



PET Administrator's manual

Participatory Extension

Created by Vilmos Kozma and Marton David Ivanyi

For ELTE-IKKK



2008 December

Contents

CONTENTS	2
INTRODUCTION	4
1.1 Agent-Based Modeling	4
1.2 MASS.....	4
1.3 PET	4
1.4 Purpose.....	5
1.5 System requirements	5
1.5.1 Minimum System Requirements	5
1.5.2 Optimal System Requirements	5
1.6 Roles	5
1.7 Login	5
1.7.1 Login failed.....	7
1.8 Navigation.....	8
1.8.1 Navigation bar (admin interface).....	8
2 MODELS	9
2.1 Importing Model Families	9
2.2 Importing Models.....	10
2.3 Creating Models	10
2.4 Model Parameters and Settings	11
2.4.1 Settings	11
2.4.2 Agents, adding agents.....	12
2.4.3 Errors and error messages while creating a model	14
2.4.4 Deleting a model.....	14
2.4.5 Public/Private model.....	15
3 SIMULATIONS	16
3.1 Building a simulation.....	16
3.2 Joining simulations.....	16
3.3 User settings (for simulations)	16
3.4 Simulation control.....	18
3.5 Controlling agents	20

3.5.1	Release agent.....	21
3.6	Reproducing a simulation.....	21
4	USER MANAGEMENT.....	22
4.1	Users.....	22
4.1.1	Creating new users	22
4.2	Groups	23
4.3	Permissions.....	24
4.3.1	Relation	24
4.3.2	Configuration	24
4.3.3	Agent.....	24
4.3.4	Group, User and Permission	24
4.3.5	Metadata	24
4.3.6	Simulation	25
4.3.7	Questionnaire, Question, Choice and Answer	25
5	GLOBAL SETTINGS.....	27
6	CONCLUSION.....	28
7	APPENDIX A - MODEL DESCRIPTIONS	29
7.1	Fire.....	29
7.2	Hunter.....	29
7.3	Zerosum	29
7.3.1	Mathematical model.....	29
7.3.2	Short analysis	30
7.3.3	Agent strategies	30
7.3.4	References for Zerosum.....	31
7.4	Repast HeatBugs	31
7.5	Configurable Repast HeatBugs	31

Introduction

1.1 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.

1.2 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 modelling 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.

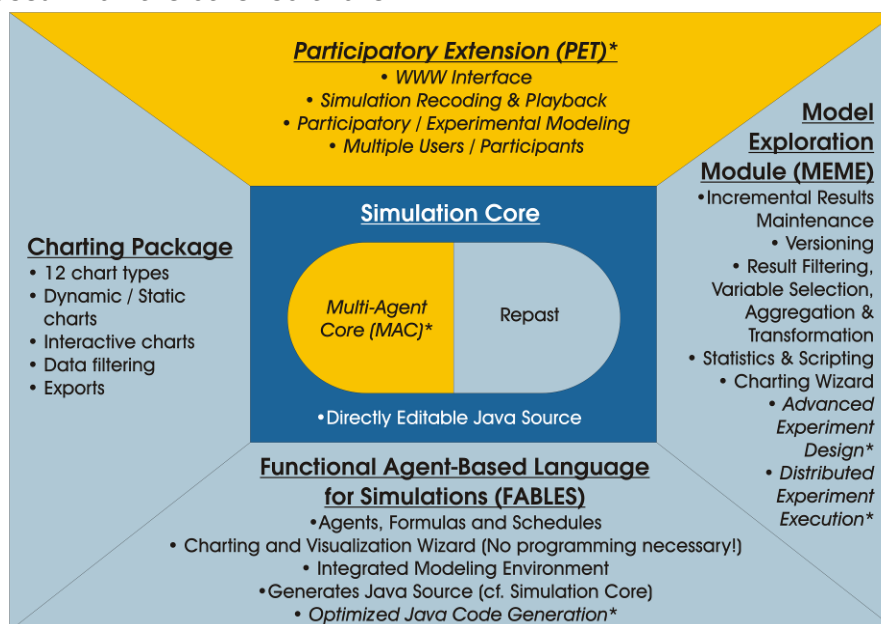


Figure 1 - Multi-Agent Simulation Suite

1.3 PET

PET is the web-based environment designed for administrating and running models and multi-agent simulations. It is based either on its native core, the Multi-Agent Core (MAC) or on Repast J. PET is part of the Multi-Agent Simulation Suite (MASS).

1.4 Purpose

This Administrator's Manual is prepared for the administrators of the Participatory Extension (PET). As this paper is intended for the administrators of the environment it does not go into the issue of programming models and provides more information than an end user would require. For information regarding programming models consult *PET Tutorial*, while end users should read the *PET User Guide*.

1.5 System requirements

1.5.1 Minimum System Requirements

To install and run PET a Java 1.5 compatible platform (Pentium 166MHz or faster, minimum 64 MB RAM) and about 200 MB of free disk space is needed. Note that on above described systems PET will perform poorly.

1.5.2 Optimal System Requirements

Equivalent to a Pentium 1000MHz or faster, 256 MB RAM and 200 MB additional (excluding Java) free disk space.

1.6 Roles

There are two main types of users in the Participatory Extension; end users who run and participate in simulations and administrators, who create models and administrate end users for example. This guide is prepared for the second group while user related issues are covered in the *PET User Guide*.

1.7 Login

In order to start PET open your web browser – Mozilla Firefox or MS Internet Explorer preferably – and type in the address provided by your system administrator. PET uses a web interface and you're required to log in to it with the username and password provided by your system administrators just like on an ordinary Internet site. The administrative login screen is titled *Admin login*, while the final user login is titled *User login*. In order to access the administrative features login at the ¹*Admin login* (see below).

¹ If PET is installed on the computer you are working on: <http://localhost/pet/admin.do>, otherwise it's <http://<url>/pet/admin.do>.

