



## **JAS library**

*Java Agent-based Simulation library*

Version 1.0

## **User's Guide**

<http://jaslibrary.sourceforge.net>

Michele Sonnessa  
([sonnessa@di.unito.it](mailto:sonnessa@di.unito.it))

## **Copyright Notice:**

*JASLibrary v.1.0 - A Java Agent-based Simulation Library  
Copyright 2002-2004 Michele Sonnessa*

*This library is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation; either version 2.1 of the License, or (at your option) any later version.*

*This library is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details.*

*You should have received a copy of the GNU Lesser General Public License along with this library; if not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA*

*Michele Sonnessa  
University of Torino  
Department of Computer Science  
Corso Svizzera 185  
Torino - Italia  
([sonnessa@di.unito.it](mailto:sonnessa@di.unito.it))*

## Index

JAS library	<b>1</b>
User's Guide	<b>1</b>
Introduction	<b>4</b>
The JAS features	5
The JAS engine	5
The external libraries	5
Installing JAS	<b>7</b>
Installing the sources and recompiling JAS	<b>8</b>
The JAS simulation environment	<b>10</b>
The JAS tool bar	11
The JAS menu	12
The File menu	12
The Simulation menu	12
The Tools menu	13
Help	13
The JAS Project Properties dialog box	14
The JAS Project tree panel	16
The JAS engine status dialog box	16
The JAS Properties window	19
The JAS built-in graphical output console	20
The JAS tools	<b>21</b>
The Parameter bug editor	21
The JAS graph editor	22
The HSQL Database manager	23

## Introduction

JAS is a simulation toolkit, specifically designed for agent based simulation modelling. Agent based models are representations of dynamic social systems realised through object-oriented computer programs. For a detailed introduction on ABM see the working papers section at <http://www.santafe.edu>.

JAS is a clone of the Swarm library, an ABM framework originally developed by the Santa Fe Institute for creating multi-agent simulations of complex adaptive systems. Until recently the Swarm project was based at the Santa Fe Institute. The development and management of Swarm is now under control of the Swarm Development Group. (<http://wiki.swarm.org>)

The libraries were originally written in Objective C. In order to make them more standard the SDG developed a Java extension, too.

According to the Swarm experience, we can assert that the most promising approach to develop AB models is represented by the use of object-oriented programming (OOP). So JAS does not define a domain-specific language: it provides the user a collection of ready to use widgets and a set of rules of thumbs to build such kind of simulations.

The JAS tools can be easily embedded in the users' models, reducing the code complexity.

Building JAS, we did not want to create a brand new way of building agent based models, but somewhat a rich and open collection of libraries to help researcher to create models and share them in an easy way. Many of the library contained in JAS have been based on open source, reliable and well tested third-party libraries.

In the JAS architecture, agents are organized and managed by fundamental components, called *models*. A model is a Java class inheriting from *jas.engine.SimModel*, creating a collection of agents with a schedule of events over those agents. JAS is able to execute more than one model at time. In fact the scheduler is unique and each model shares it. This allows to create complex structures whereby agents of different models can interact with each other.

A good Swarm tutorial by Hala Al-Bakour and Sheri Markose<sup>1</sup> says that:

The Swarm architecture is based on an internal model Swarm and an external observer Swarm. These two aspects of the artificial world is clearly separated in the Swarm system. The objective of the special 'observer' agents is to observe other objects via the probe interface. These objects can provide both real-time data presentation and storage of data for later analysis. The observer agents are actually swarms (a group of agents and a schedule of activity) and a complete experimental framework is obtained by combining the model and observer apparatus.

The distinction between model and observer is a very useful paradigm and JAS protocol suggests to use this approach, too. Technically it can be implemented simply defining two models (two classes inheriting from *SimModel*), with one of them playing the role of the observer.

The present guide suggests a step-by-step procedure in building agent-based models with JAS. Following the suggestions the user might obtain benefits in models portability.

