Action factory's VPE – v0.3b

Table of Contents

Table of Contents

Action factory's VPE − v0.3b

Visual programming tools

[PDF version]

Action factory's VPE is a set of graphical user interfaces, to help designing actions and action factories, and to visually design or execute a sequence of actions mostly defined as a "computational path".
All the tools are built on a toolkit, which should enable easy building of dedicated interfaces for those who need it.

Comments, bugs, fixes to *t_chevalier@libertysurf.fr*.

Requirements

[home]

Action factory's VPE is written in Python. For the GUI, it uses the Tk widget set, through the Tkinter python interface. It also uses the Python Mega−Widget set, which is a set of advanced widgets based on Tk for Python.
Finally, it uses Fnorb, which is a CORBA 2.0 compliant ORB for Python (mostly written in Python itself).

Now here's the blunt list of the products you'll have to install prior to using Action Factory's VPE:

- **Tcl vs 8.0** or later (I used 8.0.5) and **Tk vs 8.0** or later (I used 8.0.5)

[GPL product, available from www.tcltk.com or www.scriptics.com]

- **Python vs 1.5.2** (I didn't checked 1.6 nor 2.0)

[GPL product, available from www.python.org or www.pythonlabs.com]

- **Pmw vs0.8.4**

[GPL product, available from www.python.org or python ressource sites (Vault of Parnassus, Starship Python...)]

- **Fnorb 1.1** (Beware Fnorb is free for non−commercial use, but a low cost license is needed otherwise)

[available from Distributed Systems Technology Centre Pty Ltd. or python ressource sites (Vault of Parnassus, Starship Python...)]

Comments, bugs, fixes to *t_chevalier@libertysurf.fr*.

[Home]

Overall presentation

[home]

Introduction

Action factory's VPE will enable you to design an executable specification of your computational path, to test it, and then to generate the C++ ActionFactory CORBA wrapping required to easily implement your specification.
A further improvement, yet to be published, will enable generation of a Python or C++ batch process controller, equivalent to the interactive data−flow process controller graphically designed.

In the following discussions, we will define 3 different type of people :

- **Users**, which are mainly in charge of running numerical simulations and analysing the results
  They often prefer explaining their needs as a process (sequence of actions, executed in a specific order), which they would like to be automated (computationnal path / data−flow approach).
- **Integrators**, which are mainly in charge of providing users with automated 'meta−tools', perfectly suited to the problem they are facing, so that they may concentrate on the physics.
  These 'meta−tools' will be assembled from current available numerical simulation tools (modelers, mesh generators, solvers, post−processors, visualizers, databases front−ends...)
- **Developpers**, which are mainly in charge of providing the core of the simulation tools the integrators will wrap into components for a better usability.
  They often prefer structure their developments in modules and classes, to ease development and maintenance (object−oriented approach).

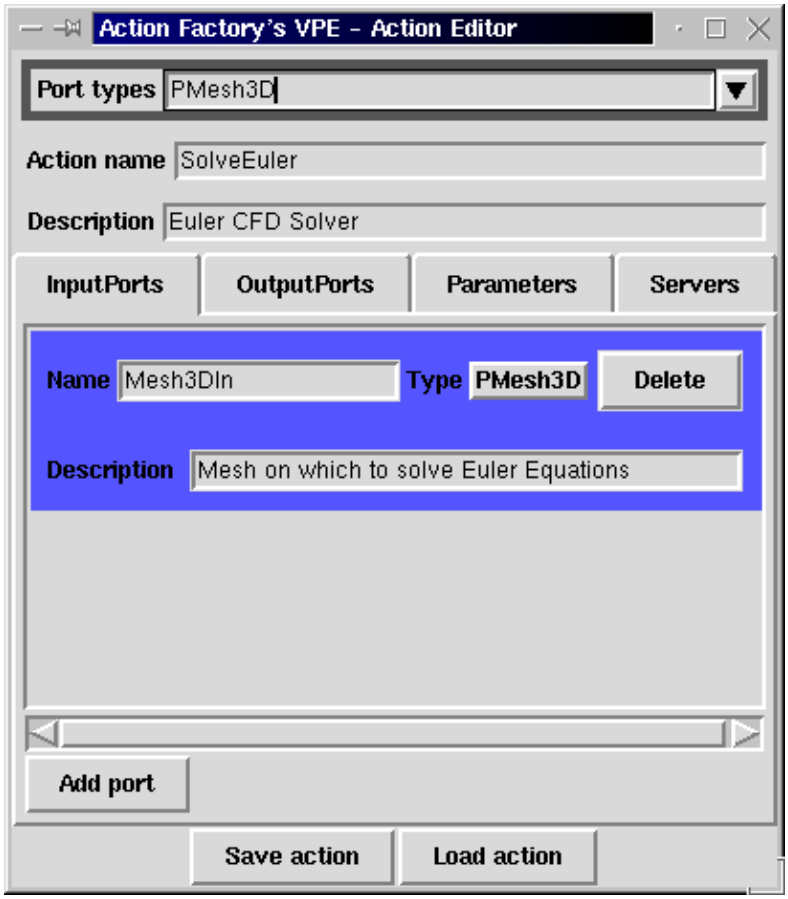
The Action Factory system is designed to help integrators filling easily the gap between **developpers** and **users**, by providing a clean data−flow to object oriented mapping, and enabling a quick and mostly automated way to build actions and computational pathes on top of existing developer's components.
While the Action Factory system is somewhat developer's oriented, the Action Factory's VPE is more user oriented. It is designed to help users and integrators to build and check actions (the steps of the process to be automated), to provide the Action Factory tools the specification they need.

Interestingly, as we will see, the specification will became directly executable as soon as the underlying implementation will be available.

