking on the appropriate column name. Clicking again on the same column name reverses the order of sorting.

Upon selecting a finished simulation from the list, the description of the selected simulation and the list of its output files (and some log files, too) appears on the right. By pressing the *Download* button the *selected* output files are transferred to the local computer running the application. If you wish to import the selected output file directly to the database of MEME press the *Import to database…* button instead. The appropriate platform-specific import dialogue appears. See in details in section 4.5 File Import.

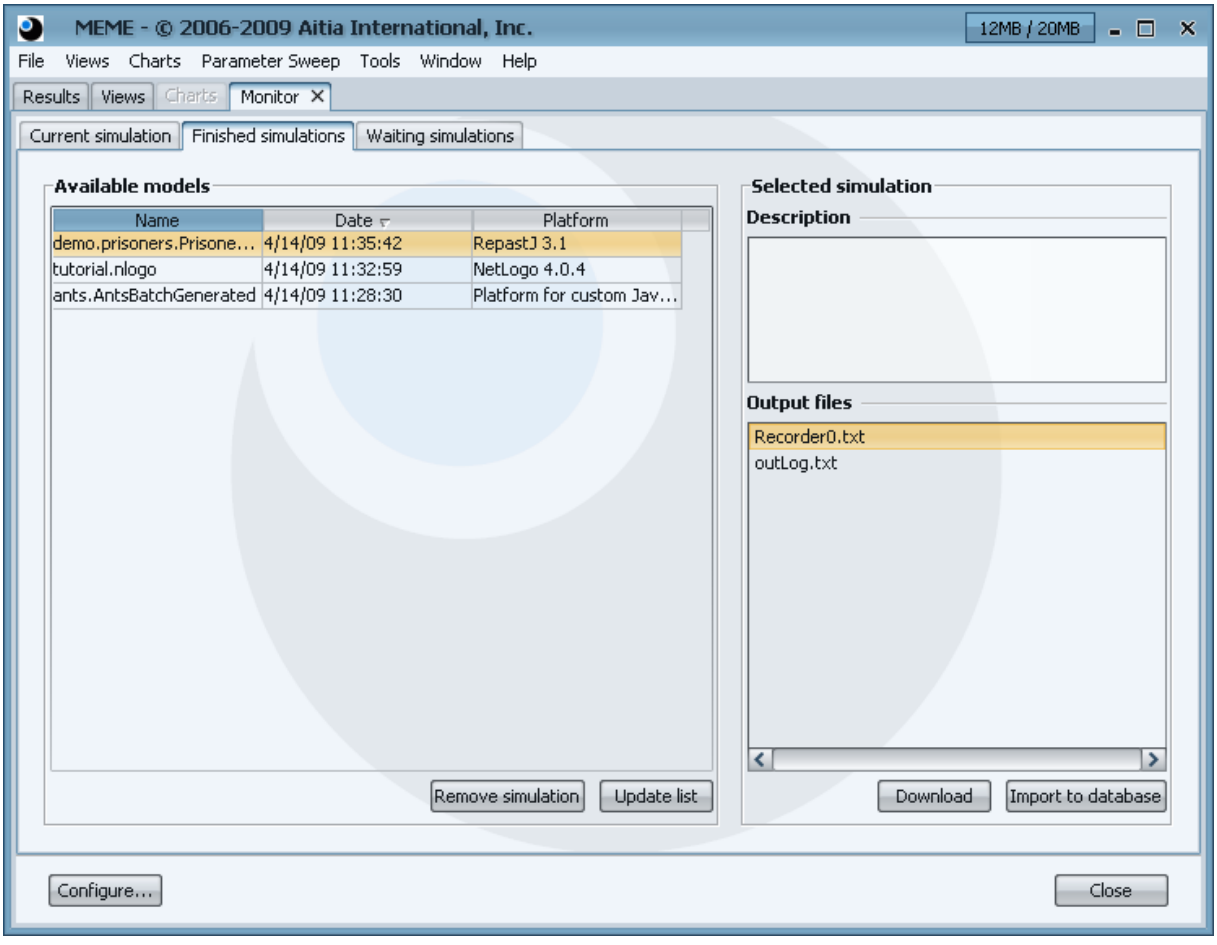


Figure 55 – Finished simulations

Note: Please do not try to import the log files (outLog.txt or errLog.txt) to the database because it may cause an error.

Simulations aborted due to an error are marked by red color and their names start with ERROR. While there are no output files for these simulations you can still download the log files that may help to find the cause of the error.

You can remove the selected simulation from the server by pressing the *Remove simulation* button. All output files are deleted and this cannot be undone, so use this button carefully.

3.5.3 Waiting simulations Tab

The *Waiting simulations* tab is shown on Figure 56.

When you start a simulation with the Parameter Sweep Wizard, the wizard sends a simulation request message to the server application. The server application stores the simulation requests in a queue and runs the simulations one after the other. The earliest request will be served first.

The *Waiting simulations* tab always shows the current content of the abovementioned queue.

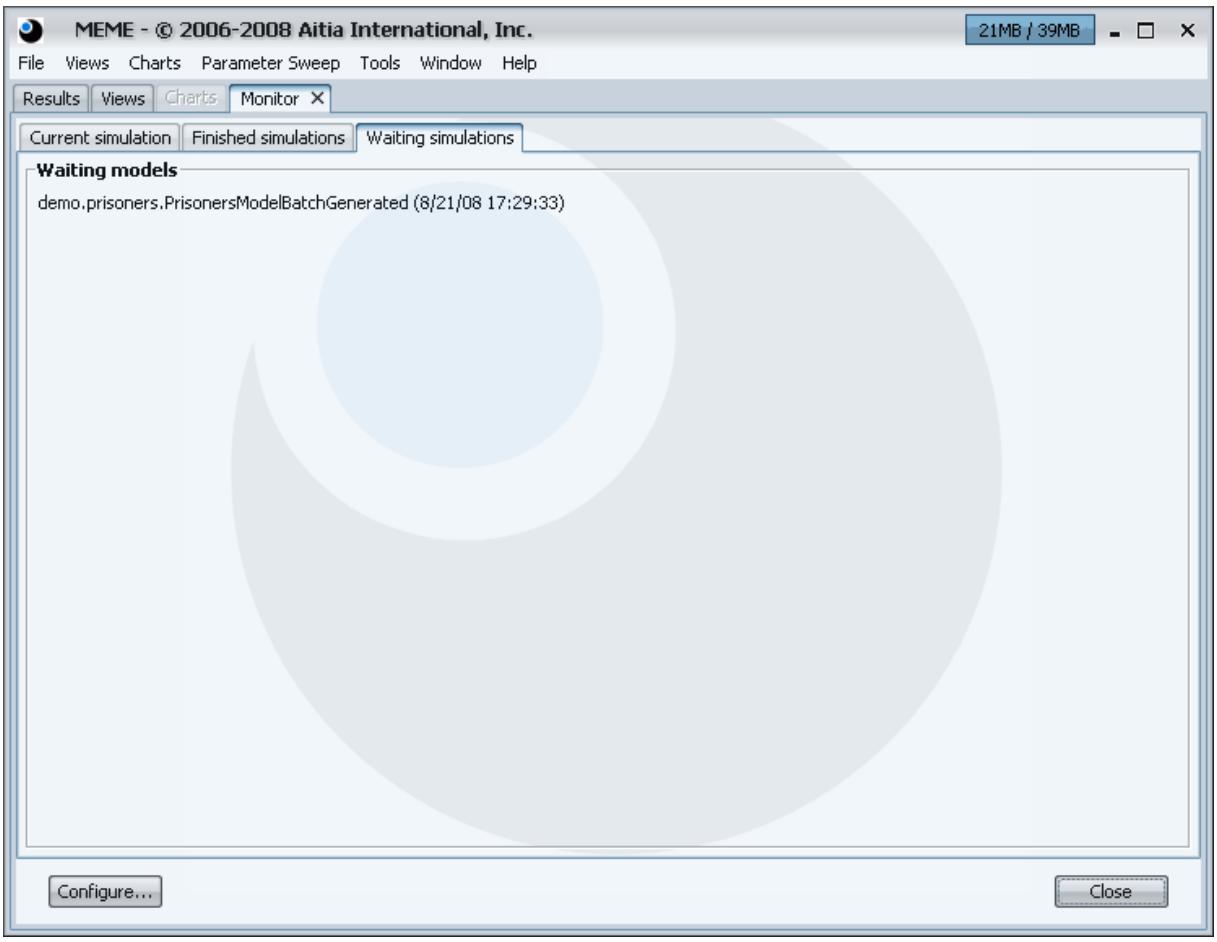


Figure 56 – Waiting simulations

3.6 Run Simulation on the Model Exploration Server

If you choose the *Run simulation on the Model Exploration Server* experiment option on the *Network* page of the *Preferences* dialogue, an *Authentication* dialogue appears after you press the *Finish* button (see Figure 57).

You have to specify the e-mail address and password here (please register at the Model Exploration Service website[3] to acquire these information), and the required service

[3] http://modelexploration.aitia.ai

mode. If you select the leased mode you also have to define the number of requested processors.

The values you might specify on the *Network* page of the *Preferences* dialogue are used as defaults here.

When you press the *OK* button, the application sends your authentication information to the Model Exploration Server. If these information is valid, the application transfers the simulation files and the server schedules the experiment. A confirmation message appears on the screen if the experiment request is approved (or a warning in case of rejection).

Currently, you can't monitor your experiment with MEME. However, at the Model Exploration Service site you can follow the progress. Furthermore, e-mail notification is sent to you if the experiment is finished (or aborted).

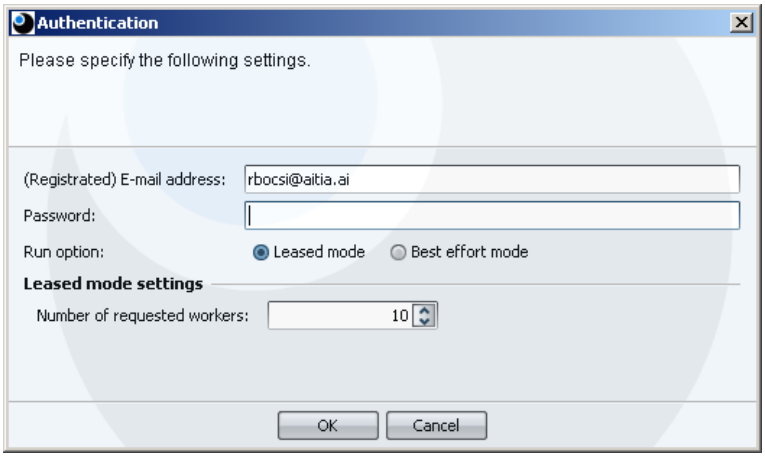


Figure 57 – Authentication dialogue

After the experiment is finished, you can download (or imported directly to the database) the results with MEME (see Figure 58)).

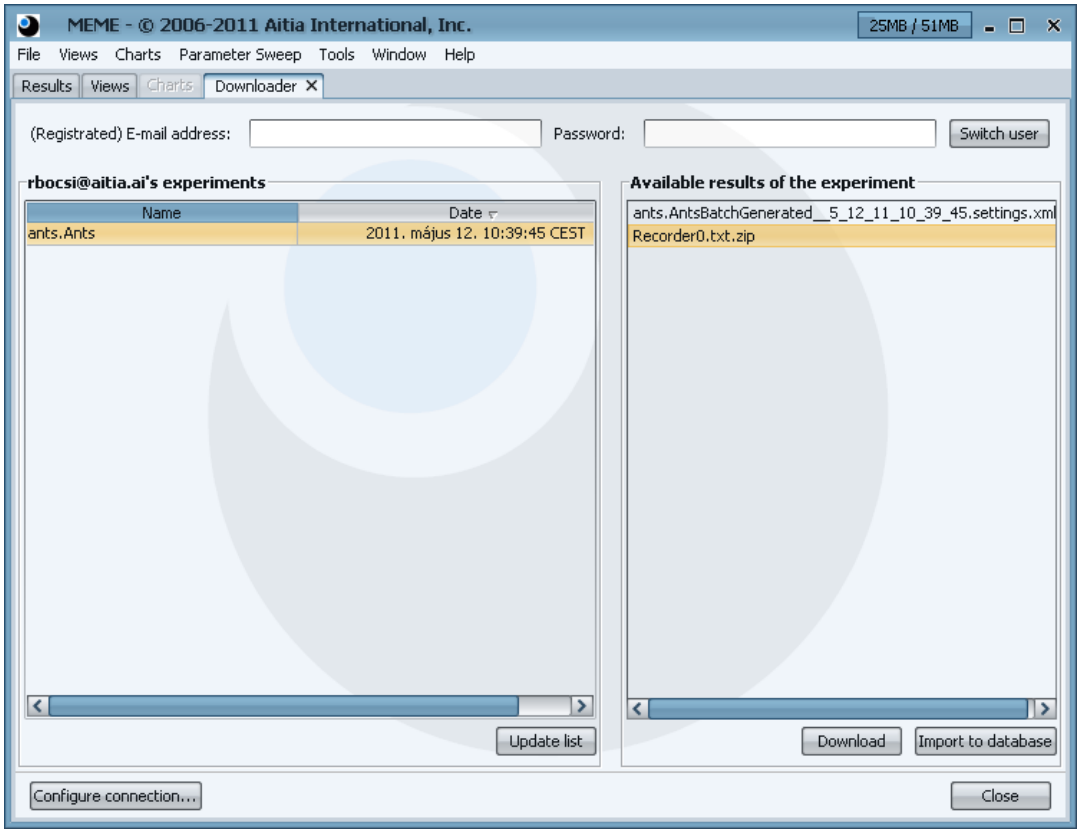


Figure 58 – Downloader tab

To open the downloader tab (see above) select the *Parameter Sweep/Download results from cloud…* menu item or press the button.

As you can see, this tab is very similar to the *Finished simulations* tab of the monitor tool previously described.

By default, it lists the finished experiments of the user specified on the *Network* page of *Preferences* dialogue. If the e-mail address and/or password is not defined there, the experiment list is empty. However, you can specify the user on the top section of this tab (or you can switch to another user's experiment list).

3.7 Create Simulation Job for QosCosGrid

Besides running the model on the local machine or an available cluster MEME can create a simulation job for a dedicated special-purpose multiprocessor computing system named QosCosGrid[4]. You can choose this option on the *Network* tab of the *Preferences* dialogue (see in 3.1.3 Network Settings section).

If this option is selected when you press the *Finish* button MEME generates the simulation job (a "Creating QosCosGrid package… Please wait!" message appears on the screen). A simulation job is a bunch of files that contain the model itself, all related resources and configuration files that describe the job.

Note: Generating the job may take some time so please be patient.

When the generating is done the following message dialogue appears:

Figure 59 – QosCosGrid job generation is done

The files related to the simulation job are available in the specified directory (a model-specific subdirectory of the *<MEME installation directory>\QosCosGridPackages* folder). You have to upload these file to the QosCosGrid website to run your simulation on QosCosGrid. See details about performing a job on QosCosGrid at the following site: http://www.qoscosgrid.eu/

3.8 Platform for Custom Java Models

The platform for custom Java Models is a support platform. It enables the users to use the capabilities of the MEME Parameter Sweep Wizard with any models created in pure Java. As you will see, this platform is very similar to the RepastJ platform from the point of view of the usage of the Parameter Sweep Wizard.

The platform for custom Java Models is not a built-in platform (it is not contained by the installer or the archive file). If you want to use, you have to download independently from our website: http://mass.aitia.ai/.

The following sections will provide all necessary information that helps you installing the platform under your MEME and preparing your model to co-operate with our tool. At the end of the chapter the platform-specific parts of the wizard is discussed.

3.8.1 Platform Installation

If you download this platform, a simple JAR file named *CustomJavaPlugin.jar* to be given (it is in a zip archive file). To install the platform simply copy this JAR file to the *PlatformPlugins* folder that is located under the installation directory of MEME.

[4] http://www.qoscosgrid.eu/

Before you can use it, you have to restart MEME and then register the platform in the Parameter Sweep Wizard (see details about registration in section 3.1.2 Platform Settings).

3.8.2 Model Preparation

The platform for custom Java Models enables the users to create and run batch experiments with any models created in pure Java. To achieve this some minimal modifications must be done on the original model which require some programming skills in Java.

- All variables of the model must have to at least ***protected*** visibility. Otherwise the Parameter Sweep Wizard cannot see them and use them in the batch model.
- There has to be a nullary constructor (a constructor with no arguments) in the model class. The constructor must set the default value of all parameters.
- The model class must implement the `ai.aitia.meme.paramsweep.platform.custom.ICustomModel` interface. See details later about the interface.
- At the end of every step of the model the `stepEnded()` method (specified by the previously mentioned interface) must be called.
- Whenever the model want to exit in a normal way (not because of an error) it has to do this by calling the `simulationStop()` method (specified by the previously mentioned interface).

3.8.2.1 The `ICustomModel` interface

The binary class (and the source code) can be found in the *meme.custom.model.jar* file which is located in the *resources* folder of the MEME installation directory. You have to extend the class path of your model with this jar file before you compile the modified model.

The interface specifies the following methods:

- `double getCurrentStep()`: Returns the number value of the current step of the simulation.
- `void simulationStart()`: Initializes and starts the simulation. It may throw `SimulationException` (defined in the `ICustomModel` interface as an inner class) if any problem occurs during the initialization or simulation starting. Please make sure that all variables using parameter values are initialized here instead of the constructor because actual parameter values are set after constructor invoking.
- `void simulationStop()`: This method must stop the entire simulation. You must use this method if your simulation wants to stop itself
- `void stepEnded()`: Implement this method with empty body. The wizard will generate the body of this method before the execution of the batch experiment.
- `String[] getParams()`: Returns the names of the parameters of the model in an array. For each name *X* contained by `getParams()` array you have to specify a public `getX()/isX()` (latter only if *X* is a boolean parameter) and a public `setX()` method in the model *OR* alternatively a public `Object getParameter(String)` and a public `setParameter(String,Object)` method. `getParameter("X")` must return the current value of parameter *X* while `setParameter("X",value)` must set the value of parameter *X* to *value*. Note that in MEME, *param* and *Param* denotes the same parameter because of the name conventions of getters/setters (e.g. `getParam()`).

3.8.3 Using the Wizard with the Platform

As it is mentioned earlier the platform for custom Java models is designed to be similar to the RepastJ platform as much as possible (for example, both platforms use the same parameter and output file format). For this reason using the Parameter Sweep Wizard with this platform is almost identical when it is used with the RepastJ platform. However, there is a slight difference:

- On the *Model selection* page you cannot specify an existing parameter file for the wizard;

3.9 The NetLogo 4.1.1 Platform

This platform enables the users to use the capabilities of the MEME Parameter Sweep Wizard with any models written in NetLogo 4.1.1. (Because of the minor differences, MEME supports all NetLogo 4.1.x versions.)

Note: Currently, NetLogo 5.0 is not supported. The previous versions (NetLogo 4.0.4 and earlier) are also incompatible with the current version of MEME. However, some earlier MEME versions supports only NetLogo 4.0.4 (see the official website for details).

The NetLogo 4.1.1 platform is not a built-in platform (it is not contained by the installer). If you need to please download from our website: http://mass.aitia.ai/.

The following section will provide all necessary information that helps you installing the platform under your MEME. After that some limitations is discussed. At the end of the chapter the platform-specific parts of the wizard is explained in details.

3.9.1 Platform Installation

- ***Step 0***: To use MEME with NetLogo 4.1.1, you have to be installed NetLogo 4.1.1 on your computer. So if you don't have one, install it. You can download NetLogo 4.1.1 from the official NetLogo website (see [11]).
- ***Step 1***: Extract the netlogoplugin.zip file to a temporary folder.
- ***Step 2***: Stop MEME if it is running.
- ***Step 3***: Start the NetLogo Platform Installer application by double-clicking on the NetLogoInstaller.bat file or typing the following command to a console: java –jar NetLogoInstaller.jar and then pressing *ENTER*.
- ***Step 4***: The graphical user interface of the application is shown on Figure 60.

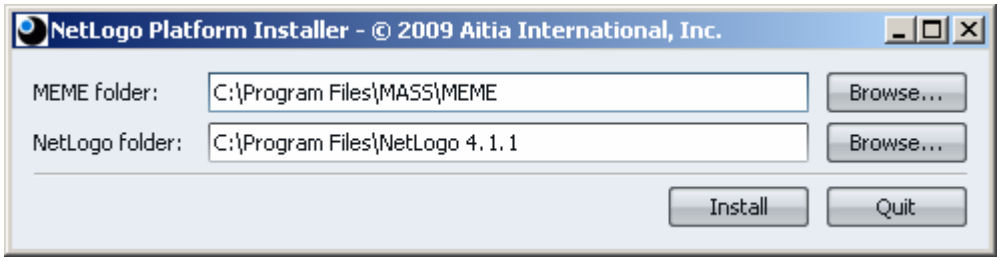


Figure 60 – NetLogo Platform Installer application

  You have to specify the installation directory of MEME and the NetLogo 4.1.1 either typing the absolute path of the directories to the appropriate field or pressing the corresponding *Browse…* button and selecting the directories from a file dialogue. Note that if you install the programs to their default location the text fields are already contains the paths so you can skip this step.
- ***Step 5***: Press the *Install* button. When installation is done a message dialogue appears.
- ***Step 6***: Before you can use it, you have to register the platform in the Parameter Sweep Wizard (see details about registration in section 3.1.2 Platform Settings).

3.9.2 Limitations of the Platform

As it is mentioned earlier the platform offer only limited support at this time for NetLogo modelers:

- It is mandatory to name the setup method of the model as *setup* and the go method of the model as *go*.
- Because of the lack of type information all reporters without arguments appear as recordable elements, however only numerical, textual or logical elements should

be added to the recorders. Otherwise importing the output of the recorder to the database of MEME may cause error or unexpected behaviour.

- NetLogo 4.1.1 supports only one recorder per experiment.

3.9.3 Using the Wizard with the NetLogo 4.1.1 Platform

This chapter discusses the changes of the Parameter Sweep Wizard if you choose NetLogo 4.1.1 platform on the *Platform selection* page instead of RepastJ.

3.9.3.1 Model Selection

At NetLogo 4.1.1 platform, you have to select the .nlogo file as model file on the *Model selection* page.

The class path list and the buttons belongs to it are disabled because the platform doesn't use user defined class path. However, if your model uses extensions that are located in the model directory and you want to run the model in a cluster or grid of computers, you have to specify the extension JARs as resources. See details about resources in section 3.2.2.2 Resources.

The name of wizard settings descriptor file of a NetLogo 4.1.1 model is very similar to the name of a RepastJ model's file as the following example shows:

The filename tutorial.nlogo__4_10_09_15_38_28.settings.xml indicates that:

- *The model name is tutorial.*
- *The model is an original (not modified) NetLogo model ("nlogo").*
- *The date of the experiment creation is 10 April, 2009.*
- *The time of the experiment creation is 15:38:28.*

If you record created data sources the name is slightly changed. Here is another example:

The filename tutorial_aitiaGenerated__2_9_10_11_42_00.settings.xml indicated that:

- *The model name is tutorial.*
- *The model is a generated based on the original NetLogo model ("aitiaGenerated").*
- *The date of the experiment creation is 9 February, 2010.*
- *The time of the experiment creation is 11:42:00.*

3.9.3.2 Parameters

At a NetLogo 4.1.1 model all interface elements that represent a numerical (e.g. a slider), a textual (e.g. a chooser) or a logical value (e.g. a switch) are shown automatically in the parameter tree (see Figure 61). If you want to use the *random-seed* as a parameter please check the *Always use 'random-seed' as a parameter* checkbox on page *NetLogo 4.1.1* of the *Preferences* dialogue (see 3.1.2.3 for details).

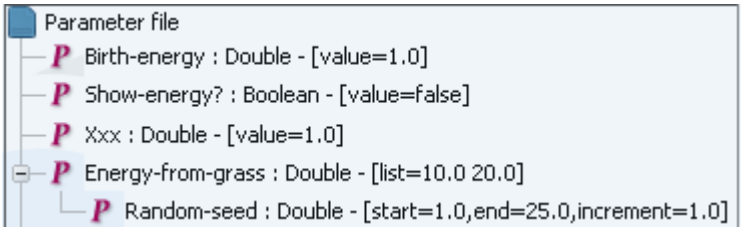


Figure 61 – Parameter tree of a NetLogo model

You can change any global variables (including the built-in ones) of the model with appropriate type to a parameter by using the *Add new parameters…* button. See details in section 3.2.4.3 New Parameters.

NetLogo 4.1.1 platform does not support the parameter level *Runs* value which was introduced at RepastJ platform. It supports only a global *Runs* value which means all defined parameter combinations will be are going to be explored *Runs*-times. This global *Runs* value can be modified at the top section of the left panel (see Figure 62).

Figure 62 – Global Runs field

For example, the parameter tree on Figure 61 with the global runs value 2 means 100 runs (2 x 25 x 2 runs) in the experiment.