---

<sup>1</sup> [http://www.essex.ac.uk/ccfea/swarm/SwarmTutorial/web/documents/swarm\\_tutorial.htm](http://www.essex.ac.uk/ccfea/swarm/SwarmTutorial/web/documents/swarm_tutorial.htm)

## The JAS features

Actually JAS provides the following features:

- It is based on a discrete-event time simulation engine.
- Thanks to a custom Java class-loader, JAS can load models without configuring the CLASSPATH environment variable.
- It is able to represent different time units (ticks, seconds, minutes, days...).
- It is equipped with a real-time engine, which can be used to implement emulation models. It is able to fire events using the real computer timer.
- It supports the XML for data input/output operations and the SVG graphic format.
- It provides a genetic algorithms library as well as an artificial neural networks one (classifier systems are *still under construction*).
- JAS provides an implementation of the [Sim2Web](#) architecture: a JAS-Zope bridge for web publishing of simulations and remote users interaction.
- The MultiRun class manages repetitive executions of a model in order to support automatic parameters calibration.
- It provides a powerful statistical package, based on the *cern.jet* package. Statistical data can be automatically collected in a database, thanks to the JAS database features.
- [Hypersonic database](#) is built-in the JAS package.
- The brand new *jas.graph* package allows to manage relational agents, supported by built-in Social Network Analysis functions.

## The JAS engine

The core of the JAS toolkit is represented by the simulation engine. It is based on the standard discrete-event simulation paradigm, which allows to manage the time with high precision and multi-scale perspective. We like to stress the important difference between discrete-time and discrete-event time representation. The tools implementing discrete-time engines, like Swarm, are particularly fast in models characterized by loops of events. They are discrete representation of continuous time models. When the events happens at different time scales the discrete-time representation is more inefficient than the discrete-event paradigm.

Thanks to its discrete-event engine, JAS represents a good compromise in simulating both discrete and continuous agent-based models.

## The external libraries

JAS project is an example of the interoperability and code re-utilisation the Open Source world makes possible: it is useless to reinvent the wheel every time we need a functionality!

The JAS library contains the following third-party libraries:

- The PtPlot library (<http://ptolemy.eecs.berkeley.edu/java/ptplot/>) is used to plot variables over time. JAS provides some wrapper classes mapping the statistical objects' interface to the plotters.
- The COLT library (<http://hoschek.home.cern.ch/hoschek/colt/>) is the random number generation engine used by JAS. Moreover it is largely used in statistical computers.
- The Metouia Look&Feel library (<http://mlf.sourceforge.net/>) is the look & feel JAS GUI is based on.

- The Apache SVG Batik library (<http://xml.apache.org/batik/>) provides JAS the capability to generate SVG images. It is used by the Sim2Web implementation (<http://wf.econ.unito.it/sim2web>).
- The JExcelApi library (<http://www.andykhan.com/jexcelapi/index.html>) is used to access Microsoft Excel spreadsheet. It is particularly useful to retrieve simulation parameters from a spreadsheet.
- The JGraphT library (<http://jgrapht.sourceforge.net/>) has been used to develop the JAS graph implementation.
- The Apache XML-RPC library (<http://ws.apache.org/xmlrpc/>) is used int JAS implementation of the Sim2Web architecture.
- The HSQLDB library (<http://hsqldb.sourceforge.net/>) is the standard database engine used by JAS.

## Installing JAS

Before downloading and installing JAS, it is necessary to check that the Sun's Java Development Kit (JDK) is present on the target machine and that its version is the 1.4 or higher.

JAS may run also with the Java Runtime Environment (JRE), but this solution does not permit to compile the simulation models.

Both the JDK and the JRE are available at the Sun Microsystem's web site: <http://java.sun.com>.

With the java environment installed on the target machine, the user can install JAS, simply unpacking the zipped jas package to a target directory.

Once JAS has been unpacked, the target directory will contain a root directory called 'JAS'. The JAS root directory is organized as the tree in figure 1.