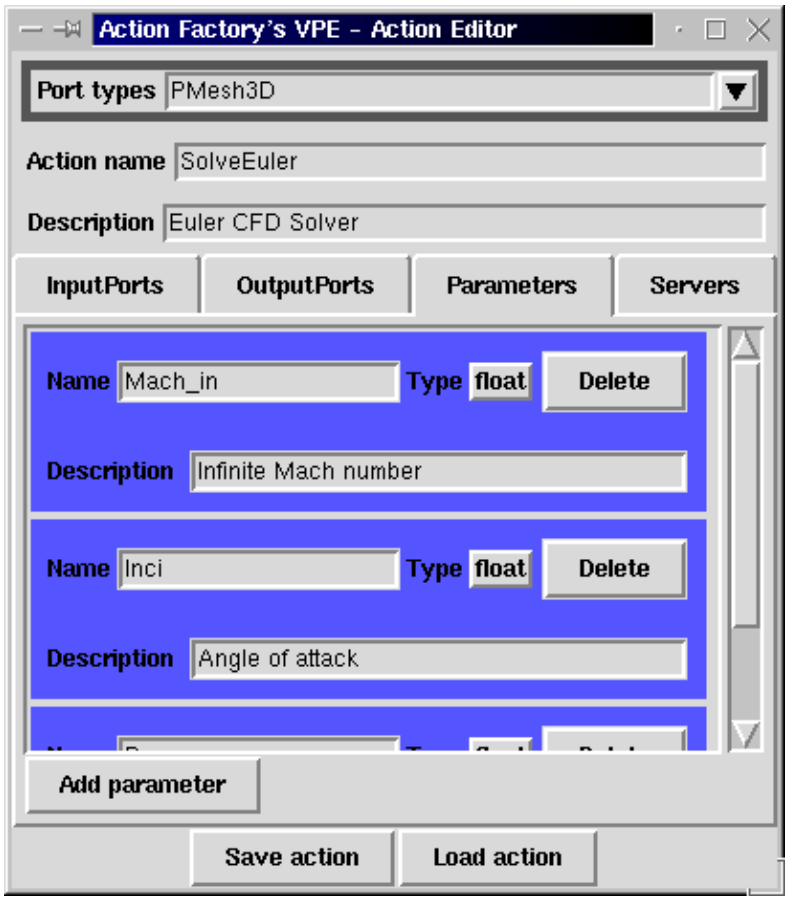
Designing the steps of the process

This step may be performed directly by users

First fill the name of the step in the process, ans its description
Then interactively specify incoming and outcoming data for this step of the process
In the ActionFactory's jargon, the step is called an **action**, and incoming and outcoming data are exchanged through typed input and output **ports** of the **action**.



Interactively specify parameters of that step of the process
In the ActionFactory's jargon, parameters of a step are called the **parameters** of the **action**
Then save your **action** description to file

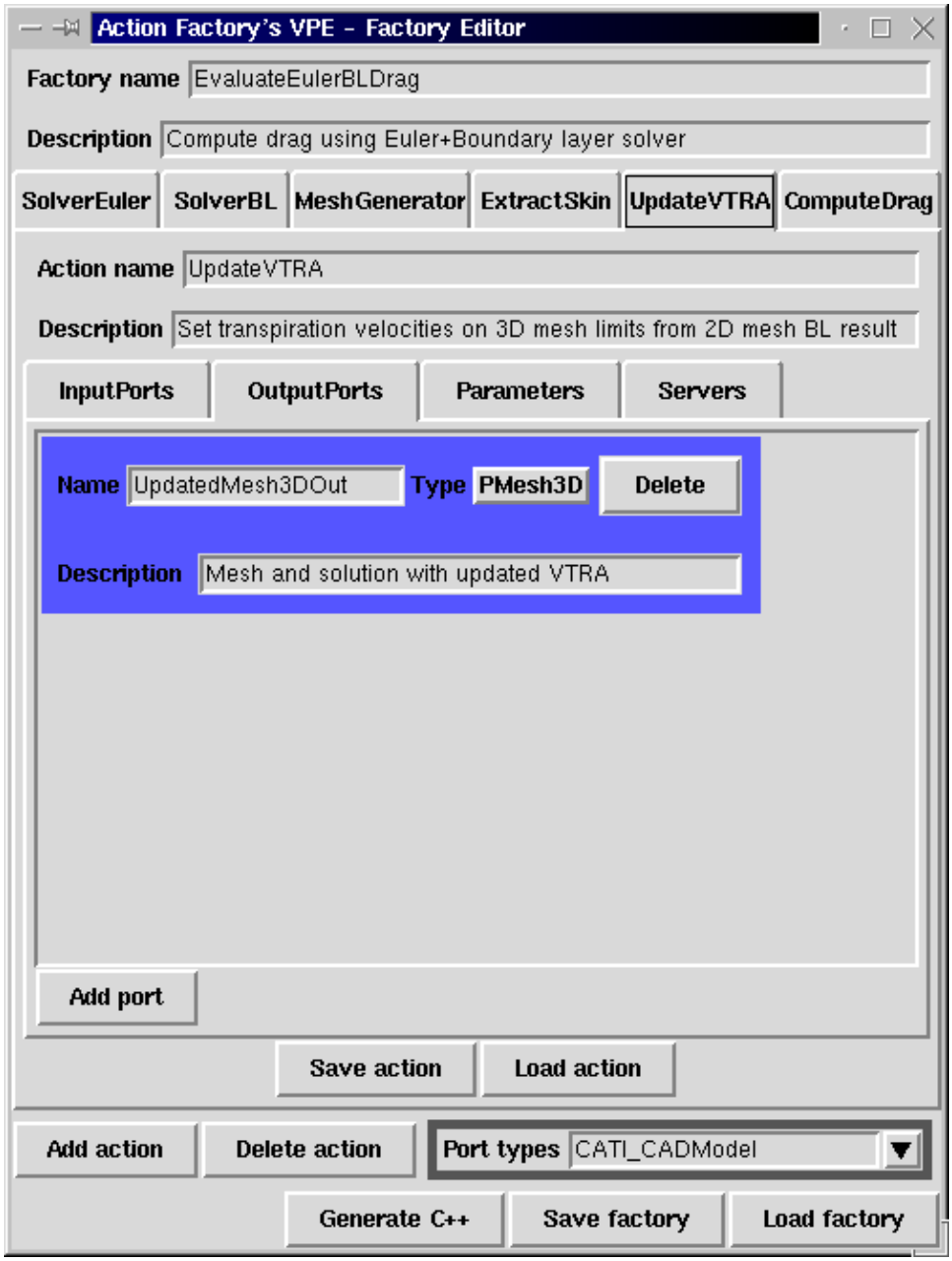


Servers may be ignored at this stage, and default save file will be <actionname>.action.xml

Packaging the elements of the process

This step may be performed directly by users

First fill the name of the process, ans its description Then interactively specify or load the steps (**actions**) of the process In the ActionFactory's jargon, all the steps (**actions**) of the process are packaged in an **action factory**.
Then save your **action factory** description to file.