MEME uses an own parameter file format for NetLogo models because NetLogo doesn't separate the definition of parameter combinations from the other parts of the experiment description. The NetLogo parameter file is a simple XML-document; here is an example generated from the parameter tree on Figure 61 with the global runs value 2:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<sweep runs="2">
  <parameter definition_type="constant"
             name="Birth-energy"
             type="Double"
             value="1.0"/>
  <parameter definition_type="constant"
             name="Show-energy?"
             type="Boolean"
             value="false"/>
  <parameter definition_type="constant"
             name="Xxx"
             type="Double"
             valid_values="1.0 2.0 3.0 4.0 5.0"
             value="1.0"/>
   <parameter definition_type="list"
              name="Energy-from-grass"
              type="Double"
              values="10.0 20.0">
     <parameter definition_type="iteration"
                end="25.0"
                name="Random-seed"
                start="1.0"
                step="1.0"
                type="Double"/>
  </parameter>
</sweep>
```

This means that the *Energy-from-grass* parameter will take two values, 10 and 20, and for both of these values the *Random-seed* parameter will iterate over the values of 0, 1, … up to 25. The model will run twice with all parameter combinations (thanks to the global runs value).

The *Xxx* parameter is a chooser parameter which means its value always must be one of the elements of the *valid_values* list. The wizard checks this constraint whenever you try to modify the parameter.

3.9.3.3 Data Collection

NetLogo 4.1.1 platform offers only limited set of recording time options (*At the end of every iterations* and *At the end of runs*) as shows and only one writing time option (*At the end of runs*). The *Advanced…* button is disabled because there is no another writing time alternative.

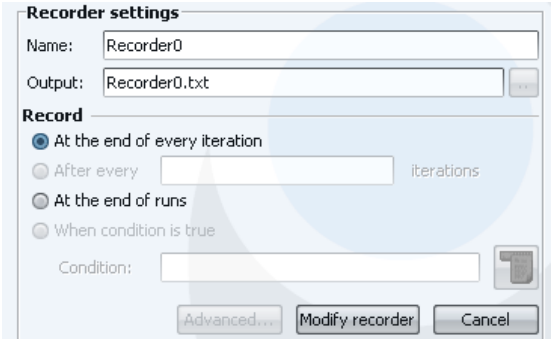


Figure 63 – Recorder settings of at NetLogo 4.1.1 platform

As it is mentioned previously variables and methods (a.k.a. reporters in NetLogo) can be used as recordable elements as well as derived data sources (see details later).

The *Variables* list contains all user-defined (e.g. *param1*) and built-in (e.g. *world-width*) global variables with appropriate type except the ones that become parameters.

The *Methods* list contains all reporters without arguments of the model. The *unknown* type means that the wizard could not identify the return type of the reporter. See also 3.9.2 Limitations of the Platform.

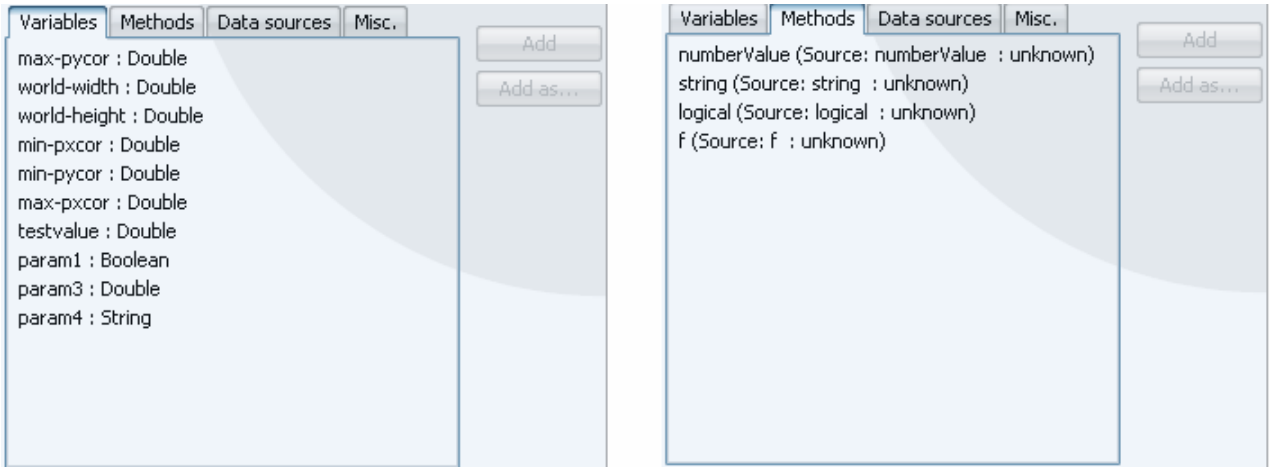


Figure 64 – Variable and Reporter recordable elements of a NetLogo 4.1.1 model

Creating alias names for recordable elements is not supported by the platform, therefore the *Add as…* button is disabled.

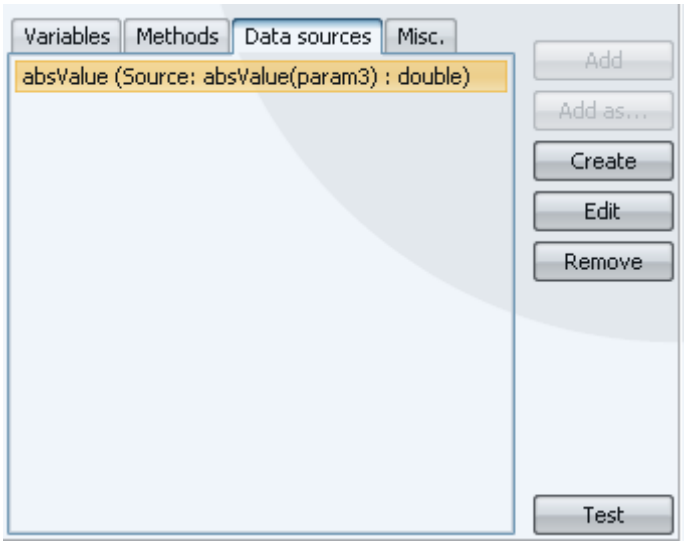


Figure 65 – Data sources tab at NetLogo 4.1.1 platform

As you can see on Figure 65 besides creating, adding, editing and removing data sources there is one more option: testing the data sources. If you use the Test button the wizard generates and runs a temporary model that uses the selected data sources. The results of testing are shown in a dialogue (see Figure 66).

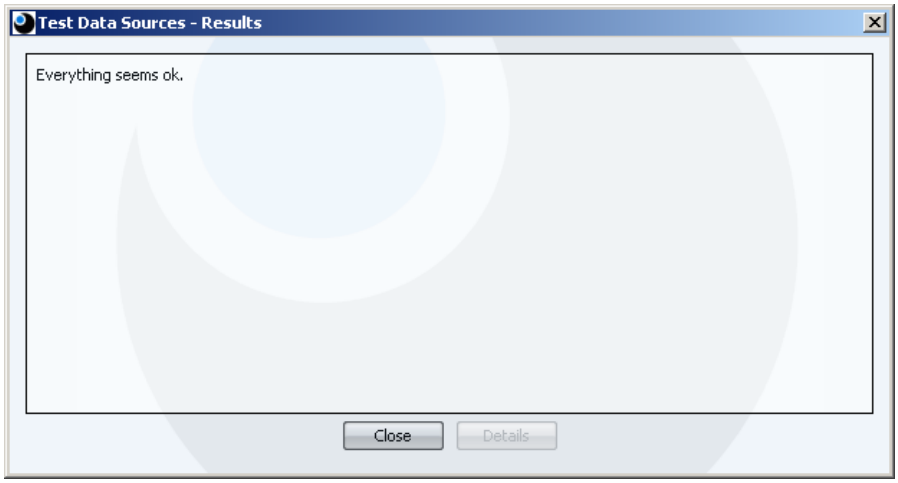


Figure 66 – Test Data Sources dialogue

If the test run terminated with no error an “Everything seems ok.” message appears (see above). Otherwise, a NetLogo-like error message is shown:

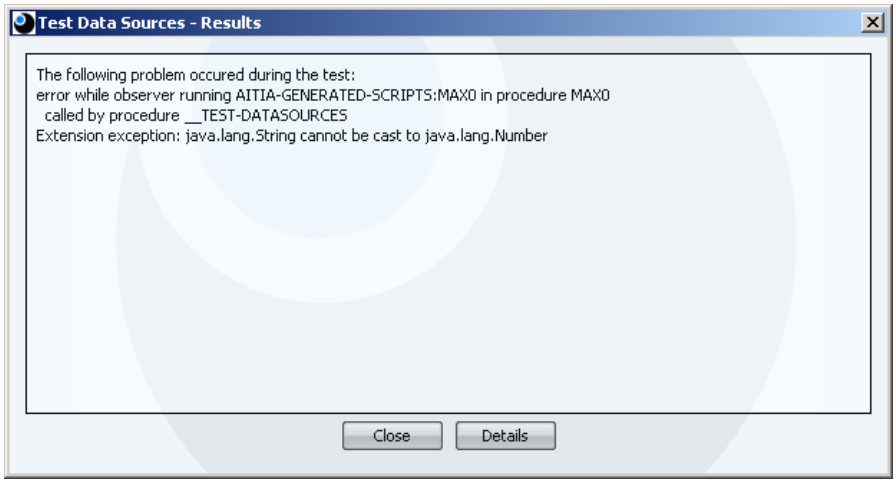


Figure 67 – Test Data Sources dialogue, error

You can gain a more detailed error message by pressing the *Details* button which shows the Java stacktrace on the screen:

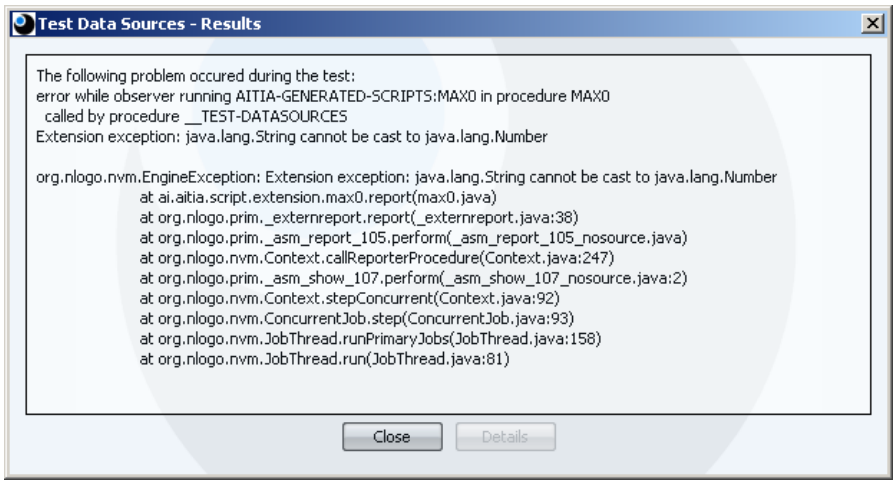


Figure 68 – Test Data Sources dialogue, detailed error

The output file of a NetLogo 4.1.1 batch experiment is a special CSV file (it extends with timestamps of the start and end time).

3.9.3.4 Creating NetLogo 4.1.1 Data Sources

Creating NetLogo 4.1.1 data sources is very similar than creating RepastJ (i.e. Java) data sources. In this section we will only discuss the differences.

Operator Parameters

- ***Element selection, First element, Last element***: Returns an element of a list or agentset.

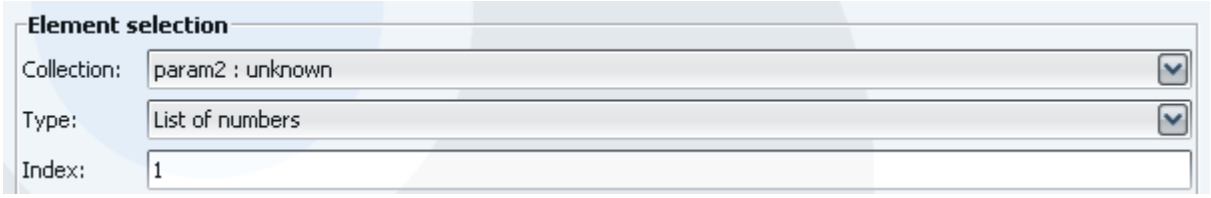


Figure 69 – Element selection

The parameters are the source list/agentset, the type of the list/agentset and the index that specifies the position of the element you want to select. For example, the parameters on Figure 69 mean the second element of the `param2` which stores numbers.

As you can see the wizard does not know the type of the selected collection so you have to specify it by selecting the proper type from the uneditable drop-down list. If you cannot find the type you search you cannot use this operator with the selected collection. The elements of the type box are the following:

- List of numbers;
- List of booleans;
- List of strings;
- List/Agentset of turtles;
- List/Agentset of patches;
- List/Agentset of links;
- List/Agentset of <*breed*>s;
- List/Agentset of <*directed-link-breed*>s;
- and List/Agentset of <*undirected-link-breed*>s

The words agentset, turtles, patches, links, breed, directed-link-breed and undirected-link-breed are come from NetLogo 4.1.1. See details about them in [11].

According to the type, the result of the operator may be a recordable element (for example an element of a list that contains numbers). The *First element* and *Last element* operators fall into this category, with no need, however, to define an index parameter.

- ***Filter***: Returns a list/agentset containing only those items of the source list/agentset for which filter condition is true.

Figure 70 – Filter

The parameters are the source list/agentset, the type of source and a filter (logical) condition. As you can see on Figure 70 in the filter condition you can use

the `%value%` expression to reference to the current element of the list/agentset. For example, the operator on Figure 70 results in a list containing only non-zero elements of the original list.

- ***Sort, Tail, Maximum places, Minimum places***: The *Sort* operator sorts the specified list/agentset into ascending order, according to the natural ordering of its elements. The *Tail* operator returns the selected list/agentset without its first elements. The *Maximum places/Minimum places* operator provides the list of indices of the maximum/minimum values in the specified list/agentset. The parameters are the same in all four cases of operators: the source list/agentset and its type.
- ***Union, Intersection***: Returns the union/intersection list/agentset of the parameters. The user interface of the parameter selection is shown on Figure 71 (there is a union definition on it, but the interface of the intersection is the same). The parameter list has to be specified in the list on the right (*Selected elements*). In order to do so, select elements from the left list (*Available elements*) then press *Add* button. Remove elements from the right list by selecting them and then pressing the *Remove* button. Note that lists/agentsets with unknown element type may appear on the left side (e.g. *agentlist : unknown*). You can define the type by clicking the right mouse button on the name and select the appropriate type from the appearing drop-down list (see Figure 72).

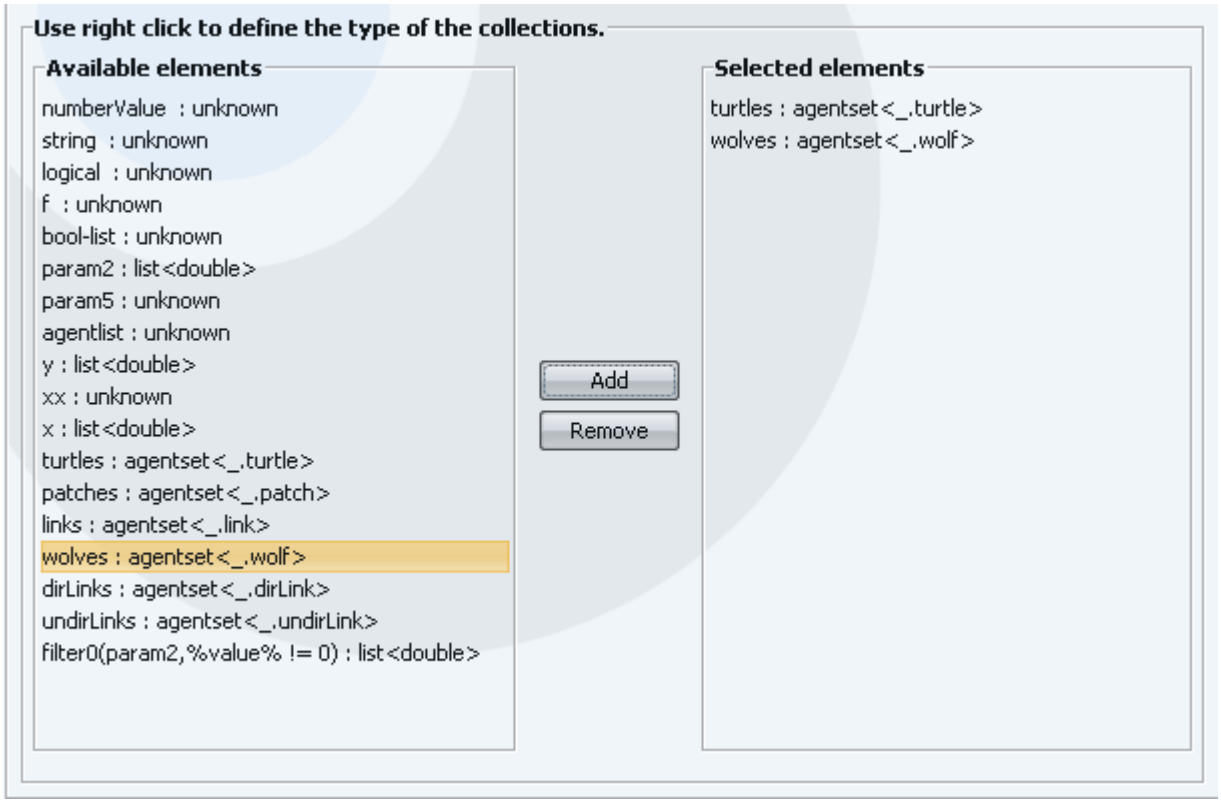


Figure 71 – Union

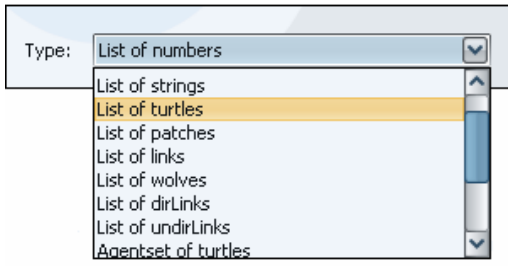


Figure 72 – Type definition at NetLogo 4.1.1

- ***Remove element, Remove element (all instances)***: Returns a list/agentset that contains the same elements than the selected list/agentset except the first instance/all instances of the given element. To do this, you must select a list/agentset, define its type and select the removable element from a drop-down list. If the selected list/agentlist contains strings or numbers you can also define the removable element by editing the drop-down list.
- ***For each***: Executes the selected reporter with each element of the specified list and creates a list from the results.

Figure 73 – For each

The source list can be selected from the first drop-down list. You must define the type of the source list. The function can also be selected from a drop-down list which contains the appropriate methods of the model and some built-in methods from NetLogo 4.1.1.

- ***Count element***: Returns the number of elements equals to the specified one in the given list/agentset. To do this, you must select a list/agentset, define its type and select the element you want to count from a drop-down list. If the selected list/agentset contains strings or numbers you can also define the last parameter by editing the drop-down list.

The ***Member selection***, ***Map selection*** and ***List*** operators are not available at NetLogo 4.1.1. The others can be used the same way than previously written in 3.2.6.1 Operators and Statistics

Statistic Parameters

At NetLogo 4.1.1 platform when you have to specify a number collection parameter you can't use the built-in *Member selection* and *List* operator since these operators are not supported. However, the *Available elements* tree may contains not appropriate elements (not numbers or number collections) because of lack of type information. It is the user's responsibility to select only appropriate elements as actual parameters of the statistic.

Scripting

Note: Scripting is for advanced users. It requires some programming skills in NetLogo.

The *Scripts* tab of the *Create data source* dialogue is shown on Figure 74.

As you can see the return type of the script can be selected from an uneditable drop-down list. There are four options there: number, boolean, String, other.

Note: If the return type is other the created data source can't be used as a recordable element, only in other data sources.

You have to define the body of the script in NetLogo instead of Java. Therefore there is no need using import declarations (related buttons are disabled).

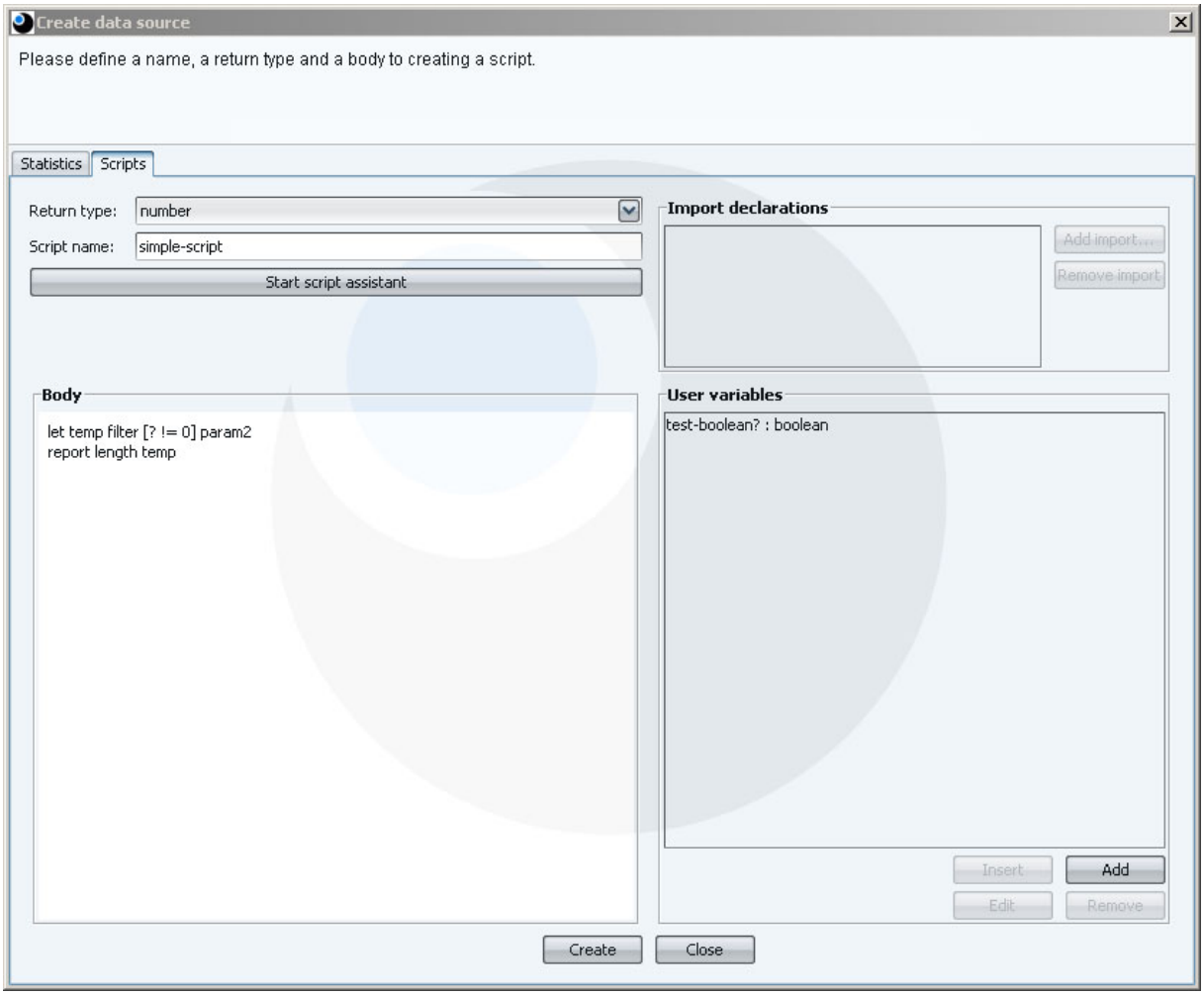


Figure 74 – Create data source dialogue, Scripts tab

4 Storing and Organizing Simulation Results

In the following chapters, "simulation result" means a bunch of raw data that was produced by a single simulation (a single run of a model).

4.1 Concept: Model Name and Version

As illustrated in 4.7 *The Results Browser* section, MEME organizes simulation results in a 3-level hierarchy. Every simulation result belongs to a particular version of a particular model. Here the "version" and "model" are names[5] entered by the user. These are the first two levels of the hierarchy. The third level holds batches. Usually a batch of runs is performed at once, generating a batch of simulation results; therefore every simulation result belongs to a batch, too. Batches are identified by numbers that are incremented automatically by MEME.

4.2 Concept: Input and Output Parameters

Simulation results are made up of values recorded during the simulation or at the end of the simulation (a single run of a model). The values are usually associated with names, hereby called as *parameters*. MEME distinguishes two kinds of parameters: *input* parameters and *output* parameters. Input parameters are used to denote values that describe/represent the circumstances in which the model is started, therefore the input values never change during a simulation. Output parameters are the factors that may change during the simulation and are the observed for recording into the output. There are three important things to know about parameters:

- Simulation results belonging to the same version of a model always have the same set of parameters. Adding results that have different parameters to the database is not recommended (although it is possible: in this case the new parameters – that are missing from the already stored results – will be inserted into all stored simulation results of that version and model, with `null` values; and if some of the existing parameters are missing from the new result, `null` values will be added to the new result, too).

  The general rule is that different parameter sets should go to different versions of the same model, or to different models (it's up to you).
- Parameter names must be different within one category, which means that there cannot be two input parameters named *X*, but *X* can be an input parameter and an output parameter at the same time. Parameter names are case-sensitive, thus *pH* and *Ph* are considered to be different. Parameter names may contain spaces and special characters, but line brakes are not acceptable.
- MEME supports numeric, textual (string) and logical (true, false) values only.

4.3 File / Database Settings

MEME stores simulation results in database(s) managed by an SQL database server it connects to at startup. By default MEME uses a built-in SQL database server which stores the data in a directory on the local file system. However, this database engine is not as powerful as many professional DMBS software, and the size of database with the built-in engine is limited to 8GB's[6]. Therefore, MEME supports various professional database

[5] Model names must be unique and max. 64 characters long. Similarly, version names must be unique within the model, and max. 64 characters long.

[6] If the size of the database exceeds 8GB's an error message is created and the application is stopped.

engines through the JDBC protocol. External database server support is tested with MySQL database engine. Other engines may work well, too, but there is no guarantee.

The *Database settings* menu item in the *File* menu allows you to select which database is to be used:

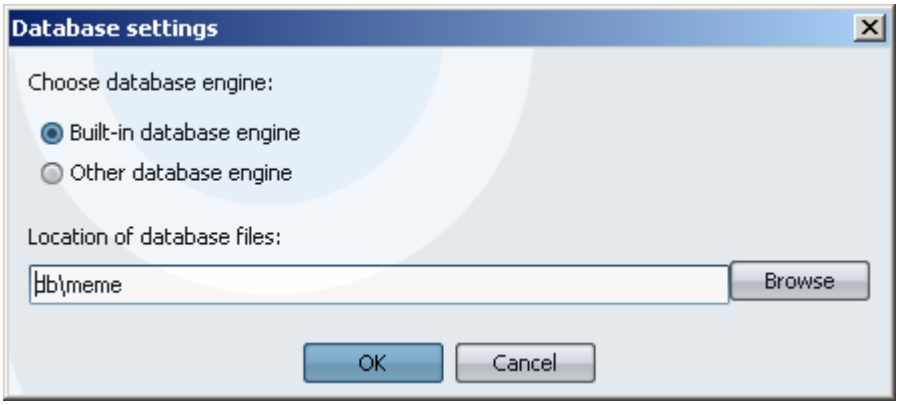


Figure 75 – Settings of the built-in database engine

If you choose the built-in database engine, you have to specify a path to a directory and a file name prefix. If the path is not absolute, it is interpreted relative to the installation directory[7] of MEME. In the above example, `db\meme` means that the database will be stored in the `db` subdirectory, in the `meme.backup`, `meme.data`, `meme.properties` and `meme.script` files.

The alternative is to use an external database server. In this case you have to specify a username, password and a JDBC connection string as shown in the following figure:

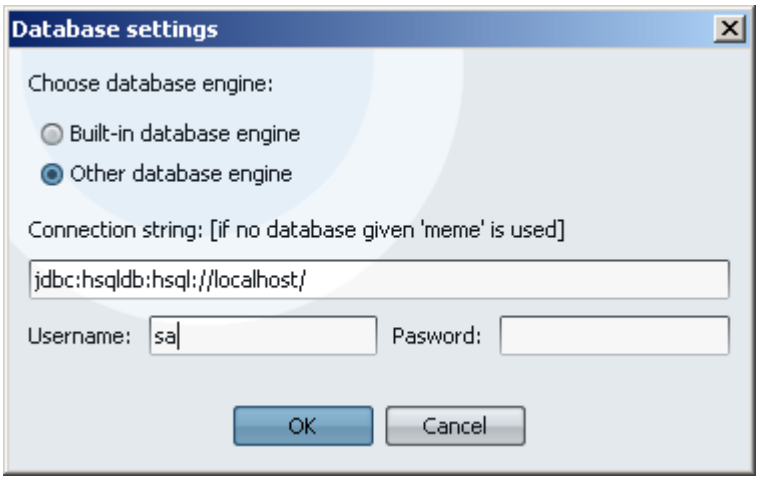


Figure 76 – Settings of an external database engine

Use this window to switch between different databases. The most recently entered value is remembered and used automatically when MEME is started.

Please note that MEME does not support transmitting data between different databases. In other words, if you have put some data in one database, you cannot move it into another using MEME.

4.3.1 MySQL Support

Besides hsqldb MEME application also supports MySQL database engine (See [8] for details). This latter can offer faster operations at times and in case of larger and more complex models the size of storable data sets is limited only by the free storage area available.

[7] By default on Windows this is the c:\Program Files\MASS\MEME\ directory.

For using MySQL the followings are needed: a database engine installed and configured, appropriate user, a database schema created and configured and an appropriate MySQL Connector/J driver.

4.3.1.1 Installation

- Version

  The version tested is MySQL 5.0 Community Server, Windows ZIP/Setup.EXE (x86) 5.0.51 and 5.1.41 (it is likely that other versions may work without problem, but no guarantee whatsoever).

- Launching installer

  With the installer bundle downloaded and launched on the first relevant dialogue screen choose the *Typical* option.

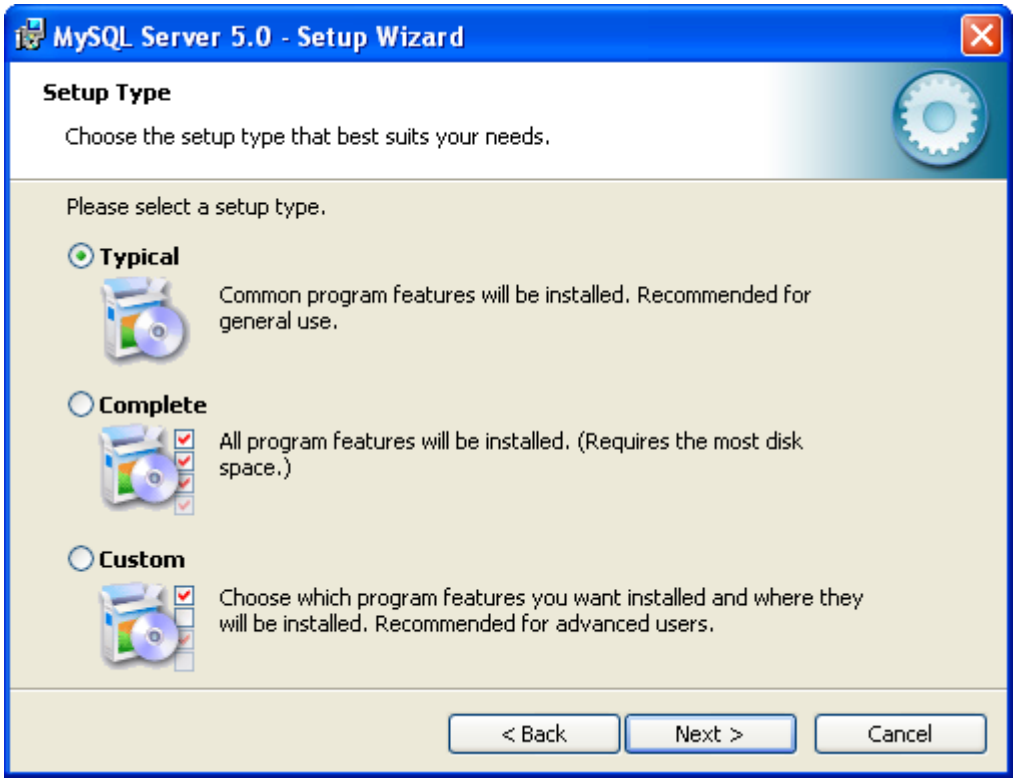


Figure 77 – Setup Type

4.3.1.2 Configuration

- Having installed MySQL its configuration wizard helps to do essential settings. The suggested values without further explanation are shown below on Figure 78 - Figure 84:

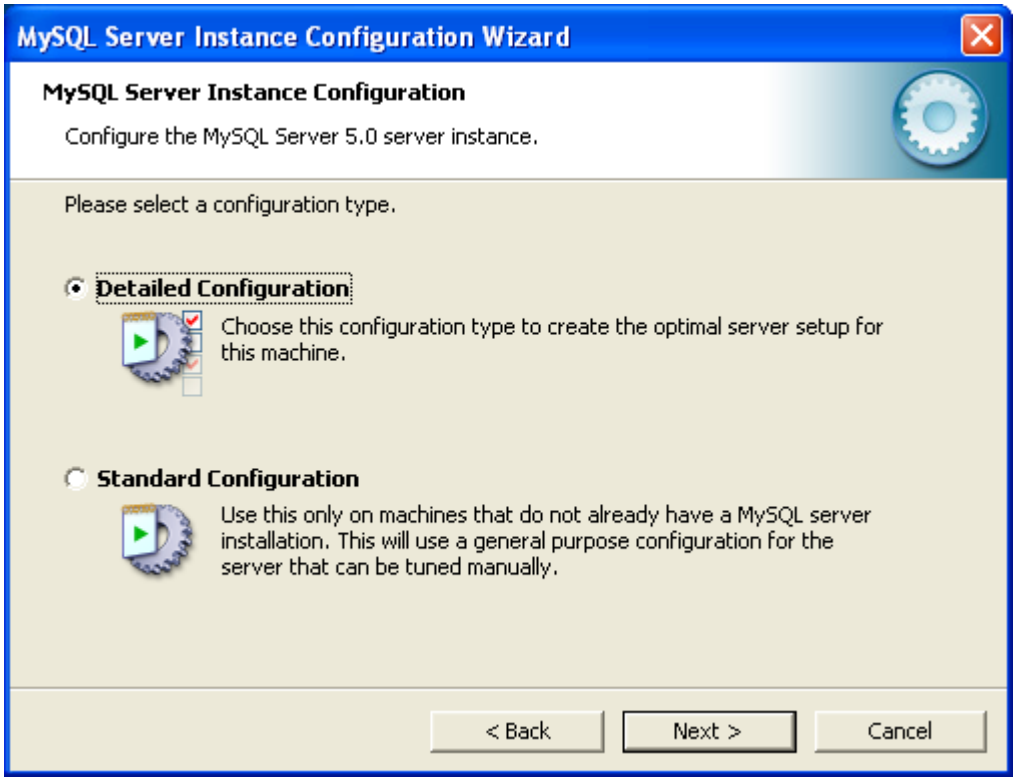


Figure 78 – Configuration type

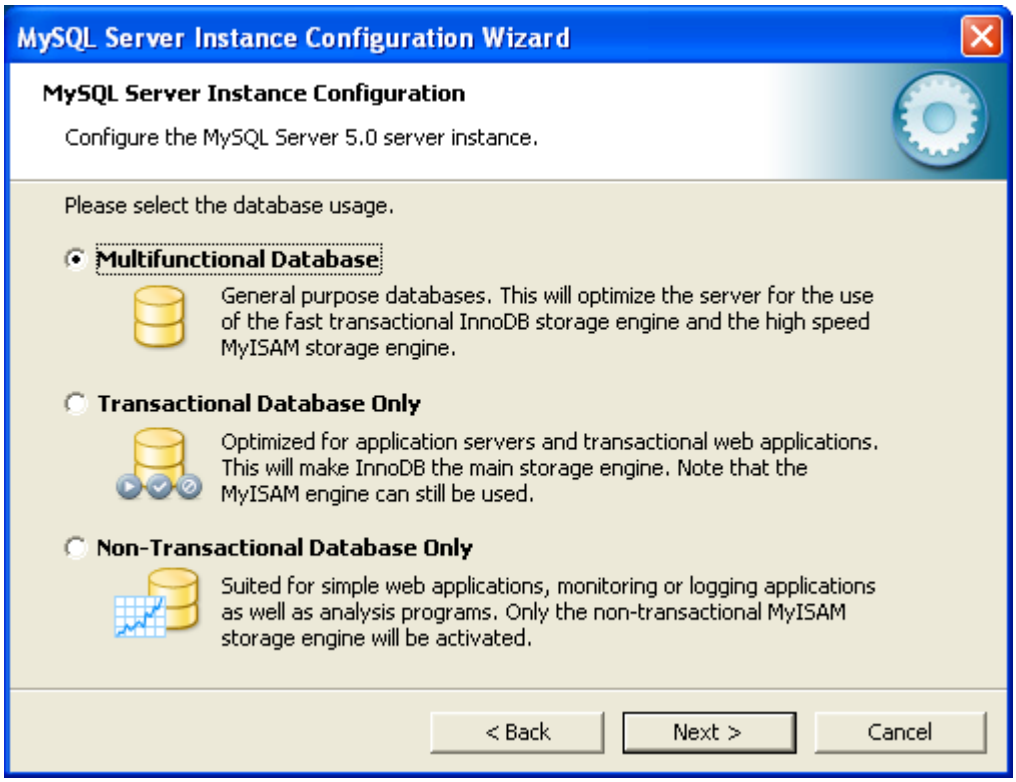


Figure 79 – Database usage

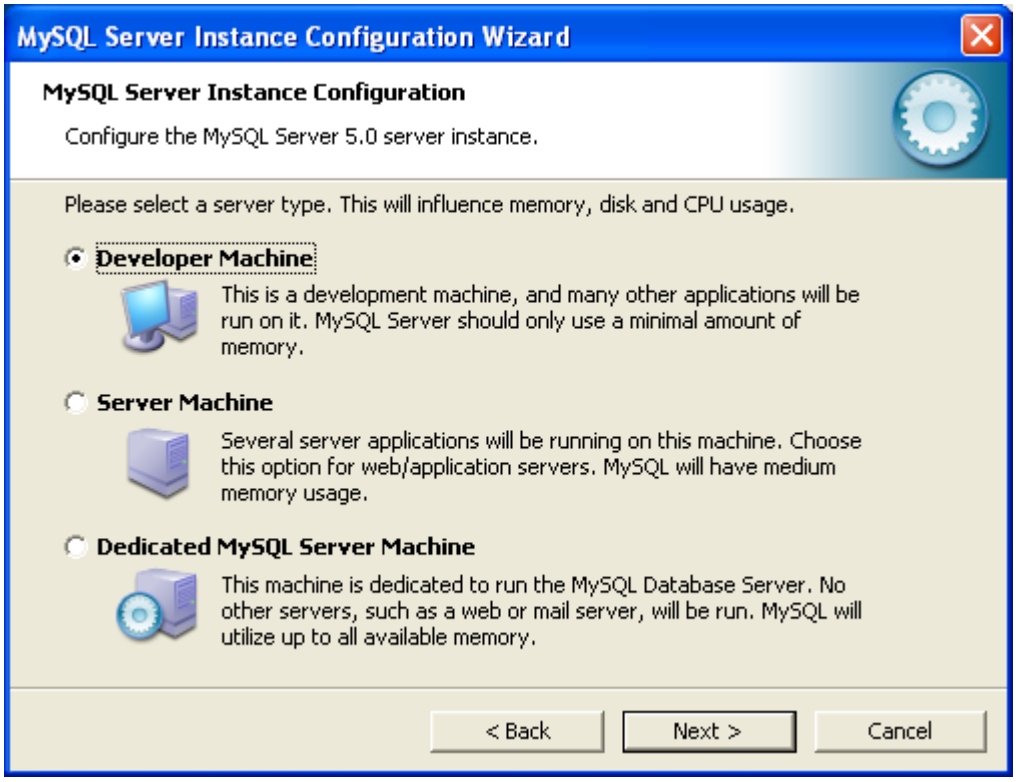


Figure 80 – Server type

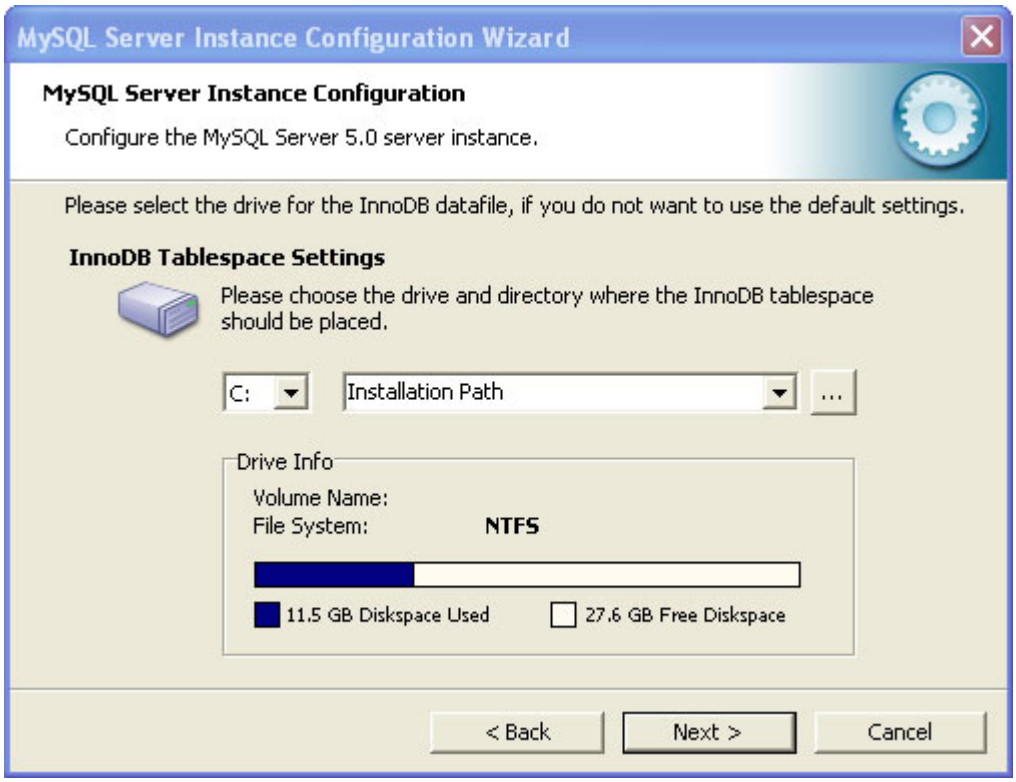


Figure 81 – Drive selection

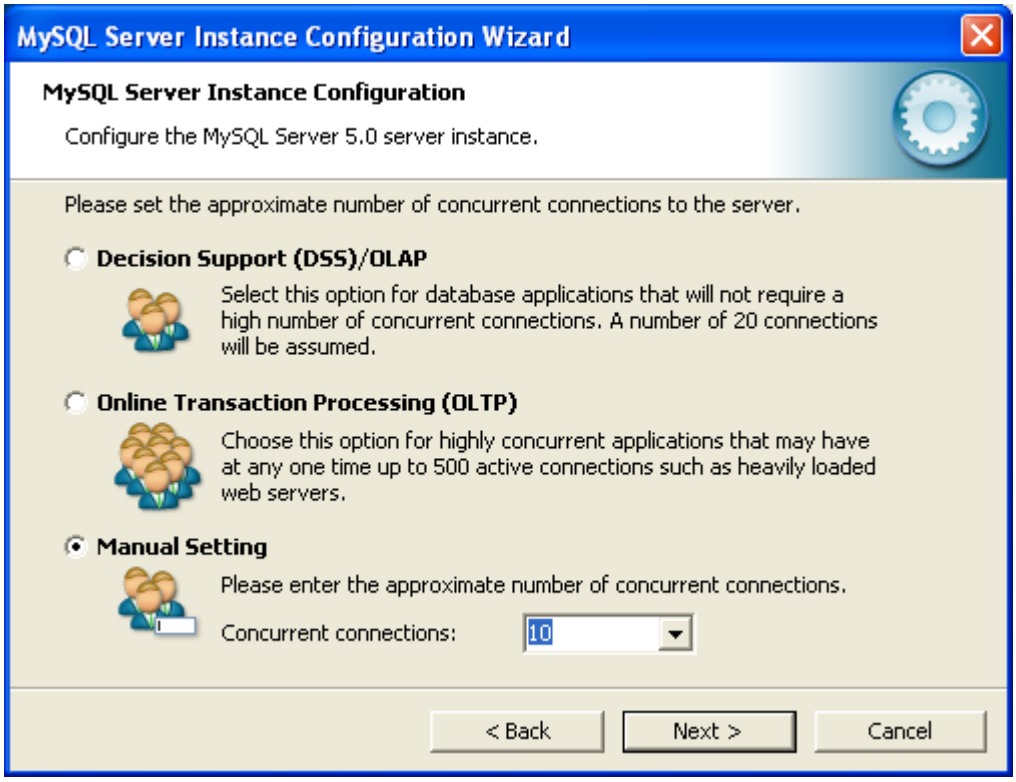


Figure 82 – Concurrent connection settings

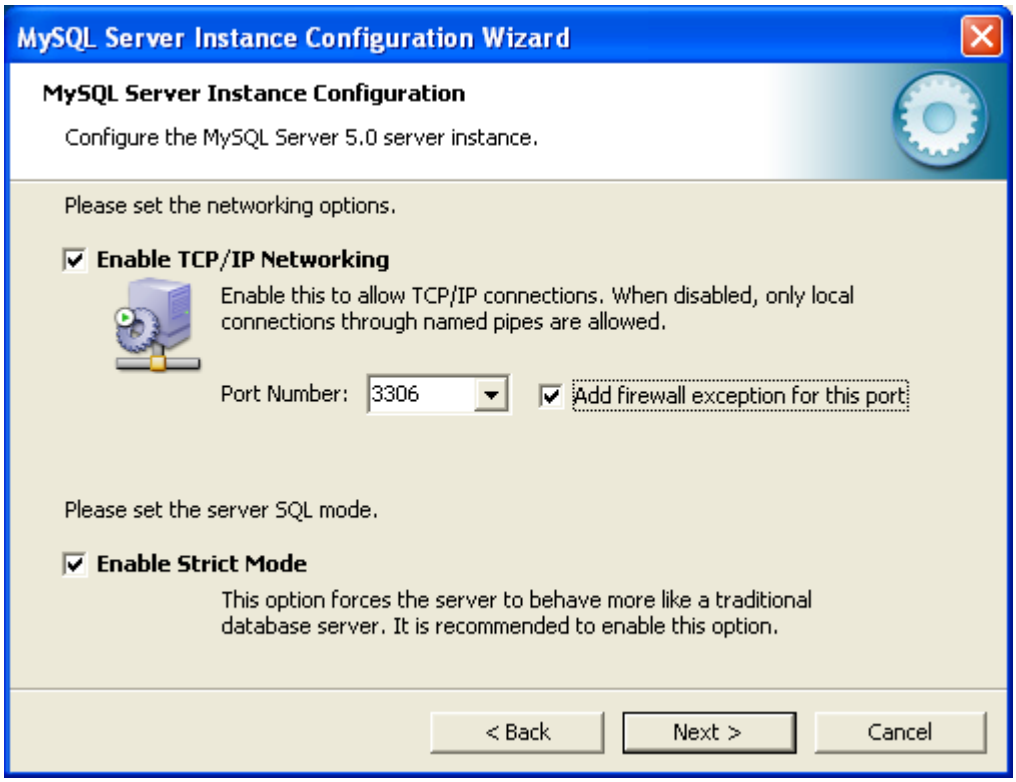


Figure 83 – Networking options

Figure 84 – Windows options

- The root user

  Administration of MySQL is done by user *root*. It is strongly advised to define a password for it.

Figure 85 – Security options

The dialogues closing installation are not shown here. On a Windows machine it might be required to allow the defined (default) port in Windows Firewall Settings.

4.3.1.3 Creation of Database Schema

As a part of database installation a graphical user interface **MySQL Administrator** is put on the system.

Note: Some versions of the installer do not contain the GUI tool. In this case you can download it independently from MySQL website: http://www.mysql.com/downloads/workbench/

For creating a schema log in to it, select *Catalogs* menu, then on *Schemata* panel click the right mouse button and select *Create New Schema.* In the dialogue displayed type in the name of your schema.

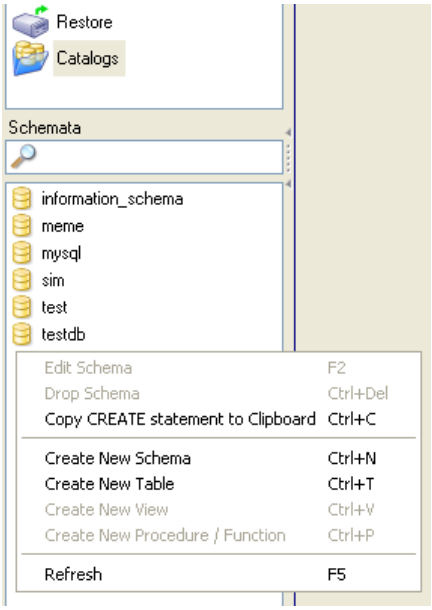


Figure 86 – Create New Schema

4.3.1.4 Creating a User

To be able to work with the schema just created we need to define a user. It is not recommended to use root as normal user!

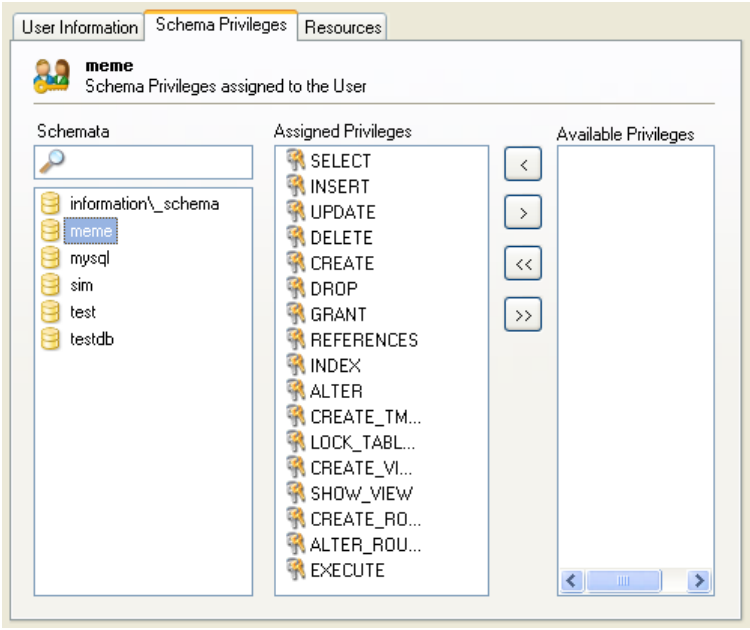


Figure 87 – Schema Privileges assigned to the User

Steps for user creation are similar to that of schema creation: choose *User Administration* menu, on the *User Accounts* panel click the right mouse button and select *Add new user*. On *User information* panel, type in necessary data. Then select *Schema Privileges* tab and on the panel displayed select the schema just created („meme" in our example). Add every elements of the list of *Available Privileges* to *Assigned Privileges*.

4.3.1.5 MySQL Connector/J

For using MEME with MySQL database engine a driver is needed to connect the two applications.

Please do the following steps:

1. Download the `mysql-connector-java-5.1.6.zip` file using the link below:

   http://dev.mysql.com/downloads/connector/j/5.1.html
2. Extract it.
3. Copy the `mysql-connector-java-5.1.6-bin.jar` to the `lib` subdirectory of the installation directory[8] of MEME.

Note: If you want to use another version of the MySQL Connector/J JAR-file rename it to abovementioned name otherwise MEME is not going to find it.

4.3.1.6 Usage MySQL Engine from MEME

- Selection of MySQL database engine

  In MEME select *File/Database settings* menu item and in the appearing dialogue select *Other database engine* option.

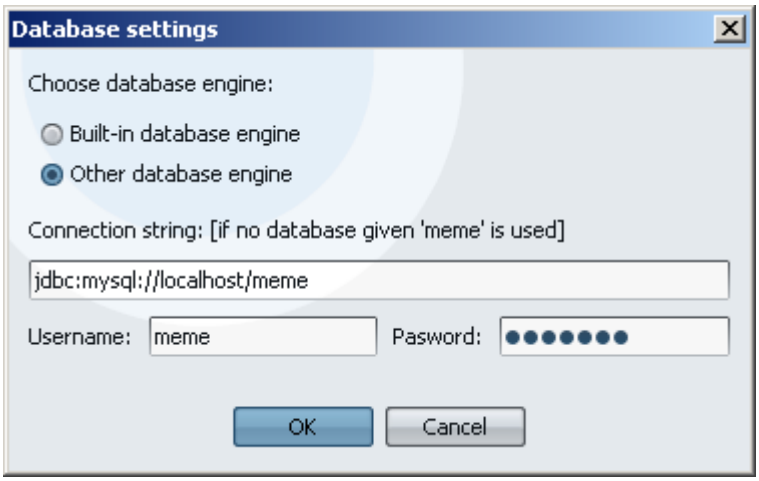


Figure 88 – MySQL database settings

- Data for database connection

  In order to connect to the schema the following connection string is required:

  `jdbc:mysql//localhost[/<schema>]`

  `schema` is the one just created. This (the one in square brackets) is optional. If not given, a default („meme") is used. The schema specified is accessed by using the username and password created and set above.

After successful installation, configuration and selection of MySQL in MEME the application is used and works as usual.

[8] By default on Windows this is the c:\Program Files\MASS\MEME\ directory.

4.4 Heap Size

In the upper right corner of the main MEME window the actual heap size / memory reserved field is displayed. MEME is configured to reserve a maximum of 512 MB's for database processing. If this turns out to be insufficient it can be changed (see 11.2 Memory Usage for reference). The program runs the garbage collector from time to time reducing the size of memory reserved by throwing out data no longer in use. To run the garbage collector manually, click on the heap panel.

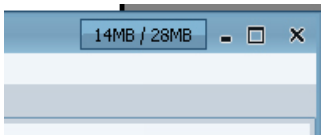


Figure 89 - Heap panel

4.5 File Import

MEME can import simulation results into the database. The *Import…* menu item in the *File* menu lists the supported formats. The following sections describe the import settings for these formats.

4.5.1 Import RepastJ Results

RepastJ allows you to observe the values of certain fields of your model during the simulation and saves the values into a log file, in a format similar to the following:

```
Timestamp: 2006.12.13. 10:19:56
Both: 1.0
Looser: -3.0
Neither: 0.0
Payoff1: 0
Payoff2: 0
StopAt: 10000.0
Strat1: 2.0
Strat2: 2.0
Winner: 4.0


"run","tick","RngSeed","payoff1","payoff2"
1,10000.0,1166001596609,10001.0,10001.0
2,10000.0,1166001596921,10001.0,10001.0
3,10000.0,1166001596953,10001.0,10001.0
4,10000.0,1166001596984,10001.0,10001.0
5,10000.0,1166001597000,10001.0,10001.0

End Time: 2006.12.13. 10:19:57
```

The *File / Import… / RepastJ result file* menu item can be used to import data from such files into the current database. It asks for the file in an open file dialogue, and loads it. The program automatically detects the delimiter string and the list of parameters and the value types. It also tries to detect which parameters are input and which are output. It displays its findings in a table where the user can adjust such settings (see Figure 91).

In order to import multiple RepastJ result files hold down the CTRL key and click on the desired files in the import file dialogue. Enter a model name and a version description (can be both numeric and textual) in order to have each file imported to the database as a new batch, or select the *Every file a new version* option. Selecting the latter versions are auto-numbered (i.e. *%* display). When importing multiple files the user doesn't have the option to declare whether a parameter is input or output (see Figure 90).

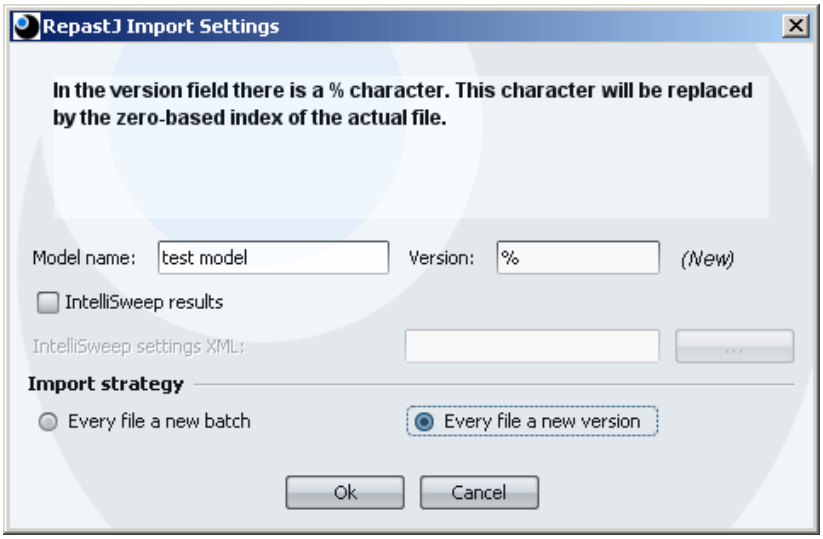


Figure 90 – Importing multiple RepastJ result file

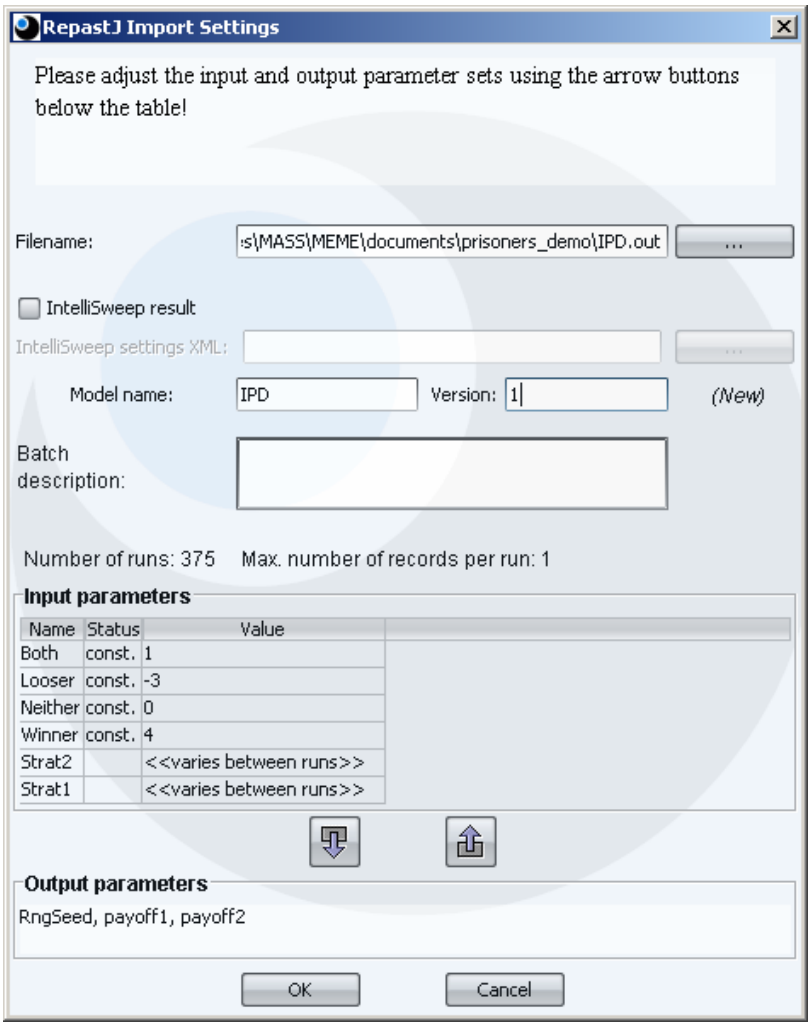


Figure 91 – Adjusting input and output parameters

Use the down/up arrow buttons located between the *Output parameters* and *Input parameters* fields to adjust which parameter belongs to which group. There must be at least one output parameter and the parameters that are listed in the header of the file cannot be changed to output parameters.

This dialogue also asks for the name of the model and version this result belongs to. As these names are restricted in length (max. 64 characters), there is also a description

field in which you can provide mode information. The given model and/or version will be created if it doesn't exist yet. Otherwise the data will be added to the existing model version as a new batch.

If the named model and version already exist, this is displayed in the label next to the *Version* field. If the parameters are different from the existing parameters the new or missing parameters are indicated in the table and in the *Output parameters* list (see also 4.2 *Concept: Input and Output Parameters* section) regarding this situation. If only the types of the values differ from the types of the existing parameters, it is not indicated, but all existing data will be converted to the wider type. (For example if a parameter named *x* already existed in the model with numeric values, and now *x* comes with textual values, all previously stored *x* values will be converted to text.)

The *OK* button can only be pressed if the name and version fields are filled in properly.

4.5.1.1 IntelliSweep Result Processing

The IntelliSweep result checkbox is there to indicate that the imported RepastJ result file is the result of an IntelliSweep simulation run. It is useful, because it creates a separate data table with the processed results of the simulation, and makes one or more charts that demonstrate or visualize information derived from the IntelliSweep runs. It also brings up the *Intellisweep result processing options* dialog if necessary, to enable the user to change some parameters of the processing of Intellisweep results.

If the checkbox is checked, you must specify the IntelliSweep settings XML file (which was automatically saved at the beginning of the simulation) in the text field. You can use the … button to browse the file system.

If you are at the *RepastJ Import Settings* dialogue after you ran the simulation on your computer, or you import result to the database from remotely run simulations, the *IntelliSweep result* checkbox and the XML file textbox are set automatically. You only have to take care of these settings, if you choose to manually import RepastJ results.

The *Intellisweep result processing options* dialog can be seen in Figure 92. It shows the processing settings for a Factorial design experiment. Other methods may or may not have this screen.

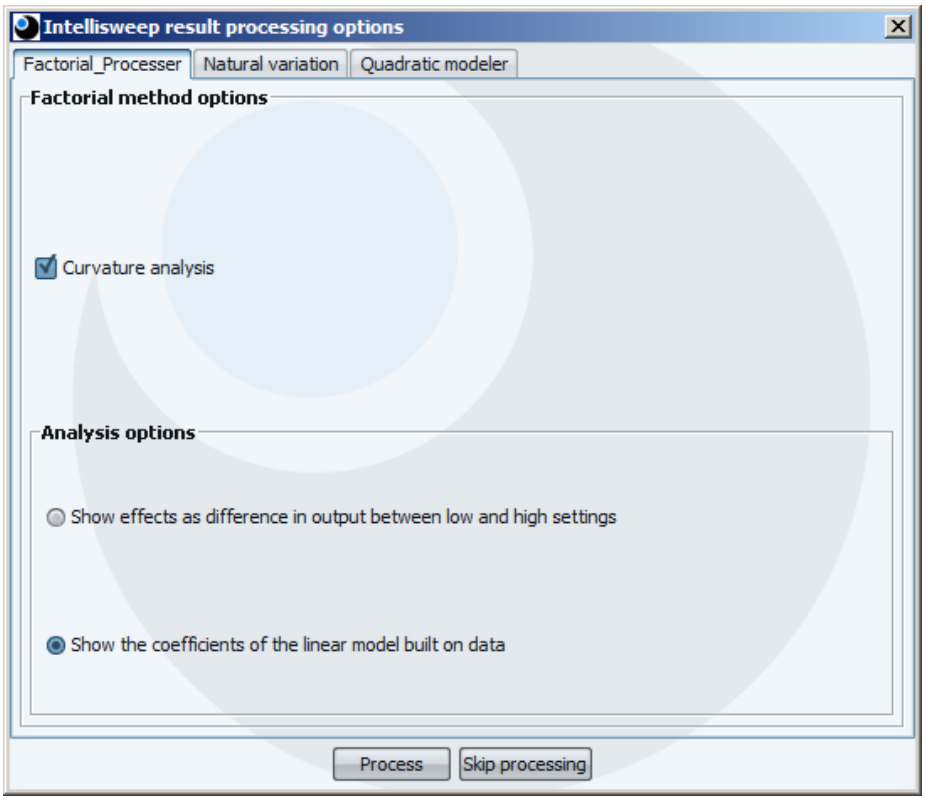


Figure 92 – Intellisweep result processing options dialog

Natural Variation Processing

If you set up random seeds with the *Random Seed Manipulator*, there will be a *Natural variation* tab on the *Intellisweep result processing options* dialog. It can be seen in Figure 93. You can select for which variables you want to perform a variation analysis.

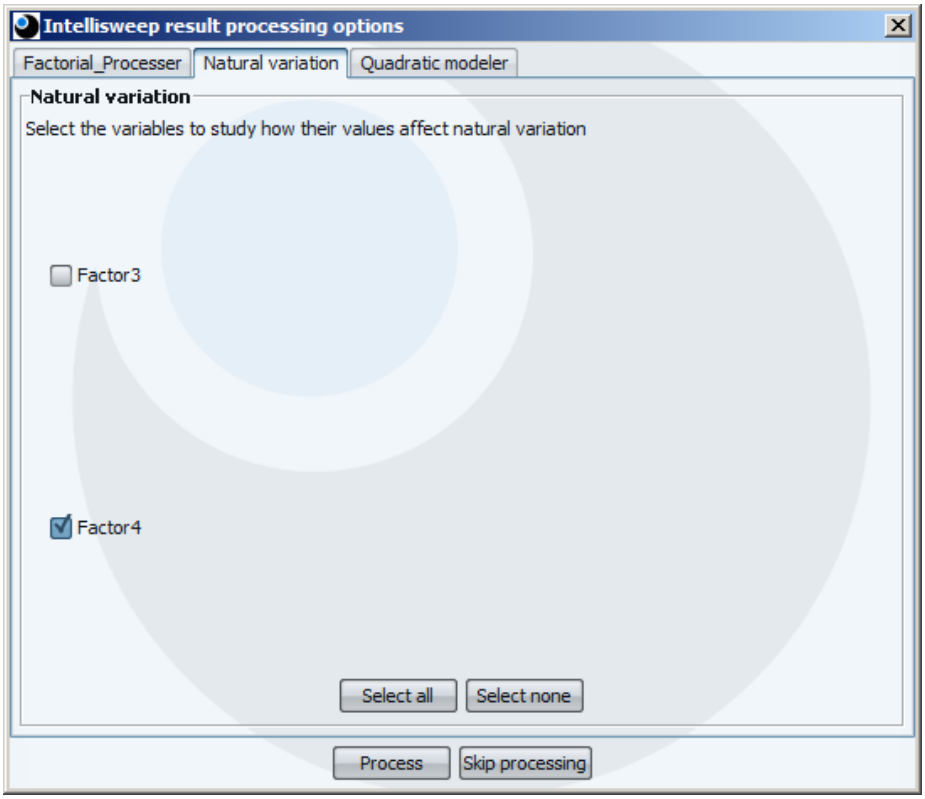


Figure 93 – Intellisweep result processing, natural variation options

Quadratic Model Builder

After importing IntelliSweep results, the user can choose to build a quadratic model of the parameters based on the result data. Response surface methods, like Box-Behnken, Central Composite or Three-level Factorial are capable of measuring enough data to produce sensible quadratic models, but you may build a model from data acquired by a different method.

You can see the setup screen of the Quadratic Model Builder that pops up after result import in Figure 94. You can choose which parameters are to be included in the model.

After result import and processing, MEME will create a version of the result you imported with the version name ending 'Quadratic_Model_XXXXXX', where 'XXXXXX' is the timestamp. Figure 95 shows an example.

The result shows the terms of the quadratic model and their coefficients. In case of multiple output columns, each will have a coefficient column in the results indicating a different quadratic model for each output column.

In case of insufficient data, or bad experiment design (e.g. using a factorial design to build a quadratic model) the model builder tries to fall back by erasing terms. At first, it gets rid of the interaction terms, then tries the same with the pure quadratic terms, then erases both from the model if it still cannot build a model. In such cases the coefficients of the erased terms are blank in the result.

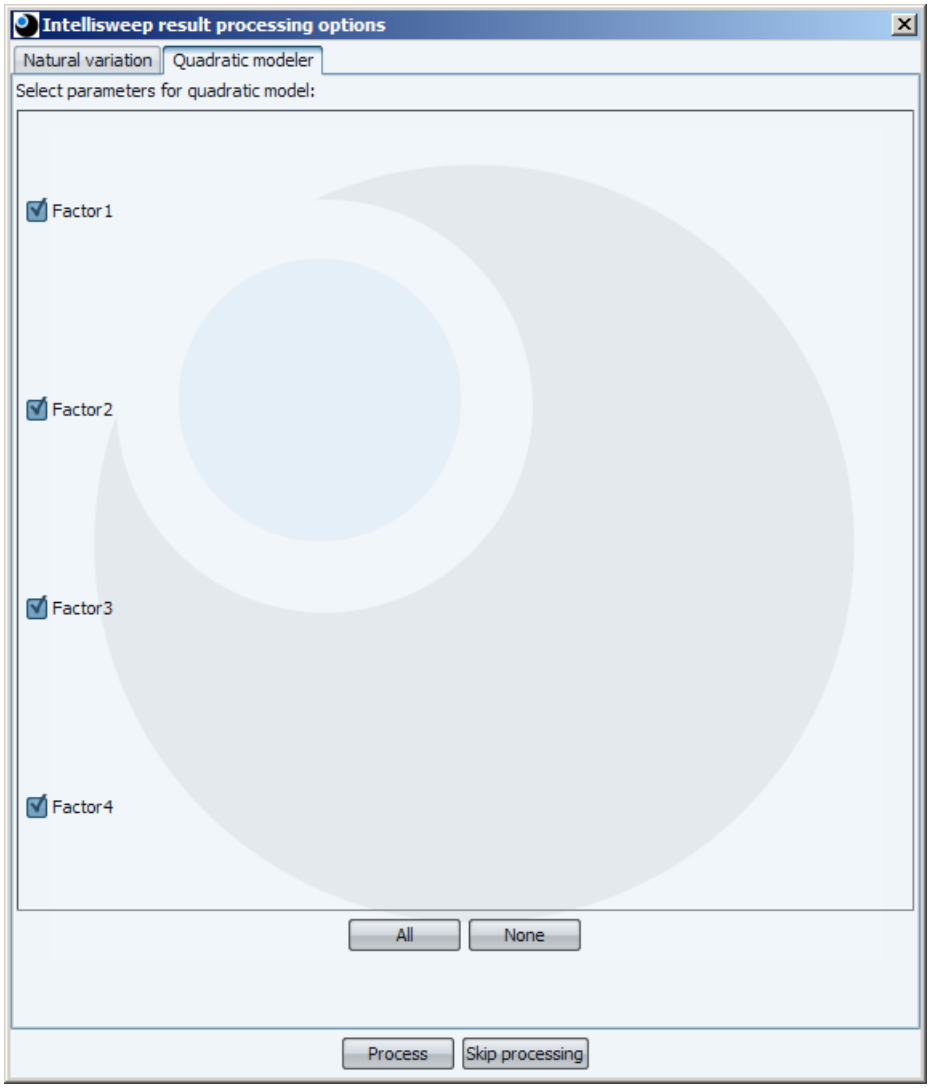


Figure 94 – Quadratic Model Builder setup dialog

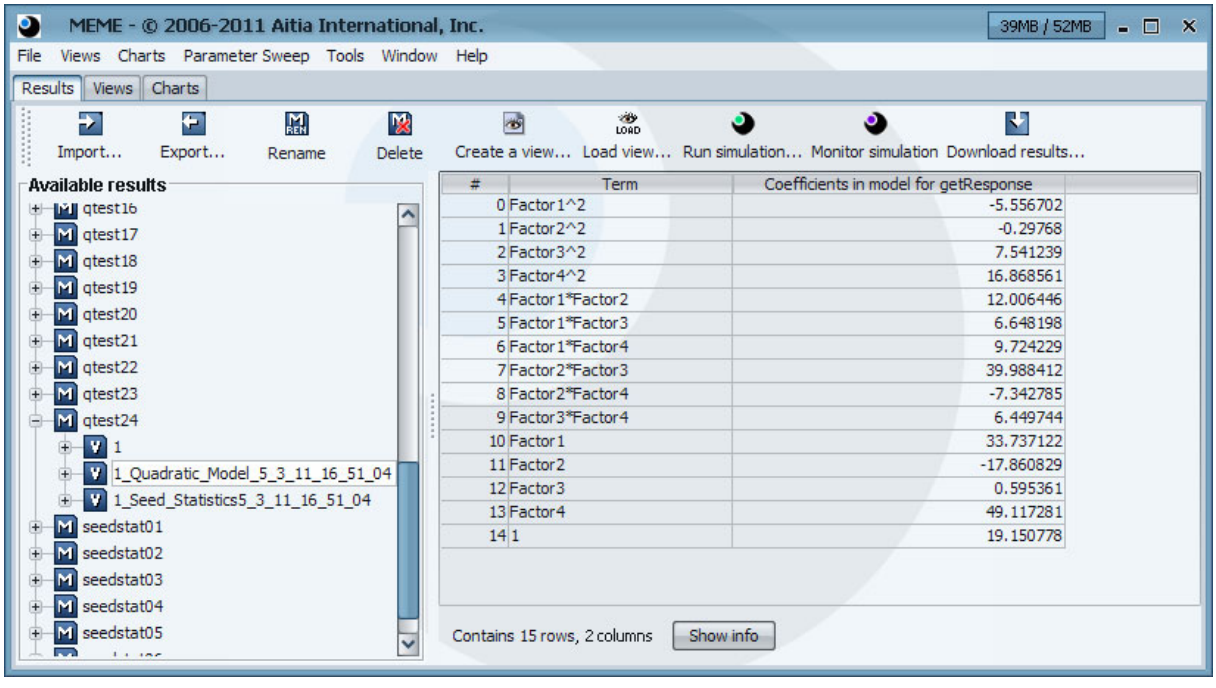


Figure 95 – Quadratic Model result

Note that depending on the number of output files of the model one of the importing dialogues above (Figure 90 and Figure 91) appears after the Parameter Sweep Wizard has finished a simulation. At local simulation running the appropriate dialogue appears automatically. At remote simulation runnings it is shown only if you press the *Import to database…* button on the *Monitor* panel. See details about the Parameter Sweep Wizard and the monitor in section 3 Creating and Running Batch Models.

4.5.2 Import Custom Java Model Results

As it is mentioned in section 3.8 Platform for Custom Java Models, the platform for custom Java models is very similar to the RepastJ platform. One of the similarities is that they use the same output file format. Therefore selecting The *File / Import… / Custom Java model result file* menu item the same dialogue appears (only the title changed) after the user selects the output file(s) in a file dialogue. See details in the previous section.

4.5.3 Import NetLogo Results

When you perform a batch experiment with the Parameter Sweep Wizard at NetLogo 4.1.1 platform it produces a special CSV file (comma separated values with timestamps of start and end time) as output:

```
"Timestamp: 04/02/2009 16:39:12"


"[run number]"|"Random-seed"|"Birth-energy"|"Energy-from-grass"|"[step]"|"f "
1|"20"|"1"|"10"|"9"|"0.8698372224360509"
2|"25"|"1"|"10"|"9"|"0.3247320815503395"
3|"30"|"1"|"10"|"9"|"0.7236025749318812"
4|"35"|"1"|"10"|"9"|"0.0652982922497436"
5|"40"|"1"|"10"|"9"|"0.708122469516378"
6|"21.25"|"1"|"10"|"9"|"0.03330578911091486"
7|"22.5"|"1"|"10"|"9"|"0.9777671103929456"
8|"23.75"|"1"|"10"|"9"|"0.9463054714402043"
9|"26.25"|"1"|"10"|"9"|"0.34352646662905717"
10|"27.5"|"1"|"10"|"9"|"0.9617232326859589"