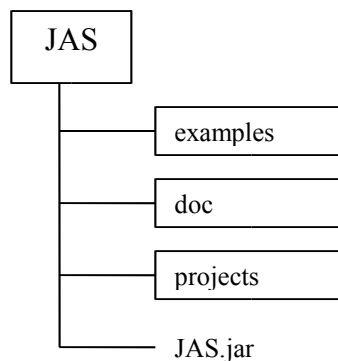


figure 1 The JAS directory tree

The *JAS.jar* file represents both the JAS application and the collection of libraries used by simulation models.

The *examples* subdirectory contains the standard example models.

The *doc* directory contains all the documentation, with the standard API of the JAS library and the APIs of the third-party libraries, distributed with JAS. All the documentation is directly accessible from the JAS program menu.

The *projects* subdirectory is a repository for the user's models. See the @@DIR

Once JAS is installed it can be executed typing the following command from the system console at the JAS root directory<sup>2</sup>.

```
java -jar JAS.jar
```

---

<sup>2</sup> The standard Java installation often does not set the java binaries directory in the machine general path,. If the command does not work, please append to the PATH environment variable the path to reach the bin directory under the java main directory. See the java documentation for more details.

WINDOWS USERS ONLY: In Windows environments java executables should be also executable by double-clicking on the jar file.

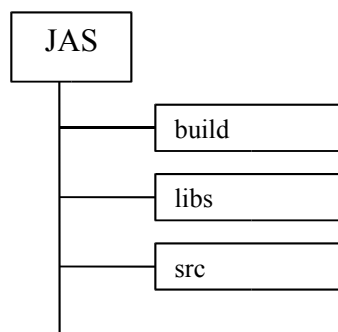
## Installing the sources and recompiling JAS

JAS can be compiled starting from the latest sources. The sources can be downloaded from the JAS download site or retrieved from the CVS repository.

Suppose JAS is already installed. We will call <JAS> the JAS main program directory. The following procedures describe how to obtain the sources using the original source package or the CVS server:

- download the source compressed package and expand it in the <JAS> directory;
- check out the “JAS” module from the `cvs.sourceforge.net:/cvsroot/jaslibrary` CVS server into the parent of the <JAS> directory, if you want to have a nightly build.

After both the above operations, the user will find the following new directories in the <JAS> directory tree:



- The *build* directory contains the *build.xml* file which is the make file used by the Java Ant system to compile java sources.
- The *libs* directory contains the third-party libraries which are included into the final *JAS.jar* package.
- The *src* directory contains the java source files of JAS.

In order to compile JAS it is necessary to install the Apache Java Ant make system. It is available at the <http://ant.apache.org/> website.

Once Ant is installed, go to the <JAS> directory and type the following commands from the system terminal:

```
cd build
ant
```



JAS has been developed with the IBM Eclipse IDE, an open source Java IDE. You can download it at <http://www.eclipse.org>. Eclipse is able to interpret the ant commands and through it the user can compile JAS without using the Java Ant.

In order to develop JAS with Eclipse, it is simply necessary to create a new project and point it to the <JAS> directory. Eclipse will automatically find all the dependencies.

## The JAS simulation environment

The JAS simulation environment is an application, through which the user can execute simulation experiments, loading models and controlling their execution.

Although a model may be executed defining a *main* method and using it as any other Java application, JAS is able to manage XML project files that are used to automatically load and execute simulation models.

An XML project file contains the path where the binaries of the model are placed, some important parameters and the list of the models to be executed. These are the information JAS needs to load a model and control its execution.