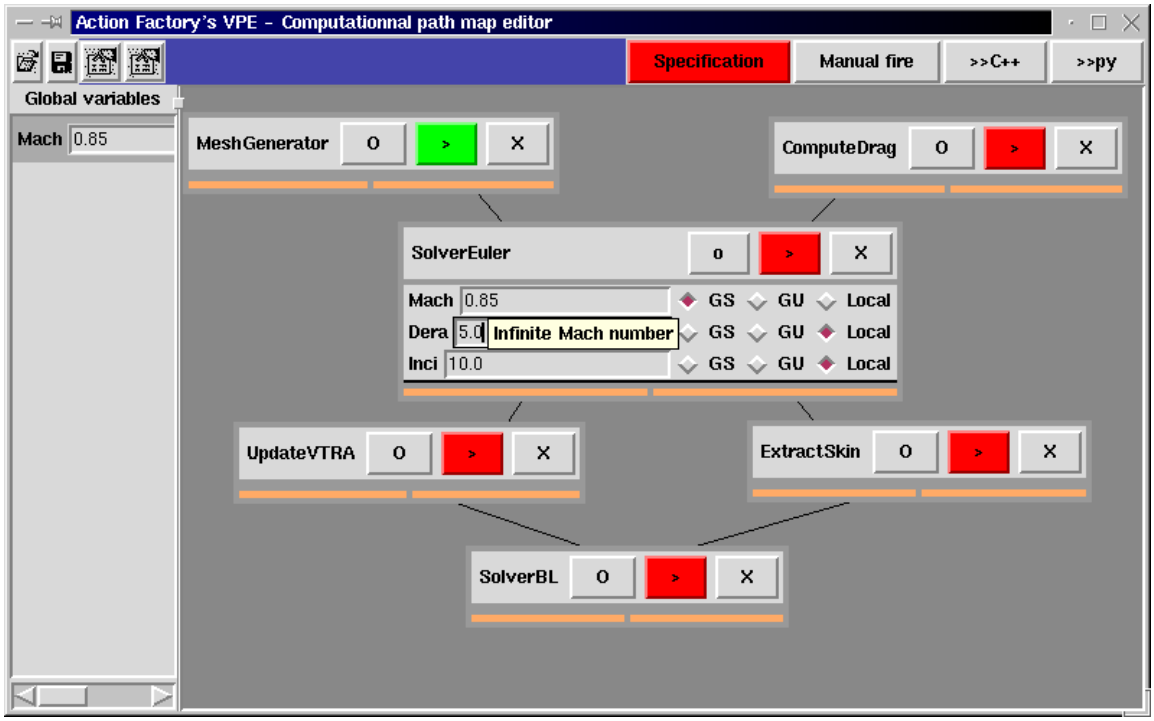


Default save file will be <factoryname>.factory.xml.

Designing the process

This step may be performed directly by users

Load your **action factory** in specification mode, to specify how various steps of your process interact : by connecting upstream action output ports to downstream action input port, you specify the order of execution and the data transmitted.



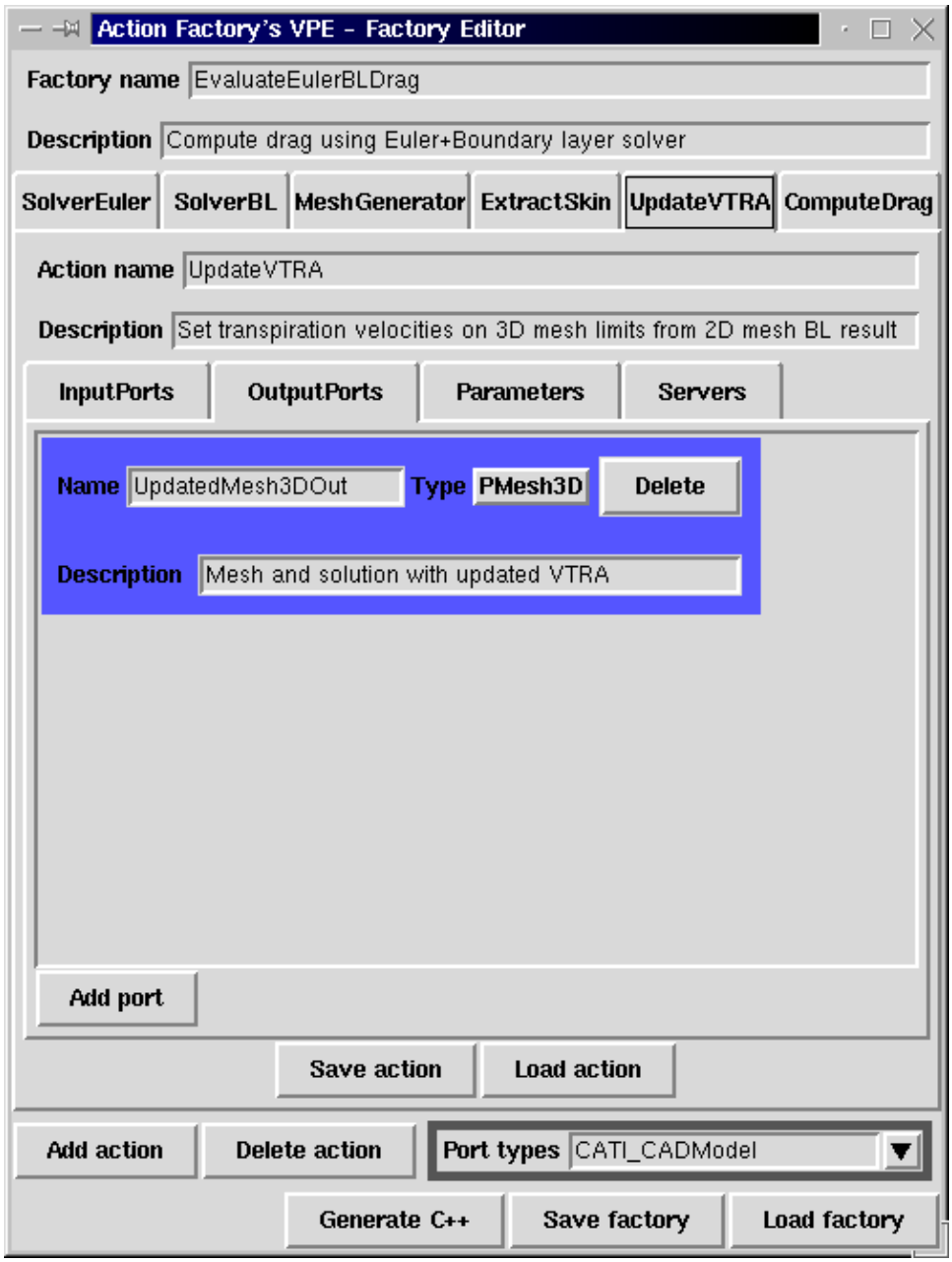
Type of transmitted data is checked, and you may test your computationnal path to get a print report of how it will behave once implemented.

