



## User Manual

Prepared by  
Róbert Mészáros, Rajmund Bocsi,  
Márton Iványi and Viktor Erdélyi



**March 2008, Budapest**

# Contents

Introduction	4
1 Before Startup	5
1.1 Installation	5
1.2 System requirements	5
1.2.1 Hardware	5
1.2.2 Software	5
2 Storing and Organizing Simulation Results	6
2.1 Concept: Model name and version	6
2.2 Concept: Input and output parameters	6
2.3 File/Database settings	6
2.4 Heap size	7
2.5 File import	8
2.5.1 Repast import	8
2.5.2 CSV import	10
2.6 File export	12
2.7 The Results browser	13
2.8 Delete results	14
3 Panels (the Window menu)	15
3.1 Layout	15
3.2 Skins, themes, watermarks	15
3.3 Local menus	16
4 Views	17
4.1 Concept: View table	17
4.2 Create view	17
4.2.1 Step 1 - Computation	18
4.2.2 Step 2 - Sorting	19
4.2.3 Step 3 - Name and description	20
4.2.4 The result	20
4.2.5 Step 0 – Base Data	21
4.3 Views/Delete	22
4.4 Views/Recreate	22
5 Charts	24
5.1 Charts/Create chart...	24
5.2 Concept: Data sources	25
5.3 The Compose, Display, Save and Cancel buttons	25
5.4 Magnifying the figure	26
5.5 Saving the figure	26
5.6 Properties	26
5.7 The Details tab	27
5.8 Charts/Open chart...	27
5.9 Chart types	27
5.9.1 Bar Chart	27
5.9.2 2D Grid	28

---

5.9.3	Shape grid	29
5.9.4	Composite grid	30
5.9.5	Histogram	30
5.9.6	Network	30
5.9.7	Rectangle Area Chart	30
5.9.8	Scatter plot	31
5.9.9	Sequence	31
5.9.10	Time Series	31
5.9.11	XY Line Chart	31
6	Scripting	32
6.1	Saving view scripts	32
6.2	Loading view scripts	32
6.3	Editing view tables	32
6.3.1	Example code	32
6.3.2	Notes	33
6.4	Beanshell Scripting	33
6.4.1	Predefined methods	34
7	Known Issues	36
7.1	MASS/MEME Parameter Sweep Wizard & Monitor	36
8	Troubleshooting	37
8.1	Reporting errors	37
8.2	Memory usage	37
8.3	Help menu	37
8.4	Progress window	37
9	Summary	38
10	References	39

# Introduction

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 MASS or Repast, 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 distilled (computed) tables which can be visualized on charts. In the near future MEME is expected to be able to run MASS/Repast simulations directly and collect their results internally, and on the other hand, to interoperate with other existing statistical software like R and Matlab. These developments will further simplify the modeling work and allow the users to focus on the models themselves.

The current version of MEME is a demo version, which offers limited functionality only. The following chapters explain the usage of these functions.

# 1 Before Startup

## 1.1 Installation

MEME is available through an installation exe file (the demo version can be downloaded from <http://www.aitia.ai/>). The installation wizard guides you through the installation process, checking for already installed versions of MEME and the appropriate Java JRE (1.5 or later) - installing it if necessary -, offers the installation of the Repast 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.

## 1.2 System requirements

### 1.2.1 Hardware

Minimum requirements: 1 GHz CPU, 512 MB RAM

Recommended: Dual-core CPU, 1 GB RAM

### 1.2.2 Software

Any system that runs Java JRE 1.5 is able to run MEME. The detailed requirements for each operating system are available at:

<http://java.com/en/download/help/5000011000.xml>

For now the demo version installer is only available for Windows 98/NT/2000/XP OS but MEME runs on all systems capable of running Java.

## 2 Storing and Organizing Simulation Results

In this document, “simulation result” means a bunch of raw data that was produced by a single simulation (a single run of a model).

### 2.1 Concept: Model name and version

As illustrated in 2.7 *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 arbitrary<sup>1</sup> names entered by the user. These are the first two levels of the hierarchy. The third level organizes the batches, because 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.

### 2.2 Concept: Input and output parameters

Every simulation result is comprised 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 those 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 parameters. 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.

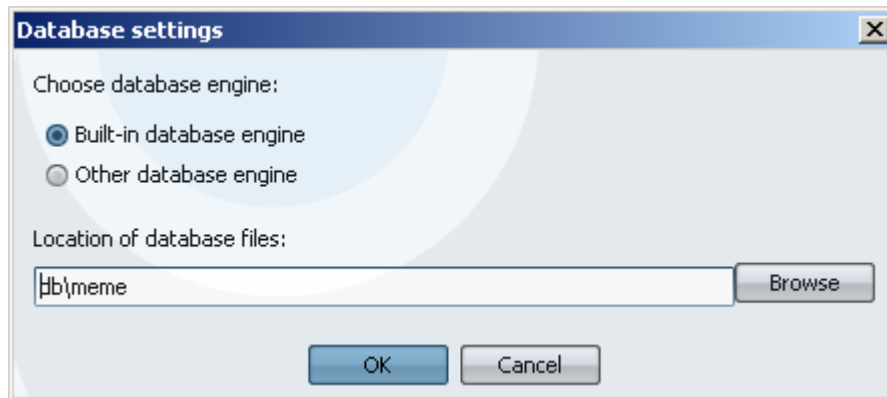
### 2.3 File/Database settings

MEME uses a database to store the simulation results. This must be a database managed by an SQL database server, to which MEME connects when started. By default MEME uses a built-in SQL database server which stores the data in a directory of the local file system. However, this database engine is not as powerful as many professional DMBS software, and the size of data with the built-in engine is limited to 8GB’s. Above that an error message is generated and the application stops working. Therefore, MEME supports professional database engines through the JDBC protocol.

---

<sup>1</sup> Model names must be unique and max. 64 characters long. Similarly, version names must be unique within the model, and max. 64 characters long.

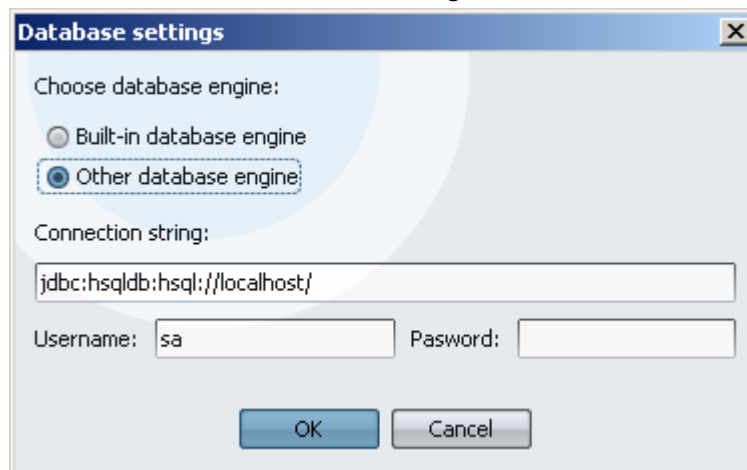
The *Database settings* menu item in the *File* menu allows you to select which database is to be used:



**Figure 1**

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 current directory. 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 2**

You can use this window to change 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 a database, you cannot move it into another using MEME.

## 2.4 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 256 MB's for database processing. If this turns out to be insufficient it can be changed, see 8.2 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.

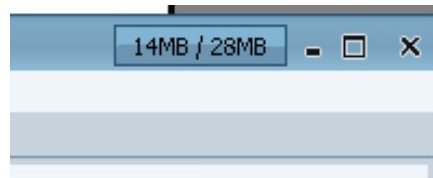


Figure 3 - Heap panel

## 2.5 File import

At the beginning, your simulation results are stored outside of MEME's database. MEME can import these recordings into the database if they are stored in a format which is known by MEME. The *Import...* menu item in the *File* menu lists the supported formats. The following sections will describe the import settings of these formats.

### 2.5.1 Repast import

Repast 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... / Repast 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 dialog, and loads it. The program automatically detects the delimiter string and the list of parameters and the types of their values. It also tries to detect which parameters are input and which are output, but it is not always possible, therefore displays it in a table what it have found. (See Figure 5!)

In order to import multiple repast 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 declaration (can be both numeric or textual) in order to have each file imported to the database as a new batch, or select the *Every file a new version* option. In the latter case % appears as version number representing the automatic numbering (0, 1, 2,..., n). When importing multiple files the user doesn't have the freedom to declare whether a parameter is input or output. (See Figure 4!)