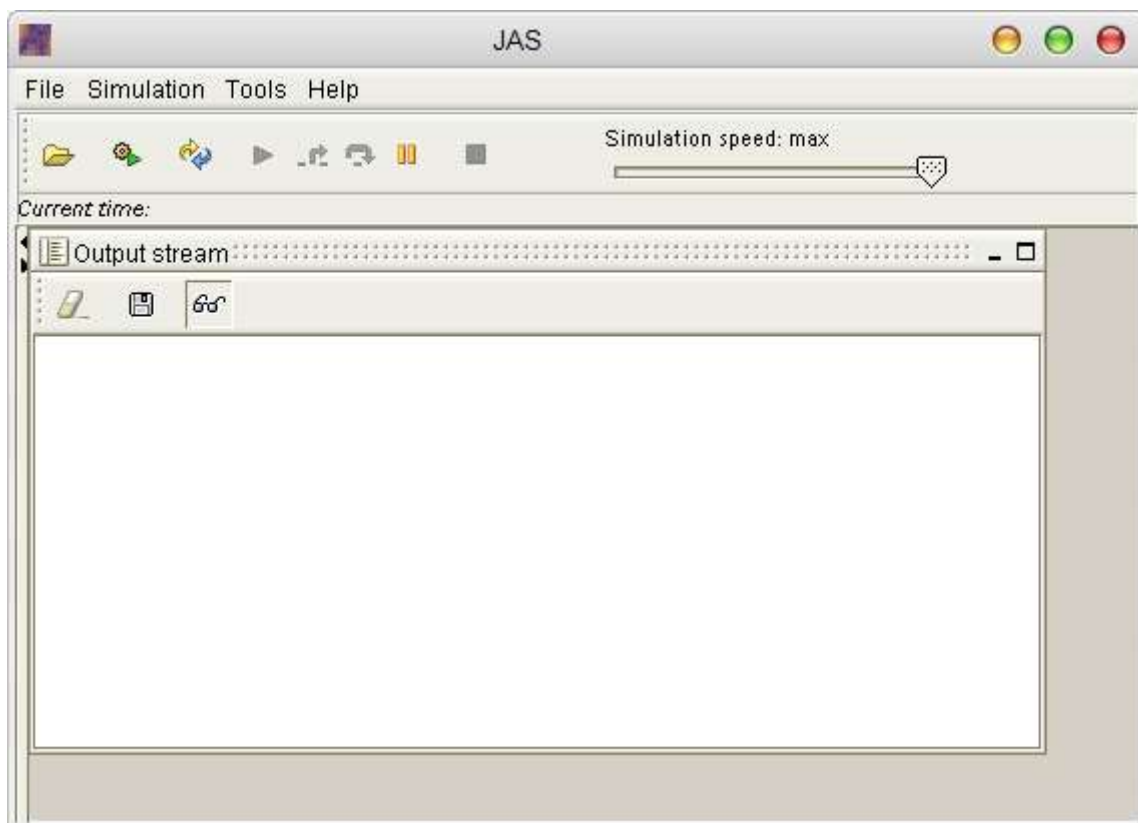










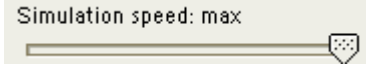
figure 2 The JAS simulation environment

The main panel of JAS is shown in figure 2. The simulation environment is made of:

- a menu through which the user may access all the program options,
- a tool bar with some shortcut to mostly used commands,
- a slider command to control simulation speed,
- a bar indicating the current simulation time,
- an output console window, grabbing and showing all the messages coming from the system output streams (*stdout* and *stderr*),
- a content pane, containing the windows created and managed by the simulation model.

### The JAS tool bar

The tool bar contains the following buttons:

	"Open project" button opens a simulation experiment project file from disk
	"Build models" button orders to JAS to invoke the <i>buildModel()</i> method of each currently loaded model.
	"Rebuild models" button order to the JAS engine to dispose the current simulation and return to the initial conditions.
	"Start simulation" button starts the simulation engine, executing the models until a stop event is raised.
	"Make one step" button asks JAS engine to raise the next event scheduled in the event list.
	"Go to the next time unit" button causes the engine to execute all the operations scheduled at current time. If more than one events are scheduled at the same time, this command cause their execution, while pressing the "Make one step" button JAS executes only the first of them.
	"Pause simulation" button puts the simulation engine in a paused mode. The experiment may continue by pressing one of the start/step buttons.
	"Stop simulation" button pauses the simulation and terminate the experiment. Technically the <i>simulationEnd()</i> method of the current models is called. The simulation may be anyway continued pressing "start" but its behaviour will depend on how the modeller managed the <i>simulationEnd()</i> method.
	<p>To the right of the execution buttons the tool bar provides a slider control which allows the user to control the execution speed. The speed can be set to a value between 0 and 200 milliseconds. When the slider is set to 0, JAS executes the simulation at the maximum computer speed.</p> <p>Where it is set to a value greater than 0, JAS waits for the specified number of milliseconds before raising each event in the event list.</p>

At the bottom of the tool bar there is a panel indicating the current simulation time. It is expressed using the current simulation time unit.