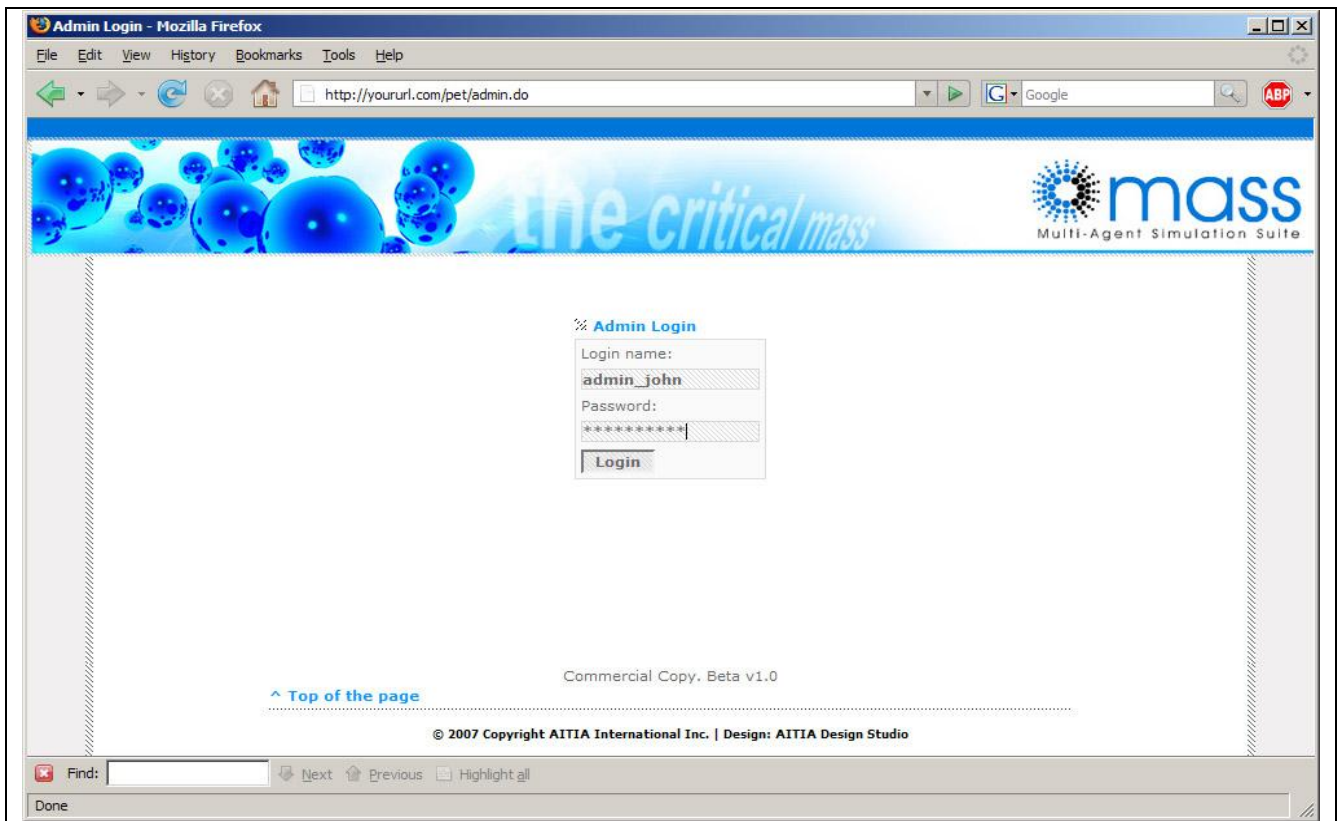


Figure 2 - Admin login

If you see login fields similar to the above, simply type in your user name and password in their respective fields and click *Login*. Upon authentication the Administrator main screen should appear (see Figure 3).

the critical mass

mass
Multi-Agent Simulation Suite

Models | Simulations | User management | User Interface | Settings | Logout

User: admin

☞ All models

Create a new model as starting from

Load model from file check version

Move	ID	Name	Family	State	Actions
1	1	Watch the Fire	Fire	Public	<input type="button" value="D"/> <input type="button" value="N"/> <input type="button" value="E"/> <input type="button" value="B"/> <input type="button" value="Build"/>
2	2	Escape the Fire	Fire	Public	<input type="button" value="D"/> <input type="button" value="N"/> <input type="button" value="E"/> <input type="button" value="B"/> <input type="button" value="Build"/>
3	3	Watch the hunt	Hunter	Public	<input type="button" value="D"/> <input type="button" value="N"/> <input type="button" value="E"/> <input type="button" value="B"/> <input type="button" value="Build"/>
4	4	Don't Be a Prey!	Hunter	Public	<input type="button" value="D"/> <input type="button" value="N"/> <input type="button" value="E"/> <input type="button" value="B"/> <input type="button" value="Build"/>
5	5	Control the Market 1	Zerosum	Public	<input type="button" value="D"/> <input type="button" value="N"/> <input type="button" value="E"/> <input type="button" value="B"/> <input type="button" value="Build"/>
6	6	Control the Market 2	Zerosum	Public	<input type="button" value="D"/> <input type="button" value="N"/> <input type="button" value="E"/> <input type="button" value="B"/> <input type="button" value="Build"/>
7	7	One Against the Market 1	Zerosum	Public	<input type="button" value="D"/> <input type="button" value="N"/> <input type="button" value="E"/> <input type="button" value="B"/> <input type="button" value="Build"/>
8	8	One Against the Market 2	Zerosum	Public	<input type="button" value="D"/> <input type="button" value="N"/> <input type="button" value="E"/> <input type="button" value="B"/> <input type="button" value="Build"/>
9	9	Repast HeatBugs	Repast HeatBugs	Public	<input type="button" value="D"/> <input type="button" value="N"/> <input type="button" value="E"/> <input type="button" value="B"/> <input type="button" value="Build"/>
10	10	Configurable Repast HeatBugs	Configurable Repast HeatBugs	Public	<input type="button" value="D"/> <input type="button" value="N"/> <input type="button" value="E"/> <input type="button" value="B"/> <input type="button" value="Build"/>

Commercial Copy, v1.3.0

Figure 3 - Administrator main screen - Models

1.7.1 Login failed

If you have mistyped either the username or the password you get the following message:

Invalid login name and/or password!

☞ Admin Login

Login name:
blah

Password:

Figure 4 - Login error

Any other kind error or error messages occurring during the login procedure are of outside causes. In these cases you need to contact your system administrator for further assistance.

1.8 Navigation

1.8.1 Navigation bar (admin interface)

PET administration uses a menu bar to help users navigate through the program. All pages and most functions can be reached through the navigation bar. The bar has three main menu items (Models, Simulations, User Management) and two additional functions (User Interface and Logout).

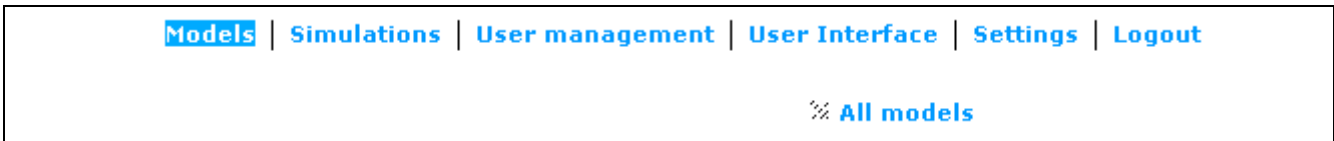


Figure 5 - Upon logging in the main screen, the main model page is opened.

The menu item's blue background denotes being active, while page navigation is implemented through the submenu items. Through *Models* model families can be imported, models and their agents can be set up and administrated and simulations can be built. (See 2 Models!)

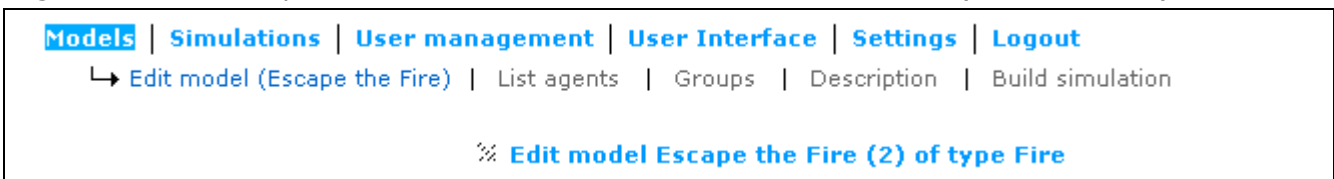


Figure 6 - Submenu and navigation.

Simulations carries simulation management and control items.



Figure 7 - Simulation Control

Through *User management* various groups can be set up with different permission levels and users can be administrated.

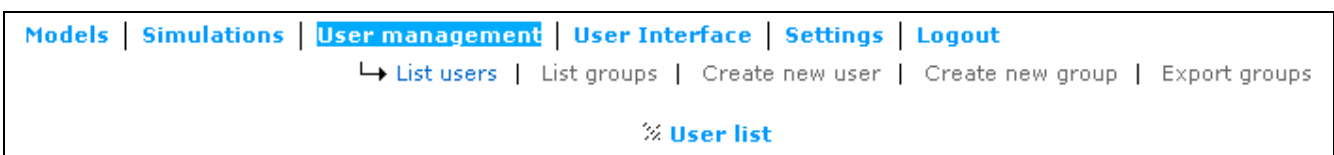


Figure 8 - User management

Global settings can administered on the *Settings* page.

1.8.1.1 Quick links

The navigation bar has two quick links on the right side (see Figure 9). The first, *User Interface* is a link to the user side that enables administrators to see their work in effect. The second is *Logout*, which implicitly terminates the current session (logs the user out in other words).



Figure 9 - Quick link to the user interface and logout.

2 Models

The Participatory Extension operates through a hierarchy of model families, models and simulations. A model family is a general model that has to be written in Java by the modeler and has to be imported into PET by an administrator. A model is a specific version generated from a model family by declaring the number of each type of agents and configuring those agents by an administrator. A simulation is an instance of a model available to be run and replay by or participate in for the users of the Participatory Extension.

While administrators see all the model families and models in the system, end users only see the models that they are allowed to run simulations from. Both administrators and end users see the models on their main screens after logging in. From other screens administrators can go to the models menu by clicking 'Models' on the navigation bar introduced earlier.