Save your specification: You will directly use it for running your real simulation as soon as the underlying implementation will be available !

Checking the Action Factory and generating the wrapper

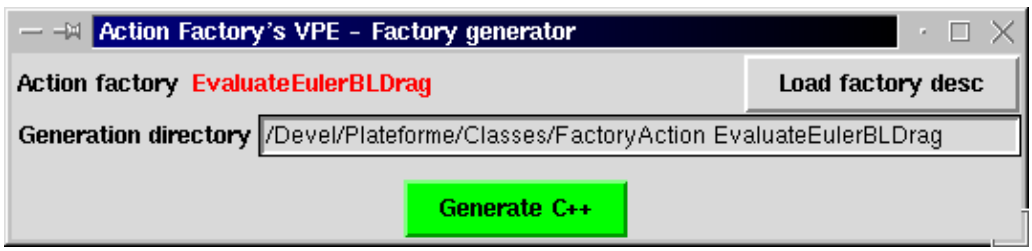
This should be performed by integrators

Check that type names are consistent with implementation type names available
For each **action**, list the components you'll need to implement the action
In the ActionFactory's jargon, all the components required to perform a step are called **servers**.
Then generate your **action factory** wrapper.



In the directory specified for wrapper generation, you'll find that all the requested IDLs, headers, sources and Makefiles have been generated.
You'll find in the source directory, source files for each specified **action**.

An alternate simple too may be used to generate/regenerate already checked factories



Implement the behaviour of individual actions

This should be performed by integrators

In each of the action implementation source file generated, look for the part to be completed : input objects are already allocated and filled, parameters too, output objects are already declared but NOT created, and all requested servers are already bound through CORBA.
You just have to invoke the appropriate methods on the servers you've asked for, to implement the behaviour of each action.
Compile your actions and the generated action factory, and package a script for your users to launch it : you're all set !

Of course, you may need to upgrade some components, or build new ones, if you have not the appropriate tools to execute the action : there's no miracle, and there will be the work of the core developpers !
The idea is that if you have the existing underlying code, capable of performing what's required by the user, it should be very easy to package it the way the user wants it (appropriate granularity / level of abstraction, appropriate level of parametrization...)

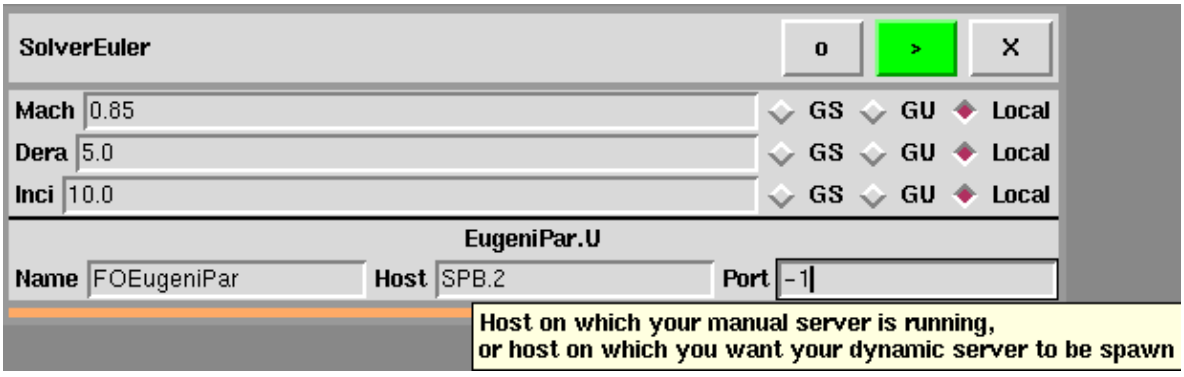
Run the process

This step may be performed directly by users

Start your action factory : because it inherit from the generic action factory, it will register itself in the naming service. If there's no naming service available, it will dump it's IOR string which you will be able to use for direct connection.

You may then from the MapEditor, in execution mode, directly select your factory within the ones available in the naming service (or connect directly by pasting the IOR).
The MapEditor will dynamically discover which actions are available in the server, which are the same than the one in the specification.

You may now load your saved computational path. You will have to specify on which host you want each action to be executed, by filling the 'server' area of each action.
Select '1' for portname if you want dynamic invocation of the underlying servers, or specify the real portnumber if you want to connect to a specific already running server.



You may now run directly the process you specified

Soon, you'll be able, once satisfied with the interactive testing of your process, to generate a batch version directly from the MapEditor, in Python or C/C++.

Comments, bugs, fixes to *t_chevalier@libertysurf.fr*.

[Home]

How to use Factory Editor

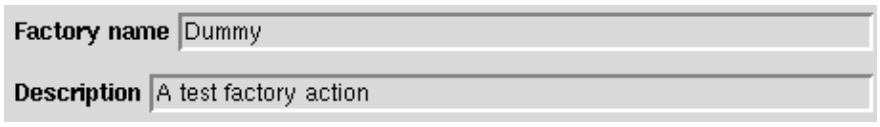
[home]

When started, the FactoryEditor will look like this:

By moving your mouse over most widgets and areas, you should have help ballons popping to help you use the tool.

Set factory name and description

Start with filling the name of your factory and some description in the upper area:



Please note that you don't have to begin your name with 'FactoryAction' : this will be added automatically.

Set used types

Then enter the type of the objects you'll have to exchange in the bottom combo−box.
Each entered type is added to the list.
Don't bother if you have typed something wrong, only types used in the actions input/output ports you'll later describe will be saved.
Don't try to be exhaustive, you will be able to complete this list at any time if you think some new type is needed.

Add an action

You're now ready to create ans specify actions, which will have to be repeated for each action you'll want to add
Push the "Add Action" button to create a new tab in the actions notebook:

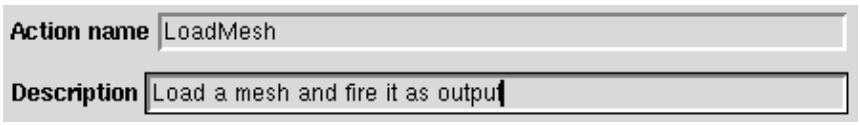
You'll be prompted to give a name to your new action:

And a new tab will be generated in the action notebook:

In case of any mistake, you can delete an action by setecting it's tab in the notebook, then push the "Delete Action" button...