**The JAS menu**

The following paragraphs describe the commands in the main menu.

**The File menu**

<b>Submenu</b>	<b>Description</b>
New project	creates a new XML project file. Through it, JAS creates an XML file containing the models to be loaded, the class paths, the simulation parameters and a short description.
Open project	opens a previously created project file, with the extension “.sprj.xml”. During the opening of the project file, JAS loads models into memory and invokes their <i>setParameters()</i> method.
Save project	saves the current project. If it has been just created, JAS will ask where to save the file, otherwise it overwrites the existing one.
Save project as	saves the current project within a new file
Close project	disposes each running model and close the current project file.
Edit project	opens the “project properties“ windows, through which the XML project file can be edited.
Compile project	this command execute the “javac” command passing to it the class paths to recompile the current project. If the java development kit has not properly configured this command may not work. In fact, the javac command must be accessible from the terminal (or command prompt) window from everywhere in the file system. See the installation section for more details. The results of the compilation are shown in the console output.
[Most recently used list]:	JAS stores the last five opened projects. Simply clicking on one of them the project is automatically processed by JAS.
Quit	closes the application

**The Simulation menu**

<b>Submenu</b>	<b>Description</b>
Build	makes JAS to invoke the <i>buildModel()</i> method of each currently loaded model.

Restart	closes each running model and reload it. During this operation JAS invokes the Java Garbage Collector to free the unused memory and flush the class loader cache, in a way that models are reloaded from the file system. This means that if the model has been recompiled while it is in the Java memory, pressing the reload menu item the last version of the classes will be loaded in memory. After a restarting, the global variable <i>Sim.currentRunNumber</i> is increased. It allows the programmer to manage the storing of data to a file, by keeping track of the different runs.
Play	starts simulation
Step	see the "Make one step" button in the tool bar
Time step	see the "Go to the next time unit" button in the tool bar
Pause	see the "Pause simulation" button in the tool bar
Stop	see the "Stop simulation" button in the tool bar
Show engine status	see the "Show engine status" button in the tool bar.

### The Tools menu

Submenu	Description
Parameter designer	opens the "parameter designer tool", which is used to design XML parameter bag files. See the developer guides for more details.
Graph editor	opens a graph editor window. See the developer guides for more details.
Database editor	opens the HSQL database manager. See the developer guides for more details.
JAS options	opens a "parameters windows", from which is possible to set a particular directory where to store simulation models.

### The Help menu

Submenu	Description
API documentation	opens the Application Programming Interface documentation of JAS in the standard javadoc format.
API libraries	contains the API documentation for the libraries distributed with the JAS package.
JAS web site	points to the official JAS web site
About	opens the about box, where there are licensing information, a list of current java environment variables and the number of the current JAS engine version.



### The JAS Project Properties dialog box

The “Project properties” windows can be called by clicking on the “File/New” menu item, to build a new XML project file. In order to edit a previous created file it is necessary to open the project and then click the “File/Edit project” menu item.

The figure 3 shows how the dialog window appears when the user creates a new project. The field “Project name” contains the name of the project.

The “Time unit” combo box specify which time unit JAS must use when it loads this project. the During the simulation, the time will be expressed using the chosen the chosen time unit. It is important to notice that using the absolute time representation in scheduling events, the time unit does only affect the format in which time is represented by the JAS Control Panel.

Otherwise, in case the events are scheduled using the methods of the *jas.engine.SimTime* class the distance between two events is measured using the selected time unit. See the “Time representation” section for more details about time management in JAS.