"End time: 04/02/2009 16:40:06"
```

The *File / Import… / NetLogo result file* menu item can be used to import data from such files into the current database. It asks for the file in an open file dialogue, and loads it.

As you can see on Figure 96, the dialogue of importing NetLogo results has the same structure than the dialogue of importing RepastJ results. There are only two main differences:

- the NetLogo result file specifies unambiguously which parameters are input and which are output so the user doesn't have to option to change the kind of the parameters;
- *Include 0th ticks (initial states)* checkbox (checked by default): if your NetLogo result file contains more line of data per run then the first one (the 0^{th}) describes the initial state of the experiment (the values of the parameters AFTER the *setup* method but BEFORE the *go* method). You can ignore these lines by unchecking the checkbox.

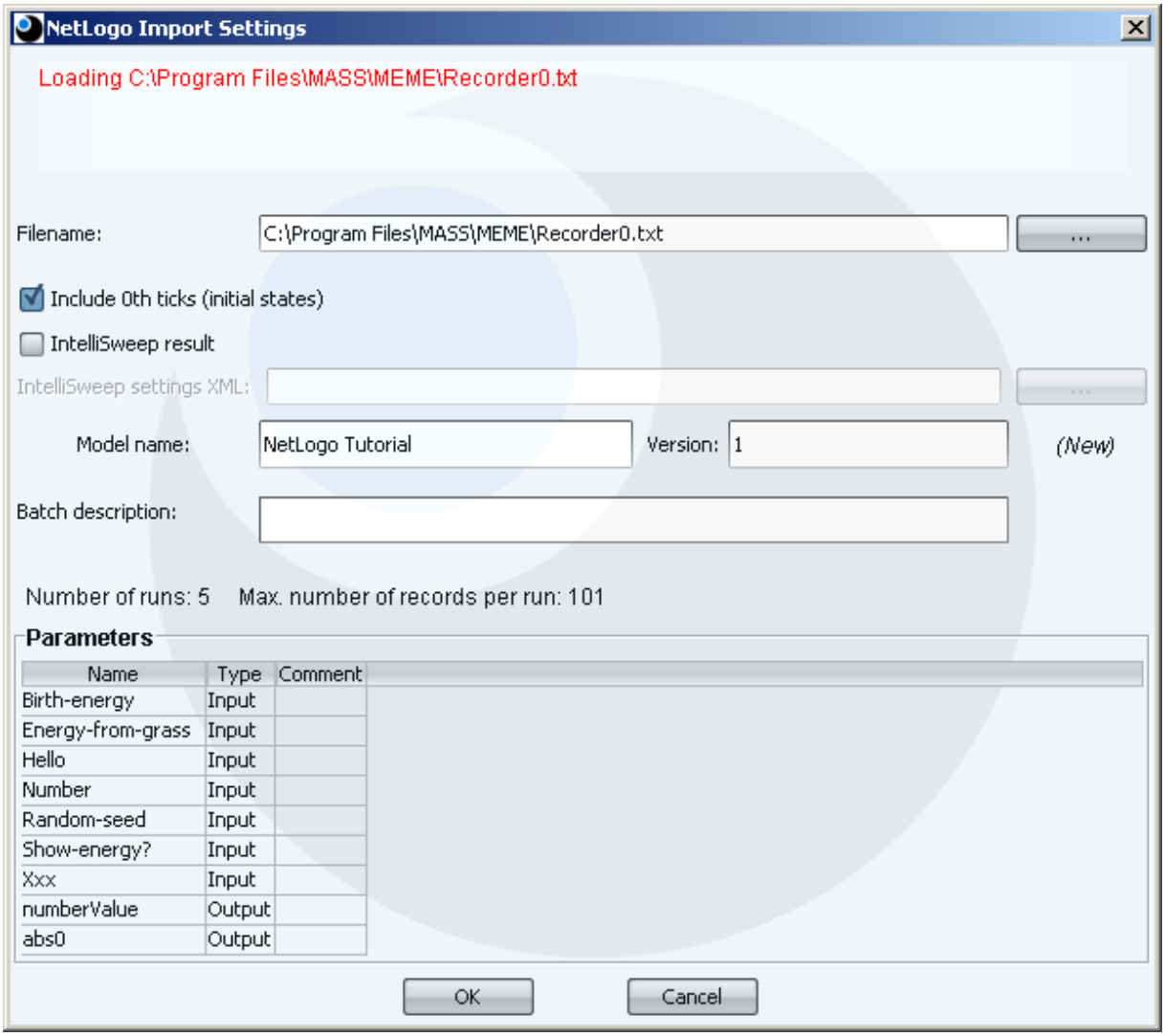


Figure 96 – Importing NetLogo result file

Importing multiple NetLogo result files is similar to doing so with RepastJ result files (the *Include 0th ticks (initial states)* checkbox can be found here too).

4.5.4 CSV Import

CSV (Comma Separated Values) files with a format similar to the following can also be imported to MEME:

```
"run","tick","RngSeed","payoff1","payoff2"
1,10000.0,1166001596609,10001.0,10001.0
2,10000.0,1166001596921,10001.0,10001.0
3,10000.0,1166001596953,10001.0,10001.0
4,10000.0,1166001596984,10001.0,10001.0
5,10000.0,1166001597000,10001.0,10001.0
```

After choosing the desired CSV file the *CSV Import Settings* dialogue appears where the following import settings can be configured: delimiters (you can choose more than one and there is an option to merge consecutive delimiters), quote string, comment character, token for empty values, reading options (w/o column names, ignoring first x lines), run options (whole file one run, number of records per run, runs by tick number). The preview field shows the first couple lines of the file with the given configuration (see Figure 97).

MEME uses the latest CSV settings as default but the various settings can also be saved for later use and loaded back again.

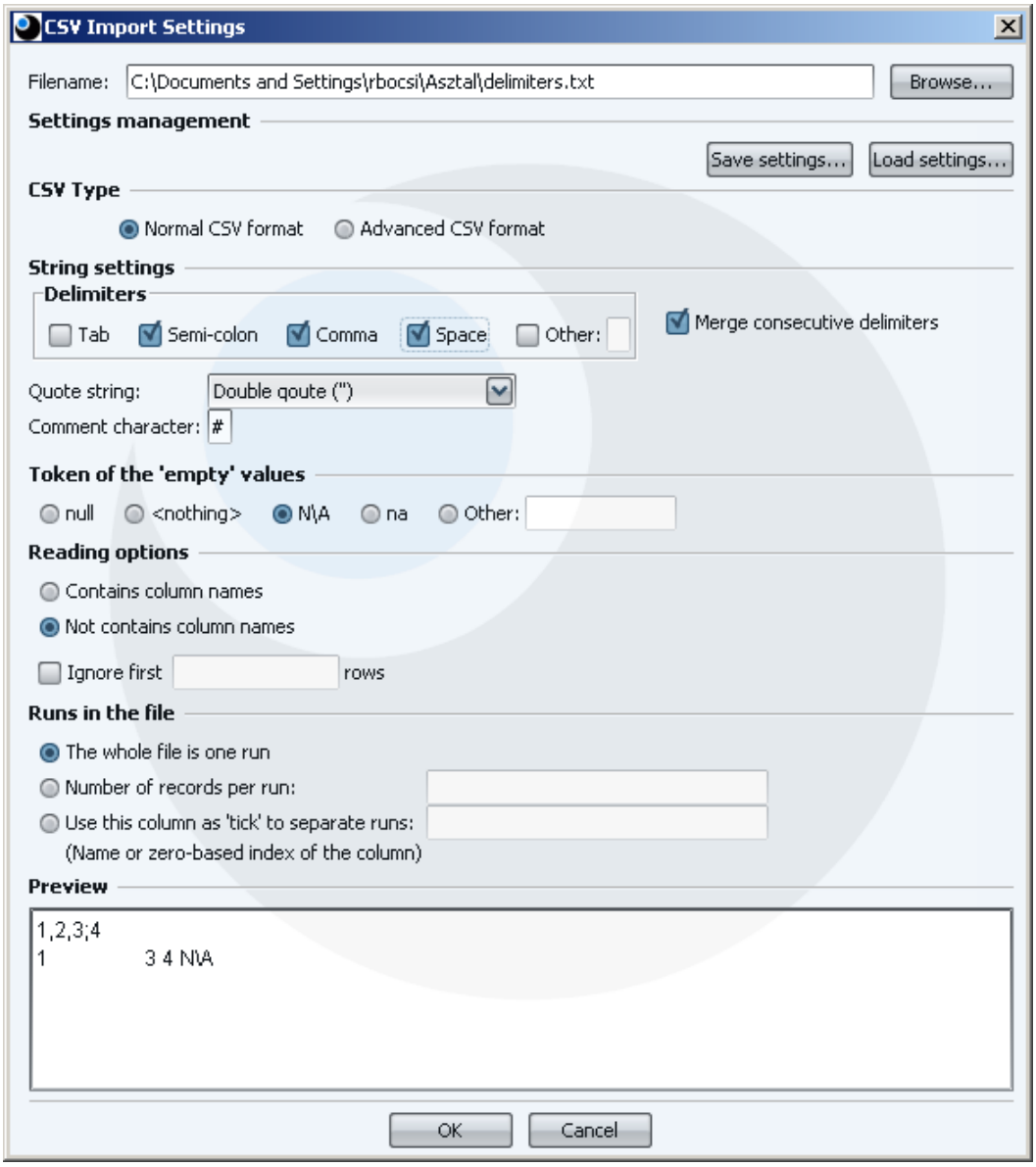


Figure 97 – Import settings of a CSV file

By pressing *OK* the CSV import dialogue switches to the model, version and batch settings already introduced in 4.5.1 Import RepastJ (see Figure 98). The only difference from the RepastJ file import is that parameters detected as *Input* can be modified to *Output* here by clicking on them in the type field and choosing *Output* from the drop-down list. Importing multiple CSV files is similar to doing so with RepastJ result files.

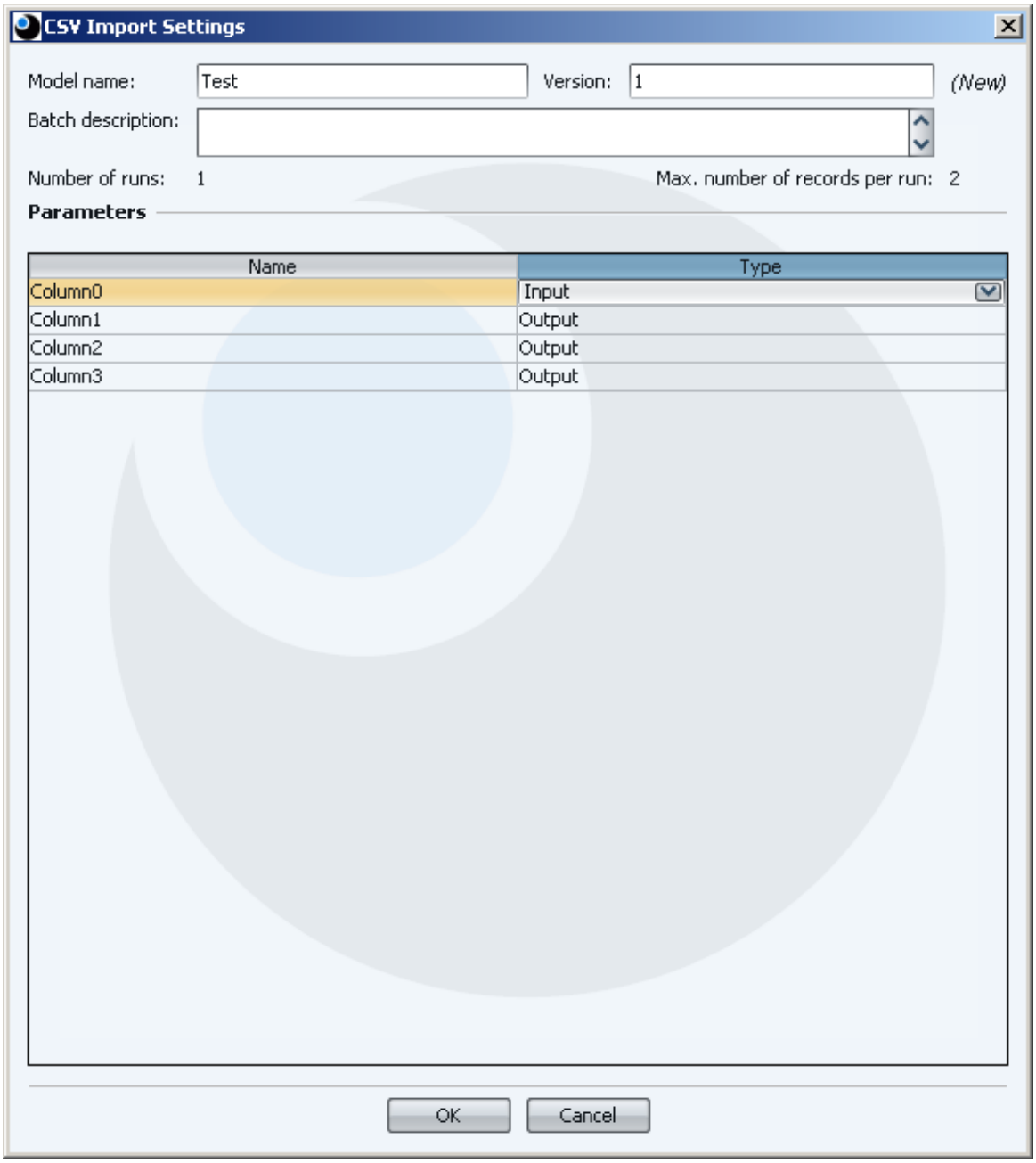


Figure 98 – Import settings of CSV data

4.5.4.1 Advanced CSV Format

MEME introduces a new – more flexible – format of CSV files that combines the benefits of regular expressions and format strings of the *printf()* method from the C programming language.

You can use this feature by clicking the *Advanced CSV format* radio button shown on Figure 97. The CSV import dialogue has changed, its new form is shown on Figure 99.

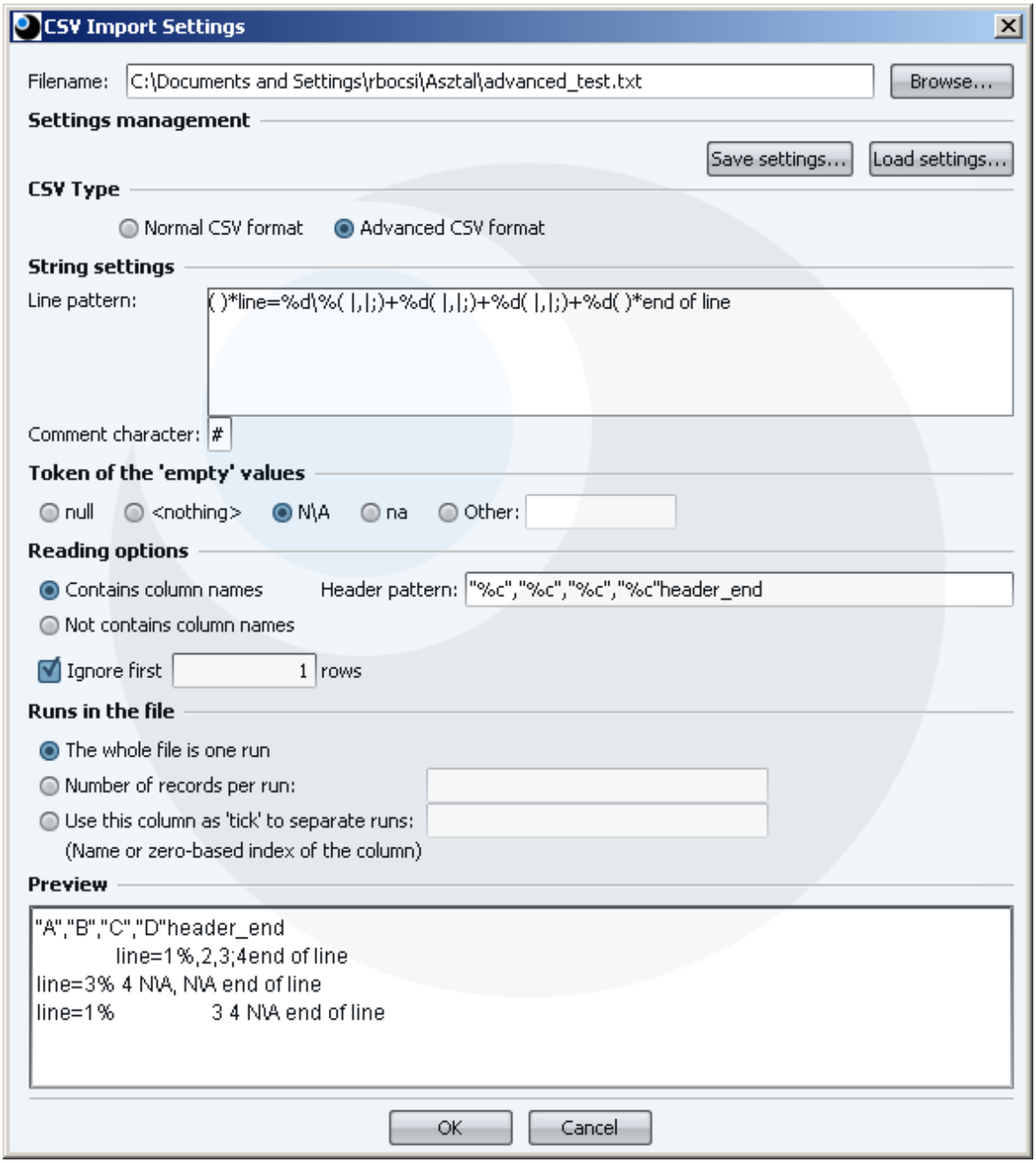


Figure 99 – CSV Advanced Import

This format uses patterns instead of delimiter and of quote string definition. This pattern is essentially a regular expression (see details about regular expressions in [9]) but it is extended with some new metacharacters (borrowed from C).

The list below introduces these:

- ***%s*** Matches any text (string). For example: “text” without the quotes.
- ***%c*** Matches any character. For example: ‘a’ without the single quotes.
- ***%d*** Matches any (decimal) integer number. For example: 1.
- ***%o*** Matches any octal integer number. For example: 0234.
- ***%x*** Matches any hexadecimal integer number. For example 0xffffff.
- ***%f*** Matches any real number. For example: 3.14.
- ***\ %*** Means the percent sign. It is necessary because % has a special meaning in the patterns.

To use the advanced CSV import feature you must define a pattern for the lines of the input file. If the file contains header information an appropriate header pattern is also necessary. The other settings on the dialogue may be familiar from the normal CSV import window.

***Example**: Let assume that we want to import the following file:*