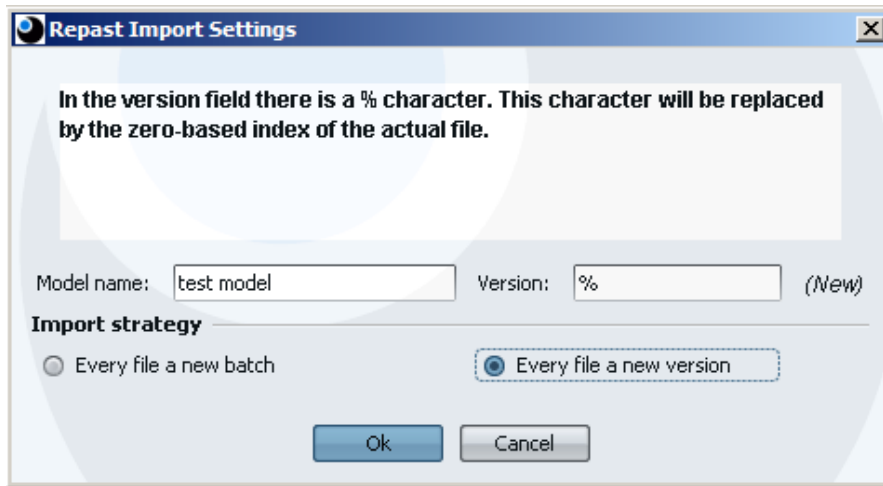


Figure 4

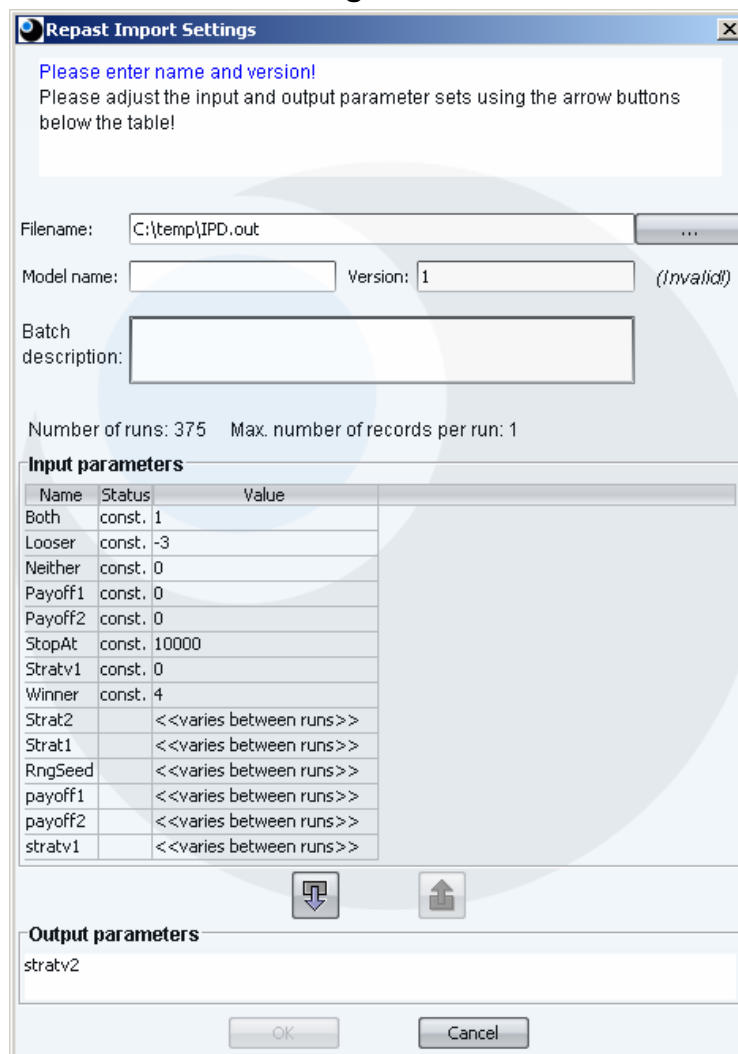


Figure 5

Use the down/up arrow buttons between the *Output parameters* and *Input parameters* boxes to adjust which parameters are input and which are output. 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 dialog 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 a description field in

which you can store longer information. The named model and/or version will be created if it doesn't exist yet. Otherwise the data will be added into a new batch.

If the named model and version exist this fact is displayed in the label next to the *Version* field. If the parameters are different from the existing parameters new or missing parameters are indicated in the table and in the *Output parameters* list (see also *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 a wider type as necessary so as to accommodate to the new data. (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 properly.

## 2.5.2 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: delimiter, 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 6!)

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

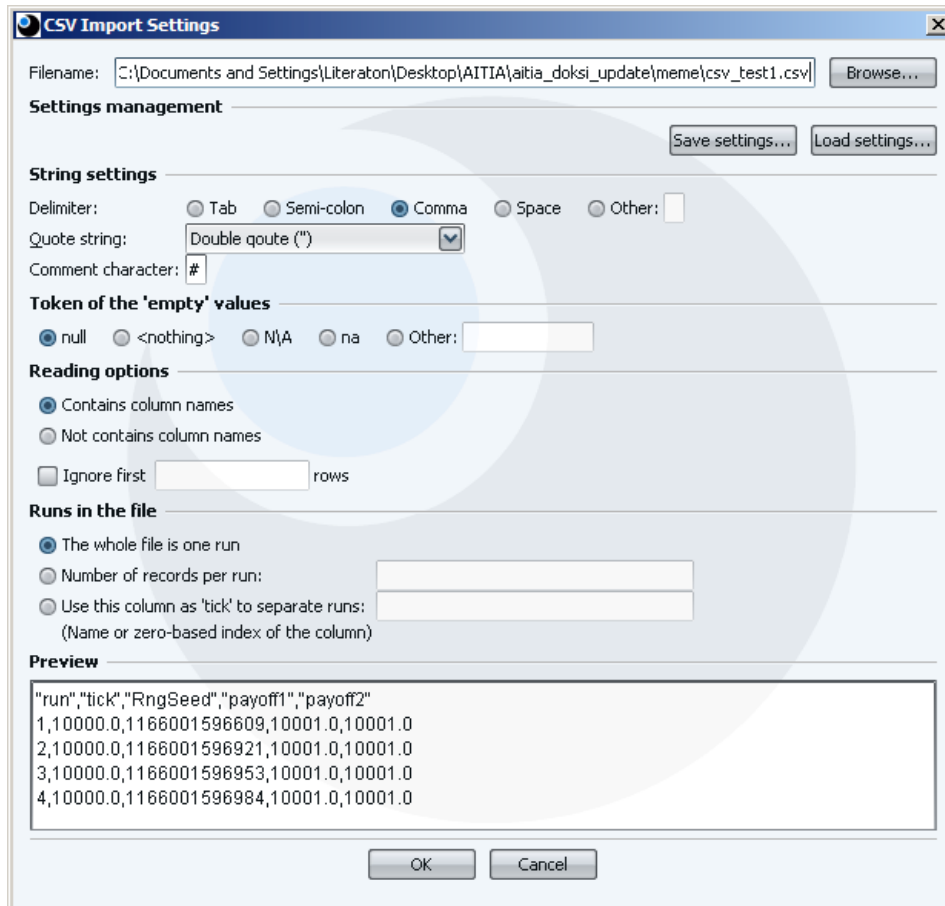


Figure 6

By pressing OK the CSV import dialogue switches to model, version and batch settings already introduced in 2.5.1 Repast import (see Figure 7). The only difference is that here parameters detected as Input can be modified to Output 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 Repast result files.

Model name:  Version:  (Invalid!)

Batch description:

Number of runs: 1 Max. number of records per run: 6

**Parameters**

Name	Type
run	Output
tick	Input
RngSeed	Output
payoff1	Input
payoff2	Input

OK Cancel

Figure 7

## 2.6 File export

View tables (see 4 Views), organized data from MEME's database, and versions (see 2.1 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 8).

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 the same settings and by providing an export directory using the default file naming rules.

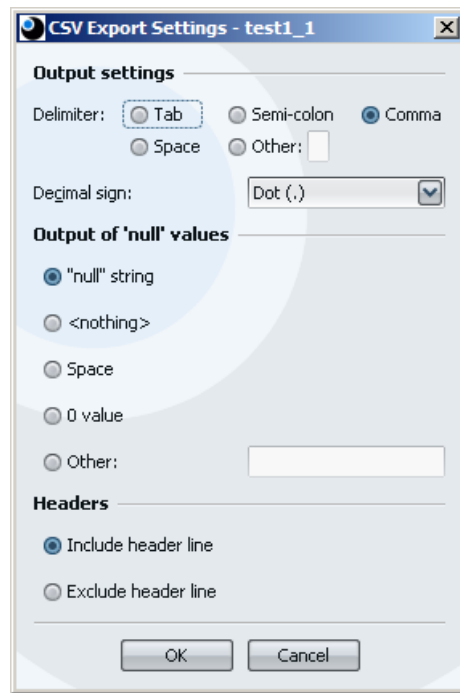


Figure 8

## 2.7 The Results browser

Once you have stored simulation results in the database you can browse them in the main window of the application:

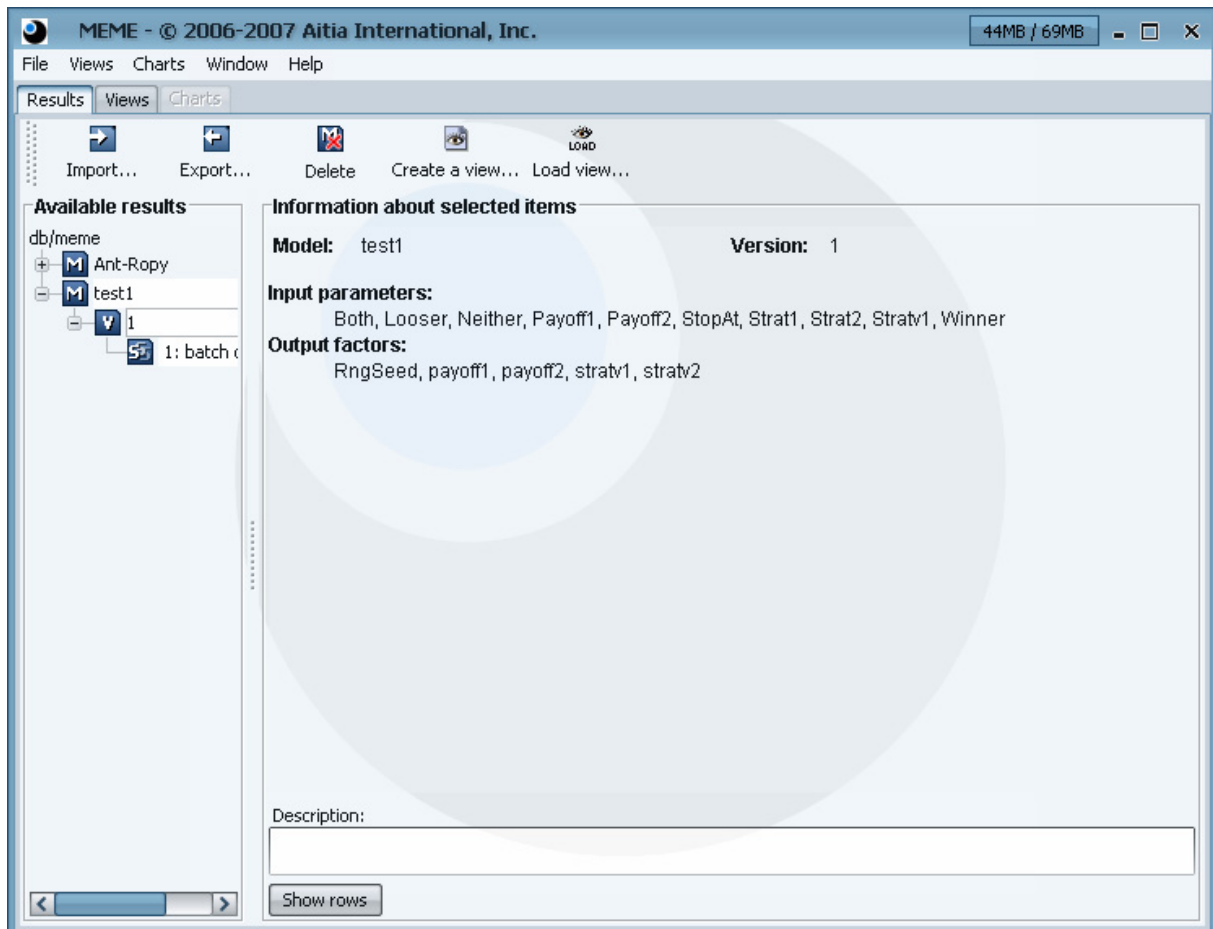

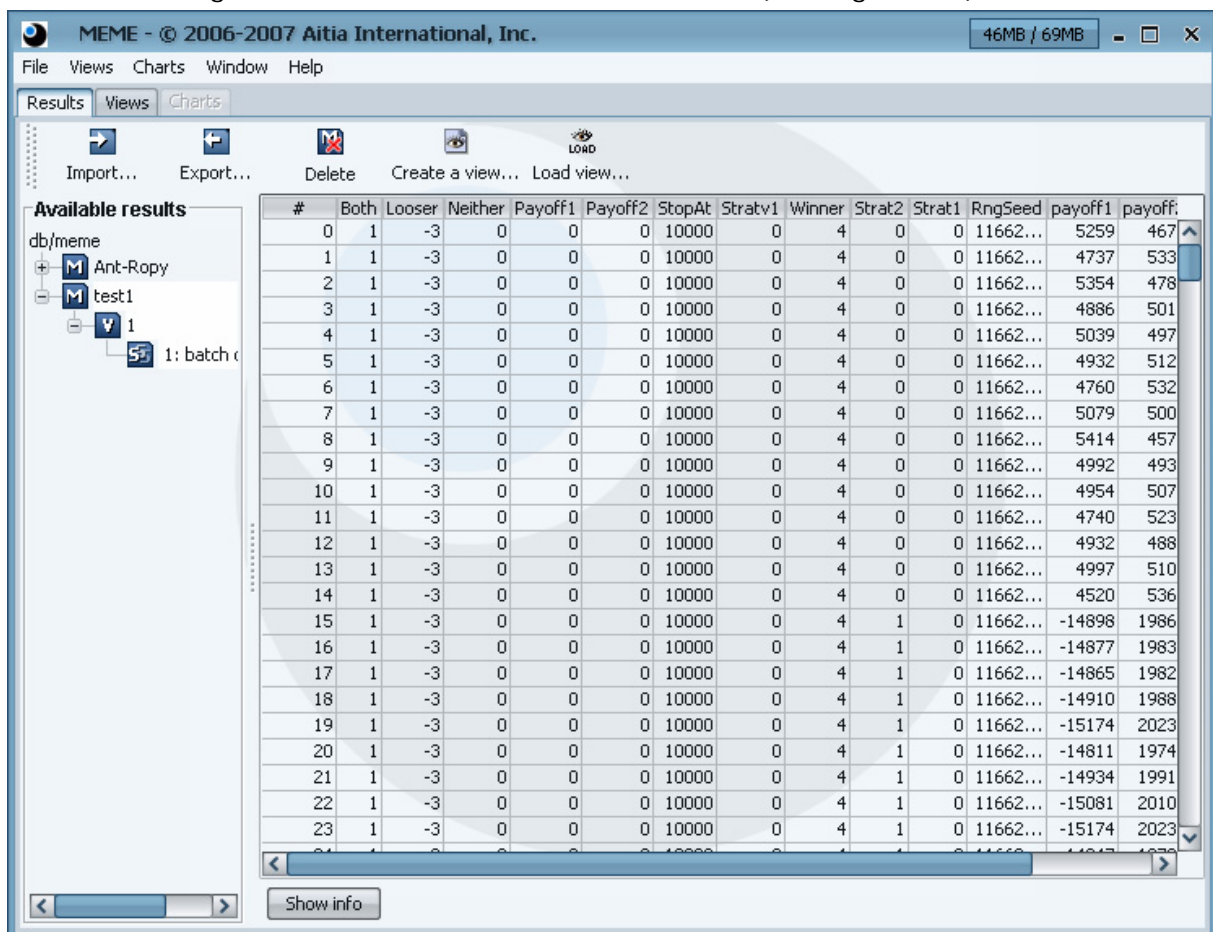


Figure 9

The results hierarchy is shown at the left side of the window in a 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 selected and information is displayed about it 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 of 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 a *view table* (see 4.1 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 10!)




#	Both	Looser	Neither	Payoff1	Payoff2	StopAt	Stratv1	Winner	Strat2	Strat1	RngSeed	payoff1	payoff2
0	1	-3	0	0	0	10000	0	4	0	0	11662...	5259	467
1	1	-3	0	0	0	10000	0	4	0	0	11662...	4737	533
2	1	-3	0	0	0	10000	0	4	0	0	11662...	5354	478
3	1	-3	0	0	0	10000	0	4	0	0	11662...	4886	501
4	1	-3	0	0	0	10000	0	4	0	0	11662...	5039	497
5	1	-3	0	0	0	10000	0	4	0	0	11662...	4932	512
6	1	-3	0	0	0	10000	0	4	0	0	11662...	4760	532
7	1	-3	0	0	0	10000	0	4	0	0	11662...	5079	500
8	1	-3	0	0	0	10000	0	4	0	0	11662...	5414	457
9	1	-3	0	0	0	10000	0	4	0	0	11662...	4992	493
10	1	-3	0	0	0	10000	0	4	0	0	11662...	4954	507
11	1	-3	0	0	0	10000	0	4	0	0	11662...	4740	523
12	1	-3	0	0	0	10000	0	4	0	0	11662...	4932	488
13	1	-3	0	0	0	10000	0	4	0	0	11662...	4997	510
14	1	-3	0	0	0	10000	0	4	0	0	11662...	4520	536
15	1	-3	0	0	0	10000	0	4	1	0	11662...	-14898	1986
16	1	-3	0	0	0	10000	0	4	1	0	11662...	-14877	1983
17	1	-3	0	0	0	10000	0	4	1	0	11662...	-14865	1982
18	1	-3	0	0	0	10000	0	4	1	0	11662...	-14910	1988
19	1	-3	0	0	0	10000	0	4	1	0	11662...	-15174	2023
20	1	-3	0	0	0	10000	0	4	1	0	11662...	-14811	1974
21	1	-3	0	0	0	10000	0	4	1	0	11662...	-14934	1991
22	1	-3	0	0	0	10000	0	4	1	0	11662...	-15081	2010
23	1	-3	0	0	0	10000	0	4	1	0	11662...	-15174	2023

Figure 10

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 of the dialog automatically (you can modify it).

## 2.8 Delete results

You can 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.

## 3 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 11 - Panels

The first one called *Results panel* displays the abovementioned Results browser. In this panel you can work with the stored simulation results. The next is called *Views panel* in which you can work with view tables that are described in 4 Views. The *Charts panel* is only displayable when you are editing a chart.