PET comes with the following simple demo models by standard:

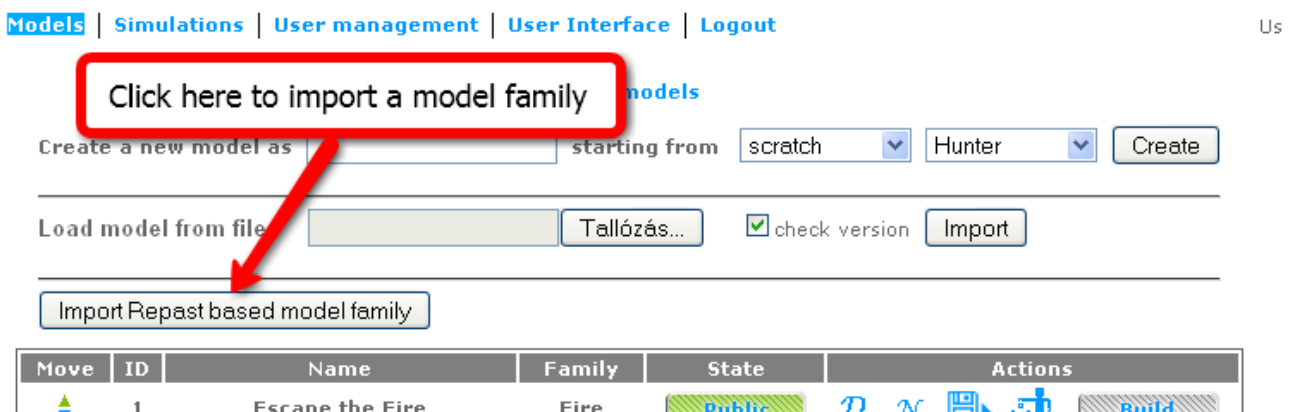
- Fire
- Hunter
- Zerosum
- Repast HeatBugs
- Configurable Repast HeatBugs

These models are described in detail in Appendix A - Model Descriptions.

2.1 Importing Model Families

To make a model family available for users to run in PET, the model-family must be imported into the system. The installation process of a model family varies with the type of model family of which PET distinguishes two:

- MAC (the native modeling framework of PET) based model families are installed manually. For more information consult the PET model writing manual and tutorial document.
- Repast based model families are installed through the administration interface by uploading a valid model family file. This can be done by clicking on the "Import Repast Based Model Family" button.



The screenshot shows the 'Models' management page. At the top, there is a navigation bar with links: Models, Simulations, User management, User Interface, and Logout. A red box highlights the text 'Click here to import a model family' with a red arrow pointing to the 'Import Repast based model family' button. Below this, there are two main sections: 'Create a new model as' and 'Load model from file'. The 'Create a new model as' section has a dropdown menu set to 'scratch' and another set to 'Hunter', with a 'Create' button. The 'Load model from file' section has a file upload button labeled 'Tallózás...', a checked 'check version' checkbox, and an 'Import' button. At the bottom, there is a table listing existing model families.

Move	ID	Name	Family	State	Actions
	1	Escape the Fire	Fire	Public	

Figure 10. – Importing model families I.

This brings the user to the model family import page where properly built Repast based model families can easily be uploaded.

Name the model-family (optional)

Import Repast based Model Family

Model family name:
(Leave this empty to apply the name used in the descriptor)

Select*.pet file containing the model family:

Select the file to upload

Figure 11. – Importing model families II.

2.2 Importing Models

Given that a model family is already written in Java the models derived from that model-family can be imported into PET. This can be done by administrators by providing the required information in the 'File upload' field and clicking on the 'Import' button. See screenshot below!

% All models

Create a new model as starting from

Load model from file 1. check version

2.

Move	ID	Name	Family	State	Actions
↑	1	Escape the Fire	Fire	Public	<input type="button" value="D"/> <input type="button" value="N"/> <input type="button" value="F"/> <input type="button" value="U"/> <input type="button" value="B"/>
↓	2	Watch the Fire	Fire	Public	<input type="button" value="D"/> <input type="button" value="N"/> <input type="button" value="F"/> <input type="button" value="U"/> <input type="button" value="B"/>

Figure 12. - Importing models: 1. Browse for model file; 2. Click 'Import'

2.3 Creating Models

When a model family is imported into PET it becomes available for administrators to create new models from it with different settings and agents in it. This is done by providing the required information in the 'Create new model' line and clicking on the 'Create' button.

1. % All models 2. 3.

Create a new model as starting from

Load model from file check version

Move	ID	Name	Family	State	Actions
↑	1	Escape the Fire	Fire	Public	<input type="button" value="D"/> <input type="button" value="N"/> <input type="button" value="F"/> <input type="button" value="U"/> <input type="button" value="B"/>
↓	2	Watch the Fire	Fire	Public	<input type="button" value="D"/> <input type="button" value="N"/> <input type="button" value="F"/> <input type="button" value="U"/> <input type="button" value="B"/>

Figure 13. - Creating models: 1. Model name; 2. 'Base'; 3. Model family

In the first field a unique name has to be chosen for the model, then a "base" (see the text *starting from* above) has to be selected. This can either be *scratch*, which will result in a model with default parameter settings and no agents, or an already existing model can be chosen to

be copied. The later creates the model with the same agents and settings as the particular model chosen in the "base" combo-box. Note that when choosing an existing model, the settings can still be modified. The third field selects the model family the model is to be created from. When all the fields are set you simply click *Create*. Note that if an existing model is chosen as base, the model family is chosen automatically.

2.4 Model Parameters and Settings

When a model is generated from *scratch* it has default settings and no agents in it. To change the settings and add agents, click on the given model's name, opening up the edit model screen.

Models | Simulations | User management | User Interface | Settings | Logout

↳ Edit model (Watch the Fire) | List agents | Groups | Description | Build simulation

⌘ Edit model Watch the Fire (1) of type Fire

⌘ Agents

Type	Class	Number	Actions
Space agent	ai.aitia.mass.demos.fire.World	1	[Icons for copy, delete, add]
Agents trying to escape the fire	ai.aitia.mass.demos.fire.Person	98	[Icons for copy, delete, add]
Fire agent	ai.aitia.mass.demos.fire.Fire	1	[Icons for copy, delete, add]

⌘ Simulation properties

Property	Class	Value
Maximum applets:	java.lang.Integer	20
Maximum number of simulations	java.lang.Integer	5
Random seed	java.lang.Long	123456
Release Policy	java.lang.Integer	Continue
Save period	java.lang.Integer	1800
Sleep between steps	java.lang.Integer	10
Timeout	java.lang.Long	300
Timeout policy	java.lang.Integer	Call on timeout
Type of Event Logger DAO	java.lang.String	Hibernate
Wait policy	java.lang.Integer	Timeout

Update properties

Figure 14 - Edit model: 1. Adding agents; 2. Changing simulation properties

2.4.1 Settings

The screen above shows the settings page for the Fire model. If you keep the default settings for the simulation it will run just fine, but you can adjust them to your liking. Here are the short definitions for the properties seen above:

- **Maximum applets:** Limits the number of Applets that can to connect simultaneously to a simulation of the selected model.
- **Maximum number of simulations:** Limits the number of simulations that can run simultaneously of this model.

- **Random seed:** This is the input for the pseudo random number generator. This enables reproducing simulations.
- **Release Policy:** Describes what happens if the user releases the agent he or she controls. Possibilities are:
 - **Continue:** PET takes back the control over the agent;
 - **Remove agent:** The agent is removed from the simulation.
- **Save period:** The auto-save period for the model.
- **Sleep between steps:** Determines the time between steps (milliseconds).
- **Timeout:** Timeframe for users controlling agents in each turn for actions. The value is given in milliseconds, which means 1000 = 1 second.
- **Timeout policy:** This property is examined by PET when a user controls an agent. If the user is not giving input for a certain amount of time (defined by the *Timeout* property) and the *Wait policy* property is set to *Timeout* then a timeout event occurs. This property defines what happens when a timeout event occurs. Possible values are:
 - **Call on timeout:** PET calls the default action for the agent. The default action is defined by the model family.
 - **Release agent:** PET takes back the control over the agent (See *Release policy* property)
- **Type of Event Logger DAO:** Events (user actions) are logged. It can be in a database (through Hibernate) or an xml file for example.
- **Wait policy:** Describes whether PET waits for the users to make their decisions in participatory simulations. This property is examined by PET when a user controls an agent. The three possibilities are:
 - **No wait:** the simulation does not wait for user input it simply continues the run;
 - **Wait:** PET waits for user input and the simulation is on hold until the user makes his decision;
 - **Timeout:** the simulation waits for a certain amount of time (defined by the value of *Timeout* property), and if the user is not giving input, a timeout event occurs. (See the *Timeout policy* property)

2.4.2 Agents, adding agents

A model created is just a frame until there are agents in it. Different types of models have different agents, but one thing is common in all of them. For technical reasons PET considers the *world*, the space the other agents exist in, to be an agent too. When creating a model where a given type of agent refers to an other type of agent, first the latter (the one referred to) has to be created. As most agents refer to the space they exist in, the space agent should be the first one added (for further instructions see below) and then can all the other types of agents be added to model. Note that in the models provided with PET there are maximum three types of agents, but the only real limitation is computing capacity in this matter.