Set action name and description

Fill the name of your action and some description in the upper area:

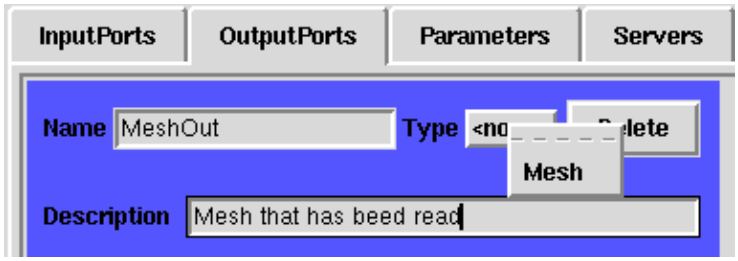


Please note that you don't have to begin your name with 'Action' : this will be added automatically.

Set action input and output ports

In the "LoadMesh" action we're currently building, there's no input port (the action just read from file and output a mesh but do not receive anything as input). We will then directly move to output ports, but input ports specification is just as output port specification, except in a different tab of the notebook.
Select the "Output ports" tab of the action description notebook, and push the "Add Port" button in the bottom area of the action description:

a new port description area will appear.
Fill the port name and description, and use right mouse key to pop the type menu. You should see in this menu all the types you entered in the "Port types" combobox (and may at this stage complete the port type list by adding new types in the "Port types" combobox if needed).

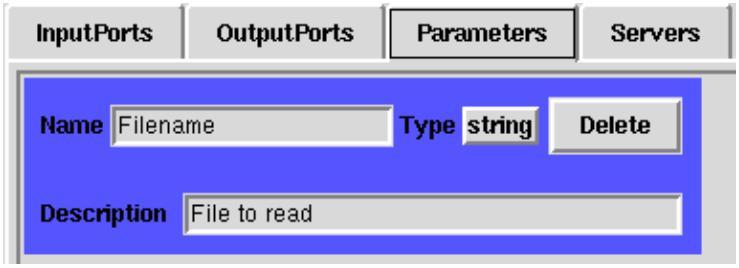


You may add as many input and output ports as you need.

Set action parameters

Select now the "Parameters" tab of your notebook, and push the "Add Parameter" button in the bottom area of the action description:

a new parameter description area will appear.
Fill the parameter name and description, and use right mouse key to pop the type menu.
Please note that the VPE tools handle various types for parameters, but that the current version of the FactoryAction only handle the "string" type.

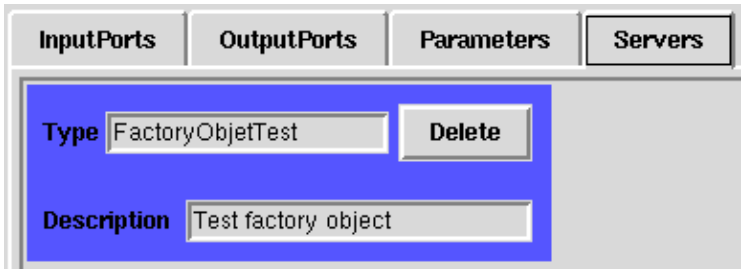


You may add as many parameters as you need.

Set action servers

If you're just designing your action factory, you may skip this step. If you're satisfied with the design and have successfully tested it in the MapEditor, then you'll need to fix the required servers as a last step before generating the C++ wrapper.
Select the "Servers" tab of your notebook, and push the "Add Server" button in the bottom area of the action description:

a new server description area will appear.
Fill the server type and description, and note that no 'FactoryObject' string is prepended to your type string.



Please note that the VPE tools handle several servers per actions, but that the current version of the FactoryAction only handle one.

Load and save individual actions

Loop over these instructions to define as many actions as you need.
Note that you may individually load/save actions, which is a convenient way to export them from one factory to another, or to build a library of actions in which to pick when designing factories. Use the "Load action" and "Save action" buttons for this :

When asked for a filename for saving, we suggest you choose a name of the form 'foo.action.xml'.

Load/save the factory, generate the C++ wrapper

Finish with saving your factory by pushing the "Save Factory" button.
You will be asked for a filename, and we suggest you choose a name of the form 'foo.factory.xml'.

If you're sure your factory is OK, you may also decide to generate the C++ wrapper for implementing it, by pushing the "Generate C++" button.
You'll be asked for a directory name under which a standard directory tree (see FactoryAction documentation) and files (IDL, headers, sources, Makefiles) will be generated.

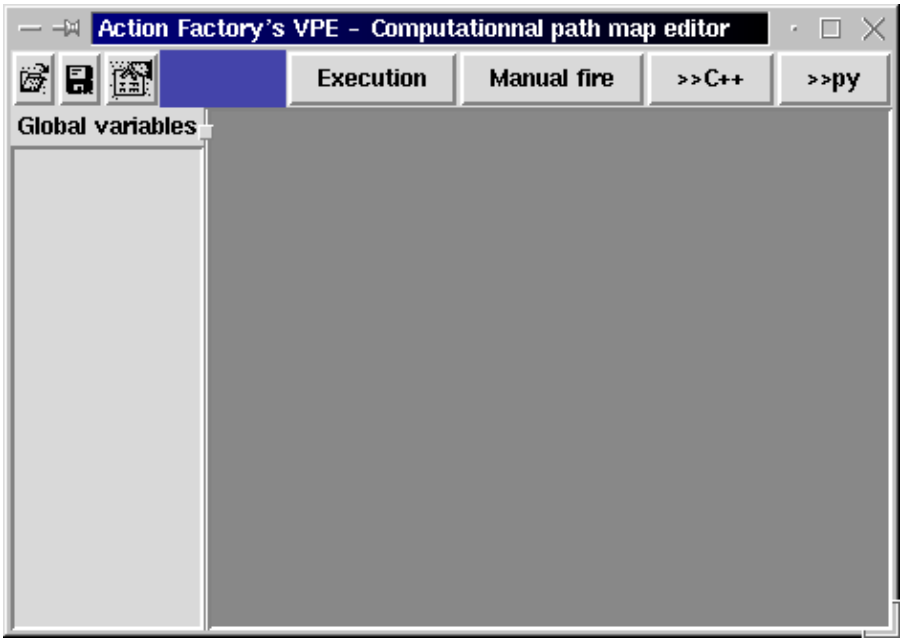
Comments, bugs, fixes to *t_chevalier@libertysurf.fr*.

[Home]

How to use Map Editor

[home]

When started, the MapEditor will look like this:



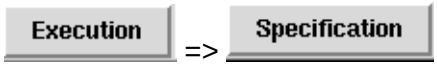
By moving your mouse over most widgets and areas, you should have help ballons popping to help you use the tool.