### 3.1 Layout

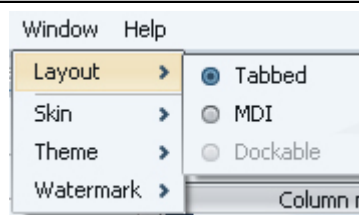


Figure 12 - Window types

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

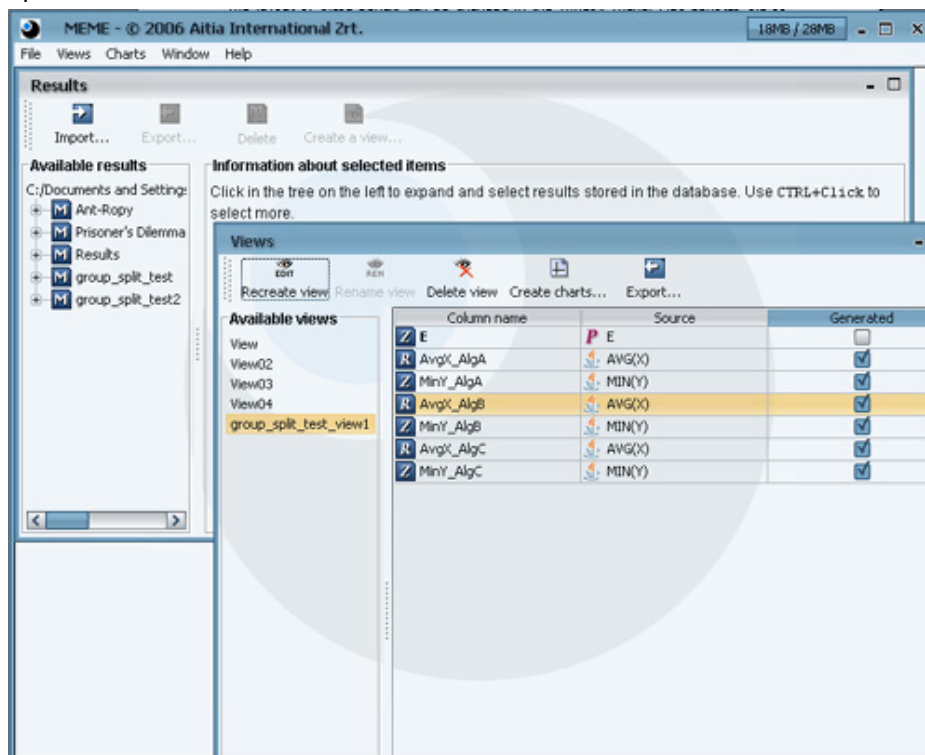


Figure 13 - MDI

### 3.2 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!

### **3.3 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 17 - Views for example.

## 4 Views


### 4.1 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 models and versions — from which the view table is created, and the set of columns that are to be included in the view table. You can also specify computations and filtering (see 4.2.1 Step 1 - Computation).

View tables are designed for chart creation: charts can be created from views only, not from results. This demo version of MEME also allows exporting the contents of a view table in CSV format (see 2.6 File export).

### 4.2 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 view...* menu item or press the  button in the *Result Panel*. This will start the *Create View Wizard*:

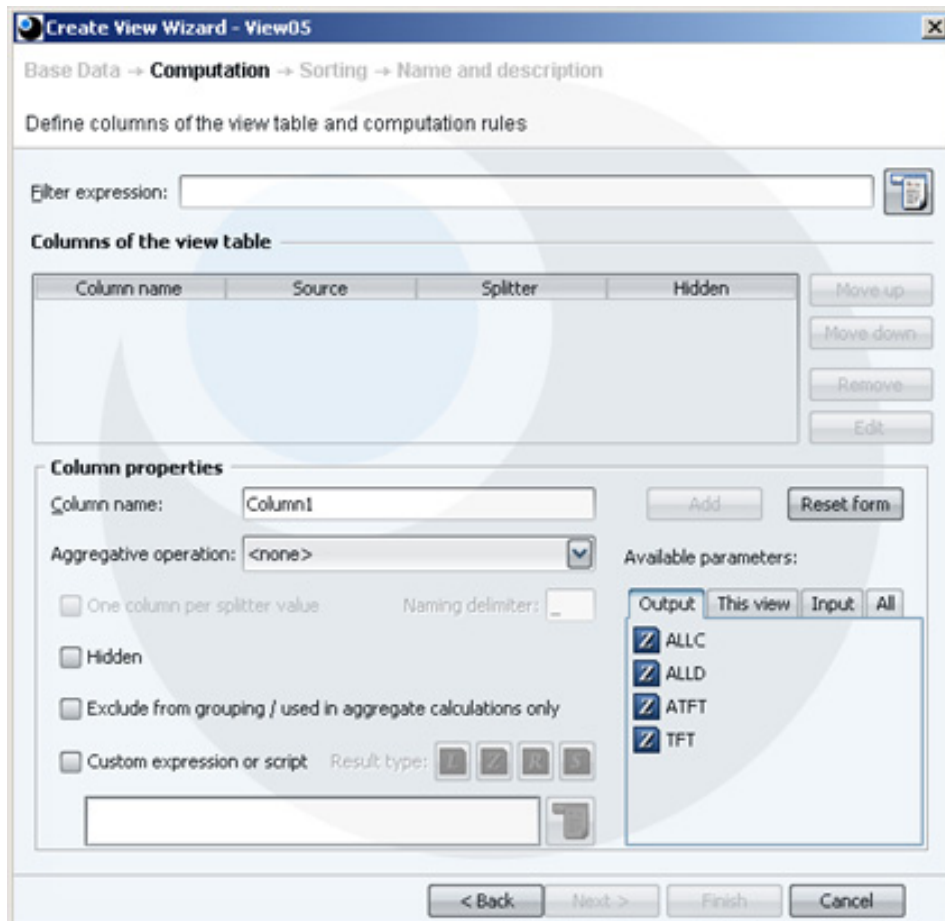






Figure 14 – Computation

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

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

### 4.2.1 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 operations (Min, Max, Average, Sum, Count) 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 `$Run$`. If the base data is from a view table (see 4.2.5 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 sense), and cannot exceed 64 characters.

By pressing the *Add* button 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*.

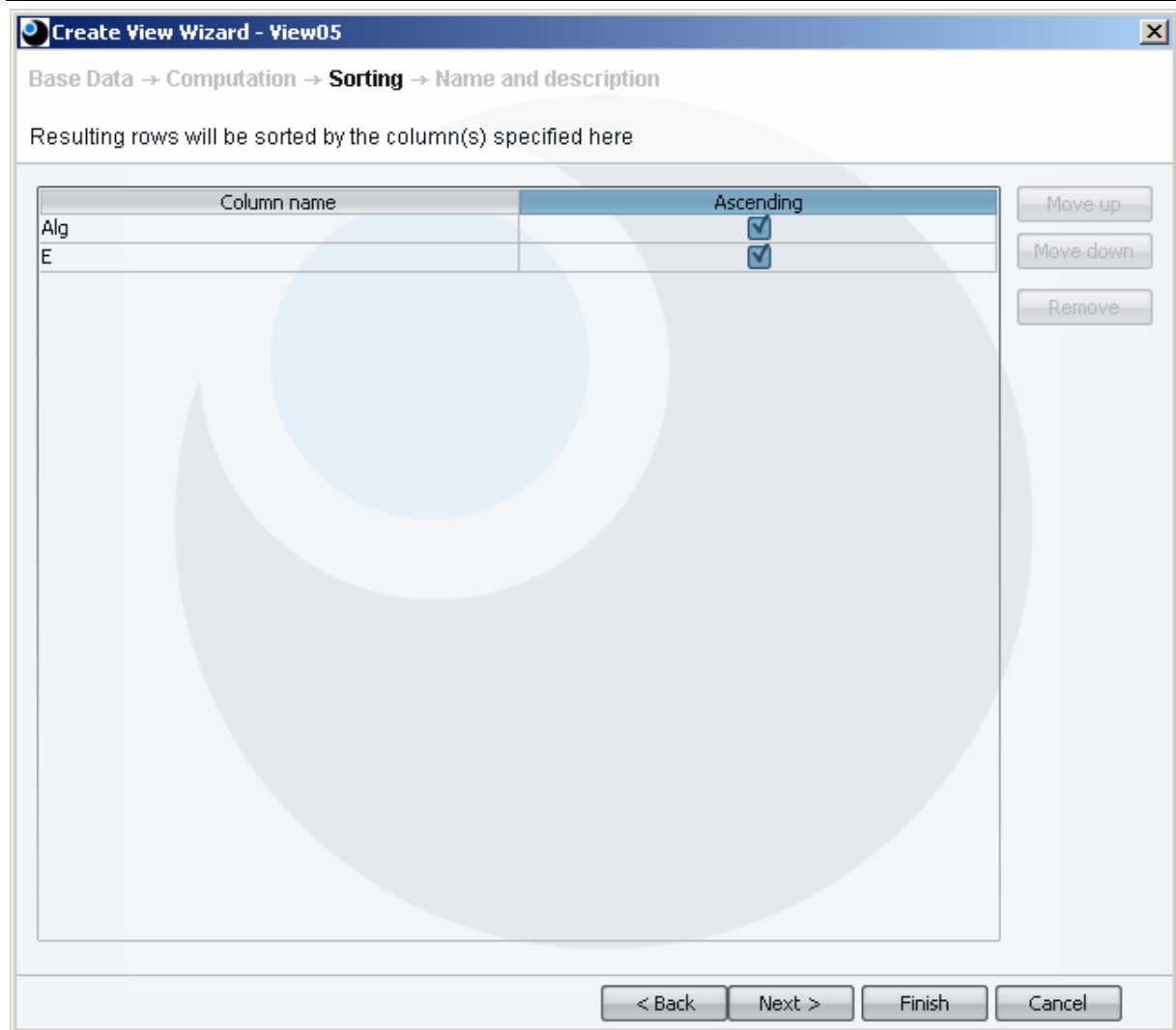
By pressing the *Edit* button or by double-clicking on its name the selected (already added) column can be renamed, operations can be done on it, can be selected to be grouped or to be the splitter. When done with the editing press *Modify* to save the changes, or *Cancel* to undo them. Use the Move 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. Move and remove buttons are inactive while editing columns.