In order to add a space agent click on the 'Add new <given_space_agent_name>' command.

⌘ Edit model Control the Market 1 (7) of type Zerosum

⌘ Agents

Type	Class	Number	Actions
ZerosumSpace	ai.aitia.mass.demos.zerosum.ZerosumSpace	1	
ZerosumAgent	ai.aitia.mass.demos.zerosum.ZerosumAgent	60	

Figure 15 - Agents: 1. List all; 2. Delete all; Add agent

Clicking on the *Add agent* button activates the agent settings screen where agent properties can be set (see Figure 16).

The *Delete agents* button deletes all instances of the given type of agents.

The *List agents* button lists all instances of the given type of agents. Clicking on one of the agents also brings on the agent properties screen enabling the user to edit the given agents settings (see below).

2.4.2.1 Agent properties

When adding or editing agents, the default or previously set agent properties can be modified, and the number of agents in the model with the same properties can be set.

On the property page there is a table with the given agent's properties and attributes. The first column denotes the actual property. The second column (Class) refers to the next column (Value) and clarifies what kind of input variable the given property takes. The fourth column (Visibility) denotes whether users see the property, while the last column (Initialized) controls whether users can or cannot set the given property while the simulation is running. (See 3.3 User settings (for simulations) for the latter.)

[Models](#) | [Simulations](#) | [User management](#) | [User Interface](#) | [Settings](#) | [Logout](#)

↳ Edit model (One Against the Market 1) | List agents | Groups | New agent (Type: Space agent)

% Add new agent - Space agent

Property	Class	Value	Visibility	Initialized
Agent Height	java.lang.Integer	<input type="text" value="6"/>	<input checked="" type="radio"/> true <input type="radio"/> false	<input checked="" type="radio"/> true <input type="radio"/> false
Agent Width	java.lang.Integer	<input type="text" value="6"/>	<input checked="" type="radio"/> true <input type="radio"/> false	<input checked="" type="radio"/> true <input type="radio"/> false
Applet Height	java.lang.Integer	<input type="text" value="400"/>	<input checked="" type="radio"/> true <input type="radio"/> false	<input checked="" type="radio"/> true <input type="radio"/> false
Applet Width	java.lang.Integer	<input type="text" value="900"/>	<input checked="" type="radio"/> true <input type="radio"/> false	<input checked="" type="radio"/> true <input type="radio"/> false
Controllable	java.lang.Boolean	<input type="radio"/> true <input checked="" type="radio"/> false	<input type="radio"/> true <input checked="" type="radio"/> false	<input checked="" type="radio"/> true <input type="radio"/> false
Detectors visible	java.lang.Boolean	<input type="radio"/> true <input checked="" type="radio"/> false	<input type="radio"/> true <input checked="" type="radio"/> false	<input checked="" type="radio"/> true <input type="radio"/> false
Future Price	java.lang.Double	0.0	<input checked="" type="radio"/> true <input type="radio"/> false	not modifiable
Number Of Agents	java.lang.Integer	<input type="text" value="0"/>	<input checked="" type="radio"/> true <input type="radio"/> false	<input checked="" type="radio"/> true <input type="radio"/> false
Real Price	java.lang.Double	0.0	<input checked="" type="radio"/> true <input type="radio"/> false	not modifiable
Visible	java.lang.Boolean	<input type="radio"/> true <input checked="" type="radio"/> false	<input type="radio"/> true <input checked="" type="radio"/> false	<input checked="" type="radio"/> true <input type="radio"/> false
Visualization	java.lang.String	<input type="text" value="None"/>	<input type="radio"/> true <input checked="" type="radio"/> false	<input checked="" type="radio"/> true <input type="radio"/> false

Add piece(s)...

Figure 16 - Agent properties

To add a world agent with default settings simply choose a visualization for it from the drop down list in the 'Visualization' row (Zerosum World in this case) and set the 'Visibility' property to 'true'. When the desired quantity is set (preferably 1 for a space agent) click the 'Add' button to include the agent in the model.

Here are the definitions for the properties of agents provided by PET as standard:

- **Controllable:** Decides whether an agent can be user-controlled or is strictly computer driven.
- **Detectors visible:** Indicates whether the detector functions are displayed on the user interface.
- **Visible:** Decides whether the agent appears as a viewable agent on the user side.
- **Visualization:** Type of visualization for the space.

All other properties of agents are model specific. As Figure 16 shows (the last two columns) each property has two attributes: Visibility and Initialized.

- **Visibility:** If visibility is set to true the users can see the value of the given property. For most properties, if you do not wish to confuse users with unnecessary information, this should set to false.
- **Initialized:** If initialized is set to true then, when a user starts a new simulation the given property will be unchangeable. If it's false the user can modify the value of that property and the value set in this page will only be used as the default value.

When a space agent is added all the other agents can be added similarly to the model by using the 'Add new <XYZ> agent' command in the 'Agent Properties' screen.

2.4.3 Errors and error messages while creating a model

- Trying to add agents without adding the agent they refer to (usually the space agent) first results in a message similar to the one below:

Agents

Add ai.aitia.mass.demos.fire.World first!

Type	Class	Number	Actions
World	ai.aitia.mass.demos.fire.World	0	+ ☒
Fire	ai.aitia.mass.demos.fire.Fire	0	+ ☒
Person	ai.aitia.mass.demos.fire.Person	0	+ ☒

Figure 17 - Always add the space agent first!

- Trying to provide inappropriate values for properties returns the user to the given page with the dissatisfactory values marked red:

Simulation properties

Property	Class	Value
Random seed	java.lang.Long	123456
Release Policy	java.lang.Integer	Continue
Save period	java.lang.Integer	1800
Sleep between steps	java.lang.Integer	10
Timeout	java.lang.Long	300
Timeout policy	java.lang.Integer	Call onTimeout
Type of Event Logger DAO	java.lang.String	Hibernate
Wait policy	java.lang.Integer	Timeout

Update properties

Figure 18 - Inappropriate value

Any other errors that might occur during the creation a model is of outside causes. They can be database or network related problems, contact your system administrator to deal with these.

2.4.4 Deleting a model

Administrators can delete models from PET. To delete a model click on the trash can symbol in any listing the model appears on and confirm delete on the popup. If there is an simulation in the suite created from the particular model it can't be deleted. Attempting to delete a model with existing simulations results in the following error message:

🔗 Repast HeatBugs models

This model is not modifiable!
 Instance of this model exist,
 please delete them or create a new one!

Copy as

Create new model as starting from scratch

Load model from file check version

ID	Name	Family	State	Actions
9	Repast HeatBugs	Repast HeatBugs	Public	<input type="button" value="D"/> <input type="button" value="N"/> <input type="button" value="F"/> <input type="button" value="U"/> <input type="button" value="Build"/>

Figure 19 – You can't delete models with simulations.

2.4.5 Public/Private model

Models are created initially in a *Private* state. Once the administrator feels that a given model is ready to be used by the public (i.e. the users), it has to be made *Public*. This is done by clicking on the corresponding 'State' button on the model listings.