```
ignored line
"A","B","C","D"header_end
#comment
               line=1%,2,3;4end of line
line=3% 4 N\A, N\A end of line
line=1%                  3 4 N\A end of line
```

How can we do that?

*First check the **Ignore first n rows** checkbox to ignore the first line. Then define # as the comment character and N\A as the token of 'empty' values.*

Use the following pattern as header pattern to parse the header:

"%c","%c","%c","%c"header_end

What does this mean?

*The header starts with a quote, then there is an arbitrary character, a quote again, then a comma. This is repeated three more times and the header ends with the text **header_end**.*

Use the following pattern as line pattern:

()*line=%d\%(|,|;)+%d(|,|;)+%d(|,|;)+%d()*end of line

What does this mean?

*The line starts with zero or more spaces, then there is the text **line=**, then an arbitrary decimal number, then a percent sign. The delimiter can be a space and/or a comma and/or a semi-colon. There is one delimiter at least but it can be more. This is repeated three more times from the arbitrary decimal number (except the percent sign after the decimal number). After that there are zero or more spaces and at the end there is the text **end of line**.*

4.6 File Export

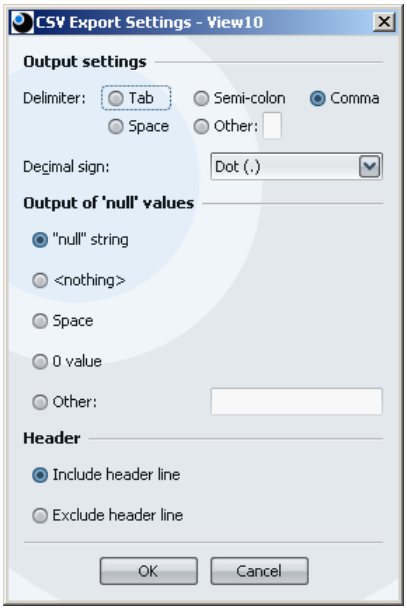


Figure 100 – CSV export settings

View tables (see 5 Views), organized data from MEME's database and versions (see 4.1 Concept: Model Name and Version) from the results database can be exported through the *File > Export…* command as CSV files with various delimiter, decimal sign, null value token and header settings (see Figure 100).

By default MEME offers *<view name>.csv* for view tables and *<model>_<version>.csv* for result data versions as file names. Multiple views and versions can also be exported with identical settings. In this case the default file naming rules are used.

4.7 The Results Browser

Once you have stored simulation results in the database you can browse them in the main window of the application (see Figure 101 below).

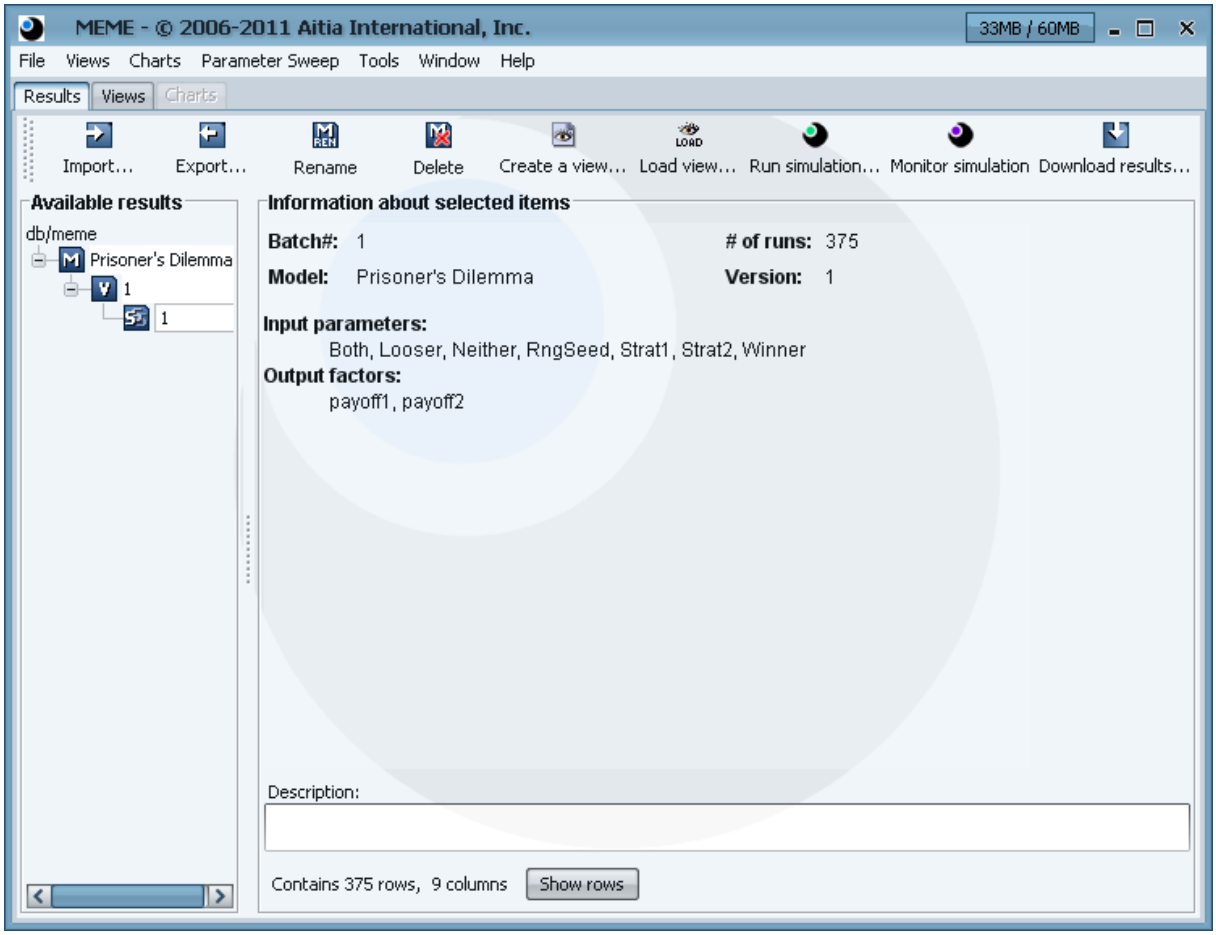


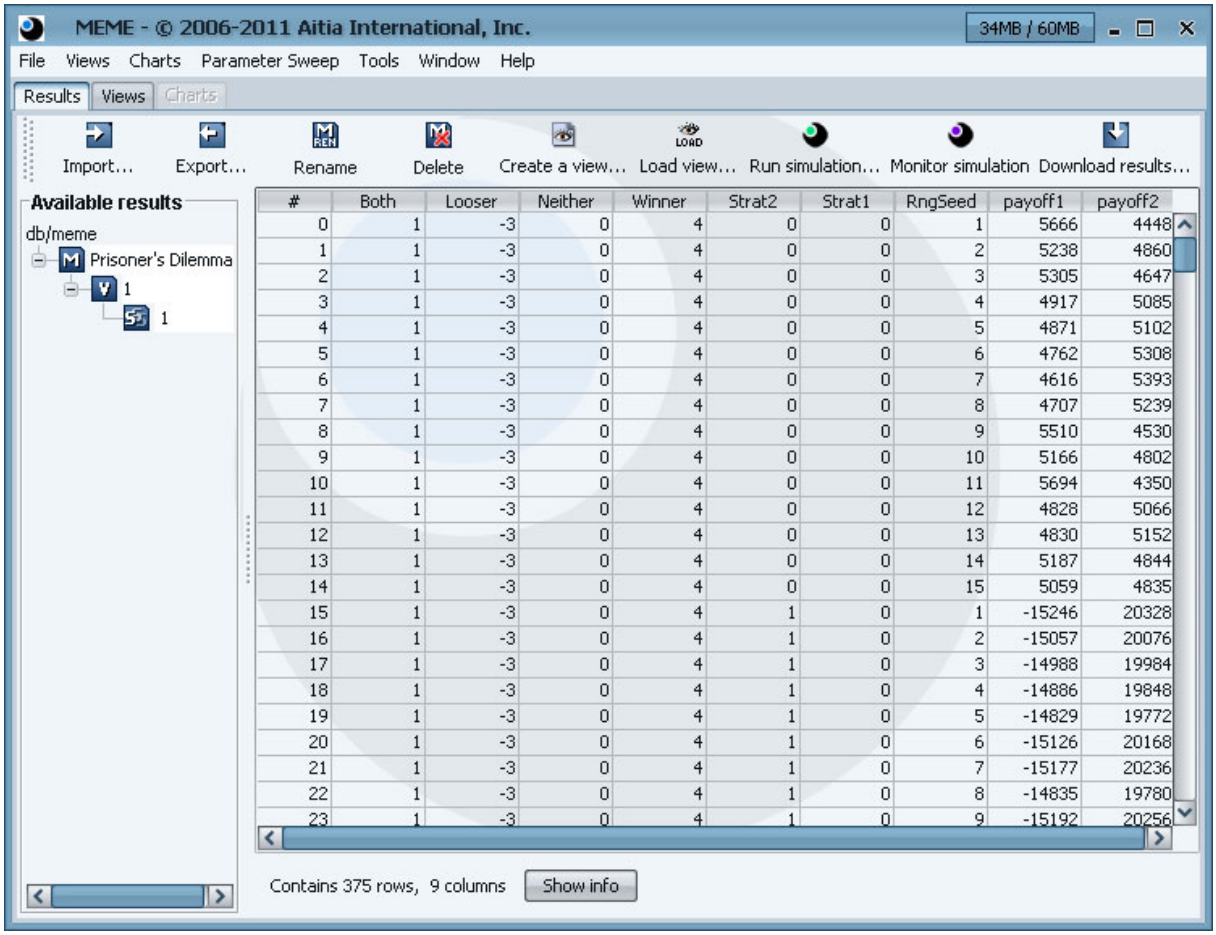
Figure 101 – Results browser

The results hierarchy is shown at the left side of the window in the tree. The root of the tree displays the current database location (*db/meme*). The model names are branched from this root. These can be expanded either by double clicking, or by single-clicking on the ⊞ node near to the name's icon. If you click on a name, it becomes highlighted and information is displayed about the selected element on the right side. Note that the *Description* field is editable.

You can select more than one branch in the tree. In general, when you select a node, all its descendants are selected too. For example selecting a model name selects all its versions and batches as well. When multiple versions are selected the information displayed is the joint list of their parameters. Selecting multiple items can be useful when you want to use data from different versions of a model, especially when creating *view tables* (see 5.1 Concept: View Table).

If selecting a version, a batch or a series of batches the *Show rows* button appears in the bottom enabling the user to observe the selected data (see Figure 102).

Note that if you select a model or version before importing a file the program assumes that the data is to be stored into that model, and therefore fills in the model name and version fields in the dialogue automatically (you can modify it though).



#	Both	Looser	Neither	Winner	Strat2	Strat1	RngSeed	payoff1	payoff2
0	1	-3	0	4	0	0	1	5666	4448
1	1	-3	0	4	0	0	2	5238	4860
2	1	-3	0	4	0	0	3	5305	4647
3	1	-3	0	4	0	0	4	4917	5085
4	1	-3	0	4	0	0	5	4871	5102
5	1	-3	0	4	0	0	6	4762	5308
6	1	-3	0	4	0	0	7	4616	5393
7	1	-3	0	4	0	0	8	4707	5239
8	1	-3	0	4	0	0	9	5510	4530
9	1	-3	0	4	0	0	10	5166	4802
10	1	-3	0	4	0	0	11	5694	4350
11	1	-3	0	4	0	0	12	4828	5066
12	1	-3	0	4	0	0	13	4830	5152
13	1	-3	0	4	0	0	14	5187	4844
14	1	-3	0	4	0	0	15	5059	4835
15	1	-3	0	4	1	0	1	-15246	20328
16	1	-3	0	4	1	0	2	-15057	20076
17	1	-3	0	4	1	0	3	-14988	19984
18	1	-3	0	4	1	0	4	-14886	19848
19	1	-3	0	4	1	0	5	-14829	19772
20	1	-3	0	4	1	0	6	-15126	20168
21	1	-3	0	4	1	0	7	-15177	20236
22	1	-3	0	4	1	0	8	-14835	19780
23	1	-3	0	4	1	0	9	-15192	20256

Figure 102 – Rows of a version

4.8 Rename Results

Use the  button to change the selected model name and/or version. A dialogue appears (see Figure 103) where you can specify the new name and/or version.

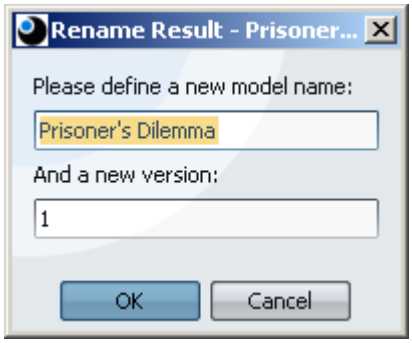


Figure 103 – Rename Result dialogue

Please note that you cannot use an existing modelname/version pair.

4.9 Delete Results

Use the  button to delete all results from the database that is contained in the currently selected batch. When a version or model node is selected, it deletes all results belonging to that version or model, including all batches.

5 Views

5.1 Concept: View Table

The “views” are computed tables that are assembled from raw simulation data. You can specify the batches of simulation results — belonging to any model or version — from which the view table is created, and the set of columns to be included in the view table. You can also do computations and filtering on the original data (see 5.2.1 Step 1 - Computation).

Charts can only be created from views. MEME also allows exporting the contents of a view table in CSV format (see 4.6 File Export).

5.2 Create View

To create a view, first select at least one batch in the *Results browser*. The new view table will be created from the data contained in the selected results.

Then select the *Views/Create a view…* menu item or press the  button in the *Result Panel*. This will start the *Create View Wizard*:

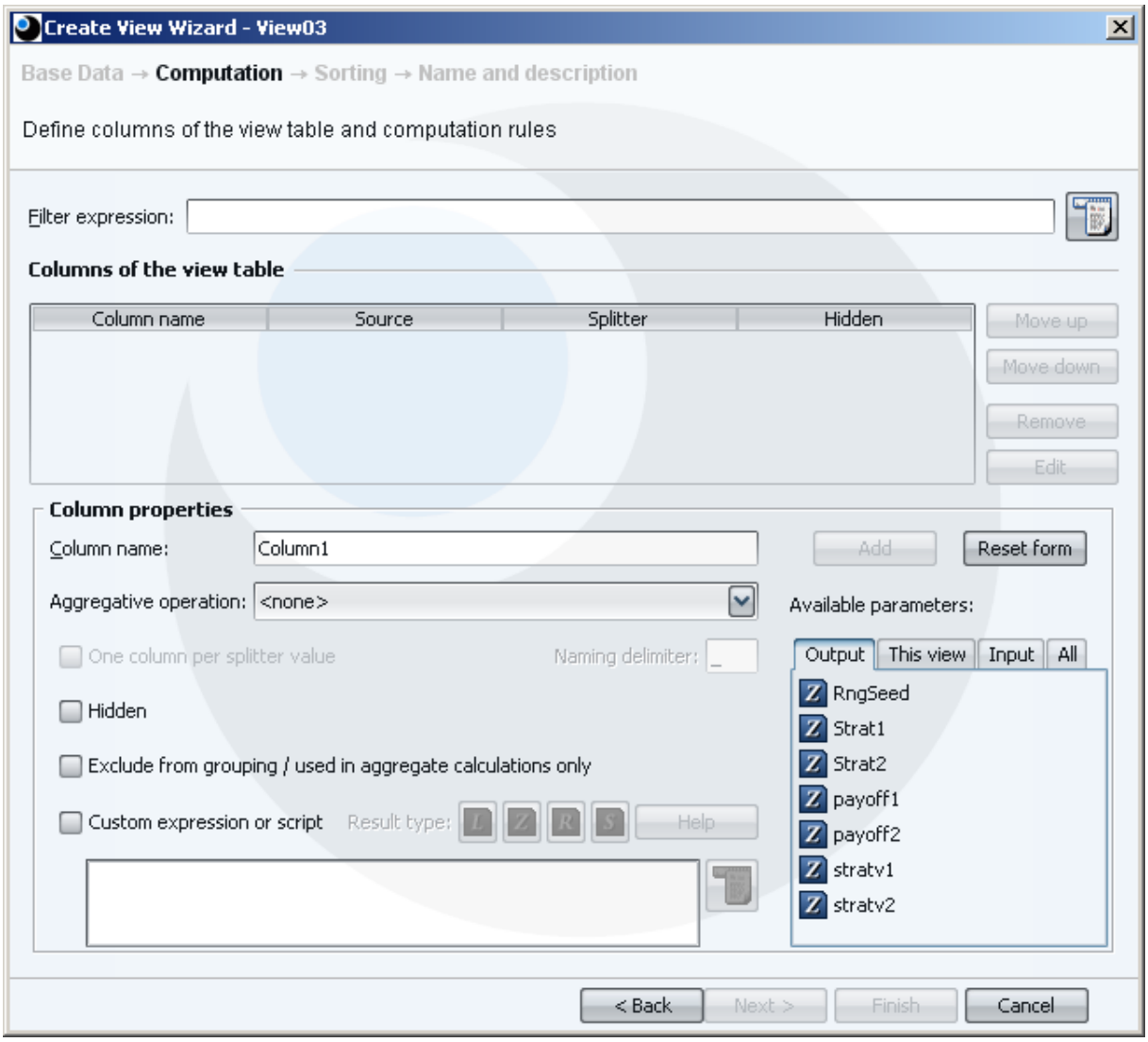


Figure 104 - Computation

Note that view tables can be created from existing view tables too by choosing the source tables on the view panel and pressing *Create a View…*.

Note that throughout the wizard the *Del* button on the keyboard has the same functionality as the *Remove* button in the program.

5.2.1 Step 1 - Computation

Add a column by selecting a parameter of a model from the *Available parameters* list. The list is broken down into *Output* and *Input* variables of the model and parameters already used in the given view. You can also see all the parameters by selecting the *All* tab.

The  ,  ,  ,  icons by the parameter and column names in the columns list indicate the data type of that column: integer, real, text or logical (boolean).

Once a column is selected you can have simple *Aggregative operation*s (minimum, maximum, average, sum, count and others, see [7] for details about the available operations) done on it. You can also set the given parameter to be excluded from grouping; hence it won't be altered by transformations done to the table. *Custom expressions or scripts* (Beanshell) can be to be run on variables to do more complex operations than the ones included in the program by default. As ticks and runs are cornerstones of simulation data, MEME has built in scripts to return them; `$Tick$` and `$Run$` (these scripts are also available from the input *Available parameters* list as Run and Tick column). If the base data is from a view table (see 5.2.5 Step 0 – Base Data) `$Tick$` returns row numbers.

If *Hidden* is selected the given column will not be listed in the view table created. If a column is a splitter it is automatically selected as hidden (this can be changed).

The columns are named by default after the parameters they are originated from, but their names can be changed by typing in the *Column name* field.

Note: Column names have to be unique (in a case-sensitive way), and cannot exceed 64 characters.

By pressing the *Add* button (or double-clicking on it in the list) the selected column is added. Once added the column can be used as a *Splitter* to group other columns of the view table based on it. Once a splitter is added you can select the variables to be grouped by selecting *One column per splitter value*.

By pressing the *Edit* button or by double-clicking on its name the selected (already added) column can be renamed, operations can be done on it, can be selected to be grouped or to be the splitter. When done editing press *Modify* to save the changes, or *Cancel* to undo them. Use the *Move up*, *Move down* and *Remove* buttons for the corresponding results.

Variables can be filtered by typing regular Java expressions in the *Filter expression* field.

Note that when a parameter is selected from the list, or a column is being edited the list tabs including the corresponding variable(s) are highlighted (in orange using the default skin and theme settings).

Multiple columns can be selected for moving, hence moving up or down all the selected columns. Note that when multiple columns are selected and the *Edit* button is pressed only the first column of the selection is being edited. The move and remove buttons are inactive while editing columns.

5.2.1.1 Editor

If you want to create a more complex expression or script and need more space than you have you can use the  button which shows a simple editor on the screen (see Figure 105).

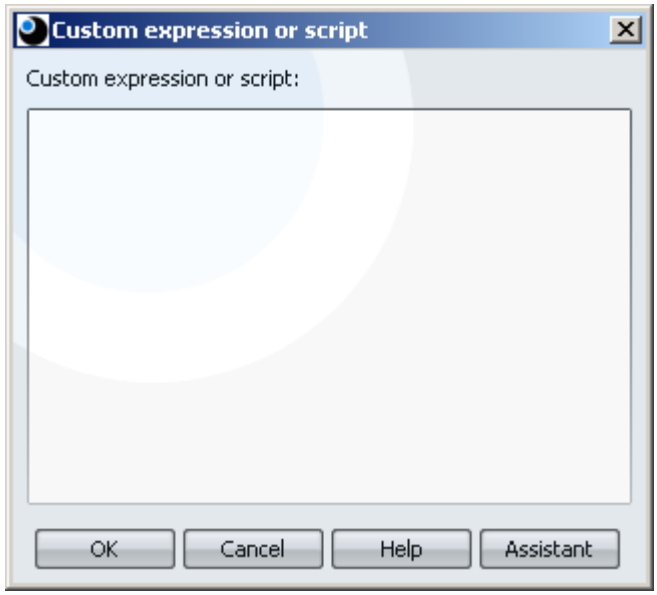


Figure 105 – Simple editor

If you need a more sophisticated tool press the *Assistant* button to start the *Script Editor Assistant* dialogue (see Figure 106).

On the top of the window there are buttons for the most used symbols (numbers, arithmetic, relational and logical operators, brackets etc.). Press any button to insert the appropriate symbol to the actual cursor position of the editing area which is in the center of the dialogue. The columns of base tables are in a list on the left part of the window. The `$Run$` and `$Tick$` can also be found here. Select one then press the *Insert* button below the list to insert the name of the selected column to the editing area.

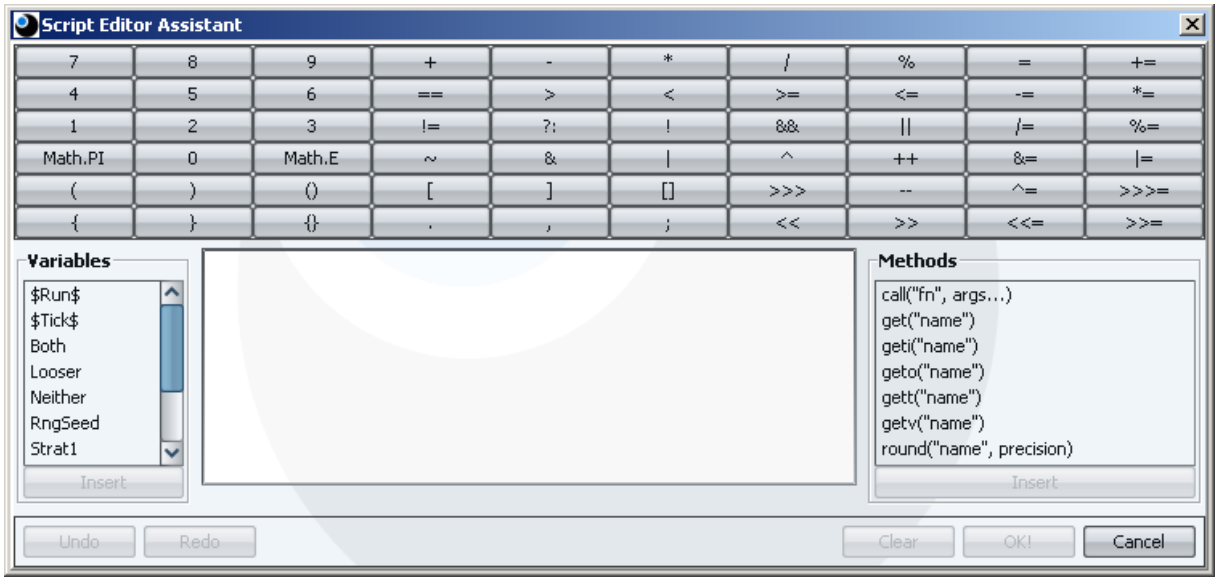


Figure 106 – Script Editor Assistant

There is a similar list for the predefined Beanshell methods on the right side of the dialogue.

You can undo any change by pressing the *Undo* button. Use *Redo* button to cancel an undo event. The *Clear* button deletes the content of the editing area.

To finish editing press the *OK!* button. The content of the editing area appears in the text area of the simple editor dialogue.

5.2.2 Step 2 - Sorting

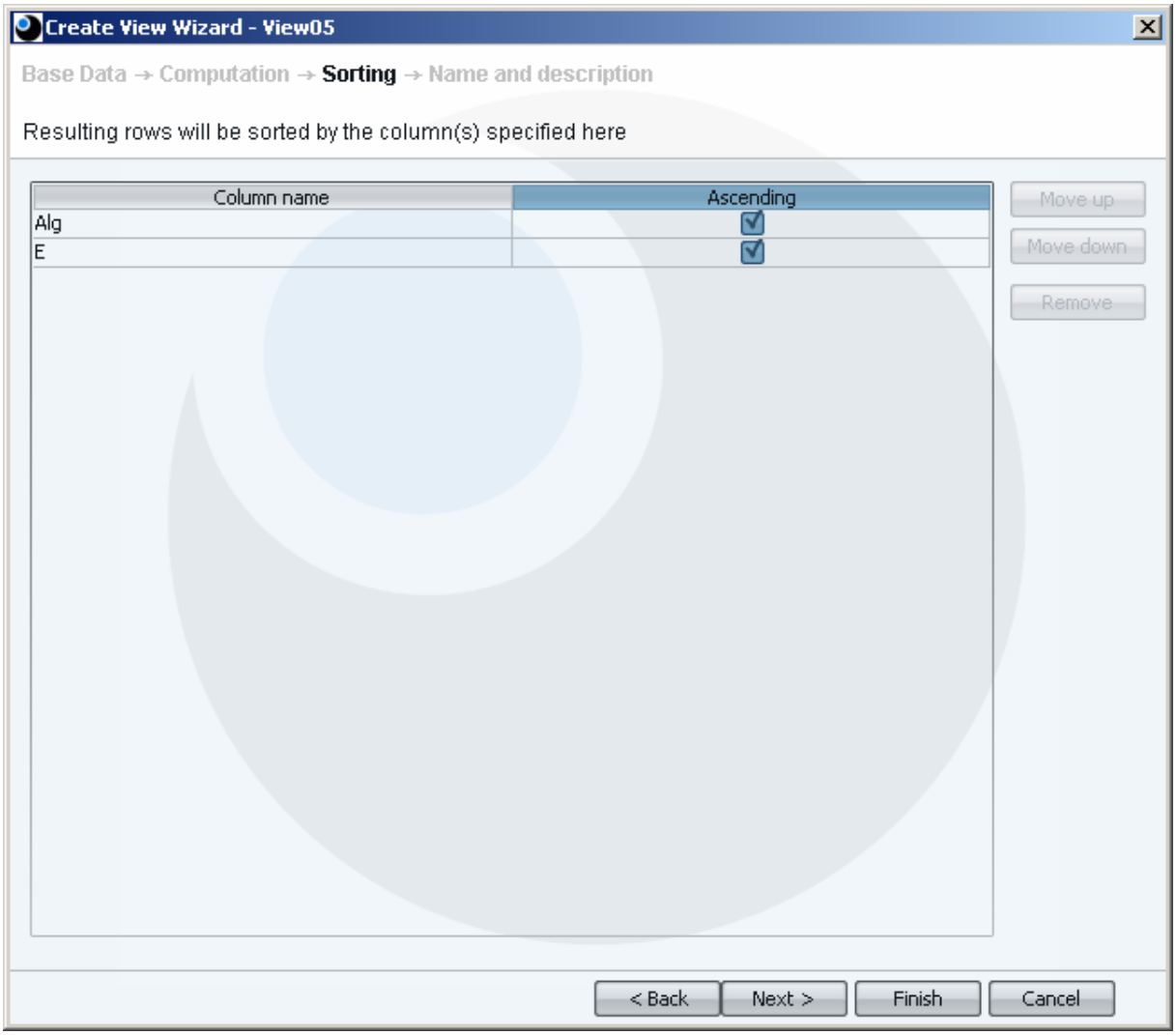


Figure 107 - Sorting

On the *Sorting* screen the layout of the view table can be set by specifying the order of the columns selected to aggregate/group the other columns. You can have these grouping variables in ascending or descending order. Select a column and press *Move up* or *Move down* to modify the layout of the view table.

5.2.3 Step 3 - Name and Description

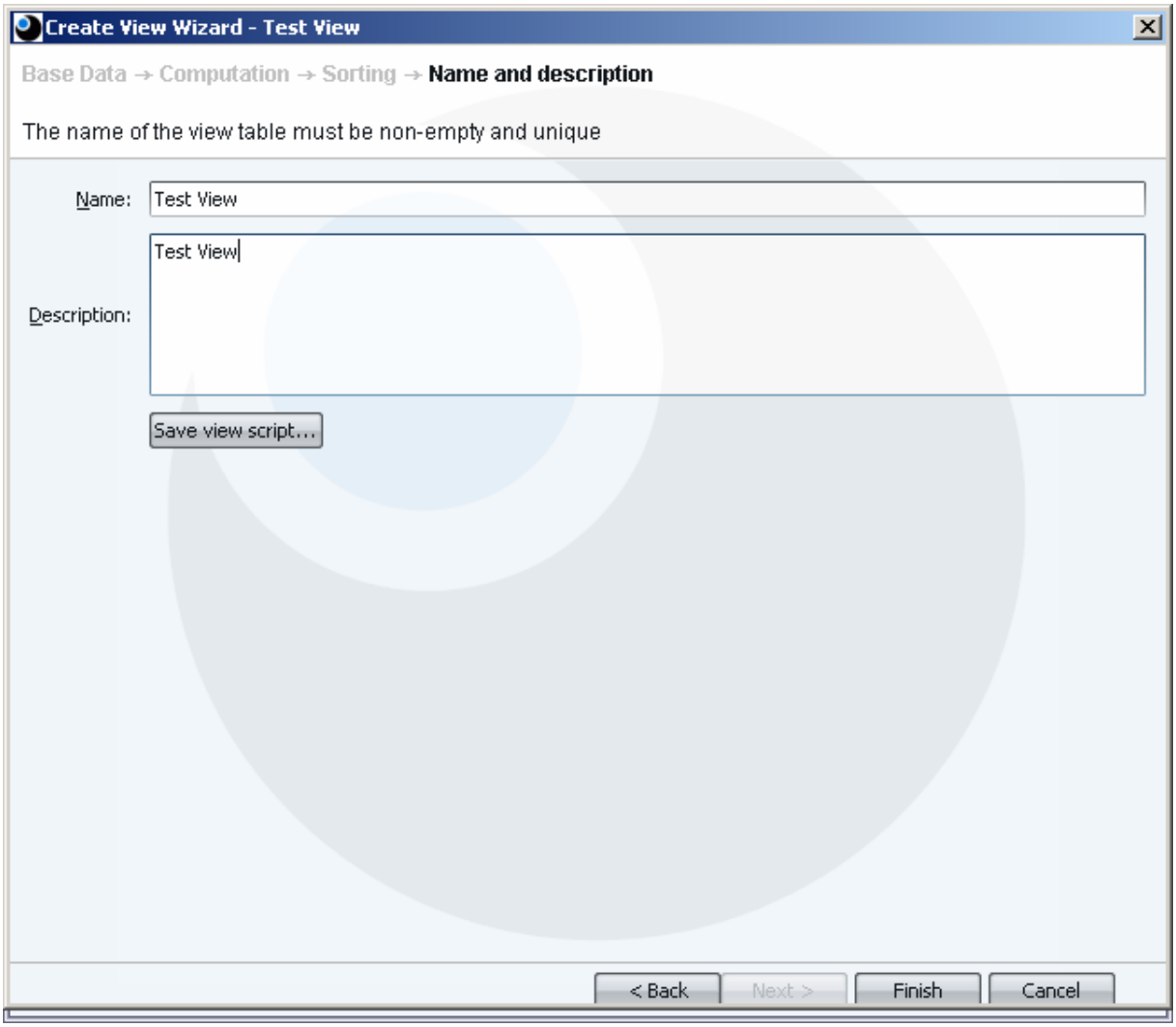


Figure 108 - Name and description

Specify the name of the view. If there's an existing view with the same name it will be overwritten when the *Finish* button is pressed (the program notifies if such view exists already). The name can be max. 64 characters long. In the *Description* field you can enter comments without limitations. Press *Finish* when you're ready.

By pressing *Save view script…* the script describing the view can be saved to a separate file. For further information see 7 Scripting.

5.2.4 The Result

Upon pressing the *Finish* button the *View Creation Wizard* is closed and the new view table is listed on the *Views* tab. MEME verifies whether the scripts used in creating the view are syntactically correct. In case of an error it returns an error message and the wizard remains open in order to enable the user to eliminate the problem. In this case when *Cancel* is pressed an empty view table is created that contains all the settings. This lets the user exit the wizard for solving the problem and opening the wizard again for finalization.

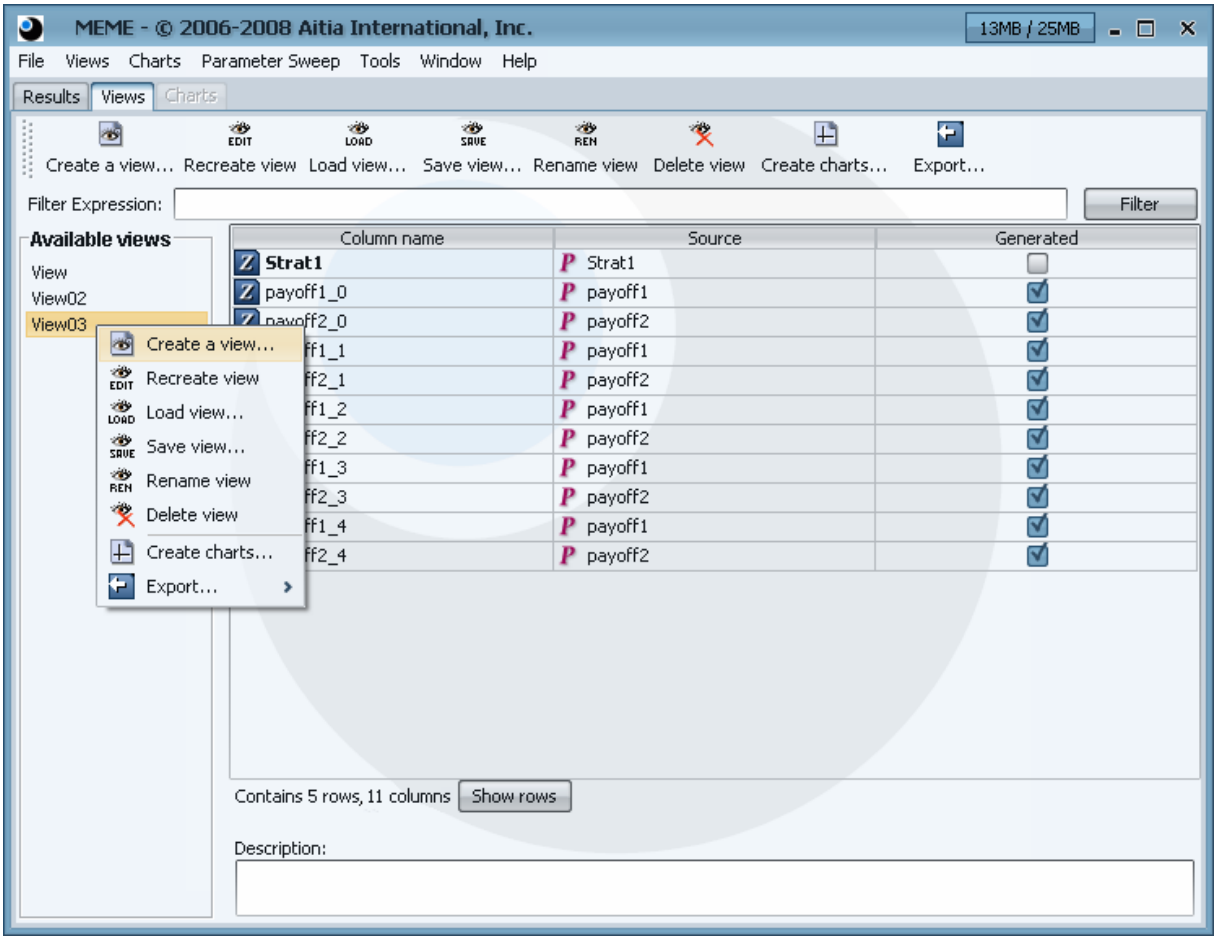


Figure 109 - Views

The *Available views* list can be filtered by typing regular expressions in the *Filter Expression* field and clicking on the *Filter* button or pressing *ENTER.*

On the figure above all the non-hidden columns are listed, and due to splitting five instances of every *payoff1* and *payoff2* value. Note that when splitting, the maximum number of columns is 10,000 (including the hidden columns and variables excluded from grouping).

Columns with names in bold letters indicate splitters (columns that aggregate or group other columns), while the aggregated columns are denoted with regular letters. The *Generated* checkmark denotes that the given column was generated by splitting.

To observe the actual view table press the *Show rows* button, to switch back press *Show columns*. From the view table charts can be created (see 6 Charts) or it can be exported to be analyzed and visualized in other programs (8 User Tools).

In order to delete, save or export multiple view tables hold down the *CTRL* key and click on the views to be selected. Selected views are highlighted and can be deleted, saved or exported at once. When multiple views are selected the data from the latest selection is displayed on the data field. Accordingly, when multiple views are selected and the *Recreate* button is pressed the latest selection is opened for editing.

Columns can be renamed by double clicking on their names in the data field. This feature was introduced to be able to do finishing touches on view tables without having to redo the whole view table creation process. Note that recreating a view table after a columns has been renamed in such a way can cause some problems and is not advisable.

5.2.5 Step 0 – Base Data

The *View Creation Wizard* starts on its second page. A result table is already selected when the wizard is started hence the base data is already given. For advanced users we provide the option of adding multiple results tables and already processed view tables as

base data. To invoke the *Base Data* page press the *Back* button on the *Computation* (starting page of the wizard) page.

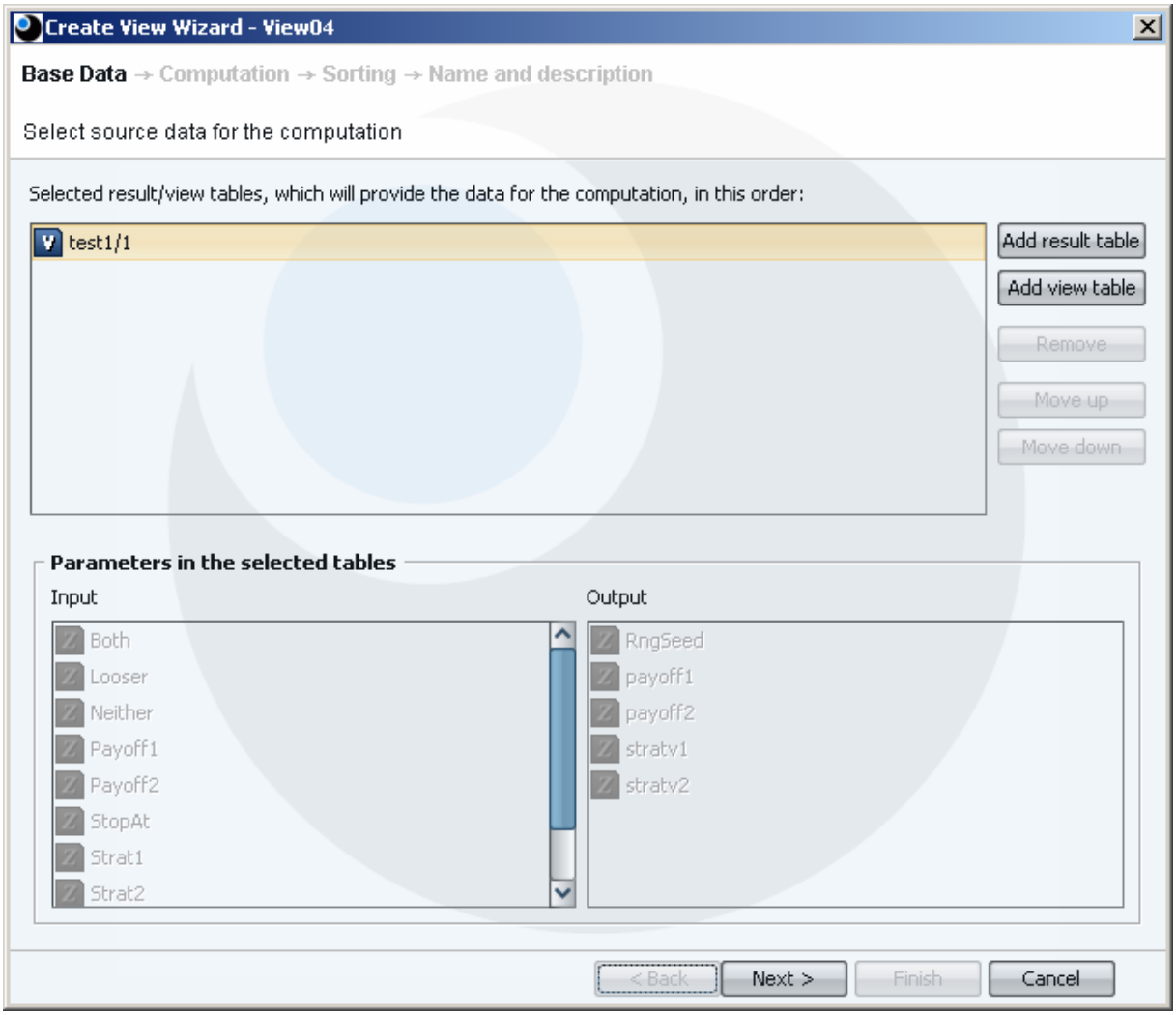


Figure 110 - Base Data

Add result and view tables by pressing the corresponding buttons, selecting the desired table and pressing *Select* on the pop-up dialogue. Select multiple view/result tables for adding, moving and removing by holding the *CTRL* key while clicking on the desired tables.

5.3 Views/Delete

As you may expect this menu item deletes the currently selected view table. All data is deleted and it cannot be undone. This function is available only when the *Views Panel* is active and a view is selected. The *DEL* key is a shortcut for this function.

5.4 Views/Rename view

Views can be renamed by pressing the *Rename view* button on the toolbar or by using the *View/Rename view* menu item. A simple dialogue appears where the new name of the view can be defined. This function is available only when the *Views Panel* is active and a view is selected.

5.5 Views/Recreate

This function allows you to append data to or remove data from a view table. (Precisely, it allows editing the rules that define the view table.) This menu item can be started when a view is selected from the list. It invokes the same *View Creation Wizard* that is

described in 5.2 Create View. MEME remembers the settings used to create the view, including which batches were used and how the columns were computed. You can modify these settings in the wizard. As a result the view table will be created again using the currently available data and settings. If the name of the view table is unchanged it is overwritten with the new settings, while if it is modified the table is duplicated with the desired changes. The latter feature is useful in comparing simulation results with different computations done on them for example.

The recreate function can also be launched by double clicking on the given view in the views listing.

Note that there are differences in handling hidden columns between creating and recreating view tables. If a column is modified to be hidden while recreating it is not automatically left out from the view table, the priority is to assure the unchanged previous sorting still works; it has to be deleted to be excluded.

6 Charts

MEME allows creating charts from view tables only. Thus you have to create a view table from simulation results before creating a chart.

6.1 Charts/Create chart...

Once at least one view table is present in the database charts can be created. Select the view table (if no view table is selected an error message is displayed) in the *Available views* list on the *Views* panel and press the toolbar button or start *Charts/Create chart...* from the menu. This enables the *Charts panel* and switches MEME to chart-creating mode:

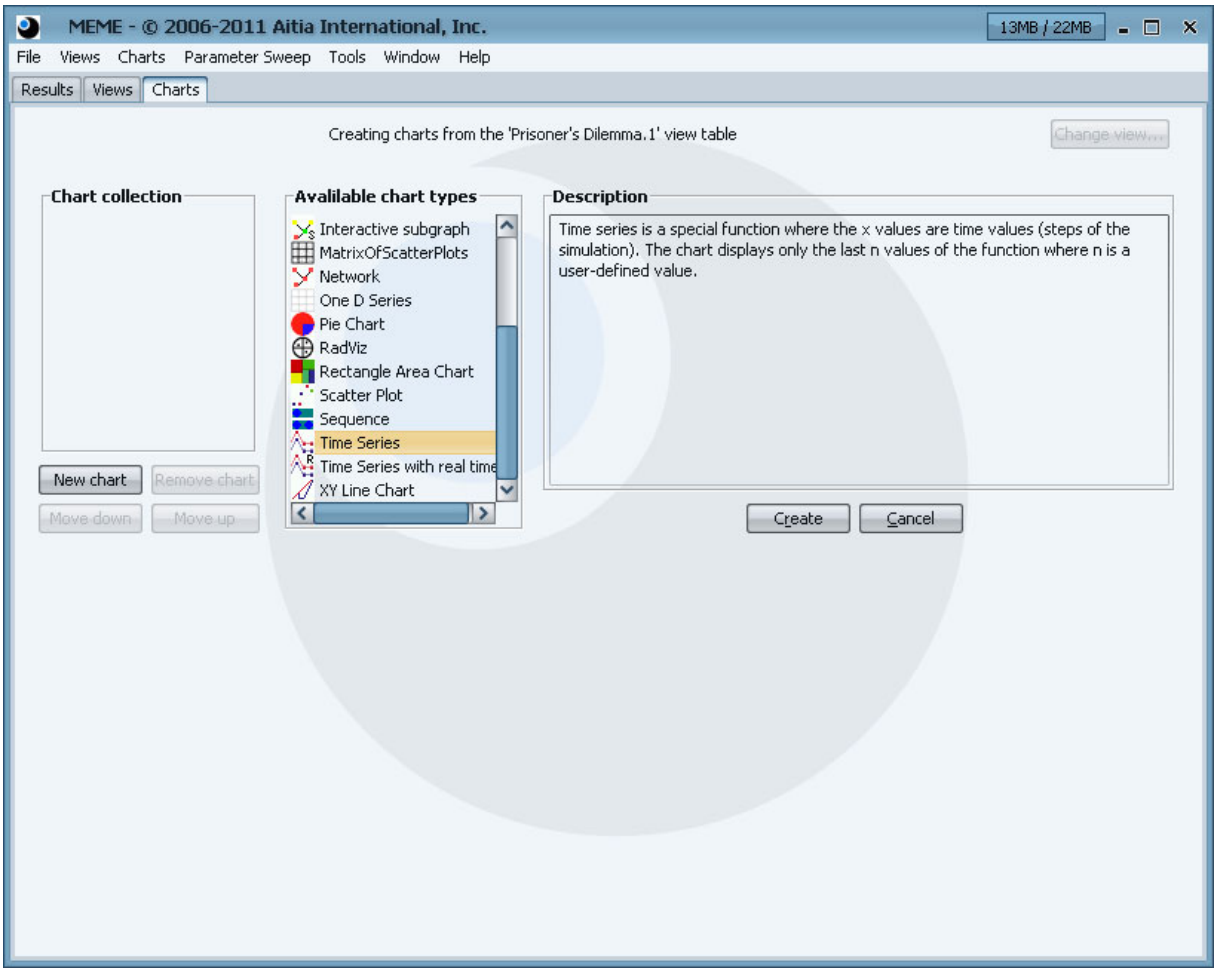


Figure 111 - Create chart

Several charts can be created from a single view table. Once created charts are listed in the *Chart collection* on the left. The settings of the current chart will be displayed on the right. The available chart types are shown in the middle. Select one and press the *Create* button or double click on it. This will add a new instance of that chart type to the collection on the left and show its settings on the right.

The *New chart* button can be used later to return to this screen and add a new chart to the collection. The *Remove chart* button deletes the selected chart from the collection. You can change the order of a created chart with the move buttons. To change chart settings simply select the designated chart in the *Chart collection* list.

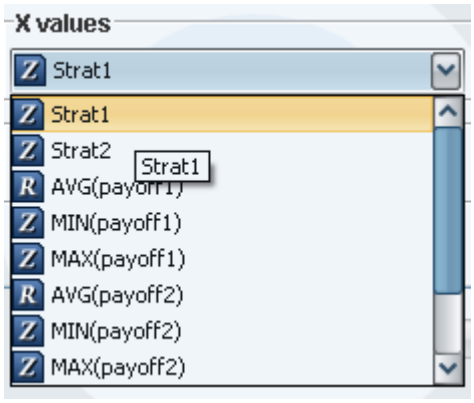
The *Cancel* button terminates the chart-creating mode and switches back to the *Views panel* deleting the unsaved charts.

After you created one or more charts from a view you can apply the created charts to another view if the stucture of the two views are similar (this means for all columns used by the charts there is a column with the same name and type in the new view). To achieve this, simply click the *Change view…* button which is enabled only when all charts in the *Chart collection* list is displayable. A view browser appears where you can select the new view. If the defined charts cannot be applied to the selected view the application shows an error message and switches back to the original view.

The following sections explain the common features of this chart-creating mode. These are necessary to understand the details of the individual chart types, which are described in 6.11 Chart Types.

6.2 Concept: Data Sources

The settings panel of each chart type contains drop-down lists allowing you to select from the columns of the view table. The selected column will provide the data for the corresponding data variable of the chart. The special # item stands for the row number of the view table. It begins at 0 and is incremented by one for every row.



Depending on the chart type and the data variable in question, there can be some difference in the actual type of the values in the view table's selected column and the type that the chart requires. In this case an automatic conversion is applied according to the following rules:

Expected type	Actual type	Conversion rule
integer	real	Real values are rounded down
numeric[9]	logic	"true" is converted to 1, "false" to 0
numeric[8]	text	See below[10]
text	numeric[8]	The numeric value in textual form
text	logic	"true" is converted to "1", "false" to "0"

The "expected type" is described in section 6.11 Chart Types for every chart type and data variable. The "actual type" of the values in a view table column is shown in the *Views* panel (see the description of the  icons in the *Create* section).

6.3 The Compose, Display, Save and Cancel Buttons

These buttons are at the bottom of the chart settings fields. The *Display* button displays the chart in a new window. It is available only if all necessary information has been entered, thus MEME has enough information to display the chart. The *Save* button saves the settings of all the charts in the collection to a file. Note that it saves only the configuration of the charts, not the data that makes up the chart figure.

The *Cancel* button immediately terminates the chart-creating mode (without saving), closes all chart-displaying windows and switches back to the *Views panel*.

[9] Numeric means integer or real

[10] An integer number is assigned to every textual value: after sorting the values (lexicographically) and eliminating repeated, identical items from the series, the serial number is assigned to every text value. This allows using textual columns where numeric values are expected.

The *Compose* button allows creating composite charts that combine numerous different chart types. For example, the following chart was created by combining an XY Line Chart and a Scatter Plot. *Compose* is available for the following chart types: XY Line Chart, Time Series (both types), Scatter Plot and Histogram.

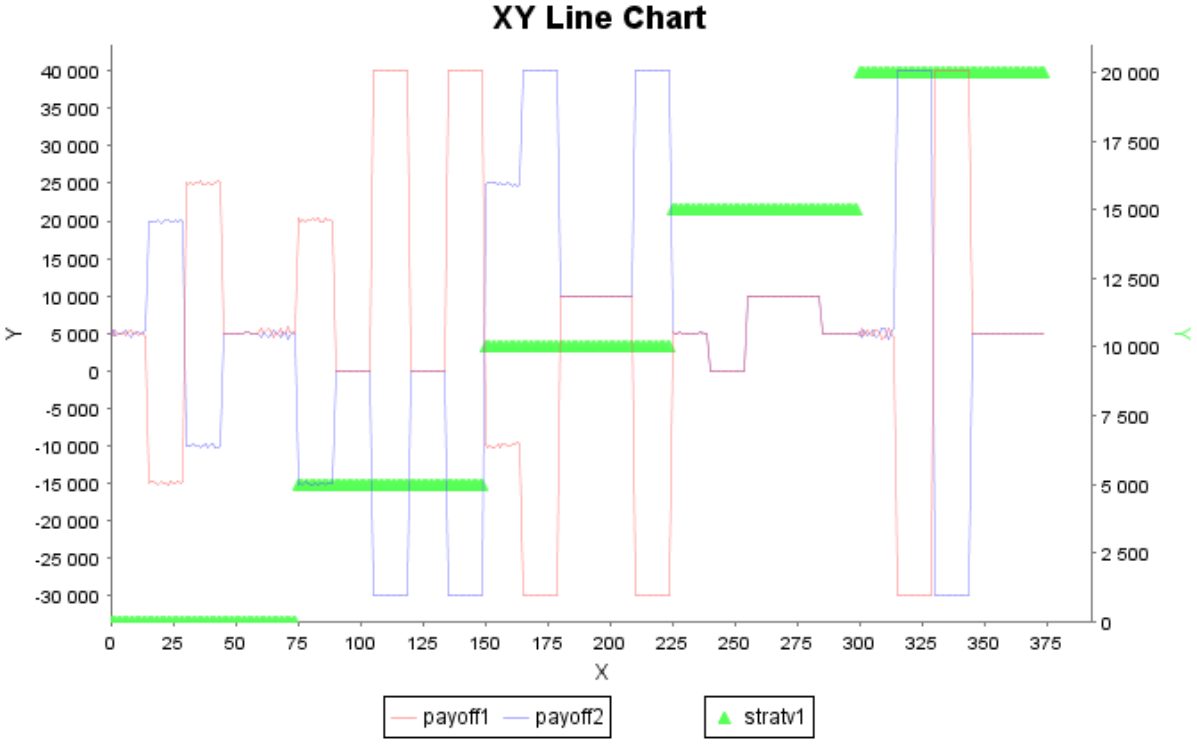


Figure 112 – A composite chart

6.4 Magnifying the Figure

You can closely examine the details of a chart by magnifying a rectangular piece. Select the desired range of the chart with the mouse (holding the left mouse button), from the upper-left corner of the rectangle towards the lower-right corner. You can return to the whole chart by doing the opposite: "draw" a small rectangle from its lower-right corner towards the upper-left corner.

6.5 Saving the Figure

All displayed chart windows have a context menu, which is available by right-clicking on the chart figure. This contains a *Save as…* menu item which can save the figure in PNG, EPS and EMF formats.

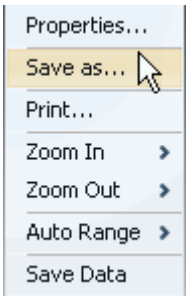


Figure 113 – Context menu

6.6 Saving the Data

The *Save Data* menu item in the previously mentioned context menu allows to the user to export the data displayed on the chart to a CSV file.

6.7 Properties

Through *Properties…* — available from the same context menu — titles and labels can be edited or changed, fonts, border and background colors can be set and axis properties can be customized.

6.8 The Details Tab

The settings of the charts are organized into a tabbed view that always contains a *Main Settings* tab, has usually a *Details tab* and occasionally can have further tabs. The *Main Settings* tab embodies the minimum information that is necessary to make the chart displayable. Further settings — like title of the chart, axis labels, colors, etc. — can be set on the *Details tab* and on the additional tabs that may be present.

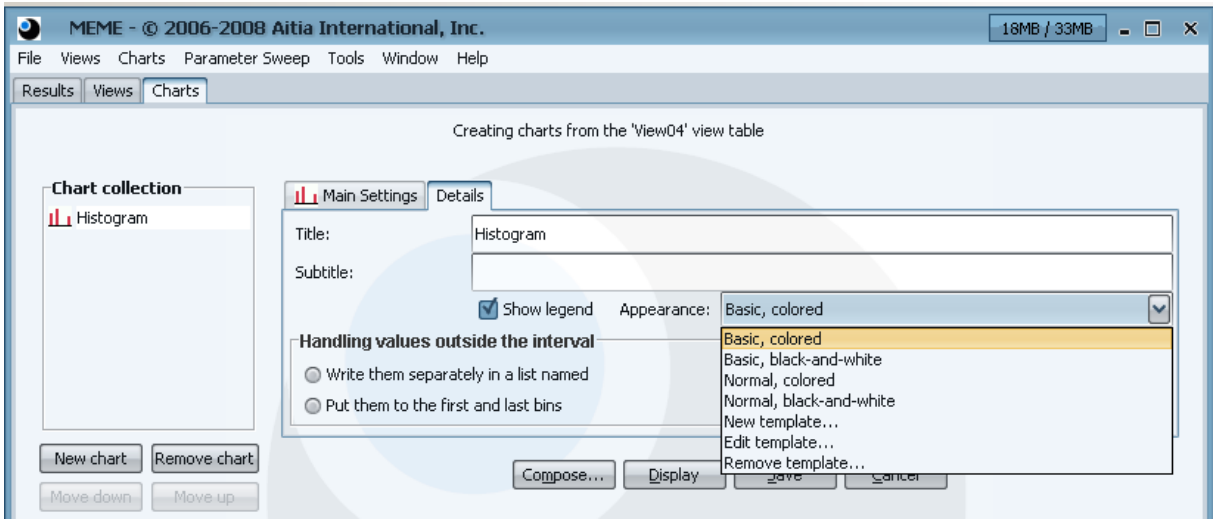


Figure 114 - Details tab

6.8.1 Appearances

It is a common requirement to be able to modify how the charts should look like. In MEME one can create templates that define the properties of the displayed charts. Then these templates can be assigned to the charts, and their visual settings can be handled together.

The properties define all the relevant parts of a chart looks: background color; symbols for points of the functions; characters for the labels; thickness, color, style of lines drawn, and much more.

Available templates can be found on the *Details* tab of each chart in the *Appearance* list. One can create new templates, as well as edit or delete existing ones (see Figure 114).

In the list there are the following previously defined chart templates:

- *Basic, colored:* Chart with colors, but otherwise basic (default).
- *Basic black-and-white:* Simple chart intended to be used in black and white publications.
- *Normal, colored:* Chart with colors and fading.
- *Normal, black-and-white:* Black and white chart with fading.

Note that for some charts (i.e. 2D Grid) there is a separate tab for appearance settings, and the predefined templates might differ.

6.8.1.1 The Appearance Template Builder

On the *General* tab the basic settings of the builder can be set, including the size, title, subtitle colors, fonts, and the background color(s) of created charts.

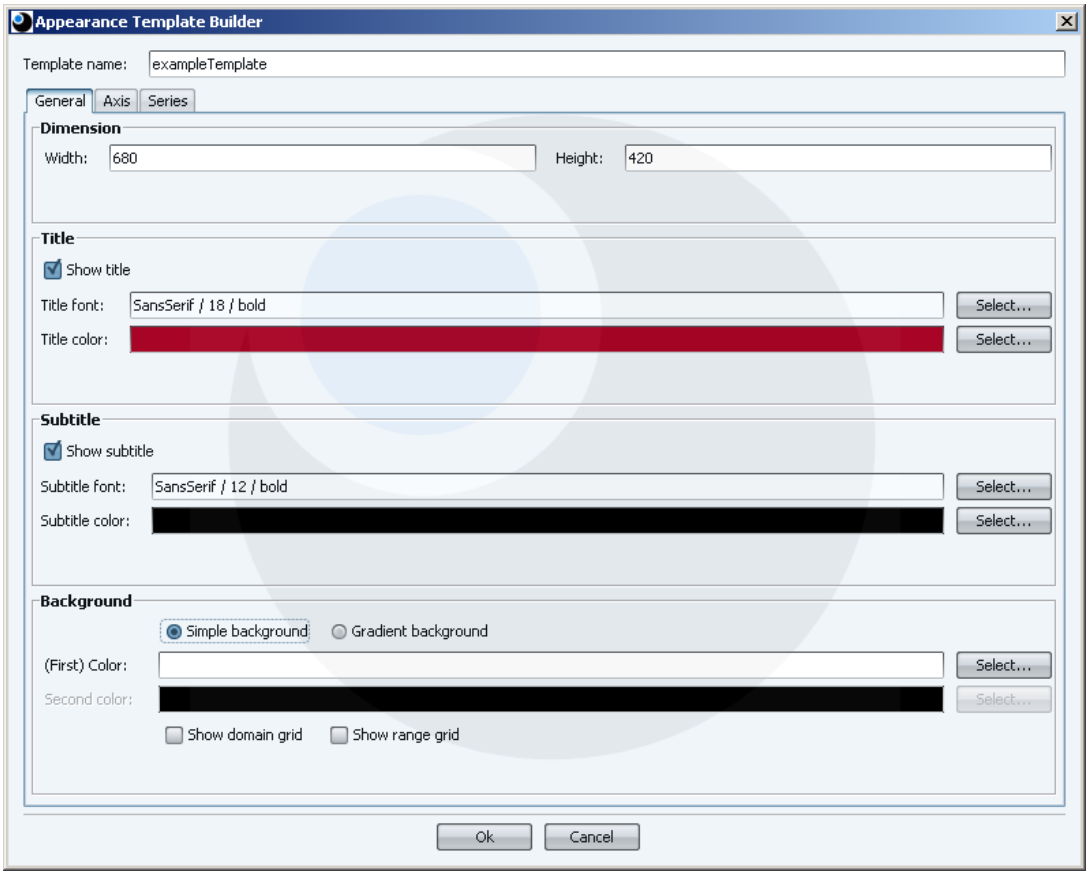


Figure 115 – General settings

Axis tab contains the settings of the displayed axis, including the fonts, colors, tick labels, tick fonts, tick markings, ranges, etc.

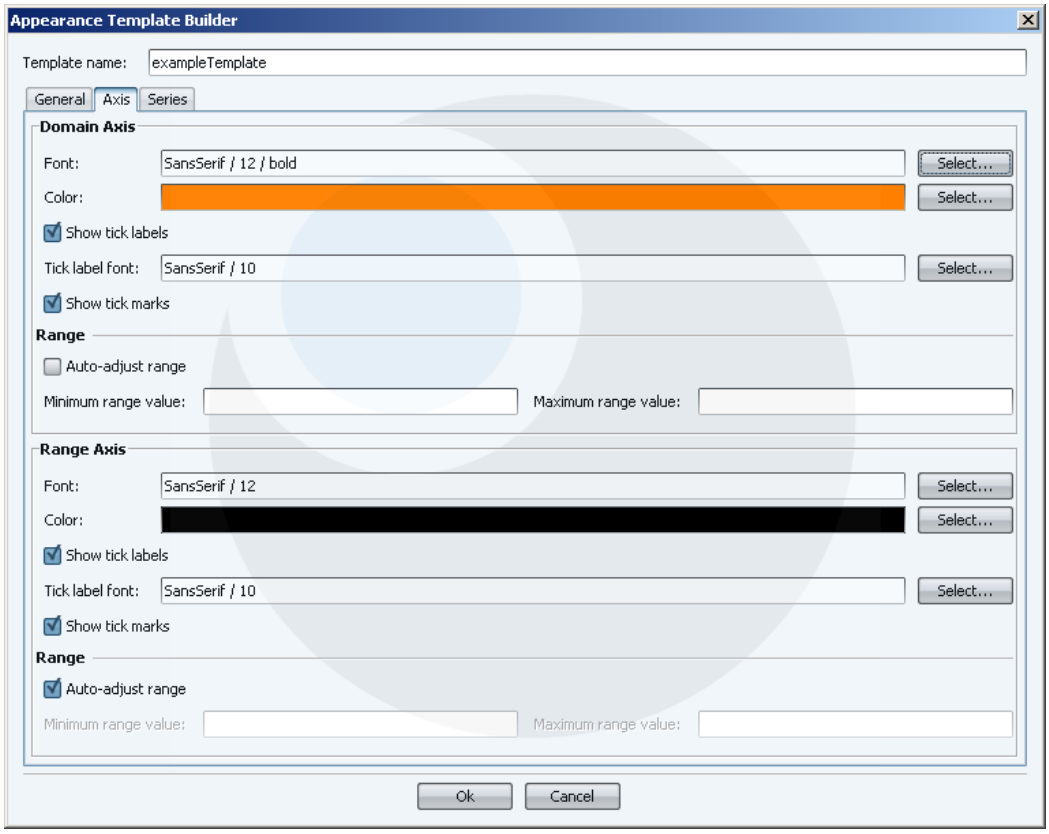


Figure 116 – Axis settings

Series tab contains the settings of the displayed values, including markings (shapes), the relative size of the markings, thickness, style and color of the drawn lines etc.

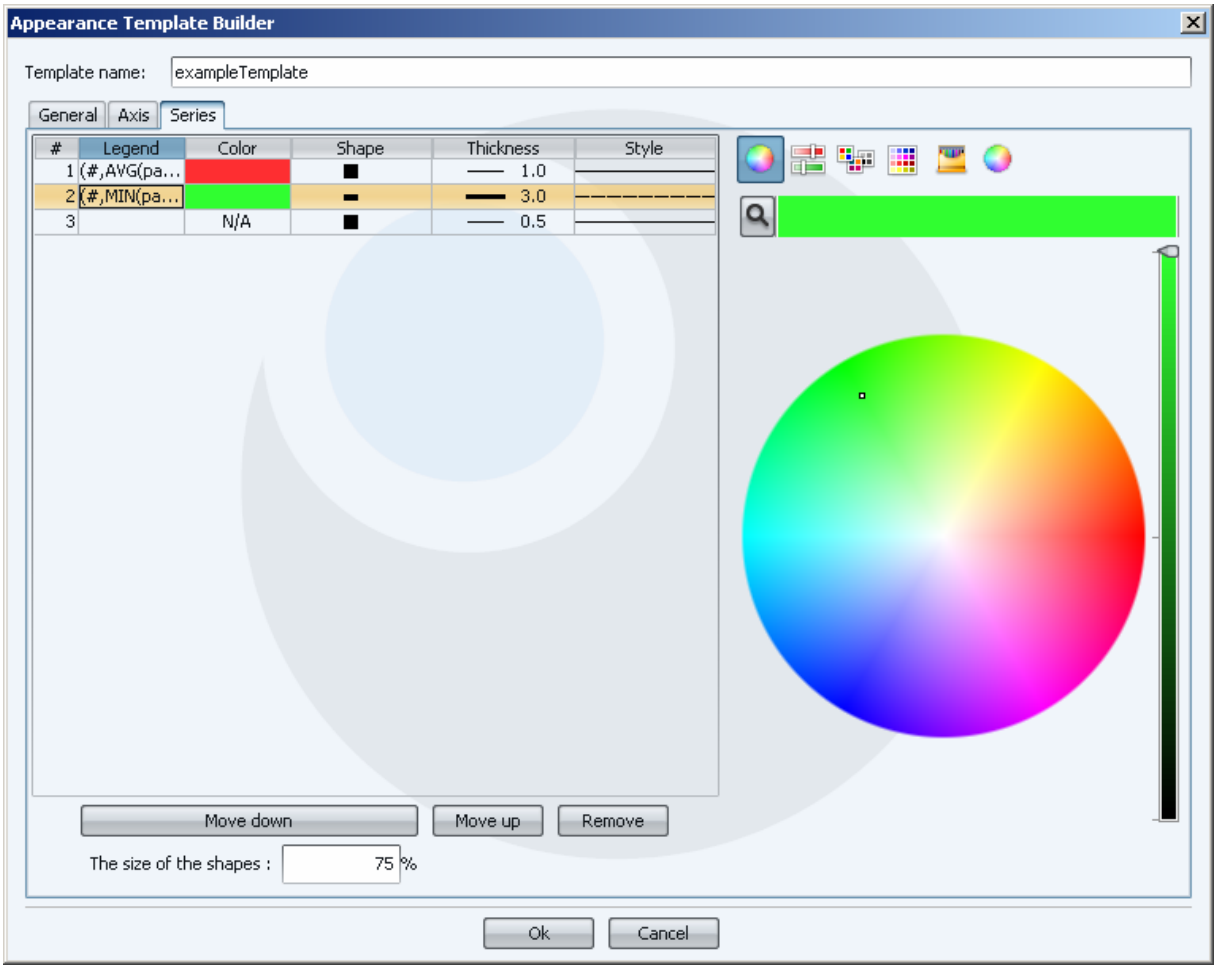


Figure 117 – Series settings

Note that some kind of charts can not use all settings defined in a template. For example, series settings are useless for the 2D Grid. Useless settings will be ignored.

6.9 Charts/Open chart…

This menu item allows the user to open a previously saved chart collection, and continue to edit or display the charts again. If *Open chart* is pressed while in chart creating mode the current chart collection is closed without saving (warning is displayed).

Note that opening chart requires that the view table with the same name and set of columns still exist in the current database, because the data is not saved with the chart configuration. If you have recreated the view table so that it contains more or less data or additional columns, the loaded chart collection will display the data from the updated view table. However, if you have deleted the view table or you have removed columns that are used in the charts you will get an error message when trying to open the saved chart.

6.10 Charts/Export charts as image…

This menu item allows the user to export images into PNG, EPS and/or EMF format from previously saved chart collections (one image per chart). A simple dialogue appears (see Figure 118) where you can specify the format of the exported images. You can choose multiple formats there.

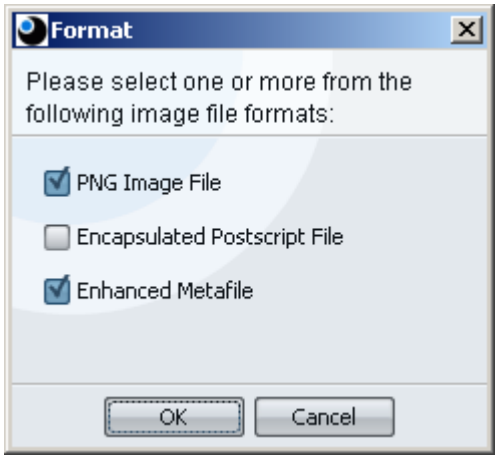


Figure 118 – Export format dialogue

The created image files appear beside the configuration files of selected chart.

Note that exporting charts requires the existence of a view table with the same name as the chart and set of columns referred to by the chart in the current database as at the *Open chart…* command.

6.11 Chart Types

6.11.1 2D Grid

The 2D Grid is a graph that is a matrix of colors. It is very useful in visualizing complex set of variables.

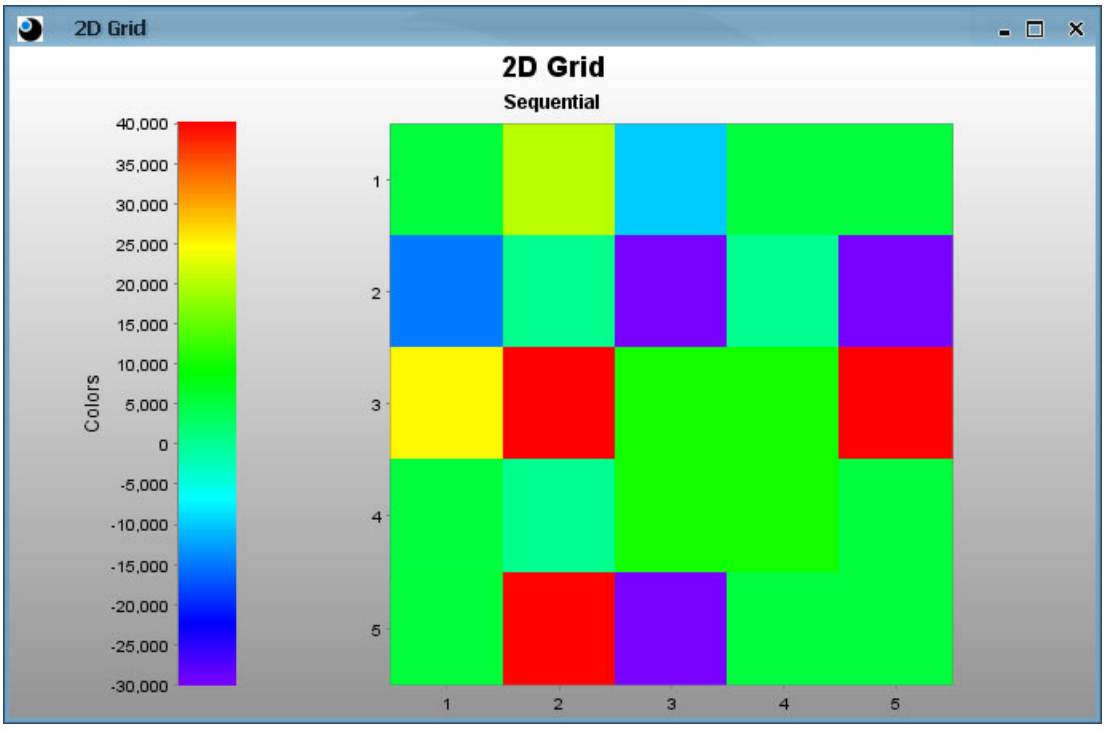


Figure 119 – A 2D Grid with sequential filling

The coloring of the grid can be random or sequential.

- When the matrix is random, three data sources are needed. These are used to form *(x,y,color)* triplets which define the coloring of the matrix. The data sources for the *X values* and *Y values* should be of integer type, while the *Color values* are expected to be real numbers. The specification of the matrix dimensions is optional; the *Width* and *Height*, if entered, are used to ignore *(x,y)* pairs that are

out-of-range. Alternatively, you can define a constant color instead of a color data source. In this case, all defined (x,y) position is marked with the specified color.

- When the coloring is sequential, only one data source (of real type) is needed. Its values define the colors of every cell one-by-one (in left-to-right or top-to-bottom order). The size of the matrix can be given manually (by specifying *Width* and/or *Height*), or specified by data sources.

The *Switch to sequential filling* and the *Switch to random filling* buttons can be used to choose between the two kinds of 2D Grid charts.

By default an adaptive Rainbow colormap is used to assign colors to the real values. This assigns blue to the smallest and red to the highest value and maps the intermediate values using a rainbow-scale, as shown in the picture above. This coloring method can be changed on the *Colors* tab of the chart settings. The alternatives are the followings:

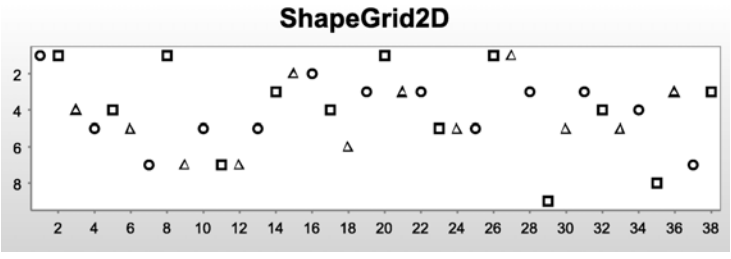
- ***Rainbow colormap:*** Enter a minimum and a maximum value, and define two corresponding colors. Intermediate values will be mapped linearly.
- ***Heat colormap:*** Similar to the *Rainbow* colormap, but only using colors from white to dark red.
- ***Real colormap:*** Similar to the *Heat* colormap.
- ***Pastel colormap:*** Similar to the Rainbow colormap, but only using pastel colors.
- ***Random colormap:*** Similar to the *Rainbow* colormap, but using only 36 colors.
- ***Simple colormap:*** A colormap where colors for minimum and maximum values can be defined and the rest is calculated in between.
- ***Table colormap:*** A colormap where individual colors for individual real values (or for closed intervals), and a default color for all different values can be specified. You can label the colormap and thus you can use it for other grids. The created colormap can be saved to file and loaded later. The file format is very simple, thus it can even be generated with an external tool if it's necessary. Here is an example:

```
#
#Wed Dec 20 03:48:55 GMT+01:00 2006
NAME=ColorMap0
67.0=-16776961
6.0=-52480
45.0=-6750055
87.0=-3342490
DEFAULT_COLOR=-1
```

Note that if you use a constant color instead of a color data source there is no need to pick a colormap so the colormap selection is disabled.

6.11.2 2D Grid with Shapes

It is much like 2D Grid, but instead of colors shapes are assigned to the real values like: ○, □, △. The expected data types of the data sources are the same. In the default algorithm shapes are assigned to values in a given order, allowing repetitions.



This default algorithm can be changed on the *Renderer* tab of the chart settings. The alternative is a manually configured set of (*value, shape, color*) triples or *(value, icon)* pairs or alternatively loading Java classes implementing the *IFigureRenderer* interface.

6.11.3 Composite 2D Grid

This is a simple combination of any number of 2D Grids and Shape grids. This means you can create a list of grid layers and the application draws these layers to the screen in the

specified order; the first one in the list will be the top layer. Every layer is a 2D grid or a shape grid according to the type of the visualization (i. e. colormap or shape renderer).

6.11.4 3D Grid

The 3D Grid is the only 3D chart in MEME which can visualize more complex set of variables than any other chart type. There are three visualization modes available:

- In ***Surface*** mode MEME draws a surface using interpolation on the given values (see Figure 120). Like at 2D Grid you can choose between random or sequential grid filling modes.

  When the matrix is random, four data sources are needed. These are used to form *(x,y,height,color)* quartets. The data sources for the *X* and *Y* values should be of integer type, while the *Height* and *Color* values are expected to be real numbers. The specification of the matrix dimensions is optional; the *Width* and *Height*, if entered, are used to ignore *(x,y)* pairs that are out-of-range.

  At sequential mode, only two data sources (of real type is needed): height and a color data source. Their values define the height and color of every cell one-by-one (in left-to-right or top-to-bottom order). The size of the matrix can be given manually (by specifying *Width* and/or *Height*), or specified by data sources.

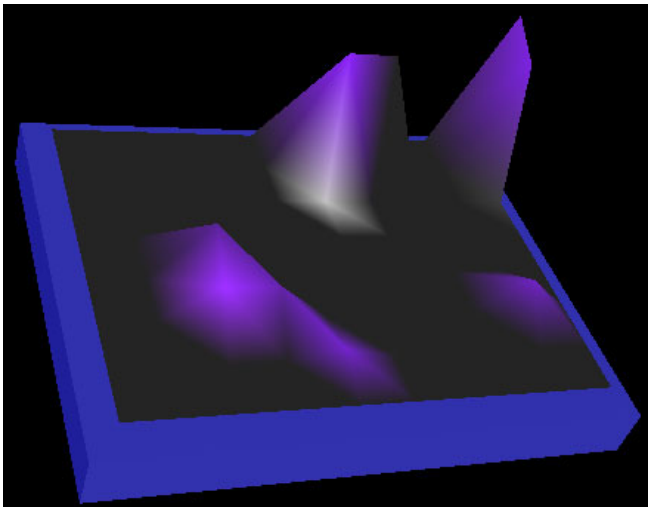


Figure 120 – 3D surface grid

- In ***Diagram*** mode columns are drawn in each cell (see Figure 121). One more data source is needed in both random and sequential grid filling mode: a shape data source which defines the shape of the base of the columns.

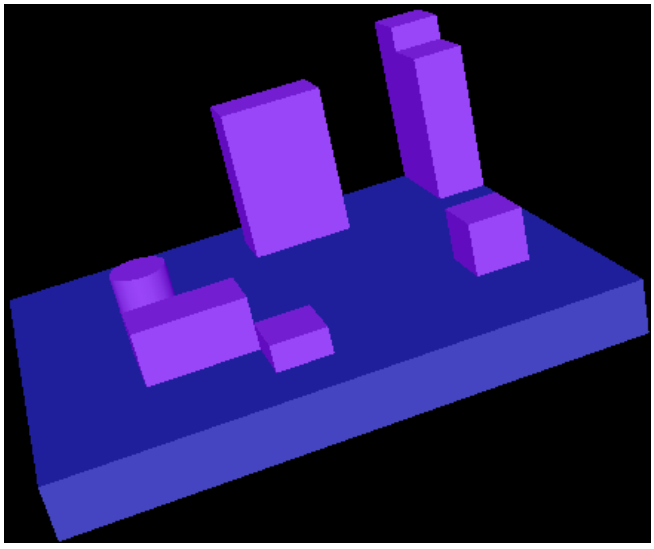


Figure 121 – 3D diagram grid

- In ***Object*** mode a *(x,y,height,color,shape,radius)* sextets (in random grid filling mode) or (*height,color,shape,radius)* quartets (in sequential grid filling mode) define the matrix (see Figure 122).

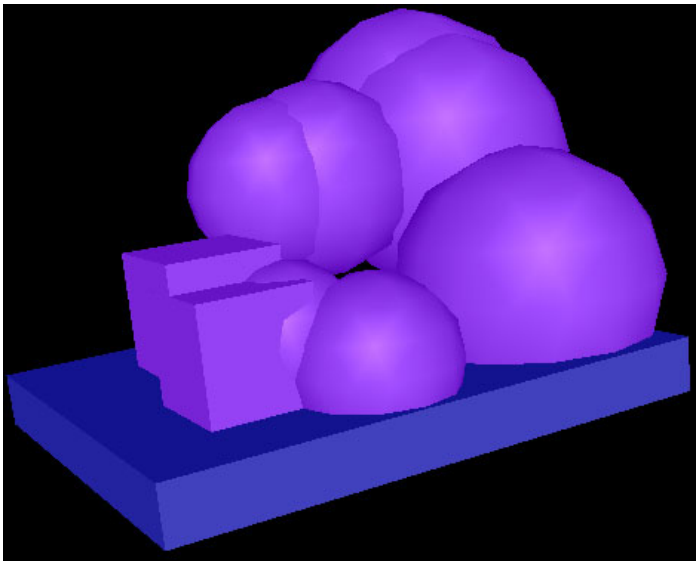


Figure 122 – 3D object grid.

6.11.5 Bar Chart

A classical chart type that allows to the user comparing different series of real values over a shared set of categories. Thus one data source is needed for the categories, and one or more data sources for the values (these are called as *Data Row*s). The category data source is expected to be of text type, its textual values are listed on the horizontal axis. The *Data Row* data sources should be of real type: these values will be represented by bars. There is one group of bars for every category, and within every group, one bar for each *Data Row* (with different colors) by default. There are other rendering options: in cumulative mode for example the *Data Row*s form one divided bar per category. Note that the categories should be different; for each textual value that occurs more than once, bars are drawn for the last occurrence only.

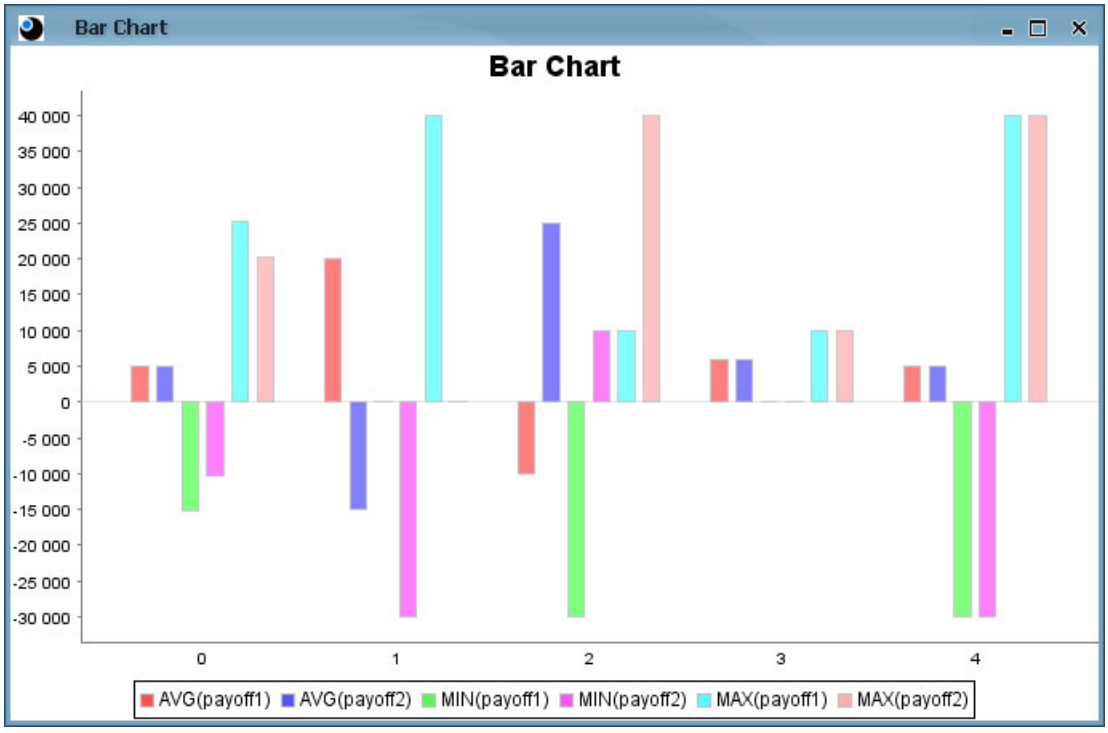


Figure 123 – A bar chart

6.11.6 Box Plot

This type depicts groups of numerical data through their five-number summaries (the smallest observation, lower quartile (Q1), median (Q2), upper quartile (Q3), and largest observation). One can separate the values by assigning them to different categories. One data source is needed for the categories, and one or more data sources for the values.

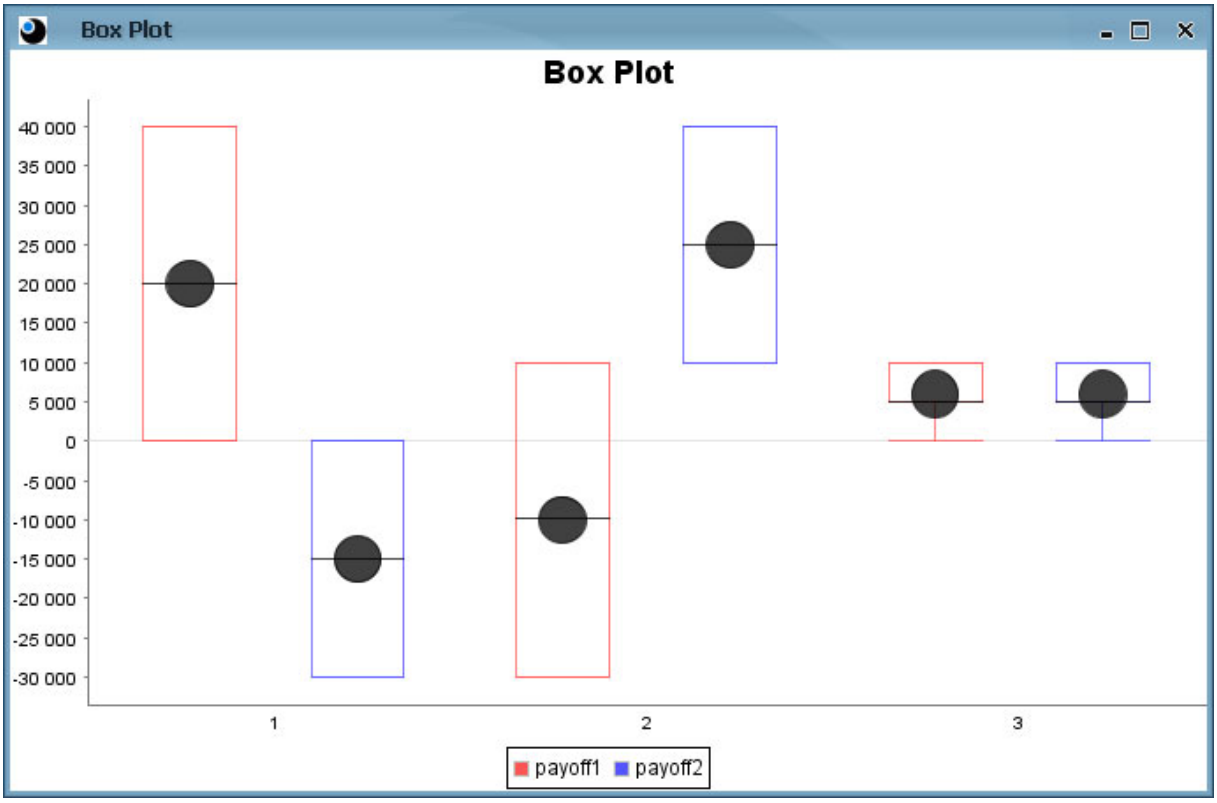


Figure 124 – A box plot

6.11.7 Histogram

The Histogram is a graphical display of frequencies. This chart takes a numeric data source and shows the density of its values within a fixed interval. The interval is divided to *n* equal subintervals ("bins") and the number (or the sum or the average) of values falling into each bin is displayed on the chart using bars of corresponding heights. This chart needs only one data source of real type. The value of *n* has to be given, while lower and upper limits of the interval and lower and upper bounds of the range could optionally be entered. Any of these can come from data source, as well.

On the *Details* tab you can choose how to handle values which fall outside of the specified interval. By default such values are completely ignored. You can define them to be listed explicitly at the bottom of the chart, or to be counted into the first/last bins.

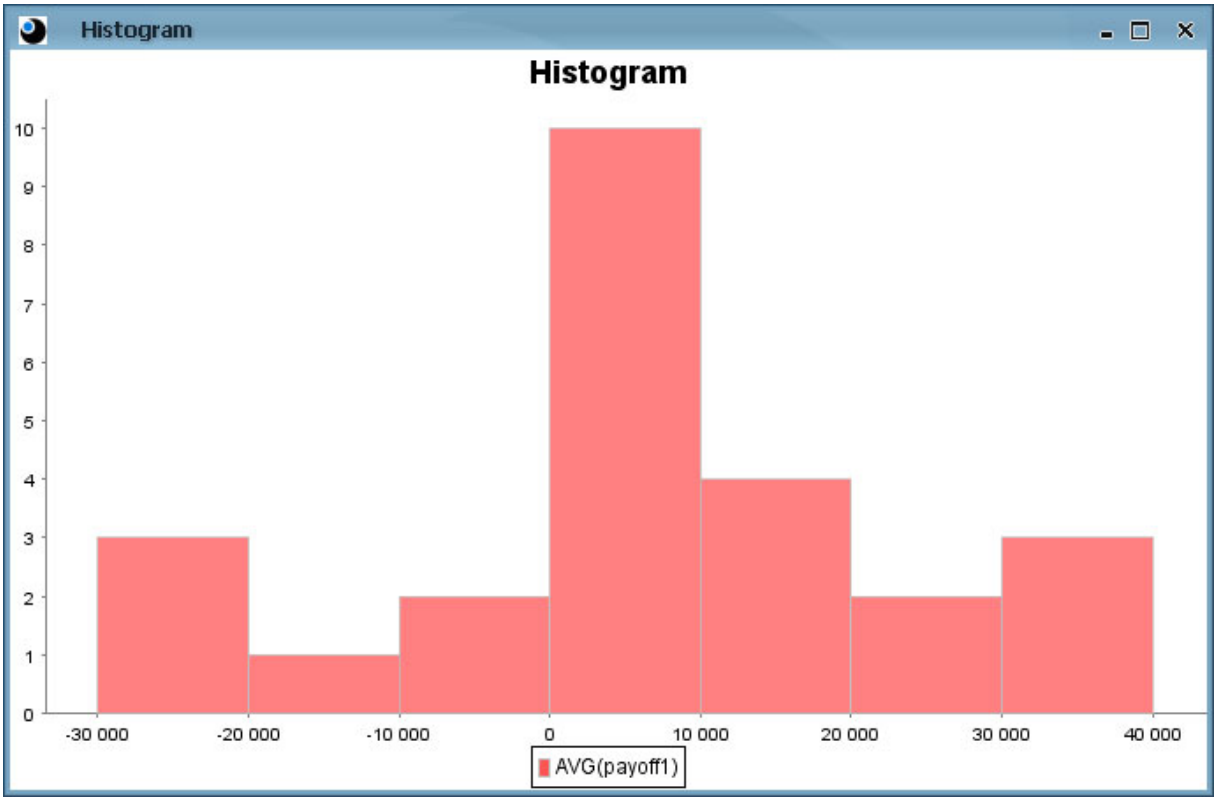


Figure 125 – A histogram

6.11.8 Matrix of Scatter Plots

Matrix of Scatter plots is an array of scatter plots displaying all possible pairwise combinations of series of values. The main diagonal contains the histograms of each series.

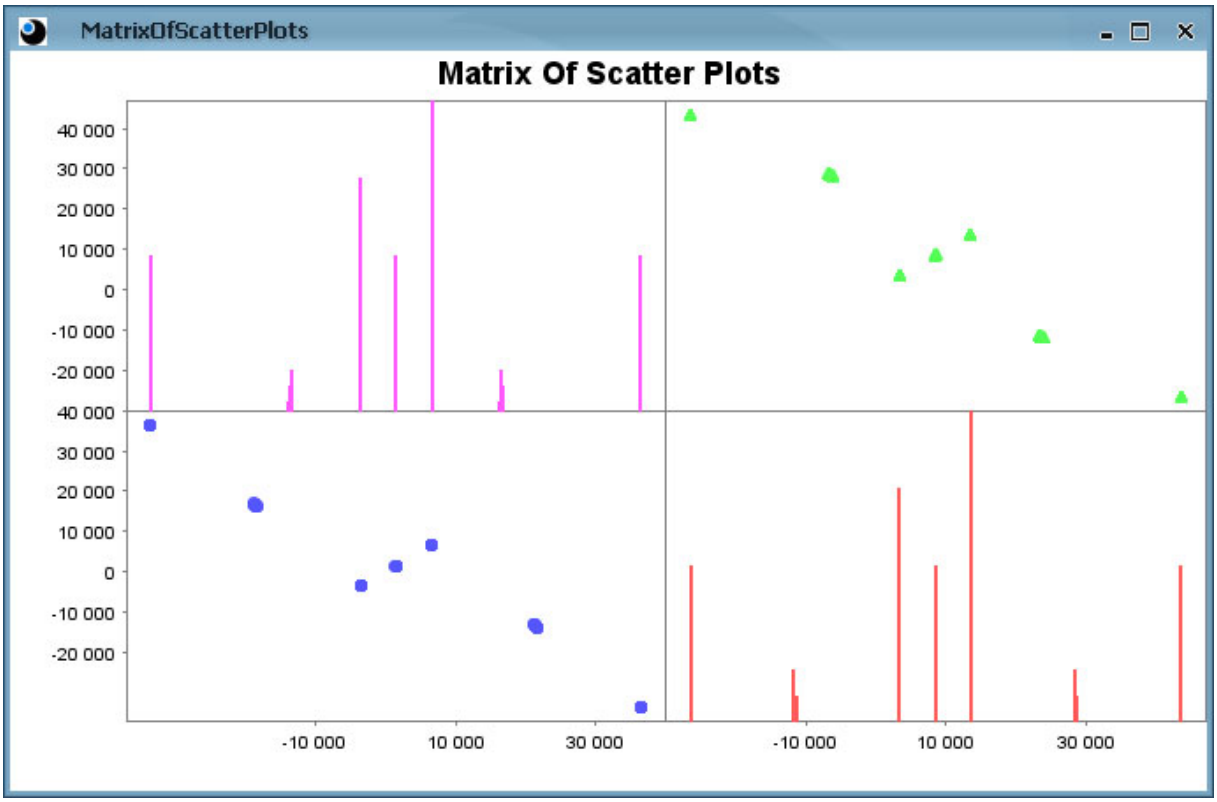


Figure 126 – A matrix of scatter plots

6.11.9 Network

A network (or graph) is a set of items connected by (undirected or directed) edges. Each item is called a vertex or a node. Formally a network is a set of vertices and the binary relations between vertices. The network takes two sets of numeric values (source and destination nodes for the vertices) and a set of string values (labels) as data sources. The number of nodes has also to be specified either by entering manually or by selecting a data source. There are six different graph-drawing algorithms implemented (*Circle*, *Fruchterman-Reingold*, *Spring*, *Kamada-Kawai*, *Kamada-Kawai 3D* and *ISOM (Meyer)*) in MEME. The type of the network can directed or undirected.

Interactive subgraph shows only a part of a graph. The user can define an item and a depth value and the sub-graph shows the given item and its neighbors in the given depth. The user can change dynamically the selected item, so complex graphs can be walked through in an interactive way.

Note that the displayed graph can be exported into a Pajek-file via the context menu of the graph display.

6.11.10 One D Series

One D Series is similar to Grid 2D Chart. It displays the values of a series with colors. One row or column (depends on the main settings) represents one state of the series. The next row (or column) represents the next state of the series, etc.

6.11.11 Pie Chart

A pie chart (or a circle graph) is a circular chart divided into sectors, illustrating relative magnitudes or frequencies or percents. In a pie chart, the arc length of each sector (and consequently its central angle and area), is proportional to the quantity it represents. All together, the sectors create a full disk. It is named for its resemblance to a pie which has been sliced.

This chart takes a set of numeric values (the sizes of the slices) and a set of string values (labels) as data sources.

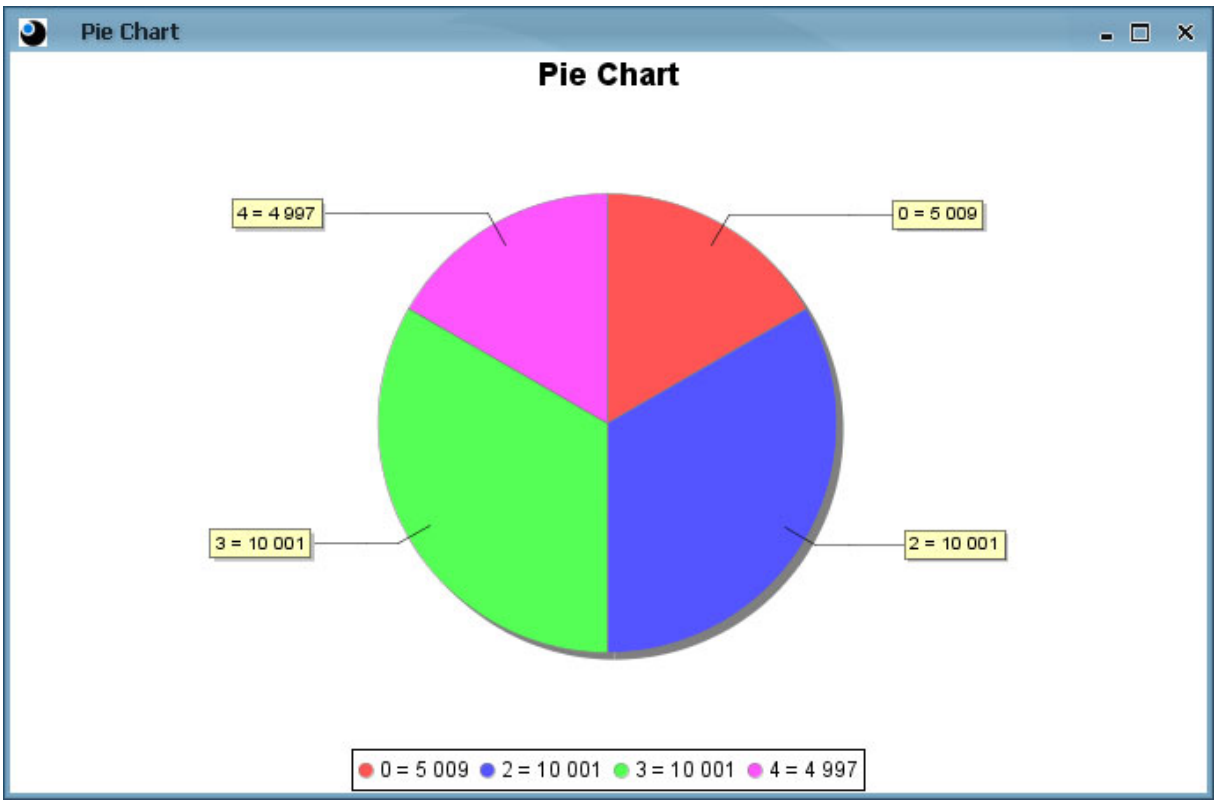


Figure 127 – A pie chart

6.11.12 RadViz (Radial Visualization)

RadViz is a method where the examples are represented by points inside a unit circle. The visualized attributes correspond to points equidistantly distributed along the circumference of the circle. The easiest way to understand the method is by using a physical analogy with multiple springs. For visualizing each data example represented by *m* attributes, *m* springs are used, one spring for each attribute. One end of each spring is attached to the attribute's position on the circumference, and the other to the position of the data point inside the circle. The stiffness of each spring in terms of Hooke's law is determined by the corresponding attribute value - the greater the attribute value, the greater the stiffness. The data point is then placed at the position where the sum of all spring forces equals 0. Prior to visualizing, the values of each attribute are usually standardized to the interval between 0 and 1 to make all the attributes equally important in "pulling" the data point. Some properties of the radviz method are:

- All the points that have approximately equal values of all the attributes after standardization lie close to the center of the circle.
- Points that have approximately equal values at the attributes which lie on the opposite sides of the circle will also lie close to the center.
- If one attribute value is much larger than the values of the other attributes, then the point will lie close to the point on the circumference of the circle which corresponds to this attribute.

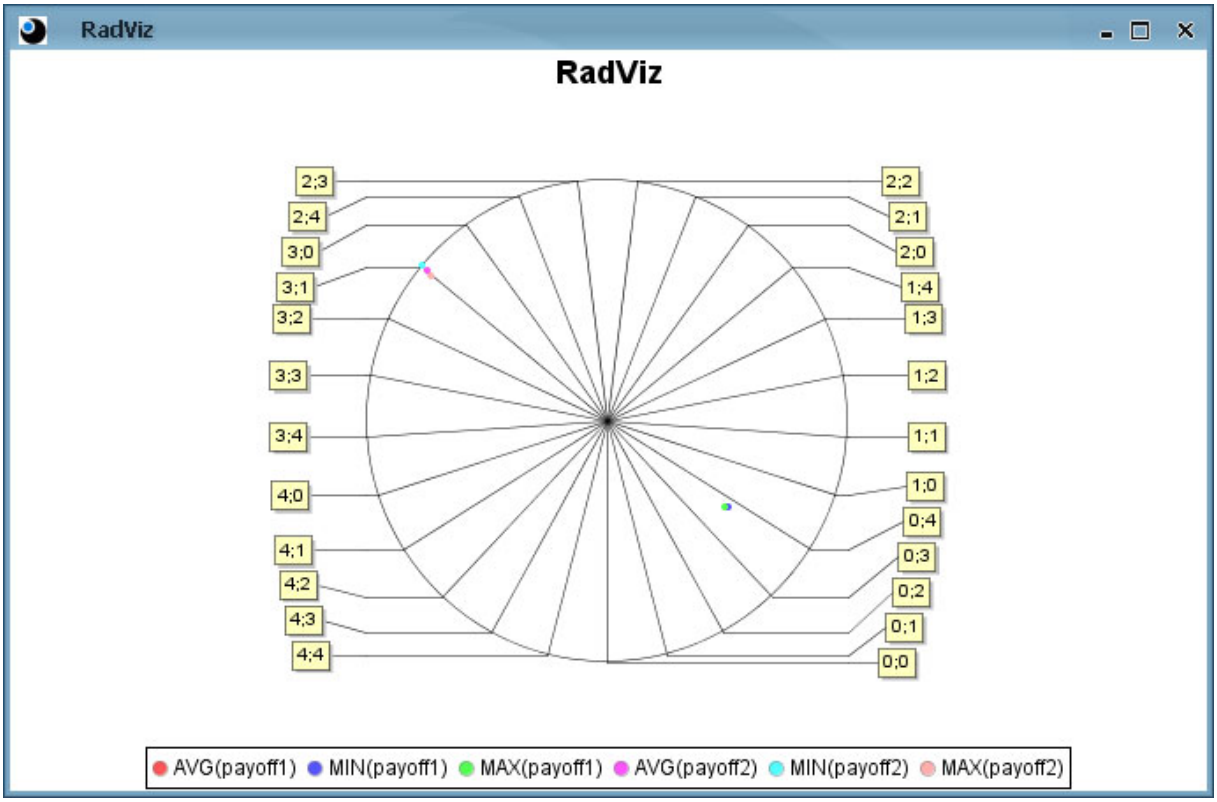


Figure 128 – A radial visualization

6.11.13 Rectangle Area Chart

It is similar to the well-known pie chart, but uses a rectangle instead of a circle. It takes a numeric (real) data source to make up a large rectangle from several small rectangles whose areas are proportional to the values of the data source. A second numeric data source is needed to determine the colors and a third one (of type text) to provide labels for the sub-rectangles. The colors are assigned to the numeric values of the second data source with the same colormap rendering methods as 2D Grid.

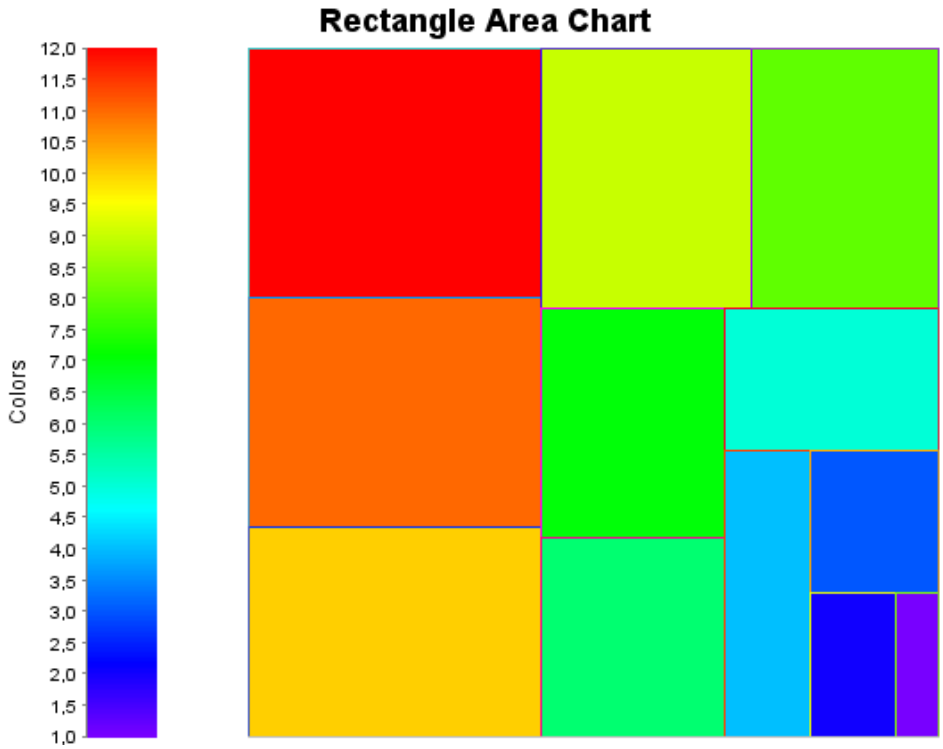


Figure 129 – A rectangle area chart

The Rectangle Area Chart originates in financial market analysis where it is often called "Market carpet" or "Map of the market". This chart is very useful in visualizing datasets with two different attributes that need to observed at once.

6.11.14 Scatter Plot

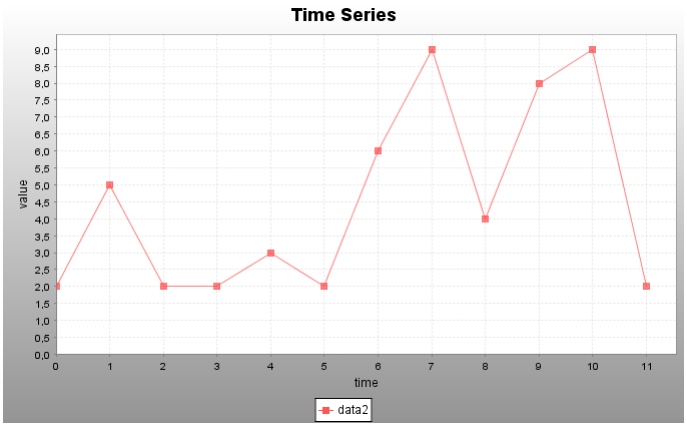
It simply displays a set of (*x*,*y*) coordinate pairs as individual points in an *xy* coordinate-system. The pairs are defined by two numeric (real) data sources, *X values* and *Y values*, just like in the case of XY line chart.

6.11.15 Sequence

A sequence chart is a visualization for ordered series of sets (data sources can only be ordered sets of numeric values). Visualization of the set elements can be customized; default, user defined or other custom rendering can be used and element colors and/or figures can be set.

6.11.16 Time Series

This chart displays the last *n* value of a series on a classical xy-axis line chart. The value of *n* is 100 by default, but it can be specified on the *Main Settings* tab. The chart expects numeric (real) data sources — these will provide the *y* values —, while the *x* values are their ordinal numbers in the series. More than one data sources can be added, in which case more than one lines will be drawn on the chart with different colors.



Time Series with real time is a special function where *x* values are (real) time values.

6.11.17 XY Line Chart

This is the classical xy-axis line chart. Two numeric (real) data sources are needed: one for the *X values* and the other for the *Y values*. The resulting $x \mapsto y$ mapping (where $x \in$ *X values*, y $\in$ *Y values*, from the same row of the view table) is drawn using a line, or a line and the nodes at the data points, or only the nodes (like at the Scatter plot). The x values need not be in ascending order, the (x,y) pairs are automatically reordered if necessary. It is also possible to draw more than one line on the same chart: use the *Add*► button to define more than one *X value*, *Y value* data source pairs.

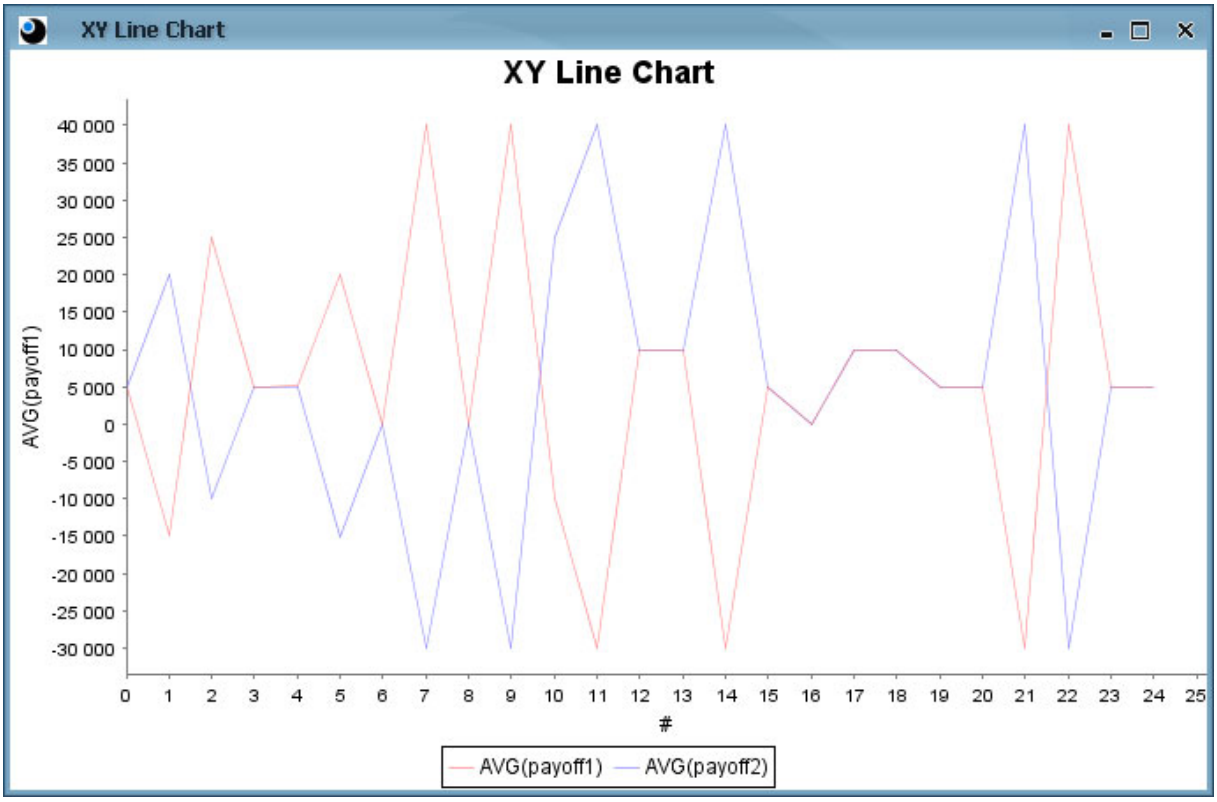


Figure 130 – An xy line chart

7 Scripting

Note: Scripting is for advanced users. It requires some programming skills and the basic understanding of the xml format.

There are three types of scripting in MEME:

- In the *Parameter Sweep Wizard* the user can create data sources to calculate derivative values (e. g. scripts written in Java or NetLogo) from "raw" information (e.g. already existing variables and methods) and then these data sources can be used as recordable elements. See details in 3.2.6 Creating Data Sources.
- Since MEME stores information describing view tables and charts in XML format, it is capable of loading table from and saving table definitions in these XML files. When there are multiple view tables that need to be modified editing these xml files can come quite handy.
- Beanshell scripting is used to describe complex column computations during the view creation.

7.1 Saving View Scripts

In order to save a view or multiple view scripts select the view table(s) to be saved and press the *Save view…* button. View scripts can also be saved by clicking on the *Save script* button in the *View Creation Wizard* (see 5.2.3 Step 3 - Name and Description).

When saving *<view table name>.xml* is offered as default name for the resulting XML file. This can only be changed when saving a single file, if multiple XML files are created the default names are used.

Note that saving the script of a view table is not equivalent to exporting (see 4.6 File Export) a view table. When exporting all the data in the view table is exported while when saving only the script describing the table is being saved.

7.2 Loading View Scripts

In order to load a single or multiple view tables press the *Load view…* button on the *Results* or *Views* panels. To load multiple view tables simply select multiple XML files in the open file dialogue.

7.3 Editing View Tables

The main objective for saving and loading scripts describing view tables is to be able to edit them. When a task requires the creation of a large number of view tables with little differences it can be quite time consuming to create the tables one by one in MEME. Also, when there are already a number of view tables in the program from different data sources but with the same settings for example, and those settings need to be changed, recreating all the tables in the program is just not reasonable. In these cases editing the scripts describing the view tables can save the modeler time and effort.

In order to help understand the XML files describing tables a well commented example code presented in the following. Comments start with `<!--` and end with `-->` and are written in `blue`.

7.3.1 Example Code