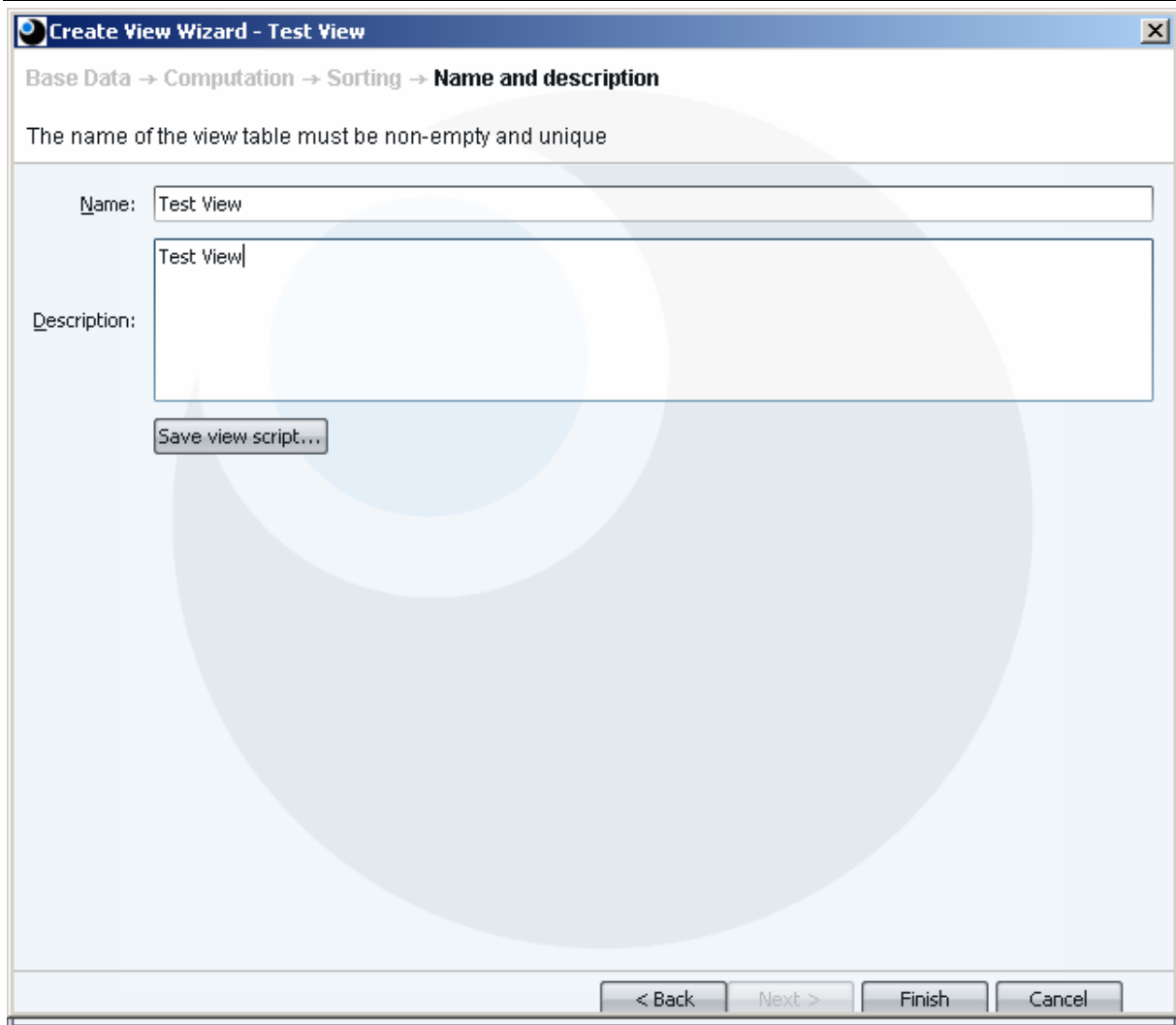
## 4.2.2 Step 2 - Sorting



**Figure 15 - Sorting**

On the *Sorting* screen the layout of the view table can be set by altering the order of 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.

### 4.2.3 Step 3 - Name and description



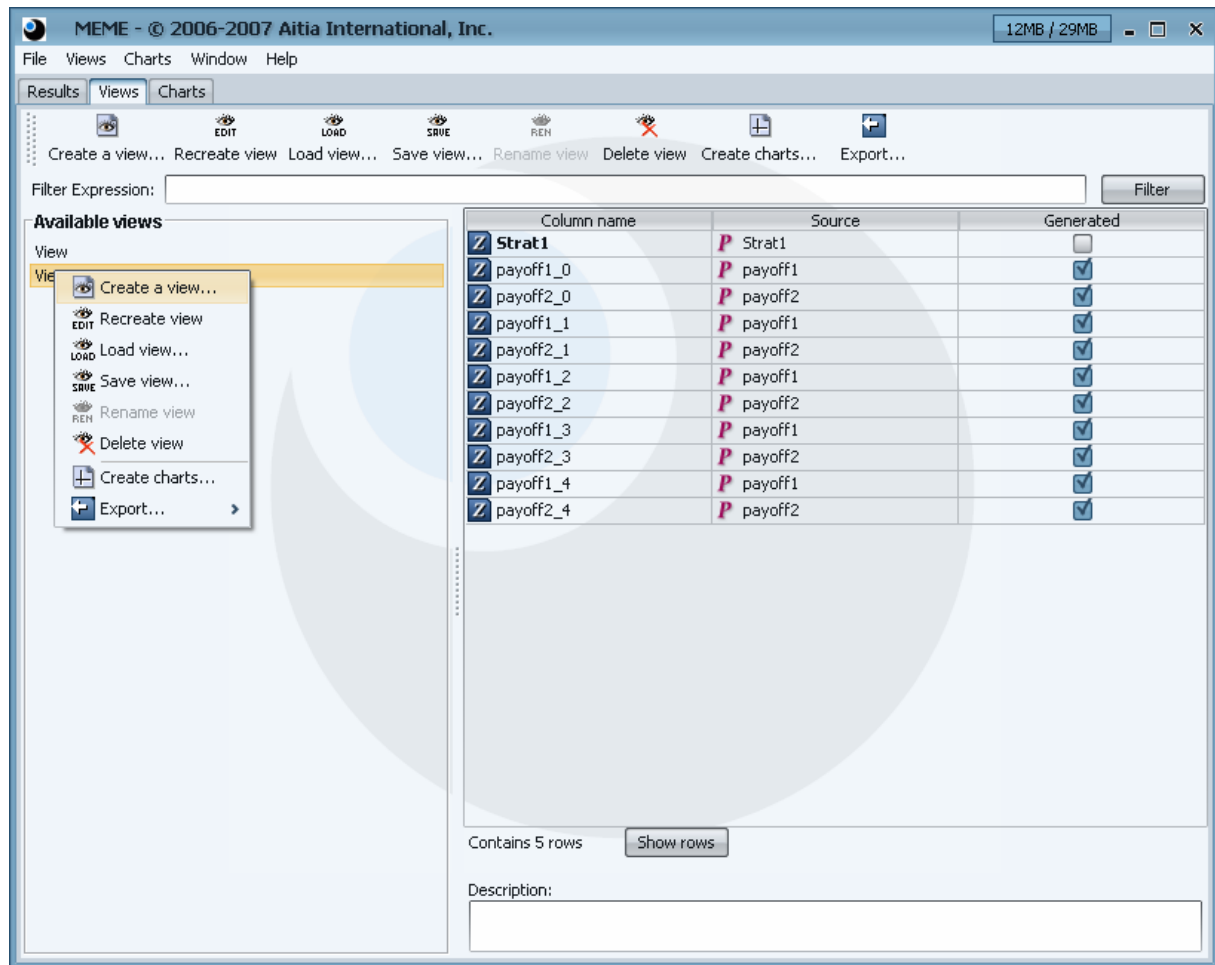
**Figure 16 - Name and description**

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 file describing the view can be saved to a separate file. For further information see 6 Scripting.

### 4.2.4 The result

Upon pressing the *Finish* button the Wizard is closed and the new view table is listed in 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 stays open in order to enable the user eliminate the problem. In this case if *Cancel* is pressed an empty view table is created but containing all the settings. This lets the user exit the wizard for solving the problem and opening the wizard again for finalization.



**Figure 17 - 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 5 Charts) or it can be exported to be analyzed and visualized in other programs (2.6 File export).