🔗 All models

Create a new model as starting from scratch Hunter

Load model from file check version

Move	ID	Name	Family	State	Actions
↑	1	Watch the Fire	Fire	Public	<input type="button" value="D"/> <input type="button" value="N"/> <input type="button" value="F"/> <input type="button" value="U"/> <input type="button" value="Build"/>
↓	2	Escape the Fire	Fire	Public	<input type="button" value="D"/> <input type="button" value="N"/> <input type="button" value="F"/> <input type="button" value="U"/> <input type="button" value="Build"/>

Figure 20 - Make a model public/private

Simulations with modifiable settings have an extra option - *Agent properties* - on their *Simulation Control* page (see Figure 23 below). Uninitialized properties can be modified before the first run or can be executed with the default values of the particular model.

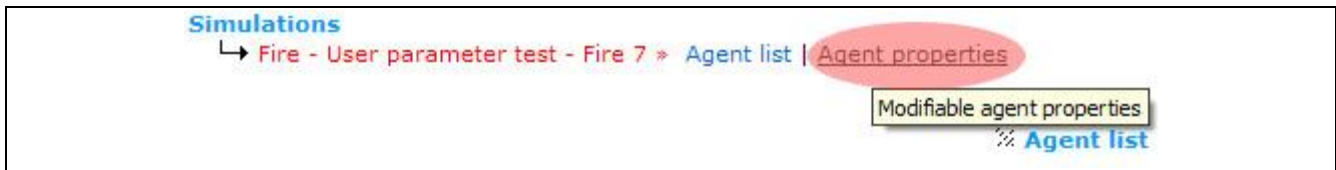


Figure 23 - Simulation with user modifiable properties.

On the *Modifiable agent properties* page (see Figure 24) all the agents with modifiable properties are listed. The agents with the same modifiable properties are grouped together. All agent properties (read-only and read/write) can be listed by checking *Show read only properties*. By default only the read-write properties are listed.

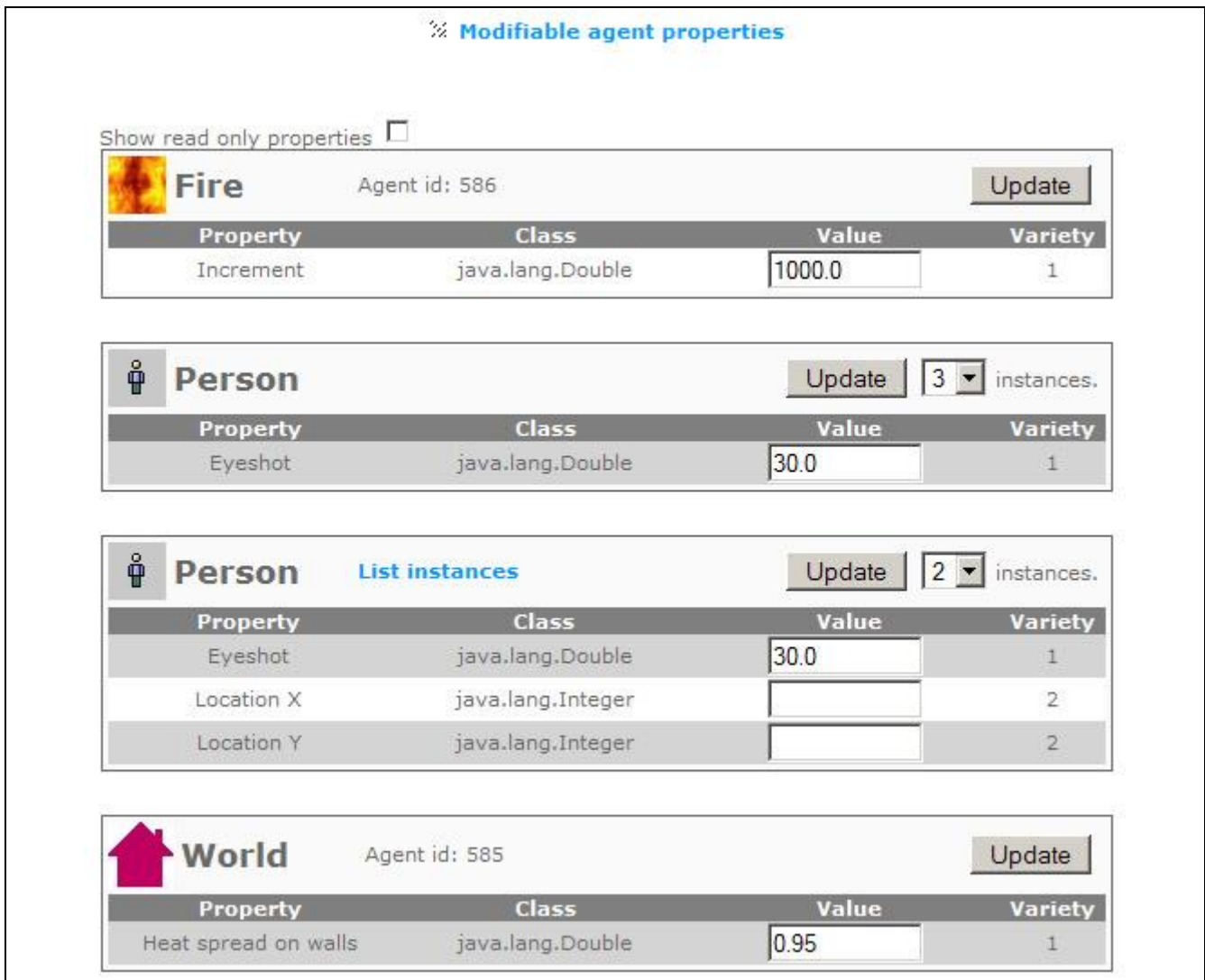


Figure 24 - Modifiable agent properties list

In order to change a property, type the desired value in its *Value* field and press the *Update* button (see Figure 25 where we set the property *Increment* to 1000.0).

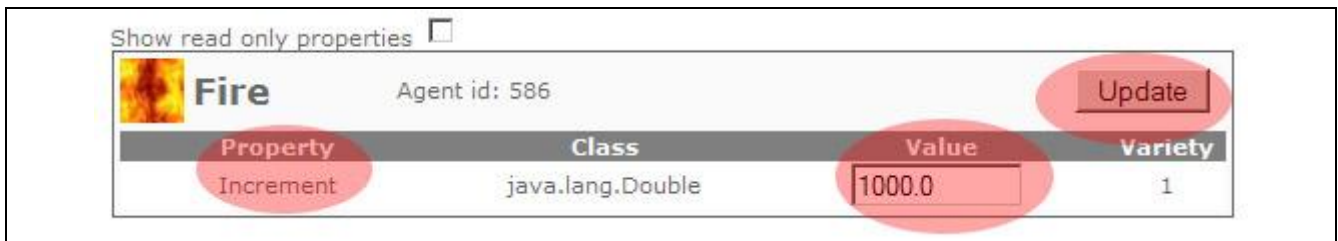


Figure 25 - Single agent, with single property.

If there are more than one agents with the same modifiable properties, the number of instances for which the value(s) is to be given can be selected through the drop down list by the 'Update' button (Figure 26).

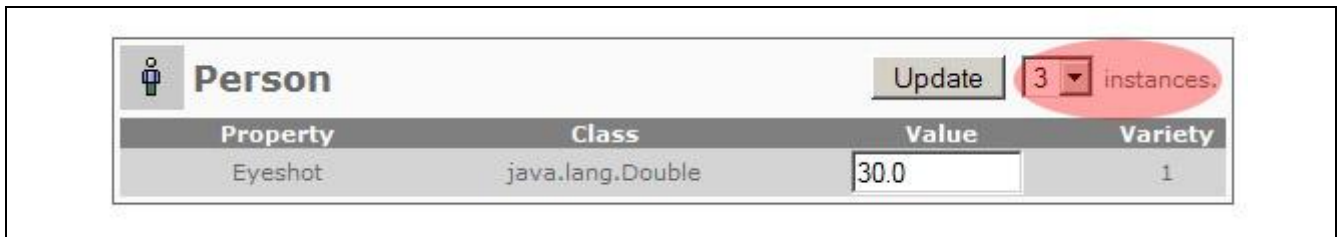


Figure 26 - Multiple agents with a single property.

With multiple agents with multiple properties it is possible to list and provide values for all instances separately by clicking on 'List instances' (Figure 27).