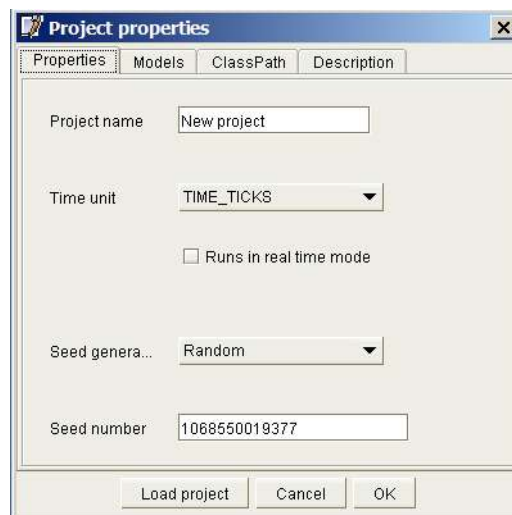


figure 3 The JAS Project properties window.

The “Runs in real time mode” check box allows the user to specify which event list manager JAS will use during the simulation. When real time mode is enabled, events scheduled in specific time in the future will happen only when the computer’s clock is equal or greater than the scheduled time. The real time is very useful to execute simulation models interacting with real interfaces as humans or robots or other computers on the network.

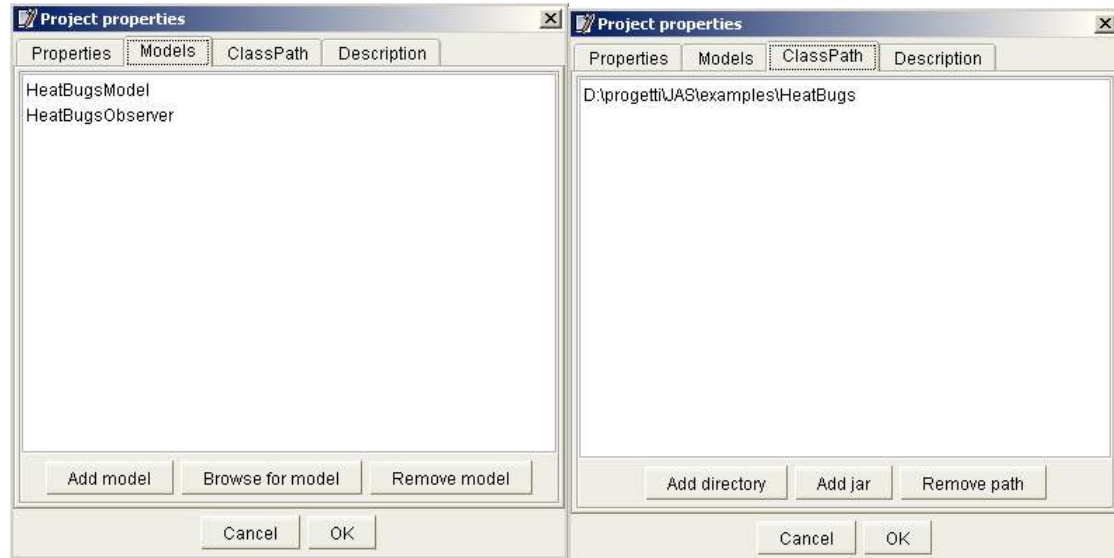
The “Seed generation” combo box is used to set the way the random number generator used by the Sim class initialise the seed number. If the user sets the “Random” mode, the next parameter is ignored and the seed number is set randomly each time a model is loaded in memory.

Otherwise, if the generation mode is “Fixed”, the seed number set at the beginning of each simulation will be set to the specified “Seed number”.

The “Models” and “ClassPath” tabs, shown in figure 4, are used to tell JAS which models the engine has to manage during the simulation and where to find the compiled classes (the .class files).

In case the model has not been typed yet, the user can use the “Add model” button, specifying the name of the model. JAS asks the user if he/she wants that JAS generates the skeleton code for the new class.

Usually the creation of the project is made, after the code has been typed and compiled yet. In this case the user can select the model browsing the file system.



**figure 4** The models and class paths.

Usually the creation of the project is made, after the code has been typed and compiled yet. In this case the user can select the model browsing the file system.

If JAS did not add automatically the class path in the “ClassPath” list, it is necessary to add all the directories where the classes used by the simulation are stored.