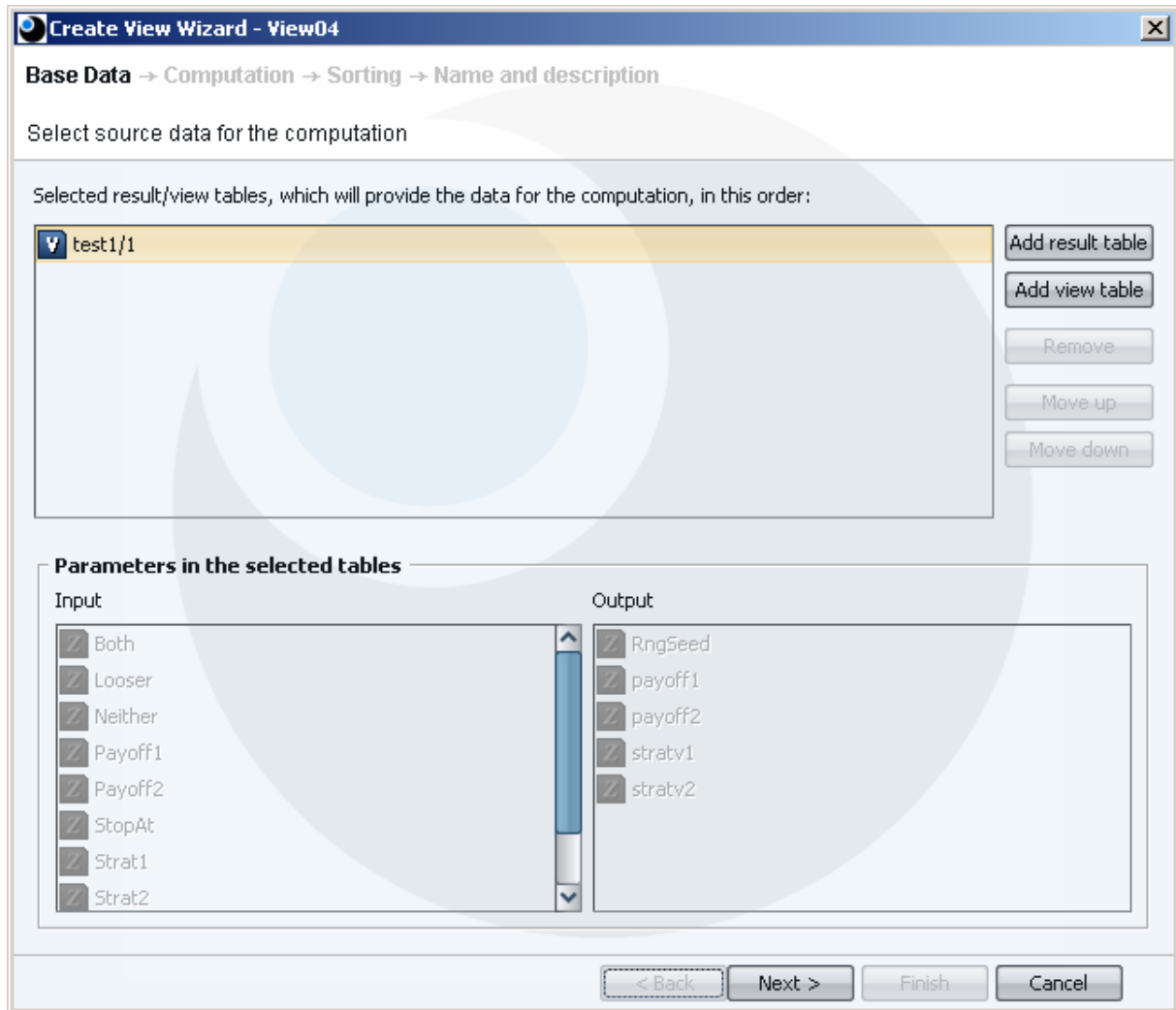
In order to delete, save or export multiple view tables hold down the *CTRL* key and click on the views to be selected. Selected views are highlighted and can be deleted, saved or exported at once. When multiple views are selected the data from the latest selection is displayed on the data field. Accordingly,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

#### 4.2.5 Step 0 – Base Data

The Create View Wizard starts on its second page, as 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 18 - Base Data**

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

### 4.3 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.

### 4.4 Views/Recreate

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

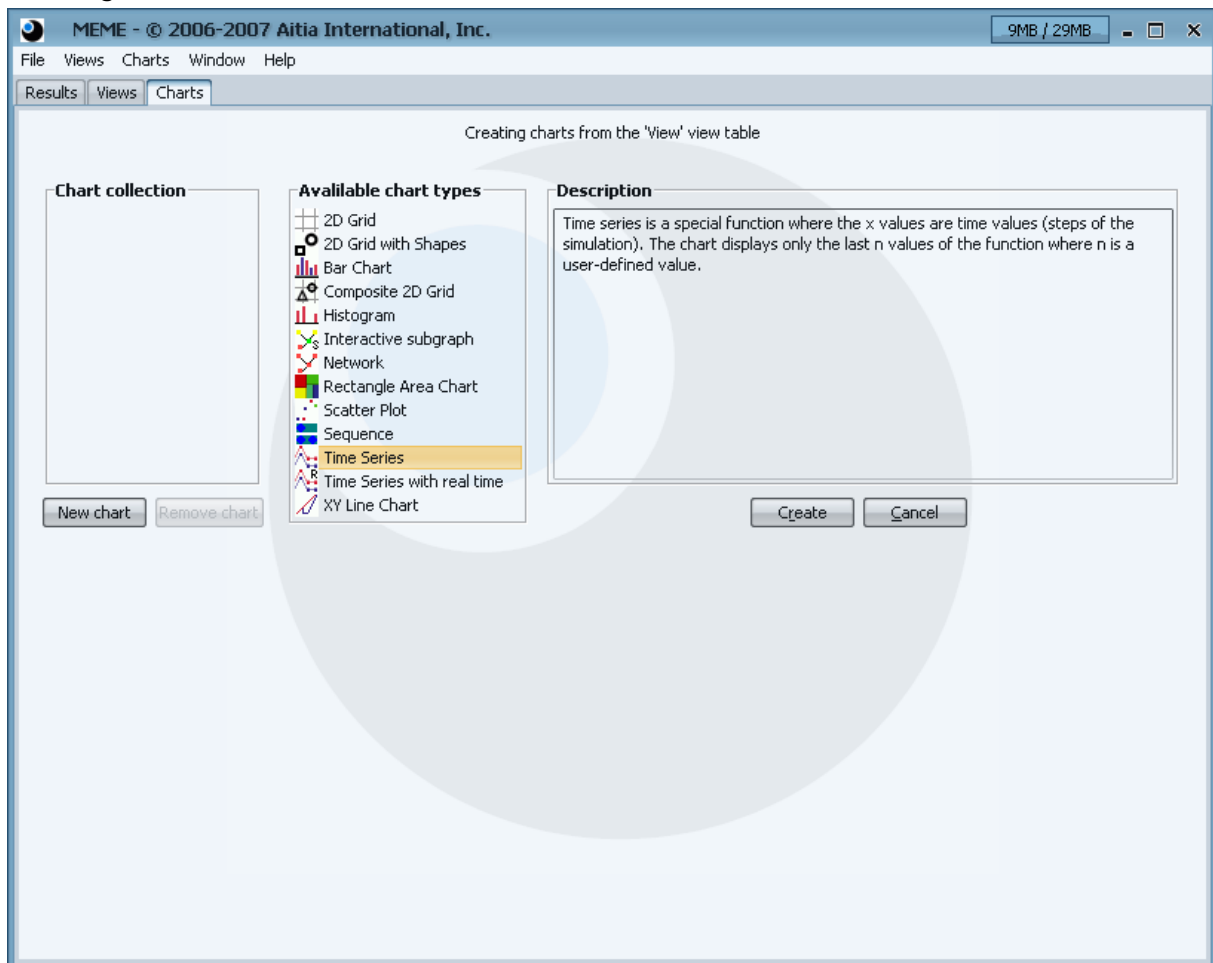
Note that there are differences in handling hidden columns between creating and recreating view tables. If a column is modified to be hidden while recreating it is not automatically left out from the view table, the priority is to assure the unchanged previous sorting still works; it has to be deleted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

## 5 Charts

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

### 5.1 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 19 – Create chart**

Several charts can be created from a single view table. Once created, these charts will be 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. To change chart settings simply select the designated chart in the *Chart collection* list.

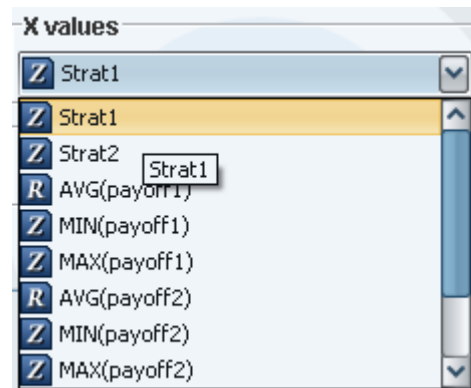
The *Cancel* button terminates the chart-creating mode and switches back to the *Views panel* deleting the unsaved charts.

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 Chart types.

## 5.2 Concept: Data sources

The settings panel of every 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 <sup>2</sup>	logic	"true" is converted to 1, "false" to 0
numeric <sup>2</sup>	text	See below <sup>3</sup>
text	numeric <sup>2</sup>	The numeric value in textual form
text	logic	"true" is converted to "1", "false" to "0"

The "expected type" is described in section 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 **Z**, **R**, **S**, **L** icons in the *Create* section).

## 5.3 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*.

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.

<sup>2</sup> Numeric means integer or real

<sup>3</sup> 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.