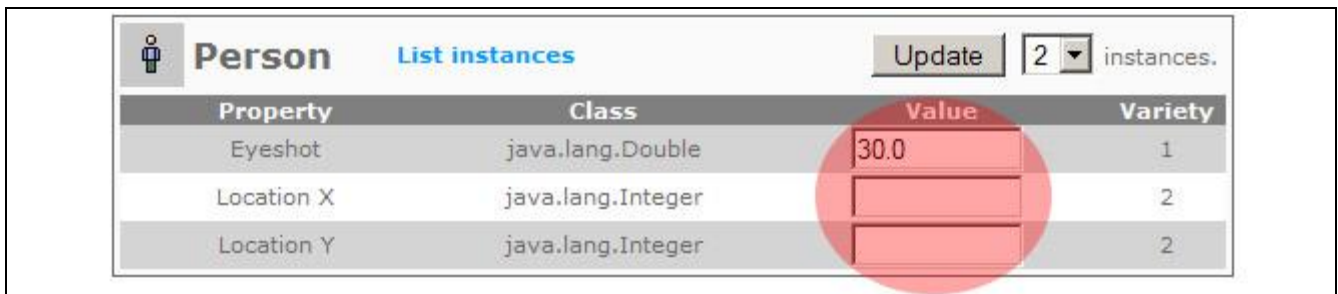


Figure 27 - Multiple agents with multiple properties.

The variety column shows the number of different values of the given property in the agents.

3.4 Simulation control

From simulation control pages users can start, pause, resume and replay their simulations, as well as observe visualizations and control their agents in participatory models.



Figure 28 - Simulation controls

The first two of four control buttons numbered on the above picture represent *Simulation mode* (1.) and *Replay mode* (2.). On the above figure the simulation hasn't been started yet. Once started by pressing button 3 the first button becomes blue (active), indicating that it is in *Simulation mode*. If paused (4.), the simulation can be switched to *Replay mode* by pressing button 2. To start the replay, simply press *Play* (3.). It can be switched back to simulation mode by pausing and pressing button 1.

The administrator's *Simulation Control* pages contain additional information compared to the one the end users see. Besides the *Visualization* and *Properties* fields (see Figure 29)

3.5 Controlling agents

In participatory models users can control all or some of the agents, while the computer controls the rest. To “get hold of” an agent *Control agent* has to be pressed in the *Simulation Control* page.

If there is only one controllable agent in the simulation PET automatically takes the user to that agent’s *Control agent* page (see Figure 30).



Figure 30 - 1. Agent control; 2. Click for visualization; 3. Actions

On the screenshot above the visualization is not embedded in the agent control page. This is rather the exception than the rule, by clicking on *Visualization* a 3D visualization applet is launched in this case.

Under *Actions* all the actions the agent can take in a turn are listed. On the screenshot above these are all related to movement, but this could be anything the creator of the model specifies. It is important to mention that when a user clicks on an action the action is not performed at once. Instead, the selected action is placed into a buffer and the action is performed when the agent is activated. In normal cases this happens only once in each simulation round. The buffer can hold only one action at the same time. If the buffer is not empty and a new action is taken, the new action overwrites the older one, and the old one is never performed.

If there is more than one controllable agent in the simulation the user is taken to the list of agents (see Figure 31) where agents can be selected for both view and control.



Figure 31 - Agent list: 1. Observe agent; 2. Control agent.

Note that on the figure above the highlighted icons are blue, which represents that agents 9, 10 and 11 are controllable.

3.5.1 Release agent

In order to release an agent – so it can manage on its own or be controlled by other users – simply press the 'Release agent' command on the 'Control agent' page.



Figure 32 - Release agent on the 'Control agent' page

3.6 Reproducing a simulation

In PET simulations can be replayed any number of times. This is possible because all actions that are taken by the users are logged, and by using the replay function the simulation always performs the same run.

4 User Management

Through *User management* both end and administrative users can be managed. Groups with complex permission settings and users with different group settings can be created. The User management pages have five main functions: list users, list groups, create new user, create new group and export groups.

4.1 Users

Choosing User management from the menu bar opens the full *User list* where all users are listed (see Figure 33). In order to filter for the members of a group select the given group's name from the *Group* column drop down list. To delete a user, press the Delete button at the end of the line. To modify user data click on the corresponding *Login* name.

Figure 33 shows a screenshot of the 'User list' interface. At the top, it says 'User list' and 'Displaying from 1 to 3 [First/Prev] 1 [Next/Last]'. Below this is a table with the following columns: ID, Login, First name, Last name, Group, and Delete. The table contains three rows of user data. Red boxes highlight the 'Group' dropdown menu (labeled '1'), the 'Login' column header (labeled '2'), and the 'Delete' button for the first user (labeled '3').

ID	Login	First name	Last name	Group	Delete
1	Testuser1	John	Testuser	administrator	[Delete]
2	Testuser2	Jim	Testuser	default	[Delete]
3	Testuser3	Jack	Testuser	empty	[Delete]

Figure 33 - User list: 1. Filter by group; 2. Login name; 3. Delete

4.1.1 Creating new users

To add new users to PET click on the *Create new user* link in the menu bar. Enter the user name, password, first and last name, select a group for the new user and press *Submit* to add the user to the database (Figure 34). Later on the user can be added to a group.

Figure 34 shows a screenshot of the 'Add user' form. At the top, there is a navigation bar with links: 'List users', 'List groups', 'Create new user', and 'Create new group'. Below this is the 'Add user' form with the following fields: Login, Password, Confirm password, First name, Last name, and Group. The Group dropdown menu is set to 'default'. At the bottom of the form are 'Submit' and 'Cancel' buttons.

Figure 34 - Adding a new user

4.2 Groups

Groups are the cornerstones of PET user management. There are four groups in PET by default. *Empty*, with no permissions whatsoever, *default*, which is an end user group, with the corresponding permissions, *administrator*, which has most of the administrative permissions and *superuser* that has each and every permission in the system. Group settings can be imported and exported as xml files. (Figure 35)

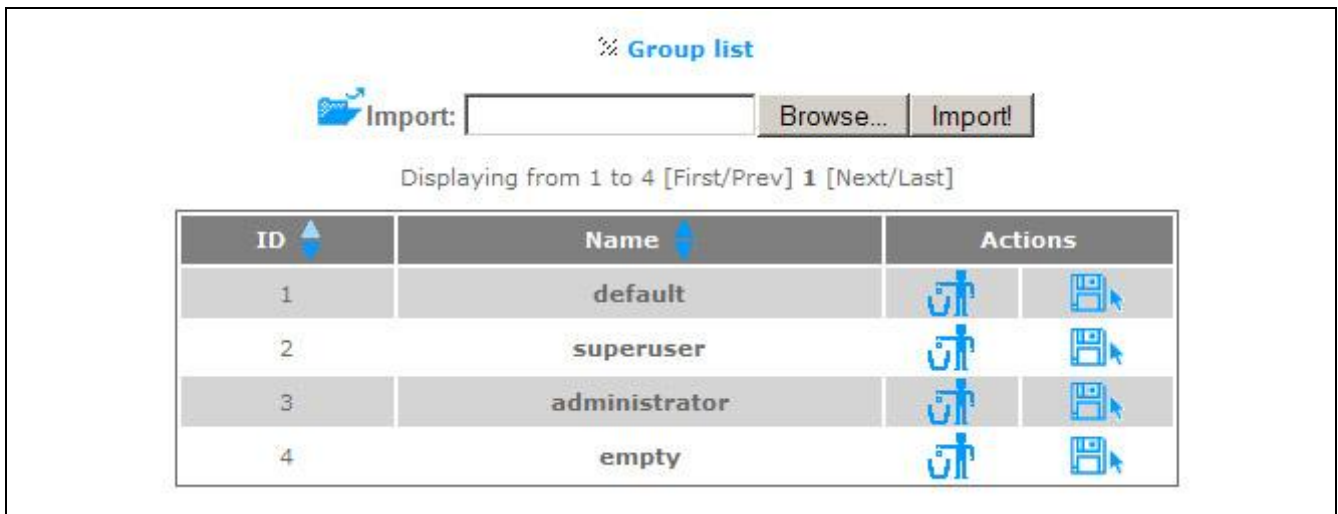


Figure 35 - Besides editing groups they can be imported through the Group list screen.