If the model makes use of some Java libraries, it is possible to add jar files to the class path. The declaration of the paths where classes are is the way JAS’ dynamic class loader searches for binaries during the simulation.

Usually Java coders have to put the paths to find classes in the system environment variable called “CLASSPATH”, in a way the Java default class loader could find the needed binaries.

This operation is boring and requires to know which models you want to execute before starting the Java Virtual Machine.

JAS bypass the problem using the XML project file and its custom class loader.

Once a new project has completely described through the project dialog box, the user can save directly the project or try to load it in memory to test if it works, pressing the “Load project” button.



### The JAS Project tree panel

When the user loads a project file, JAS shows on the left side of the simulation environment a tree panel similar to the one in figure 5.

The project tree panel gives the user a short view on:

- the list of java files contained in the directories specified by the ClassPath section in the project file. Each file can be directly edited by double-clicking on it. JAS is not equipped with a code editor, although it allows the user to define a default code editor in the 'JAS Options' window (See JAS options description for more details);
- the list of models specified the Model section of the project file. By double-clicking on a model item JAS opens a probe on the current instance of the model;
- the list of windows whose position is currently managed by JAS (the windows added using the *addSimWindow()* method in *buildModel()* method).

The tree panel can be hidden or resized dragging the bar on the right or using the arrows on the top of the bar.

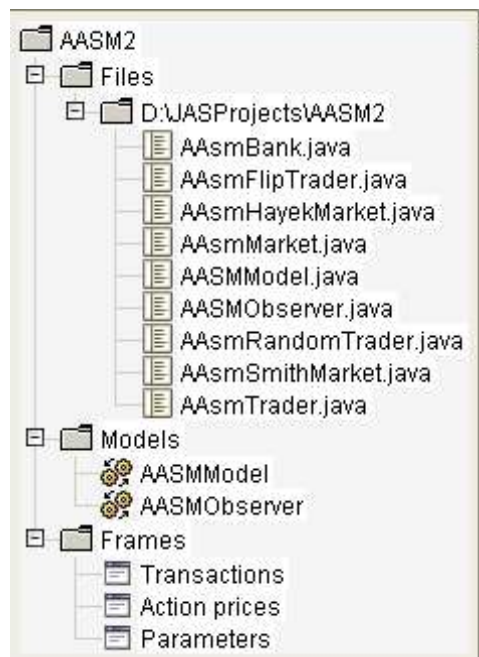


figure 5 A view of the project tree panel.

### The JAS engine status dialog box

The JAS simulation environment is a graphical interface interacting with the JAS simulation engine. The *jas.engine.SimEngine* class is manages the execution of the simulation model, so it is very important to have the possibility to look at its current status.

The engine status dialog box, shown in figure 6 and in figure 7, gives the user a lot of information. It monitors the seed number used to synchronize the random generator at the beginning of the simulation and it can be changed directly, modifying the value or generating a new one through the "Generate seed" button.

The “Time unit” combo box shows the current time unit and allows the user to change it. Let’s notice that changing the time unit may result in a unpredictable results if the model is based on a specific time representation.

The “Run” field counts how many times the current model has been restarted. It may be changed manually and it is useful for the modeller to store simulation outcomes in different output files using the current simulation run counter.