```
<viewrule version="1.0.70221">
  <name> name of the view </name>
  <description> narrative text </description>
  <data>
     <table name="res" version="101-200" />      <!-- results table -->
     <table name="view3" />                      <!-- view table -->
     <!—have any number of result or view tables here -->
  </data>
```

```
  <columns ordering="0 1" grouping="true">
    <!-- The 'ordering' attribute is optional. The numbers are
         ordinal numbers for the columns below. 0 represents the first
         column. Empty by default. The 'grouping' attribute is of
         boolean value (true/false), false by default. -->
    <column datatype="12.64.0" grouping="false" hidden="false"
    aggrfn="AVG" splitter="false">
      <!-- datatype="12.64.0" means varchar(64). This datatype string
           is generated/interpreted by ColumnType. -->
      <!-- grouping=”true” and aggrfn<>"" are mutually exclusive -->
      <!—if splitter=”true” then grouping=”true” -->
      <name> colX </name> <!-- column name in the view table -->
      <splitted>_</splitted>
        <!-- Only used for splitted columns, results in
             hidden="false" and grouping="false" -->
        <!-- there can be more than one projection_* term, but then
             there is no *_script term, there can only be one of those -->
      <projection source="input"> anInputColumn </projection>
      <projection source="output"> anOutputColumn </projection>
      <projection source="view"> previousColumnName </projection>