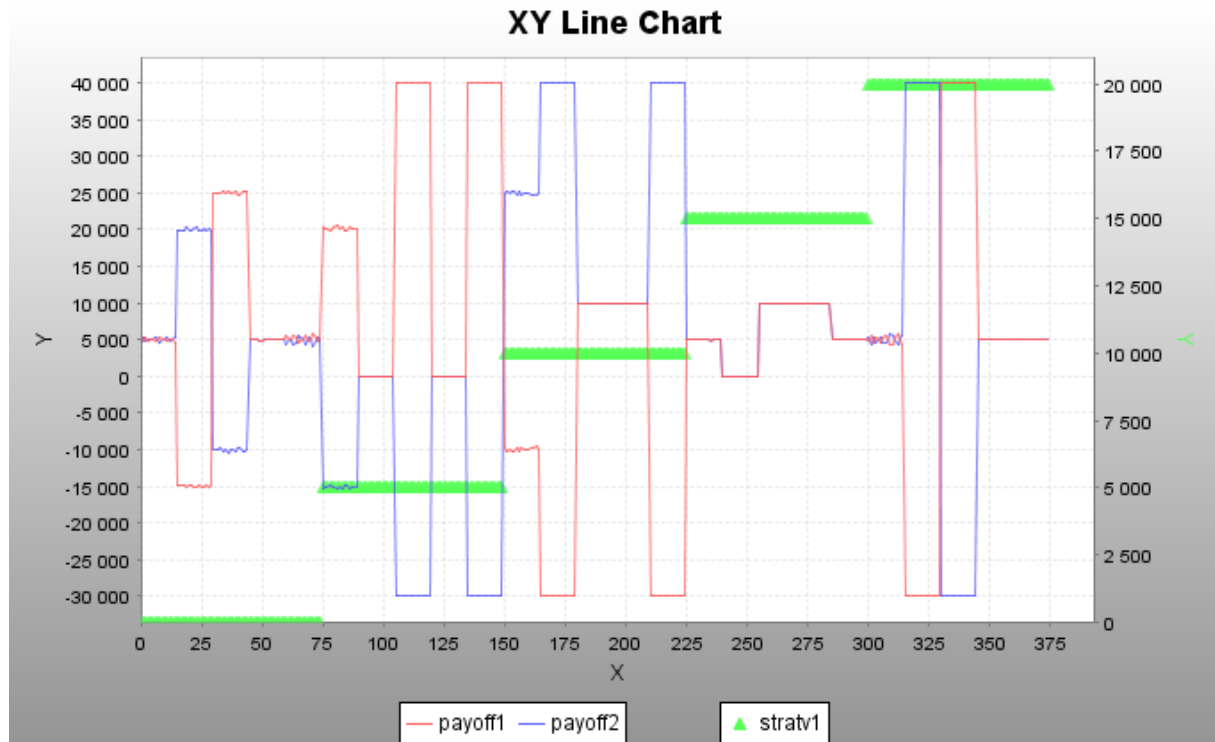


Figure 20

## 5.4 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.

## 5.5 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 and EPS formats.

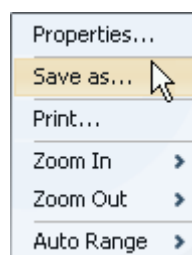


Figure 21

The *Properties* menu displays a dialog window in which you can change some basic properties of the chart like title, background color, axis labels etc.

## 5.6 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.

## 5.7 The Details tab

The settings of the charts are always divided to a *Main Settings* and a *Details tab* (sometimes there are further tabs too). 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.

The appearance of the chart can also be set on the details tab. The available settings are:

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

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

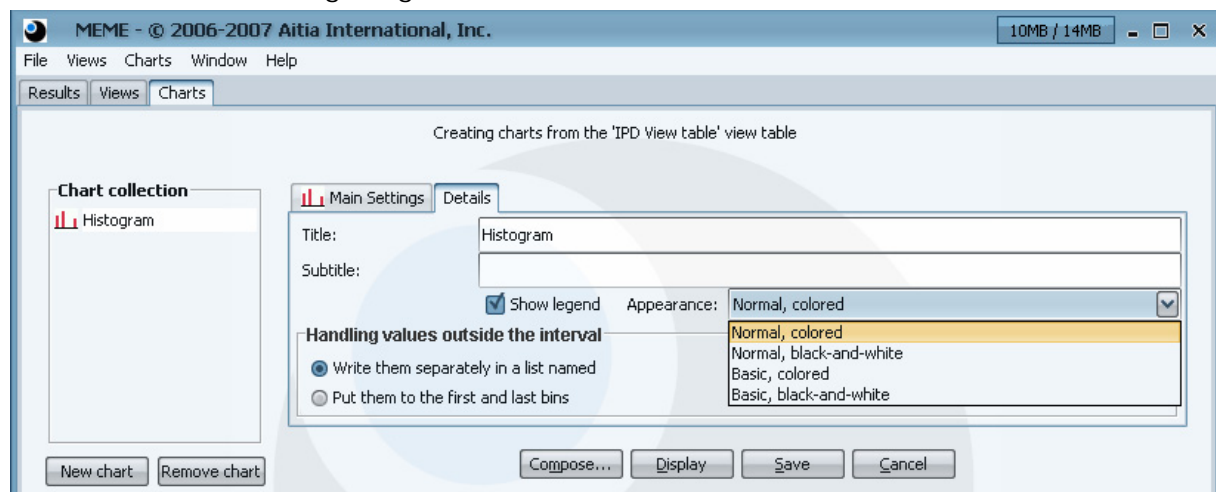


Figure 22 - Details tab

## 5.8 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.

## 5.9 Chart types

### 5.9.1 Bar Chart

A classical chart type that allows 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 Rows*). 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 Rows* 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.

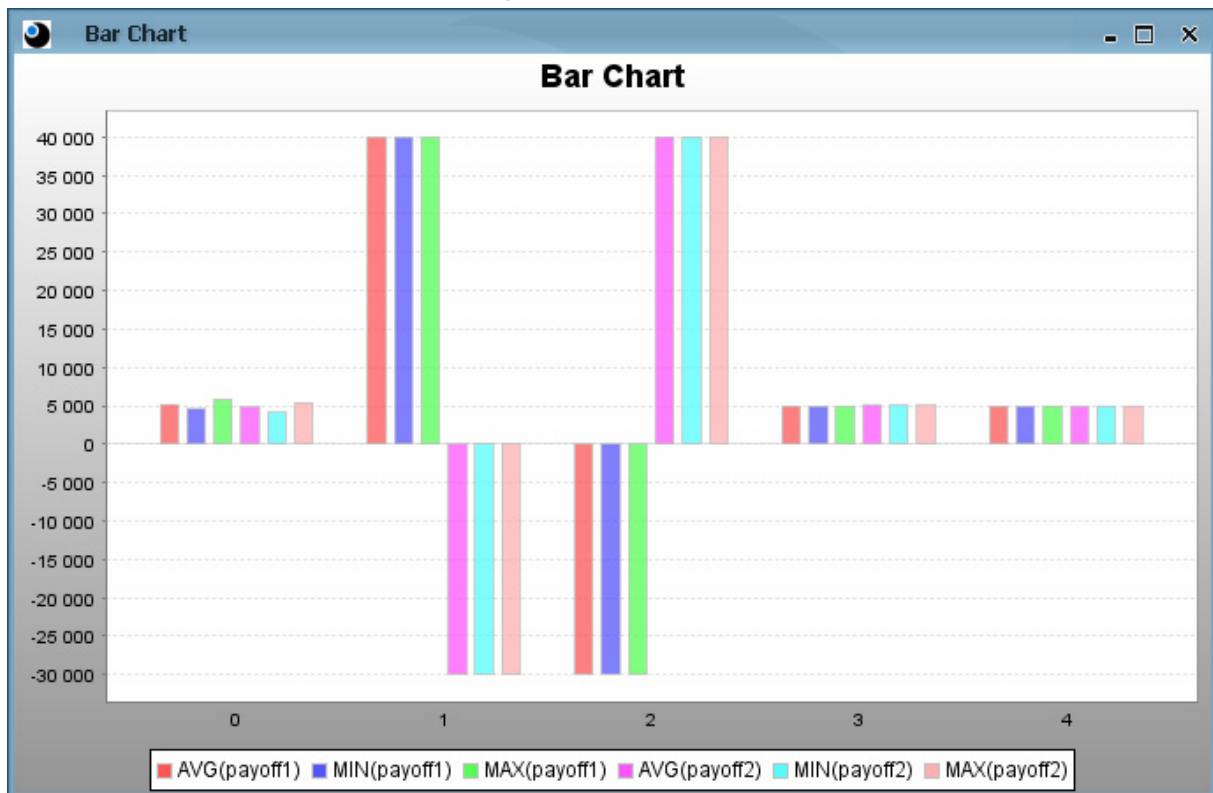
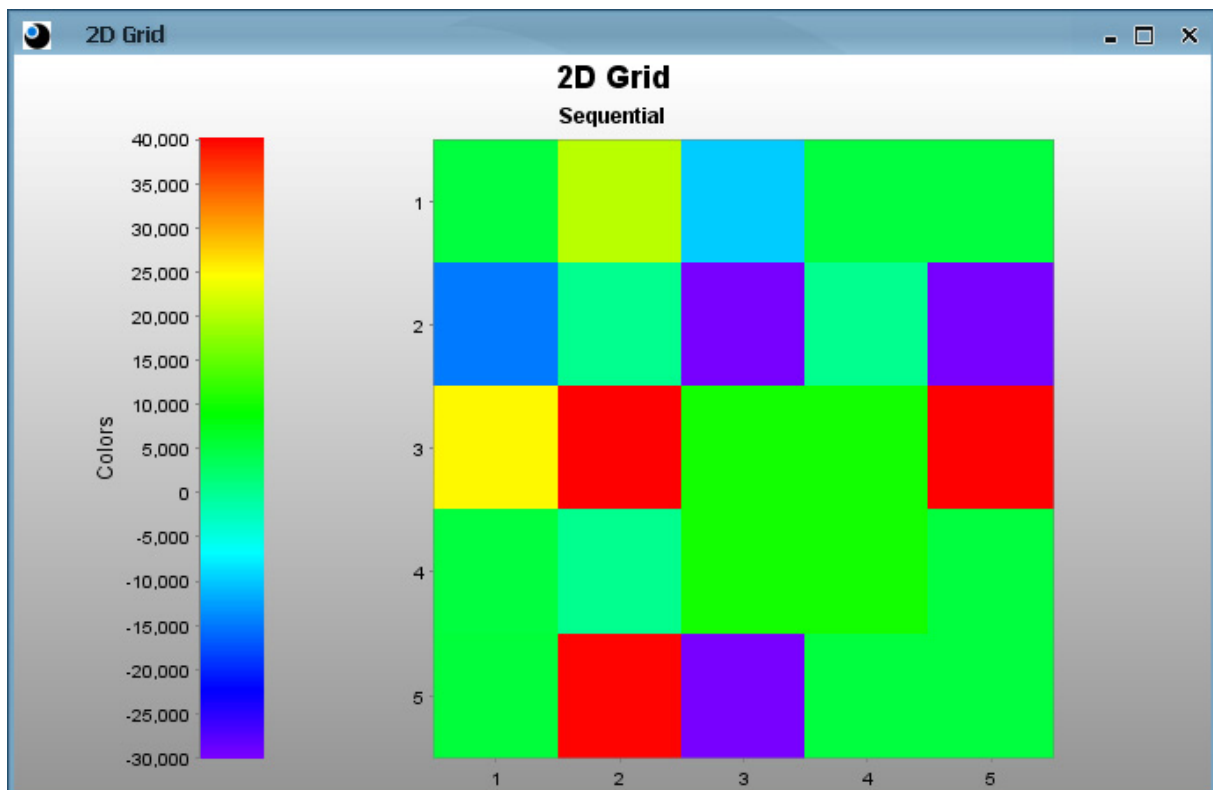