In order to create a new group Click *Create new group* in the navigation menu, type in a group name and provide the desired permission settings and press Submit. By selecting the just created group permissions can be added to it. Note that a user can belong to only one group. See below for detailed permission descriptions.

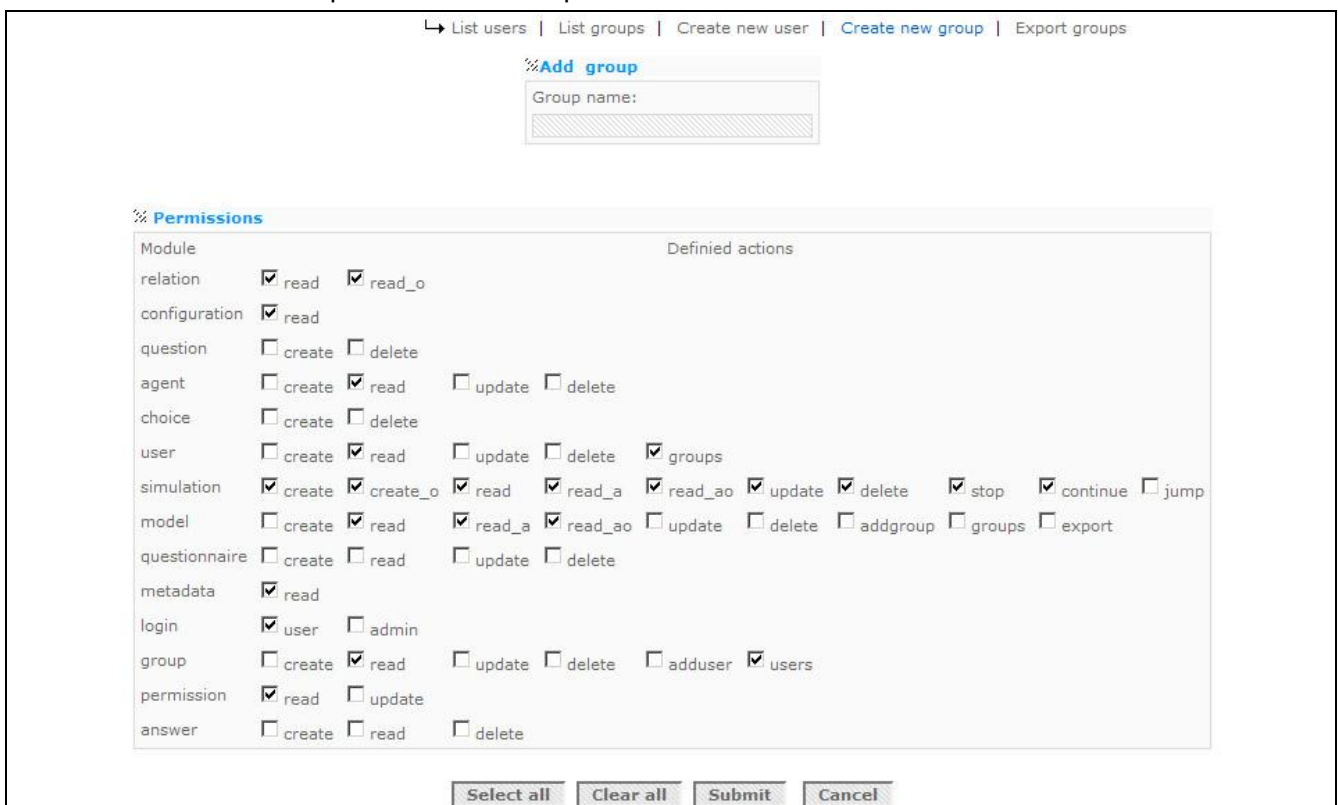


Figure 36 - Permission settings

4.3 Permissions

4.3.1 Relation

The *Relation* module stands for operations performed on relations between users and existing simulations. This basically means the listing of users playing various simulations built from models.

- Read: User is allowed to see other users assigned to simulations.
- Read_o: User is allowed to list all simulations he/she is allowed to play with.

4.3.2 Configuration

Determines whether the user can see PET configuration.

- Read: User is allowed to read configuration files.

4.3.3 Agent

The *Agent* module stands for operations performed on agents stored in the database while creating models. This has nothing to do with agents during simulation!

- Create: User is allowed to create an agent.
- Read: User is allowed to see data on agents. Without this the user is not allowed to see the agents neither on the admin, nor the user side.
- Update: User is allowed to update existing agents.
- Delete: User is allowed to delete existing agents.

4.3.4 Group, User and Permission

These modules stand for operations performed on users and workgroups.

4.3.4.1 User

- Create: User is allowed to create users.
- Read: User is allowed to see all users.
- Update: User is allowed to make changes in other users information.
- Delete: User is allowed to delete a user.
- Groups: User is allowed to see which workgroup a given other user is assigned to.

4.3.4.2 Group

- Create: User is allowed to create groups.
- Read: User is allowed to see existing groups.
- Update: User is allowed to make changes in group management.
- Delete: User is allowed to delete workgroups.
- Adduser: User is allowed to add a user to a group.
- Users: User is allowed to list all users in a group.

4.3.4.3 Permission

- Read: User is allowed to see other users' permission settings.
- Update: User is allowed to change other users' permission settings.

4.3.5 Metadata

Metadata module stands for operations performed on metadata created from imported configuration files.

- Read: User is allowed to read metadata files.

4.3.5.1 Login

Determines whether the user can login to the administrator pages and/or the user pages.

- User: User is allowed to login at the user side.
- Admin: User is allowed to login at the administrator side.

4.3.5.2 Model

The *Model* module stands for operations performed on models in the database.

- Create: User is allowed to create models.
- Read: User is allowed to see all models.
- Read_ao: User is allowed to see all models which are assigned to the group the user is assigned to.
- Read_a: User is allowed to see all models, including models assigned to groups the user is not assigned to.
- Update: User is allowed to change model settings. Note: If there is a simulation created from a given model, the settings can not be changed as long as the simulation exists.
- Delete: User is allowed to delete existing models. Note: If there is a simulation created from a given model, it can't be deleted as long as the simulation exists.
- Addgroup: User is allowed to add groups to models.
- Groups: User is allowed to see all users and groups allowed to see a given model. (Not available in current version.)
- Export: User is allowed to export models.

4.3.6 Simulation

The *Simulation* module stands for operation performed on simulations in existing models.

- Create: User is allowed to create simulations from existing models.
- Create_o: User is allowed to create simulations under his/her own name from existing models.
- Read: User is allowed to see all simulations not belonging to him/her and information concerning other users assigned to them.
- Read_ao: User is allowed to see simulations owned by his/her.
- Update: User is allowed to update simulation settings.
- Delete: User is allowed to delete simulations.
- Stop: User is allowed to stop simulations.
- Continue: User is allowed to continue paused simulations.

4.3.7 Questionnaire, Question, Choice and Answer

These modules stand for operations performed on while creating, answering and managing questionnaires. Note that questionnaires are not supported in this version of PET.

4.3.7.1 Questionnaire

- Create: User is allowed to create questionnaires.
- Read: User is allowed to see questionnaires.
- Update: User is allowed to change questionnaires.
- Delete: User is allowed to delete questionnaires.

4.3.7.2 Question

- Create: User is allowed to create questions.
- Delete: User is allowed to delete questions.

4.3.7.3 Choice

- Create: User is allowed to create choices.
- Delete: User is allowed to delete choices.

4.3.7.4 Answer

- Create: User is allowed to answer questions (fill out questionnaires).
- Read: User is allowed to read answers from other users.
- Delete: User is allowed to delete answers.

5 Global settings

PET has some settings which has affect on the global system. These settings are located under the *Settings* menu item as shown in Figure 37.

[Models](#) | [Simulations](#) | [User management](#) | [User Interface](#) | [Settings](#) | [Logout](#)

⌘ Global settings

Maximum number of concurrently running simulations:	<input type="text" value="10"/>
Maximum number of concurrently connecting applets:	<input type="text" value="50"/>
Delete the existing Repast based simulation if its tick count is greater than:	<input type="text" value="500"/>
<input type="button" value="Update values"/>	

Figure 37 The Settings page

These settings are:

- **Maximum number of concurrently running simulations:** Limits the number of simulations that may run concurrently in PET.
- **Maximum number of concurrently connecting applets:** Limits the number of applets that may connect concurrently to simulations in PET. Simulations can be in any state.
- **Delete the existing Repast based simulation if its tick count is greater than:** Restoring a Repast based simulation can be extremely slow when PET is starting. The time required to restore such simulations is linear with the simulation's tick count property (shows the simulation's progress). Therefore, to avoid endless application start this setting limits the restored repast based simulations by checking is tick count property. If this property value is greater the set value for this setting then the simulation is deleted.