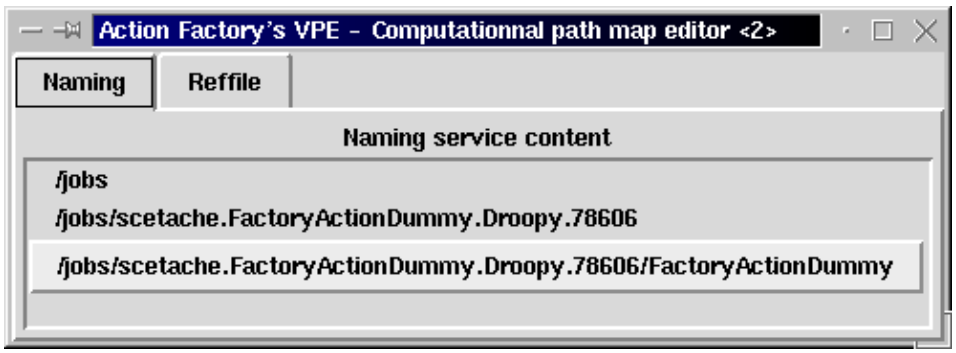
Loading action descriptions

The first thing you have to do is to load some factories.
You've two options there :

- either you've just specified your factory, have no underlying implementation, and want to check your factory specification or build your process specification to check it.
  You'll then use the MapEditor in ***specification*** mode.
- or your integrator has provided you with an implementation of your specified factory, and you want to use it to run real computation with the MapEditor as a process controller.
  You'll then use the MapEditor in ***execution*** mode.

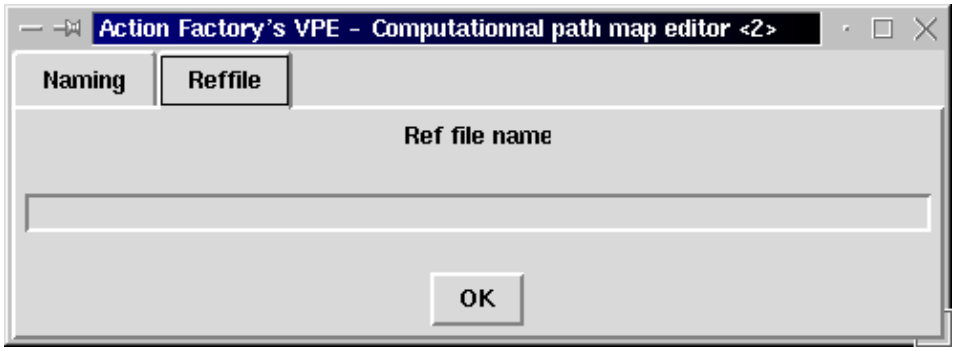
Be aware that as soon as you've loaded any factory in ***specification*** mode, you won't be able to switch back in execution mode (unless restarting the MapEditor), because it means that some loaded components have no underlying implementation.
Be aware that if the MapEditor may handle a mix of several factories in a process, the underlying FactoryAction is not yet capable of that, so you should currently restrain to using a single action factory at a time.

In ***specification*** mode	To switch to ***specification*** mode, you will first have to push the "Execution" button, which label should change to "Specification":  You'll now select the right loading button (which help balloon should read "Load actions from configuration files"):  And select your saved XML description of your action factory from the FactoryEditor visual tool.
In ***execution*** mode	Remembrer you should have started your FactoryAction, using the script provided by your integrator. You'll now select the left loading button (which help balloon should read "Load actions from a running action factory"):  You now have to choose the way for the MapEditor to connect to your running FactoryAction. If your FactoryAction registered itself in the ORB naming service (let's see that as a "Yello page" directory of running servers), you sould see it available in the "Naming" tab:



Just press the button corresponding to your Factory in the list.
The MapEditor will then connect to the FactoryAction, and discover on the fly which actions this FactoryActio implements.

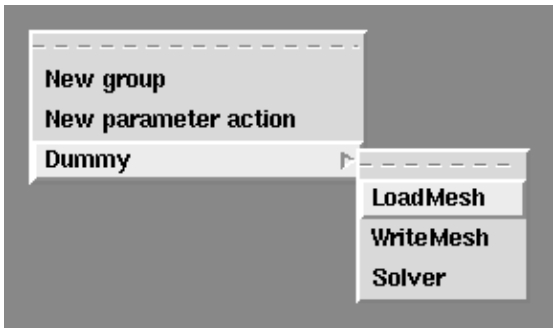
If your FactoryAction does not register itself in the ORB naming service, then you should select the "Reffile" tab, and enter the name of a file where the "stringified IOR" of your FactoryAction is stored.
Enter "foo" if yout IOR is stored in file "foo.ref":



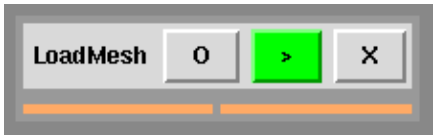
You should check that with your integrator, but an IOR looks like that:
`IOR:0000000000000.......5365727669636500`
The MapEditor will then connect to the FactoryAction, and discover on the fly which actions this FactoryActio implements.

Creating and interacting with action boxes

We will now have to define the actions that take part in our process.
Press the right mouse key in the background of the MapEditor:



Discard for the moment the first two entries ("New group" and "New parameter action"). You should find below the name of the loaded factory, with a sub−menu detailing all of the actions available from this factory.
Select one of the actions, to make an action box appear in the working area:



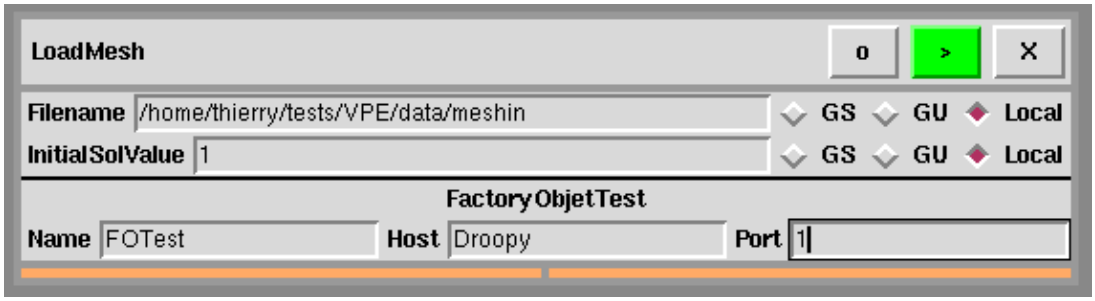
Each time you'll select an action from the background popup menu, you'll create such an action box. You'll so be able to create an action box for each of the steps of your process.
Now, let's see the basic way of interacting with these action boxes, to organize our workspace ares :