figure 6 The engine status dialog box

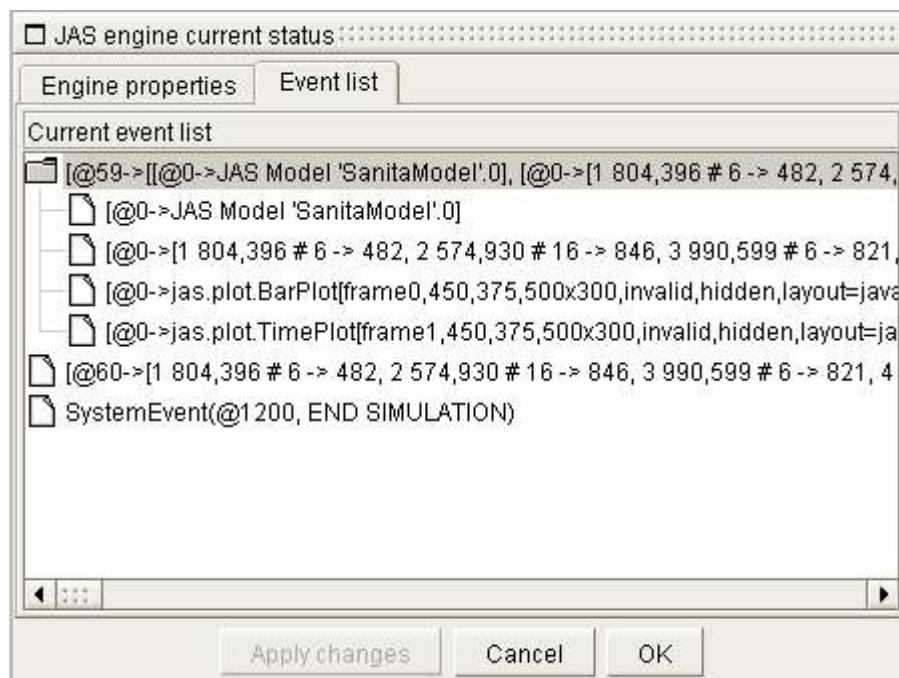


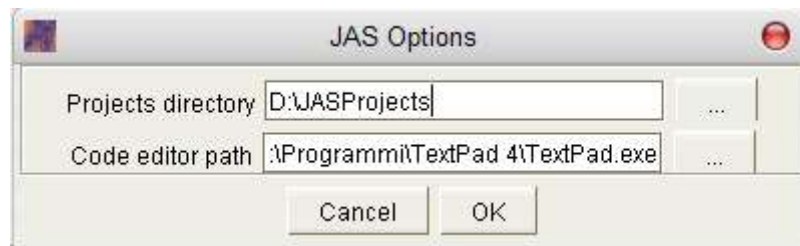
figure 7 The event list status

The “Event list” tab shows the current status of the engine core object: the *eventList*. Through this view the user can check which is the sequence of events will be fired at the next simulation step. The events are represented by a string beginning with the @ character indicating the absolute time of the event and a string relative to the instruction will be invoked during the event firing.

In case the event is a group of events (*jas.events.SimGroup*) the event is represented by a folder containing the list of the events in the group.

When this dialog box is opened the simulation engine is stopped and the execution can be performed only step by step, pressing the step button: this is due to synchronization problems during the inspection of the *eventList* by the graphic viewer.

### The JAS Properties window



The JAS properties window is accessible clicking on the Tools\JAS options menu item.

JAS comes with a *projects* directory within its installation path, even if the user can define another path to store her own projects.

The definition of a project file is useful when a model has to be send to another user. In fact, JAS stores relative paths to the file when they are under the JAS root directory or when they are under the user projects directory. Saving a model outside these folders, the XML project file will contain absolute paths, causing some problems when the model is copied on other machines.

So if the user saves his/her models in the JAS\projects directory the “Projects directory” option is not to be set, otherwise the user has to specify the path for the custom folder.


Through this window the user can set the default application used to edit the source code when she/he clicks on a file from the project tree.

**NOTICE:** The external source code editor is opened by JAS passing to the operating system a command with the executable path and the file to be updated, as parameter. If the editor does not accept parameters from the command line it cannot be used with JAS.

### The JAS built-in graphical output console




The output console grabs the standard I/O streams and shows them in the built-in text area. The java *System.in* and *System.err* streams are redirected to the output console. So the JAS simulation environment is able to show the console outputs. This feature is particularly useful when JAS is executed via the `javaw` command which bypasses the system output console (actually this command is only available on Windows platforms).

The console appends continuously the outputs received from the streams. If the user wants to freeze the output windows, she has simply to press the  button. When the button with the glass icon is not in a pressed state the output windows stop grabbing system output.

The  button allows to clear the whole content of the window.

The  button allows to save the current content of the window to disk.

The output window cannot be disposed, but it can be iconified. Notice that when iconified the window continues in grabbing output, if the  button is pressed.