6 Conclusion

PET is simulation software providing a quality platform for modelers to create participatory simulations in which individuals and synthetic agents can co-operate simultaneously. Its user interface is intuitive and easy to use allowing the users to learn managing the software quickly. The interface developed for administrators is easily manageable and the hierarchy concept of model families, models and simulations is intuitive. This document describes the administrative tasks and introduces the system from the administrator's aspect.

7 Appendix A - Model Descriptions

This appendix describes the model-families and their configured model instances coming with the default system installation.

7.1 Fire

This model is a simple example of simulating emergency situations. The model's 'world' is a building with rooms that are separated by walls. All of a sudden a fire breaks out at a certain location in the building. Fire spreads at a constant rate, but walls slow it down a little. Everybody in the building tries to find an exit to flee. Some succeed, but usually some don't. We provide the model with 2-D and 3-D visualization for the agents and space. The simulated agents don't know the location of the exits and only have a limited vision. However, they follow each other, try to move away from fire and explore their neighborhood.

Two somewhat different ready-to-run models are provided with PET under the Fire model family:

1. *Watch the Fire*: You can be an observer of a spreading fire and the agents trying to escape from a building seen from above (2D visualization).
2. *Escape the Fire*: You are one of the agents trying to escape the fire. The building is displayed from within (3D¹ visualization)!

7.2 Hunter

Hunter is a basic model for two types of agents (technically three, as PET considers space as an agent too). Some agents are *hunters* while others are *preys*. They "live" in a two-dimensional periodic space. As their names imply, the hunters try to catch preys, meanwhile preys will try to avoid them. When a hunter catches its prey, the prey disappears and is replaced to a random location in the space, or removed from the game depending on model configuration. Hunters and preys are very similar in their behavior to some extent, they both look for the (closest) opposite type of agents each turn, but hunters move toward preys, while preys move away from hunters.

The programming of the Hunter model is described in detail in *PET Tutorial!*

Two somewhat different ready-to-run models are provided with PET under the Hunter model family:

1. *Watch the Hunt*: You are an observer of the hunt.
2. *Don't Be a Prey!*: You are controlling one of the preys in the hunt. Try to avoid the hunters!

7.3 Zerosum

The Zerosum (Huber and Kirchler; Yoneda et al) model is a simple information-based zero-sum game on the futures market. It reveals a basic fact that the most and least informed players in the market earn more, than the mildly informed ones.

7.3.1 Mathematical model

The game is played by $2M + 1$ players.

1. At the beginning of each round Player i ($0 \leq i \leq 2M$) shows his reservation price R_i , by which he makes the following contract: he buys (sells) a unit of a future product if R_i is higher (lower) than the actual price P .

¹ Running 3D visualizations launches an external Java application. Please allow for the visualization applet to download before the first run in 3D. Note that users might experience problems while running 3D visualizations with some video settings.

2. The auctioneer gathers all the reservation prices and declares the median of them as P so that demand equals supply in the futures market.

3. After the future market is closed, the true value of the commodity V is revealed. It is determined exogenously as the sum of $2M$ stochastic variables: X_1, X_2, \dots , and X_{2M} , which are determined to be 0 or 1 independently with one another with the same probability 0.5 for each round.

4. Those who bought (sold) the commodity in the future market must sell (buy) it in the spot market at the true value V to close their accounts. Hence each player's profit is determined as soon as V is revealed. Those players with $P < R_i$ buy the commodity at P in the future market and sell it at V in the spot market so that they each earn $V - P$ of profit, while those players with $R_i < P$ sell the commodity at P in the future market and buy it at V in the spot market so that they each earn $P - V$ of profit. Needless to say, profit is loss if it is negative and the sum of all profit is always zero: $\Sigma(P - V) + \Sigma(V - P) = 0$.

We assume that before he determines R_i , Player i can correctly see the first i values of X_1, X_2, \dots , and X_i (Player 0 cannot predict any of them). Apparently Player i has an advantage over Player j ($j < i$), for the former knows what the latter knows (X_1, X_2, \dots , and X_j) as well as what the latter does not (X_{j+1}, X_{j+1}, \dots , and X_i). Hence, we will refer to i as "information level".

7.3.2 Short analysis

Player $2M$, the most informed player who can see every X_i , most probably earns positive profit in the long run. In fact he can choose ΣX_i as his reservation price R_{2M} ; then, as is readily checked, he earns zero profit if P happens to be equal to V or positive profit in more plausible cases where $P \neq V$. However, Player 0, the least informed player who knows none of X_i may not suffer negative gain in the long run. He may always continue to choose such a high (low) R_0 that makes him buy (sell) in the future market for every round or he may choose R_0 randomly for each period. In either case he is able to expect zero profit in the long run, because $P < V$ and $V < P$ is realized with the same probability.

A question emerges from these special circumstances, i.e., from the fact that the game is zero-sum. If the most informed player can expect a positive profit while the least-informed player can expect zero profit, some middle-informed player must suffer a loss. Yet it seems strange that a player who is better informed earns less profit than a player who is less informed.

7.3.3 Agent strategies

The agents play the following two simple strategies:

The expected-value strategy:

$$R_i = (\Sigma X_i) + 0.5 \times (2M - i)$$

Here the right-hand side represents the value of V that Player i expects; the first term is the sum of the stochastic variables, whose values Player i knows, while the second term stands for the expectation of the sum of X_{i+1}, X_{i+2}, \dots , and X_{2M} , whose values he does not know.

The extreme strategy:

$$R_i = (\Sigma X_i) \text{ with probability } 0.5$$

$$R_i = (\Sigma X_i) + (2M - i) \text{ with probability } 0.5$$

Here (ΣX_i) represents the minimum value of V that Player i expects while $(\Sigma X_i) + (2M - i)$ stands for its maximum value that Player i expects.

Four somewhat different ready-to-run models are provided with PET under the Zerosum model family:

1. *One against the market 1:* All the agents are controlled by the computer you are an observer on the market. This is a one against the masses setting, where all but one agent plays the expected-value strategy.
2. *One against the market 2:* This is basically the same as 'One against the market 1' but the strategies are transposed, most agents play the extreme strategy.
3. *Control the market 1:* Users can control agent number 0, 10, 20, 30, 40, 50, 70, 80 or 90 out of 100 agents. All the other agents play the expected-value strategy.

4. *Control the market 2*: This is about the same as the previous one, but the artificially controlled agents play the extreme strategy.

In the latter two simulations users can control agents, hence make pricing decisions in the market.

7.3.4 References for Zerosum

Jürgen Huber and Michael Kirchler "The Value of Information in Markets with Heterogeneously Informed Traders – and Experimental and a Simulation Approach", presented at 9th Workshop on Economics and Heterogeneous Interacting Agents(WEHIA2004), Kyoto University, Kyoto, Japan

Hiroyasu Yoneda, Gen Masumoto and Sobei H. Oda: „Marginal Contribution of Information to Profit in a Zero-sum Game“, presented at *EES2004: Experiments in Economic Sciences - New Approaches to Solving Real-world Problems*

7.4 Repast HeatBugs

This model family demonstrates how easy to run an existing Repast model for PET.

The original heat bugs simulation consists of heat bugs - simple agents that absorb and expel heat and a heatspace, which diffuses this heat into the area surrounding the bug. Heat bugs have an ideal temperature and will move about the space in attempt to achieve this ideal temperature. The simulation has a display on which the user can see how bugs are moving and the space as diffusing heat.

7.5 Configurable Repast HeatBugs

The Configurable Repast HeatBugs is the same as Repast HeatBugs except that - as its name suggests - it demonstrates the configuration possibility of agents via the administration interface. Models derived from this model family can be configured with arbitrary number of heat bugs agent and each heat bug can be configured uniquely.