- First, by dragging the left mouse button on the background, you'll move the complete area.
- By dragging the left mouse button on the name of the action box, you'll move this box alone.
  (just clicking with left mouse will pop the action box to the front)
- By pushing the "***O***" button, you'll open the action box to edit various parameter values (we'll see that in detail later), and pushing the "***o***" button again will close the action box to it's previous state.
- By pushing the "**>**" button, you'll ask the action to be executed.
  Note that when the button is not green, the action can't be executed because data are missing on the input port.
  If (like on the previous figure) the button is green, it means that there are data on the input ports, available and ready to be processed; or that the action has no input port so it is always ready to fire.
- By pushing the "***X***" button, you'll delete this action box.
- By pressing the right mouse key on the lower ***left*** orange rectangular area, you'll get a popping menu with the available input ports.
  (we'll see later how to use it for connecting actions).
- By similarily pressing the right mouse key on the lower ***right*** orange rectangular area, you'll get a popping menu with the available input ports.
- By pressing the middle mouse key on either orange rectangular port area, you'll get on the standard output (listing) a dump of the connection status.

Help balloons should help you to remember these commands.

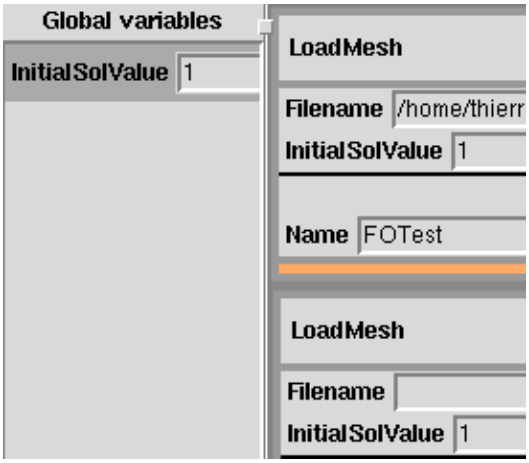
Action boxes internal details

Let's see now how to set the details of each action box. Fist, you'll need to open it using the "***O***" button, which shoud make your action box look like :



The first lines shows the parameters, that were specified as required for the action (here in this figure "Filename" and "InitialSolValue"). For each of them, an entry field allow interactive input of their values.
At the end of each parameter line, is a radio−button with three mutually exclusive options: "***GS***", "***GU***" and "***Local***". For understanding these, please create two identical action boxes (two instances of the some action), and open them.

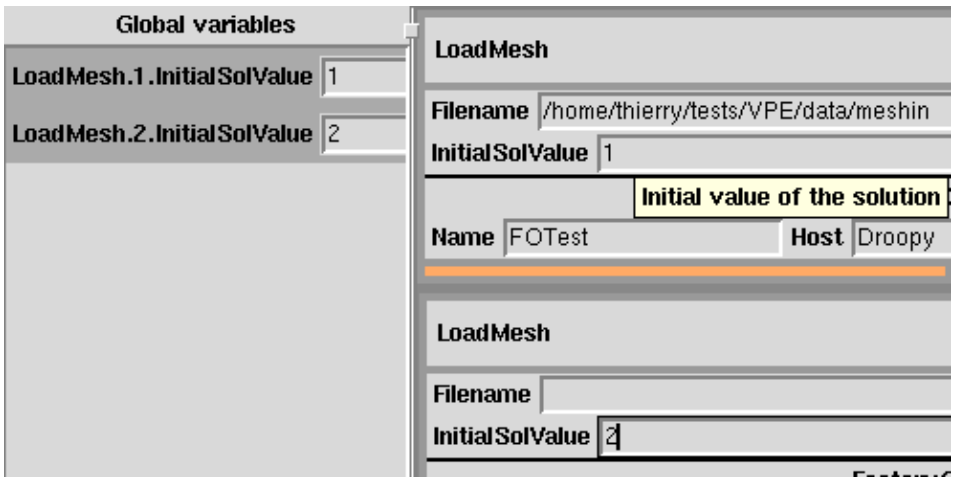
- "***GS***" stands for "Global Shared". When this option is selected, the corresponding parameter is replicated in the "Global Area", at the left of the MapEditor.



Set one of your parameters as "Global Shared", and try changing it's value : any change in the action box area is repercuted instantly in the global area and vice−versa.
Set now **the same** parameter as "Global Shared" in your other action box : the three values are now linked. Changing the value in any of the two action boxes or the global area immediately update the two others.
The parameter value is shared globally over the simulation, among all of the

action boxes which declared it "Global Shared". This is usefull to insure that some parameter values are unique among the simulation (physical parameters such as altitude, or numeriacl parameters such as global iteration number).

- "***GU***" stands for "Global Unshared". When this option is selected, the corresponding parameter is replicated in the "Global Area", at the left of the MapEditor.
  But he's now prefixed by it's action name and internal unique number.



Set one of your parameters as "Global Unshared", and try changing it's value : any change in the action box area is always repercuted instantly in the global area and vice−versa.
Set now **the same** parameter as "Global Unshared" in your other action box : another different field is set in the global area (differing by the action internal unique id, seen here as a number). This secong global field is linked to the second action box parameter in the same way the first is linked to the first action box parameter.
The parameter values is shown globally regarding to the simulation, but their values are not shared. This is mostly usefull from the user−interface point of view, to bring in the global area some important parameters of the simulation for which you don't want to have to find the action box and open it to edit the parameter value.

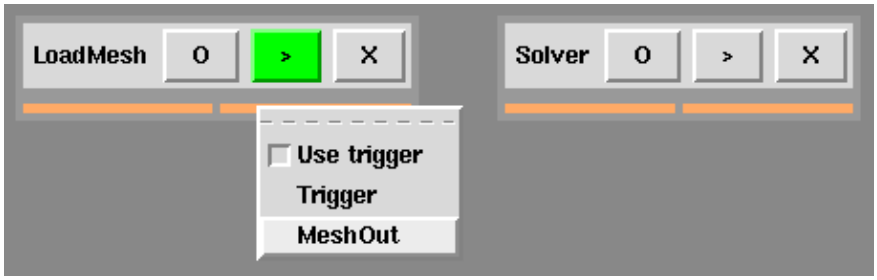
- "***Local***" stands for "Local" (!). Well, that's the default behaviour : the only way to change a "Local" parameter is to open the action box and set it's value.