      <scalar_script datatype="..."> beanshell expression </scalar_script>
        <!-- 'datatype' is only used when aggrfn!="" -->
      <group_script> beanshell expression </group_script>
        <!-- in case of group_script the grouping="false" and aggrfn="" is
             mandatory-->
    </column>
  </columns>
  <condition>
    <scalar_script> col1 == 3 </scalar_script>
  </condition>
</viewrule>
```

7.3.2 Notes

On datatype coding (`datatype="12.64.0"`): The first number is the type code, the second and third numbers are parameters (example: size of the VARCHAR; 0 if not in use). The types are described in detail in the java.sql.Types class. Our amendments:

- ***boolean*** is represented by java.sql.Types.CHAR
- ***integer*** is represented by java.sql.Types.INTEGER
- ***long*** is represented by java.sql.Types.BIGINT
- ***double*** is represented by java.sql.Types.DOUBLE
- ***string*** is represented by java.sql.Types.VARCHAR

If a script referring to a result table is changed to refer to a view table all variables need to be changed to output as there are no input variables in view tables. For the columns affected `<projection source="input">` has to be changed to `<projection source="output">`.

7.4 Beanshell Scripting

Note that Beanshell scripting is for advanced users only as it requires some programming skills.

For a detailed Beanshell manual see the Beanshell homepage [6]. This manual describes only the MEME-related details to help you create custom expressions and scripts in the *View Creation Wizard*.

Scripts can contain multiple lines. Their return value will always be the value of the last evaluated assignment or method calling or the value that is given back with a return statement.

There are two types of Beanshell scripts in MEME: scalar and aggregative. A scalar script returns a single value while an aggregative script returns an array. In scripts you can refer to the parameters with their names, because MEME creates a global variable for

each parameter. If you use a parameter in a scalar script, its value will be a single value (`double`, `int`, `String` etc.). On the other hand, in an aggregative script a reference to a parameter means an array (`double[]`, `int[]`, `String[]` etc.). For example, while in scalar script `p+3` can be a valid expression, in an aggregative script it is not. On the other hand `p.length` is only valid in an aggregative script.

7.4.1 Predefined Methods

Since ticks and runs are cornerstones in simulation data, MEME has built-in scripts to return them: `$Tick$` and `$Run$`. Following is the list of other predefined methods you can use.

- `get("name")`: Returns the value of the parameter named `name`. There can be a difference between the return value of this method and the value of the variable created from the `name` parameter:
  - The `get("name")` always returns the original value of the parameter, even if the value of the global variable `name` is changed. (Except in an aggregative script when the variables are arrays and its elements are replaced and not the whole array.)
  - If the name is not a valid Java identifier (for example *Column0.1* or *Column()*), therefore there is no global variable name, you cannot access the parameter with its name, the method must be used.

  If parameter `name` doesn't exist or name is `null`, the script will throw an error.
- `geti("name")`, `geto("name")`, `getv("name")`: Returns the value of the parameter named `name` from the input/output/this view category. This is necessary when there are more parameters with the same name but in different categories. In this case you can access the first parameter only via the global variable `name` or the `get("name")` method. The scope order is the following: *Output*, *This view*, *Input* (same as the order of the list of available parameters in the wizard). For example, the parameter named `name` in the *Output* category hides the parameter with the same name in the other categories.
- `gett("name")`: Returns technical parameter values, which are the following:
  - `batch`: The current batch number or null if the current input row doesn't belong to a batch (e.g. there is no corresponding row in the view table).
  - `run`: The current run or null if the current input row doesn't belong to a run.
  - `tick`: The current tick in case of a result table and the ordinal number *(#)* in the case of a view table.
  - `model`: The name of the current model or null if the current input row doesn't belong to a model.
  - `version`: The current version or null if the current input row doesn't belong to a model.
  - `starttime`: The start time (long) of the current run or null if the current input row doesn't belong to a result table
  - `endtime`: The end time (long) of the current run or null if the current input row doesn't belong to a result table.
  - `view`: The name of the current view table or null if the current input row doesn't belong to a view.
  - `isgroupfirst`: True if there is grouping and the current row is the first in the group, otherwise false.
  - `isgrouplast`: True if there is grouping and the current row is the last in the group, otherwise false.
  - `isblockfirst`: true if there is blocking and the current row is the first in the block, otherwise false.

  - `isblocklast`: true if there is blocking and the current row is the last in the block, otherwise false.

- `call("fn", args…)`: Calls the aggregation operation named `fn`. For example, the result of the `call("AVG", p1, p2)` is the average of *p1* and *p2*. In `args` you can enumerate arbitrary expressions, not just parameter variables. If you want to call `fn` with 11 or more arguments, you must use a `Object[]` as args. `"fn"` is the name of the aggregative operation, the same that appears in the Aggregative operation drop down list with or without brackets. You can also use the fully qualified name of the Java class that implements the aggregative operation (for example, `"ai.aitia.meme.builtinvcplugins.Avg"`).
- `round("name", precision)`: Rounds the value of the parameter named `name`. `precision` is the maximum number of digits after the decimal sign. This is an optional argument; without it the operation returns the closest integer.
- `sout(msg)`: Prints `msg` to the log file. About the log file see section 9.1. `msg` is not necessarily a string, it can be an arbitrary object or primitive value too. Please note that all `System.out.println()` and `System.err.println()` prints to the log file too.

8 User Tools

The User Tools function of MEME allows the user to execute external programs with data coming from our application.

For example, you can pass data of a view table to an external program as a command line argument. In this case, MEME first exports the view table into a temporary CSV file and passes the filename of the created file to the external program.

This way it can interoperate with other existing statistical software like R and Matlab.

In the current version of MEME there is no predefined user tool. However in the near future we will create some useful package of user tools to extend the statistical capabilities of the application. On Figure 131 the demo user tool package are shown in the menu which can be downloaded from our website: http://mass.aitia.ai/.

The User Tools facilities can be accessed through the *Tools* menu:

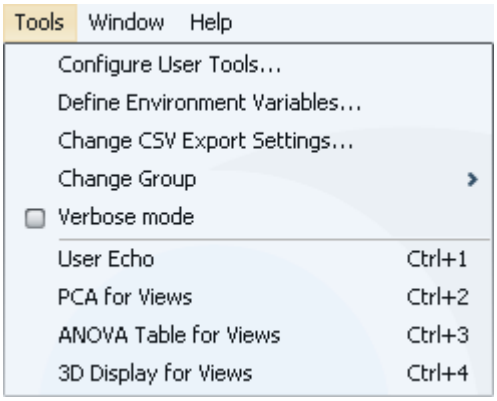


Figure 131 – Tools menu

8.1 Defining Environment Variables

Environment variables are a set of named values that can help you configuring new user tools.

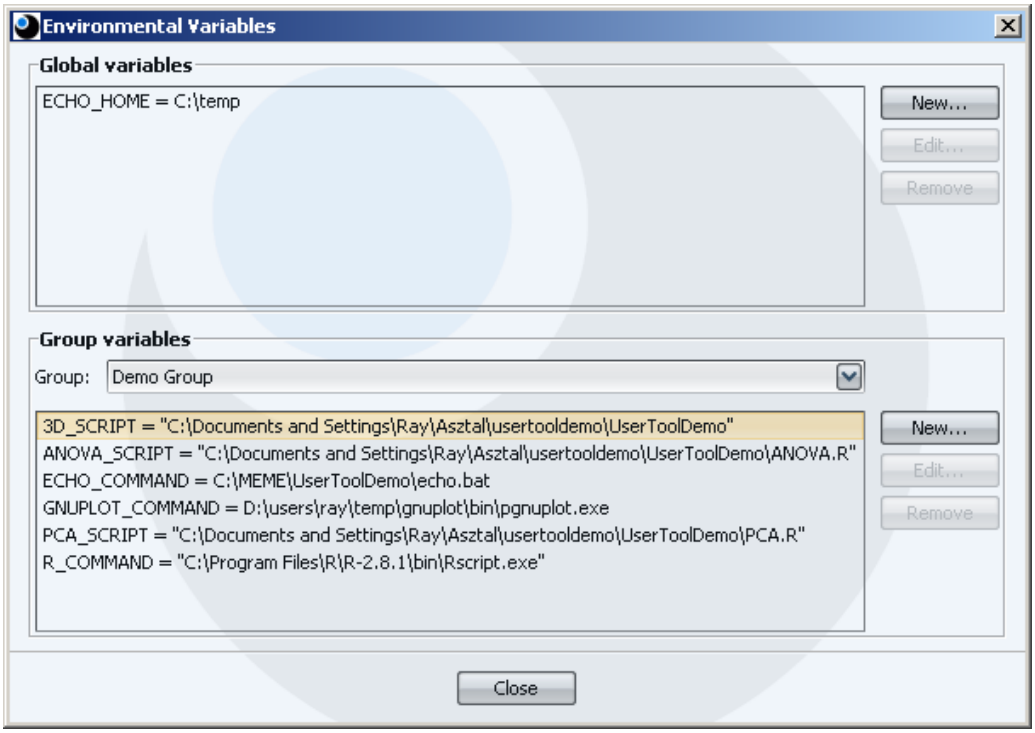


Figure 132 – Environmental Variables dialogue

You may define them if you need to use some kind of information (e.g. path of a file or folder) more than one times in one or more user tools or if you want to create an abbreviation.

To create an environment variable (or edit/remove an existing one) select *Tools/Define Environment Variables…* menu item. The *Environment Variables* dialogue is shown on Figure 132.

As you will see in the next section user tools are classified into groups. According to this fact MEME uses two levels of environment variables: a global level and a group level.

A global environment variable is visible for all user tools while a group variable is visible for those user tools that are the same group than the group variable.

A group variable may have the same name than a global variable but in this case it will mask the global one if it is used in a user tool from the same group.

To create a new environment variable (global or group-level) press the appropriate *New…* button. If you want to define a group-level variable first you have to specify the group in the drop-down list. To edit an existing environment variables select it from the corresponding list then press the *Edit…* button. In both cases a new dialogue appears (Figure 133) where you can define name-value pair.

You can delete a defined environment variable by selecting it from the appropriate list and pressing the *Remove* button.

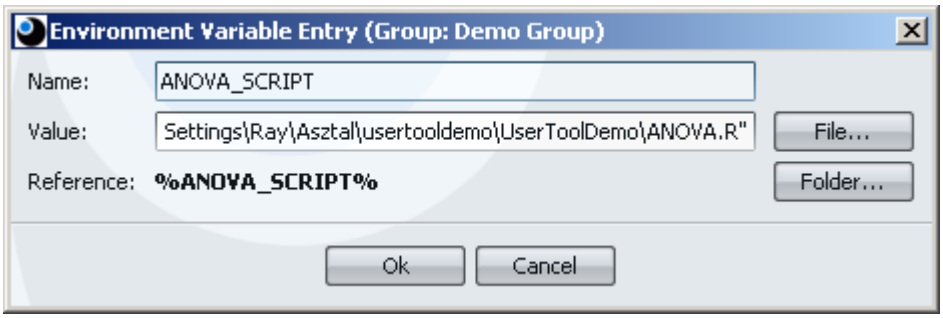


Figure 133 – Creating or editing an environment variable

On the *Environment Variable Entry* dialogue the title contains whether the variable is a global one or a group-level one. In the latter case the name of the group is also appears.

If the name is already used in the group (or in the set of global variables if you define a global one) the program asks for override confirmation. If the value is a file path or a folder path you can use the corresponding button to select the appropriate file/folder.

Please note that on the *Reference* line is shown how you can use the environmental variable in a user tool definition (the name of the variable between percent signs).

8.2 Configuring User Tools

Select *Tools/Configure User Tools…* menu item to create a new user tool (or to edit an existing one). The *User Tools* dialogue appears (see Figure 134).

User tools are classified into groups. There are ten predefined groups in MEME. While creation of new groups are not possible, you can customize the name of existing ones by pressing the *Change name…* button on the *User Tools* dialogue.

The content of a group selected from the *Group* drop-down list appears in the *Tool items* list. The currently selected tool can be removed by pressing the *Remove tool* button. Use *Move up*/*Move down* buttons to change the order of tools.

By pressing the *Define variables…* button the *Environment Variables* dialogue appears (see details in section 8.1 Defining Environment Variables).

The settings of a selected tool appears in the bottom section (*Selected tool*), where they can be edited (see in details below). To create a new tool press the *Add tool* ► button and select a type (*Document* or *Program*) from the popup menu. A new user tool with a default name (menu text) is added to the list.

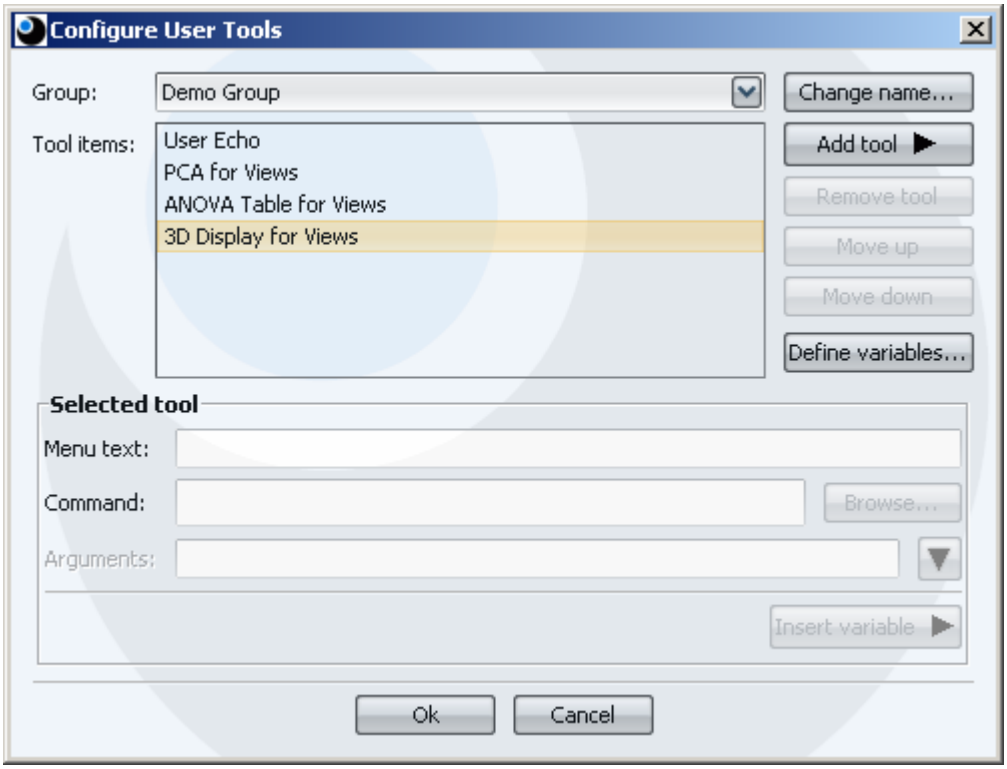


Figure 134 – The User Tool dialogue

Note that at most ten user tools can be added for a group (ensuring a hundred of various user tools from MEME at the same time), exceeding this limit will display a warning message.

8.2.1 Document User Tool

A *Document* user tool is simply a link to an existing document you want to open quickly when you use MEME. To define a *Document* user tool you have to specify a name (that is shown in the *Tools* menu) and the path of the desired document (use the *Browse…* button to select the document file).

You can use defined environment variables in the path of the document either by entering manually the reference of the variable (the variable name between percent signs) or by pressing the *Insert variable* ► button and selecting the appropriate environment variable from the popup menu. Note that this menu enumerates first the environment variables belonging to the selected group and then the global ones.

MEME currently supports only PDF, TXT and HTML formats.

Note that the defined document will only open if an appropriate helper program capable to handle the document is installed on your computer.

8.2.2 Program User Tool

To define a *Program* user tool you have to specify a name (menu text) and the path of the program (you can also use the *Browse…* button here).

You can specify command line arguments in the *Arguments* field (separate arguments with spaces).

You can use defined environment variables in the command and the arguments fields either by entering manually the reference of the variable (the variable name between percent signs) or by pressing the *Insert variable* ► button and selecting the appropriate environment variable from the popup menu. Note that this menu enumerates first the environment variables belonging to the selected group and then the global ones.

If you press the  button next to the *Arguments* field a popup menu appears (see Figure 135) that contains the predefined arguments of MEME.

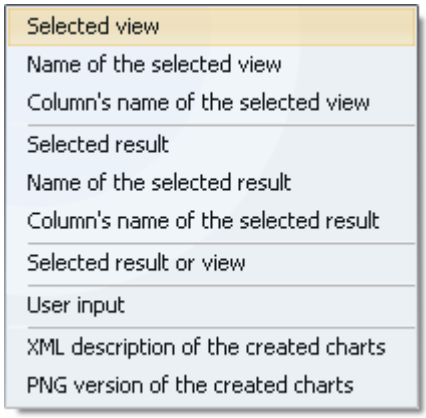


Figure 135 – Arguments menu

If you select an item from the list a meta-argument (like `$View`) is inserted into the *Arguments* field. When you use the tool the meta-arguments is replaced with the actual argument value.

The following list explains the usage of these meta-arguments:

- ***Selected view*** (`$View`): Each selected view table is exported into a temporary CSV file and the meta-argument is replaced by the path of this file. For example, if you select two view tables from the *Available views* list and start a tool which has a `$View` command line argument the application creates a CSV file from both table then executes the tool twice: first with the path of the first CSV file, then with the second one. After that it deletes the files.
- ***Name of the selected view*** (`$ViewName`): The meta-argument is replaced by the name of each selected view table one after the other.
- ***Column's name of the selected view*** (`$ViewCol(n)`): The meta-argument is replaced by the name of the *n*-th column of each selected view one after the other. *n* is the index of the columns (1 is the index of the first column and so on). Note that spaces between *n* and brackets cause warning.
- ***Selected result*** (`$Result`): Same as *Selected view,* however instead of view tables each selected (model - version) pair is exported. For example, if you select a model from the *Available results* tree which contains two versions, the application creates two CVS files and executes the tool twice.
- ***Name of the selected result*** (`$ResultName`): The meta-argument is replaced by the name of each selected (model - version) pair one after the other. The syntax of the name if the following: `<model name>:<version>`.
- ***Column's name of the selected result*** (`$ResultCol(n)`): Same as the *Column's name of the selected view,* however in this case the application uses the columns of the selected (model - version) pairs.
- ***Selected result or view*** (`$ResultOrView`): If the *Results panel* is the active panel when starting the tool this becomes the same as *Selected result* argument. Otherwise it is equal to the *Selected view* argument.
- ***User input*** (`$Input(text)`): The meta-argument is replaced by the user provided text. Before the tool starts an input dialogue (see Figure 136) appears where you can specify the current content of the argument. The `text` parameter is the name of the argument ("My test parameter" in the example) that helps to distinguish the input arguments if you use more than one.

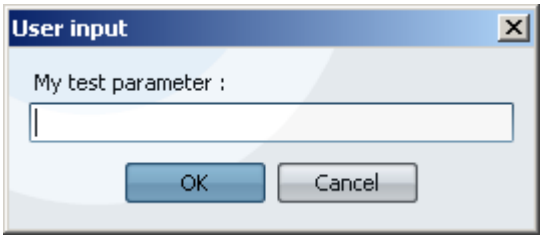


Figure 136 – User input dialogue

- ***XML description of the created charts*** (`$ChartXml`): The configuration of the defined charts is exported into a temporary XML file (see 6.3 The Compose, Display, Save and Cancel Buttons) and the meta-argument is replaced by the path of the file.
- ***PNG version of the created charts*** (`$ChartPng`): Each defined charts is exported into a temporary PNG image file (see 6.10 Charts/Export charts as image…) and the meta-argument is replaced by the path of the created files one after the other.

8.2.2.1 Using Multiple Arguments

You can use more than one meta-argument in the *Arguments* field. In this section examples are shown to explain how MEME interprets multiple arguments.

Let's assume we have an echo program that takes its command line arguments and writes them to the standard output. We specify two arguments: `$ResultName $ViewName`.

Example 1

Selected models: model:1 and model:2 (it is a model that contains two versions)

Selected views: view0

The output is:

model:1 view0

Example 2

Selected models: model:1 and model:2

Selected views: view0 and view1

The user tool runs twice and the output is:

model:1 view0

model:2 view1

By default MEME creates a list for all meta-arguments and then gets the first element from each list and runs the user tool with them. Then it gets the second elements from the lists, and so on. If any of the lists depletes MEME stops.

If you want to force MEME to use parameter combinations instead of lists use meta-arguments with `$$` prefix instead of `$`.

For example, if we use the abovementioned echo program with `$$ResultName $$ViewName` arguments we get the following outputs:

Example 1

model:1 view0

model:2 view0

Example 2

model:1 view0

model:2 view0

model:1 view1

model:2 view1

8.3 Starting User Tools

The created user tool can be started from the *Tools* menu. Each user tool from the actual group is listed there, below the general menu items. You can change the actual group through the *Tools/Change group* submenu.

Programs may write to the standard output. If you start a program of this kind from MEME its output is redirected to the log file of MEME. You can see this log anytime by using the *Help/Show log* menu item. Output lines of user tools are marked with `[Output]` prefix.

8.3.1 Changing CSV Export Settings

User tools may use CSV files that contain data of selected result or view tables. MEME creates these files before it starts the user tool and always uses the actual CSV export settings. You can view or edit these settings by using the *Tools/Change CSV Export Settings…* menu item. The same dialogue appears (see Figure 100) than that of when exporting data manually.

8.3.2 Verbose Mode

If a user tool does not work as you would expect, check this flag before trying again. With the *Verbose Mode* flag checked more information is recorded in the log file about the execution of the user tool, namely:

- Exact command with the actual value of command line arguments (marked with `[Verbose]`);
- error messages (marked with `[Error]`, if any available);
- exit code.

8.4 Predefined User Tool Packages

As it is mentioned earlier MEME is able to use predefined user tool packages.

The installation of a user tool package is very simple; the package contains an XML file that you must copy to the *UserToolPackages* folder which is located in the installation directory of MEME.

MEME will recognize the new user tool package at the next startup. At this time you have to select a user tool group; MEME installs the new tools into this group (see Figure 137). It also changes the name of the group.

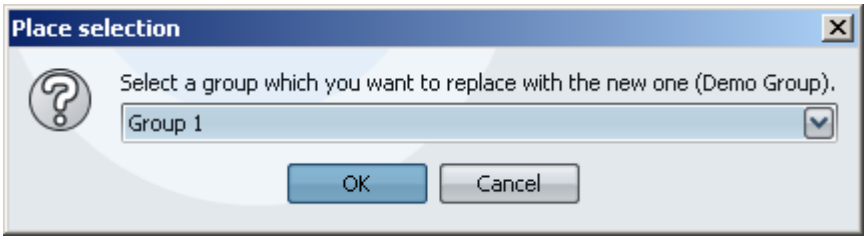


Figure 137 – User tool place selection dialogue

The user tools may use some kind of environment variables (see details about environment variables in section 8.1 Defining Environment Variables). The last step of the installation process is to define the value of the variables required by the predefined user tools.

The user interface of the user tool installer contains a description field where the meaning of each environment variable is shown. This helps you to specify the correct values.

Note that you can change the value of all defined environment variables by selecting the *Define Environment Variables...* item from the *Tools* menu if you made a mistake when you specify them.

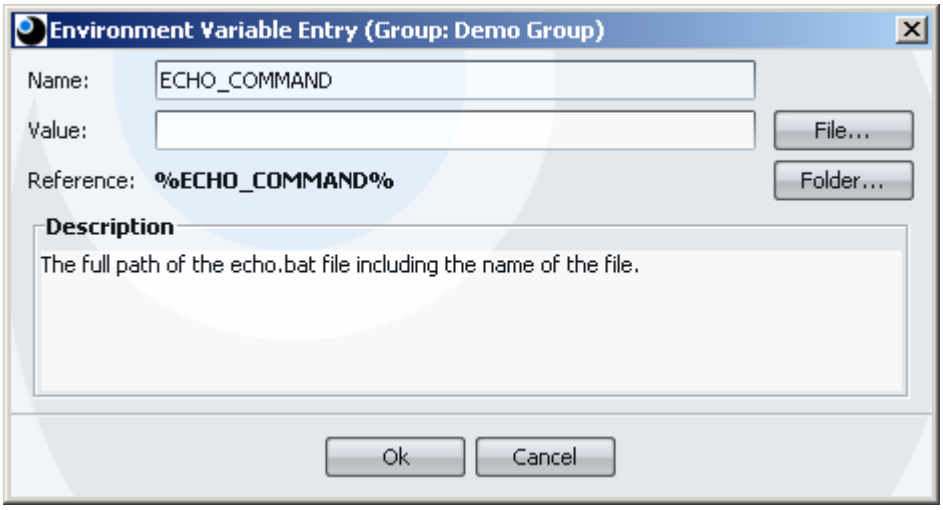


Figure 138 – Defining environment variables for predefined user tools

After every required environment variables are defined the installing is done and the predefined user tools are accessed through the *Tools* menu (you may have to change the actual group to the defined one first).

9 Frequently Asked Questions (FAQ)

9.1 General

Question: Where can I find the log file?

Answer: The log file of MEME (named MEME.log) is in the installation directory[11] of MEME. You can see its content from MEME by using the *Help/Show log* menu item.

Question: I'm using MEME to process huge amount of data and I am experiencing very poor performance. Is there any way to improve the performance?

Answer: MEME uses HSQLDB as built-in database engine which is sufficient for storing and processing data of small or medium experiments. However if you want to more you can change the database engine. MEME supports mySQL, too.

Question: I use MEME on Linux and I can't open the documentation from the application. Neither starting a *Document* user tool. What is the problem?

Answer: MEME uses the default PDF/TXT/HTML viewer application installed on the system. For many windowing systems installing the *libgnome2-0* package would solve the problem (i.e. on a Debian box use the `apt-get install libgnome2-0` command as root).

Question: I have Windows Vista on my computer. When I start MEME it shows the following message immediately: **Error in Thread: MEME-Worker-Thread Null pointer**. After this I can't close the application (only from the Task Manager).

Answer: I assume you install MEME to the default directory (c:\Program Files\MASS\MEME). The problem is that Vista does not allow creating the new database of MEME in a subdirectory of Program Files without proper rights. You have two options to bypass this problem: a) install MEME to another location like c:\MASS\MEME, b) start MEME as Administrator.

***Question*:** I would like to create mass amount of similar charts (e.g. to get an overview of a series of data for each simulation run), but it would be an exhaustive task to make them manually one after another. Are there any alternate ways to perform this task?
***Answer*:** Yes, there is a workaround. MEME is capable of saving a chart descriptor in XML. You can use this as a template to generate other charts. It requires minimal programming skills, however, you can use any preferred programming language. Please refer to the *MEME Tutorial: Generating similar charts from different datasources* for further details.

9.2 Parameter Sweep Wizard

Question: On the page *Data collection* of the wizard I define a stop condition, but I don't add any recorders to my RepastJ model. Will the wizard generate a new model?

Answer: No. Model generation happens only if you add recorders and/or new parameters to the model.

Question: I select a RepastJ batch model (with a hand-coded stop condition) then I define recorders and another stop condition in the wizard. When will the generated model be stopped?

Answer: In this case, both stopping condition are active. Any of them becomes true (the simulation reaches the time step specified), the simulation will be stopped.

Question: I started two long simulations with the wizard on a dedicated cluster. Will the simulations run in parallel? How does the monitor display their progress?

Answer: When you start a simulation with the wizard, the wizard sends a message to the server application about the simulation request after it transfers all relevant files. The

[11] By default this is the c:\Program Files\MASS\MEME\ directory on Windows.

server application stores the simulation requests in a queue and runs the simulations one after the other. The earliest request will be fulfilled the first. The monitor always shows information about the simulation running at the time.

Question: I'd like to run my model on the local host and follow its progress, but the *Start monitor after wizard is closed* option on the *Network* page of the *Preferences* dialogue is checkable only in distributed mode. Why?

Answer: If you run models on the local host, the MEME shows progress automatically with the *Local monitor*. The *Start monitor after wizard is closed* option relates to the distributed mode when you run models on a remote cluster.

Question: When I change something on the *Network* page of the *Preferences* dialogue and then I press the *OK* button the application does not respond for a long time. Why?

Answer: When you press the *OK* button, the application checks your settings including trying to connect to the server computer. Depending on the network settings and traffic this may take some time. The same situation may occur when you change the settings of the monitor.

Question: I ran a RepastJ batch model (with a hand-coded recorder) on the local computer with the Parameter Sweep Wizard. When I try to import the results to the database I get the following error message: **Error while reading file, in line 34: inconsistent input - too many values**. What is the problem?

Answer: This may happen if your computer uses comma as decimal symbol (according to the locale and language settings), because RepastJ uses these settings when it writes out results to the output file. Unfortunately, RepastJ also uses comma as a delimiter by default, so the format of the output file will be invalid. As a workaround, set the locale and language settings to US English and run again your model. Note that if you create the batch model with the *Parameter Sweep Wizard* this problem never occur.

Question: I run a model with the *Parameter Sweep Wizard*. I define one recorder that records at the end of every iterations and writes the recordings into the output file after every recording. My experience is that the simulation runs very slowly. Why?

Answer: This means the model writes a file at the end of every iterations and writing a file is a slow operation. Leave the write time to "At the end of the runs" next time you run a model.

Question: I try to run a model with MEME. After I press the *Finish* button, an error dialogue named 'MEME Errors' appears with a message such as **in Thread: Simulation-Thread <short description of the error>**. What can I do?

Answer: In most cases, when you see an error in the Simulation Thread it is caused by the model not MEME. See the log file for detailed error message. Try to start your model without MEME. If it runs properly in standalone mode, the most likely problem is that you use a resource file in your model referencing it with its relative path. When MEME runs your model the relative paths is resolved against the installation directory of MEME. Changing the location of the resource files may solve your problem.

Question: I create and run a RepastJ experiment with MEME. When I try to load the wizard settings to the MEME using the *Load wizard settings…* button the following error message appears: **Error while loading settings file. Reason: ai.aitia.meme.paramsweep. utils.ParameterParserException: Illegally formatted parameter file at line: <a line number> Expected '}'**. What is the problem?

Answer: This problem is occurred when you have a string parameter in the experiment and its value contains '-' character. This is a known issue in the parameter file parser of the RepastJ. Eliminating the '-' characters in the wizard settings XML file may solve the problem.

Question: I try to run my RepastJ model with MEME, but the model throws an exception. Running model in GUI mode works fine. What is the problem?

Answer: Please make sure that all variables using parameter values is initialized at `begin()` method of your model instead of the `setup()` method because actual parameter values are set after `setup()` invoking.

Question: I try to run my pure Java model with MEME, but the model throws an exception. Running model in GUI mode works fine. What is the problem?

Answer: Please make sure that all variables using parameter values is initialized at `simulationStart()` method of your model instead of the constructor because actual parameter values are set after constructor invoking.

9.3 View Creation Wizard

Question: In the result table I have a column that contains real values with too high precision (the number of the digits after the decimal sign is too large). How can I round these values to a lower precision?

Answer: On the *View Creation Wizard* check the *Custom expression or script* checkbox then write the following line to the text area below the checkbox:

```
round(name,precision);
```

where `name` is the name of the above-mentioned column and precision is the maximum number of digits after the decimal sign. After that you should rename the script at the *Column name* text field. Press *Add* button to add the rounded column to the view.

Question: I create and run my model with the *Parameter Sweep Wizard* and selected a method (named `max()`) as a recordable element. Now, I want to create a Beanshell script and use `max()` in its body. But I can not. The program always shows an error message whenever I write `max()`. How can I reference `max()` from a Beanshell script?

Answer: Because of the brackets `max()` is not a valid identifier in Beanshell so MEME can not create a global parameter from `max()`. You can, however, reference it with the predefined Beanshell method `get("name")`: use `get("max()")` instead of `max()`. See in details in section 7.4 Beanshell Scripting.

***Question*:** I would like to create a view that contains recorded values of a variable in each simulation tick, ordered into different columns per runs. How can I do it?
***Answer*:** Create two *Custom expression script* columns, the first with the value `$Run$`, and the second with the value `$Tick$`. Mark the `$Run$` column as *Splitter* to aggregate results (it also sets the column *Hidden* by default). Add the required variable as a third column, mark it as *One column per splitter value*, and the resulting view will contain the values of the appropriate variable each simulation tick, ordered into different columns per runs.

9.4 User Tools

Question: I want to use an R script as user tool in MEME on Windows. The user tool has one parameter: the selected view or result. I try to receive the CSV file that contains the data of the selected view or result with the following line in the R script: `read.csv(commandArgs()[1])`. The problem is that the R does not find the file. What can I do?

Answer: You have to use `commandArgs()[6]` instead of `commandArgs()[1]` in your script if you want to access the first parameter of the user tool. The explanation is the following: when you call `RScript.exe <script-file> <user-param1>`, the `RScript.exe` calls the `RTerm.exe` with some switches and passes its own parameter to the `RTerm.exe`. For example:

```
RTerm.exe --slave --no-restore --file=<script-file> --args <user-param1>
```

Because the `commandArgs()` array contains the program name (`RTerm.exe`) too and the indices start with 1, the `<user-param1>` is the 6th in the array.

10 Known Issues

10.1 MEME does not respond

In some situation MEME executes long operation without displaying a progress window first. It seems MEME hangs, but it does work, there is just no feedback.

The abovementioned situations are the following:

- Pressing *OK* button on the *Preferences* dialogue after changing something on the *Network* page.
- Pressing *OK* button on the *Monitor configuration* dialogue after changing some settings.
- Pressing *Import to database…* button on *Finished simulations* tab of the *Monitor panel*.

11 Troubleshooting

11.1 Reporting Errors

MEME is under rapid development, therefore any feedback about errors and requests is appreciated, and, on the other hand, new fixed versions are made available regularly. If you have experienced a software error or have suggestions about features, please send it in an e-mail to one of the following address:

mailto:mass@aitia.ai

mailto:mass-support-request@lists.mass.aitia.ai

If you send an error report, please include the `MEME.log` file from the installation directory of MEME, and also describe the steps that you have taken in the program before the error occurred.

11.2 Memory Usage

Maximum heap size reserved for database processing is set to be 512 MB by default in MEME. Note that this is not the maximal memory usage of the program, with the default settings that can go up to about 600 MB. When processing tables with 200,000+ data rows the default heap size might turn out to be insufficient. If your computer has 1 GB of RAM or more feel free to modify the maximum heap size for faster database processing by right clicking the MEME icon, selecting *Properties* from the pop-up and changing the *Target* line from "`javaw.exe –version:1.6+ -Xmx512 -Djava.library.path=./lib/j3d –jar MEME.jar`" to "`javaw.exe –version:1.6+ -Xmx<desired amount> -Djava.library.path=./lib/j3d –jar MEME.jar`".

Note that what mentioned above is valid only for the case of built-in database usage. Working with an external database server the default memory is more than enough.

11.3 Help Menu

This manual and the MEME Tutorial are accessible from the *Help* menu. Note that these PDF documents will only open if Acrobat Reader or other PDF-capable program is installed on your computer. To download Acrobat Reader go to:

http://www.adobe.com/products/acrobat/readstep2.html

11.4 Progress Window

The time left in the *Progress window* is an over estimation, it usually starts out with a higher value than the processing will eventually take unless there is fluctuation in the size of memory and CPU time required by other applications hence available for MEME. In this case the estimation is not dependable; it can jump up and down from time to time on the given conditions.

12 Summary

MEME is a tool for dealing with batch runs of simulations and the data produced. It allows the user to store and organize the raw data in database(s), and to create distillated (computed) tables which can be visualized on charts.

MEME can also run FABLES/RepastJ/NetLogo 4.1.1 simulations directly and can collect their results, moreover it interoperates with other existing statistical softwares, like R and Matlab, via CSV exporting and external program execution facilities.

Some plans for improving MEME in the near future includes the followings:

- Implementing most effective intelligent parameter space exploration strategies.
- Support for other simulation platforms such as Repast Simphony, MASON etc.

13 References

[1] MASS – Multi-Agent Simulation Suite. © AITIA International Inc.
http://www.aitia.ai/services_and_products/simulation_systems/mass
[2] Repast - Recursive Porus Agent Simulation Toolkit
http://repast.sourceforge.net/
[3] ProActive
http://proactive.inria.fr/
[4] R - The R Project for Statistical Computing
http://www.r-project.org/
[5] Matlab
http://www.mathworks.com/
[6] Beanshell Scripting
http://www.beanshell.org/
[7] Colt
http://dsd.lbl.gov/~hoschek/colt/
[8] MySQL
http://www.mysql.com/
[9] Java Regular Expressions
http://java.sun.com/docs/books/tutorial/essential/regex/
[10] Space-filling designs
http://www.spacefillingdesigns.nl/
[11] NetLogo
http://ccl.northwestern.edu/netlogo/