Figure 23

### 5.9.2 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 24**

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 in this case; the *Width* and *Height*, if entered, are used to ignore  $(x,y)$  pairs that are out-of-range.
- 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 random filling* and the *Switch to sequential filling* buttons can be used to choose between the two kinds of Grid 2D charts.

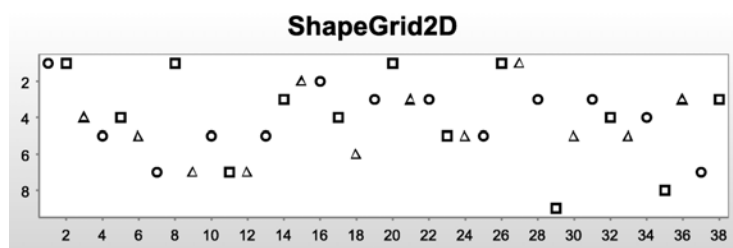
By default an adaptive Rainbow colormap is used to assign colors to the real values. This assigns blue to the smallest and red to the highest value and maps the intermediate values using a rainbow-scale, as shown in the picture above. This coloring method can be changed on the *Colors* tab of the chart 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 minimum and 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, and a default color for all different values can be specified. 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
67.0=-16776961
6.0=-52480
45.0=-6750055
87.0=-3342490
DEFAULT_COLOR=-1
```

### 5.9.3 Shape grid

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.

### 5.9.4 Composite grid

This is a simple combination of 2D Grid and Shape grid. This means that four data sources are used in the partial-matrix case — to compose  $(x,y,color,shape)$  quartets —, and two in the full-matrix case — one for cell colors and another for the shapes.

### 5.9.5 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 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, while lower and upper limits of the interval can be entered manually. It can also be set whether the number, the sum or the average of the values falling in to each bin is visualized.

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

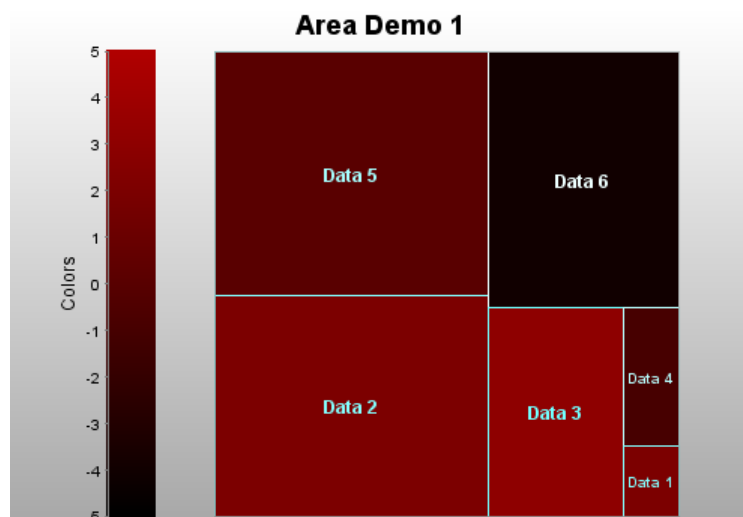
### 5.9.6 Network

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. 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.

### 5.9.7 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.



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 be observed at once.

### 5.9.8 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.

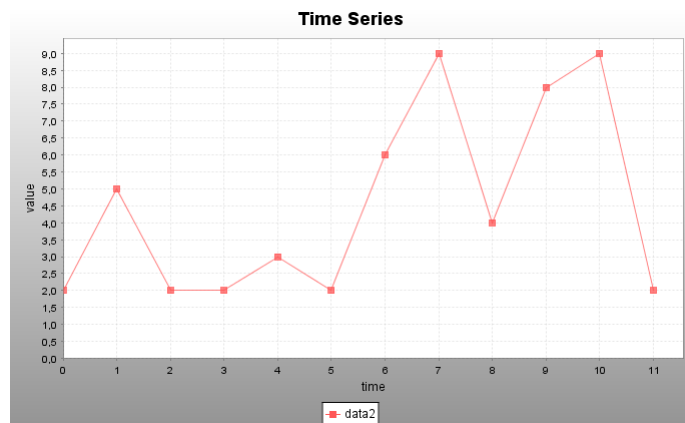
### 5.9.9 Sequence

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.

### 5.9.10 Time Series

This chart displays the last  $n$  value of a series on a classical  $xy$ -axis line chart. The value of  $n$  is 10 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.



### 5.9.11 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 is drawn using a line (where  $x \in X$  values,  $y \in Y$  values, from the same row of the view table), similarly to the Time series but without the nodes at the data points. 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.

## 6 Scripting

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

There are two types of scripting in MEME:

- Since stores information describing view tables and charts in xml based files, it is capable of loading tables from and saving tables 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.

### 6.1 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 Create View Wizard (see 4.2.3 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 2.6 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.

### 6.2 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 file in the open file dialogue.

### 6.3 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.

#### 6.3.1 Example code

```
<viewrule version="1.0.70221">
  <name> name of the view </name>
  <description> narrative text </description>
  <data>
    <table nane="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 ColumType. -->
  <!-- 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> coll == 3 </scalar_script>
</condition>
</viewrule>

```

### 6.3.2 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">`.

## 6.4 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 [5]. 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 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 every 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.

### 6.4.1 Predefined methods

Since ticks and runs are cornerstones of 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`), 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 the you can access the first parameter only via the global variable 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 on 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 only 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`").

## 7 Known Issues

### 7.1 MASS/MEME Parameter Sweep Wizard & Monitor

In the introduction we writes that MEME is a tool for dealing with batch runs of simulations, but currently it can't create a batch version of a Repast model and start an experiment (a batch run).

The MASS/MEME Parameter Sweep Wizard application can. With it you can create a batch version of a Repast model in a simple way and start experiments on the local host or a grid or cluster of computers. In the later case, you can use the MASS/MEME Monitor application to follow the progress of your simulation and to download the results. After the end of the simulation, you can use the *File / Import / Repast* result file menu item of the MEME to import the results to the database.

The installion wizard of the MEME offers the installion MASS/MEME Parameter Sweep Wizard and the MASS/MEME Monitor applications.

In the near future we will integrate these stand-alone applications with the MEME; after that the MEME will collect the results automatically.

## 8 Troubleshooting

### 8.1 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 the support address of MEME.

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.

### 8.2 Memory usage

Maximum heap size reserved for database processing is set to be 256 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 400 MB. When processing tables with 100,000+ 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:1.5+ -Xmx256 -jar MEME.jar"` to `"javaw.exe -version:1.5+ -Xmx<desired amount> -jar MEME.jar"`.

### 8.3 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>

### 8.4 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.

## 9 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 distilled (computed) tables which can be visualized on charts.

In the near future MEME is expected to be able to run MASS/Repast simulations directly and collect their results internally, and on the other hand, to interoperate with other existing statistical software like R and Matlab. These developments will further simplify the modeling work and allow the users to focus on the models themselves.

## 10 References

- [1] MASS – Multi-Agent Simulation Suite. © AITIA Internation Inc.  
[http://www.aitia.ai/services\\_and\\_products/simulation\\_systems/mass](http://www.aitia.ai/services_and_products/simulation_systems/mass)
- [2] Repast - Recursive Porus Agent Simulation Toolkit  
<http://repast.sourceforge.net/>
- [3] R - The R Project for Statistical Computing  
<http://www.r-project.org/>
- [4] Matlab  
<http://www.mathworks.com/>
- [5] Beanshell Scripting  
<http://www.beanshell.org/>