At the bottom of the action box is the underlying servers description. If you're in **specification** mode, you may forget that part, but in **execution** mode, you'll have to fill it.
Check with your integrator for filling this part, which should be improved in next release. Especially, you'll need to know if you've an automatic activation available or not (it's recommended and available with most commercial packages such as Orbix and Visibroker. We provide a custom one for Orbacus in FactoryAction).
If you've an automatic activation available, you'll just have to fill the name of the underlying servers to be activated, the host on which you wish them to be spawned, and leave the port number to "1".
If you've no automatic activation, you'll have to start manually the underlying servers,

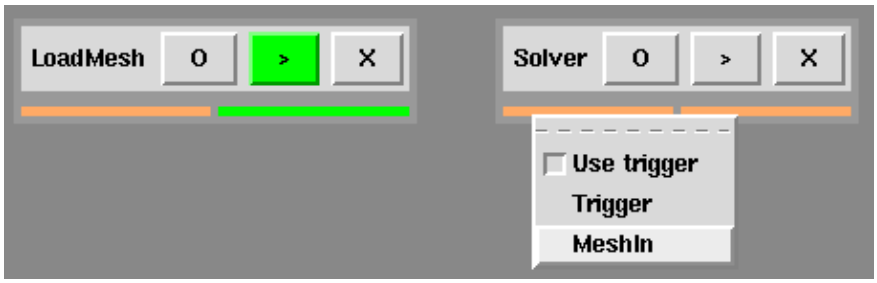
then specify their name, the host on which they were started, and their listening port number.

Connecting action boxes

Let's now connect action boxes together, to streamline the process and express how data flows from action to another.
Push right key mouse on the output port (right) bottom orange area of the upstream action, and select the data to be flushed downward :



Then push right key mouse on the input port (left) bottom orange area of the downstream action, and select the data to be accepted as input :



The two boxes are now connected, and you may check that connection prerssing middle mouse button either on upstream action output port area or downstream action input port area (and look at the listing) :

Note that you may delete the link by just clicking on it (you've to be precise !) with the left mouse key.

Loading and saving a map

You may now save your defined computationnal path, by selecting the usual "save" icon in the upper toolbar (i.e. the floppy) :

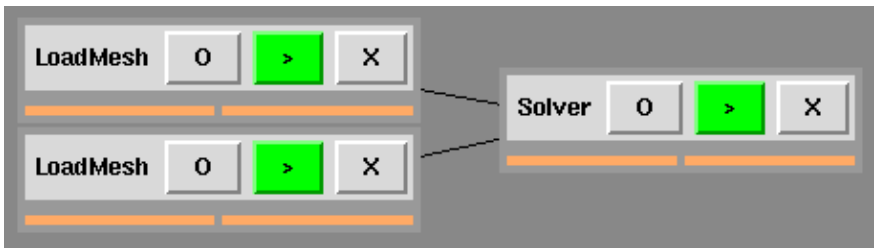
You'll be able to reload it later in the MapEditor using the usual "open" icon.

Firing action boxes

Now clicking on the fire (">") button of upstream action (in the exemple shown "LoadMesh") should send data downstream and make next action (here "Solver") ready to fire (the ">" button should pass to green) :

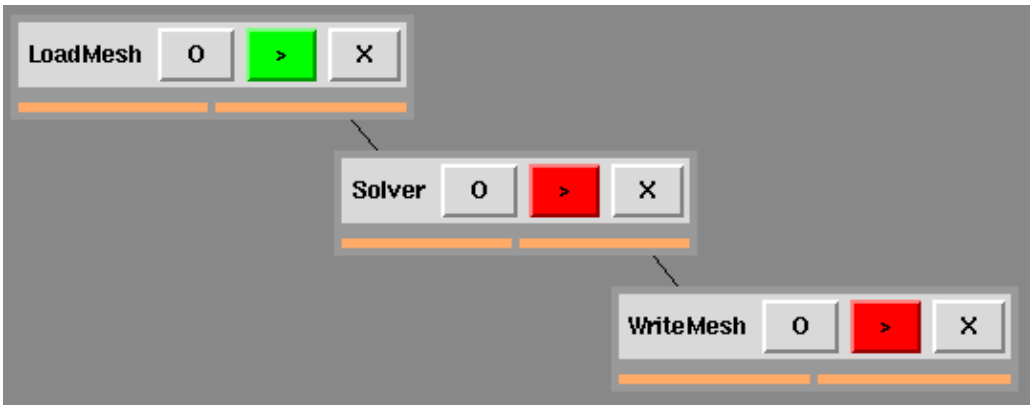
Clicking on the fire (">") button of downstream action ("Solver") should start this action and consume the available input data (the ">" button should pass to red) :

If you try to fire N times the upstream action, you'll be able to fire N times the downstream action before the fire button turns to red, because the inputs are buffered.
The same if you've two upstream actions filling a downstream action and fire both upstream actions, two data will be buffered on the input port of downstream action, allowing her to fire two time (collector default behaviour) :



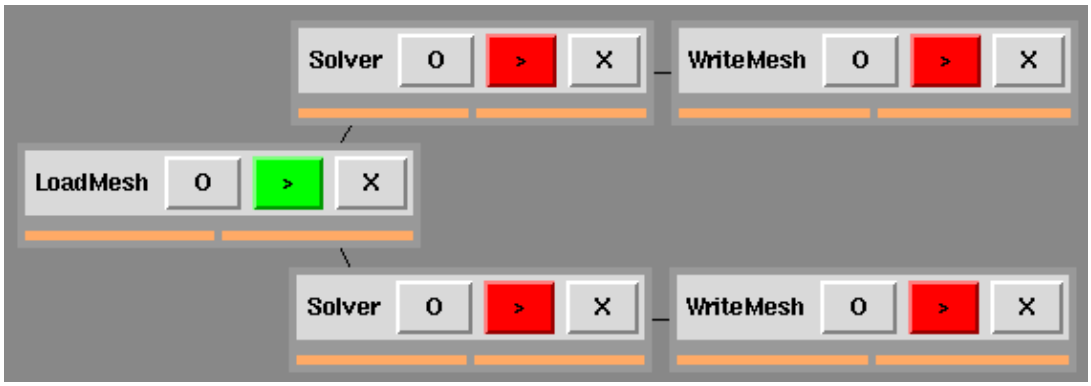
Auto−fire and parallel pathes

If we build now a chained sequence of three actions :



You may fire all three modules one by one in manual mode, or press button "Manual fire" in the upper toolbar. The button should switch to "Auto fire" :

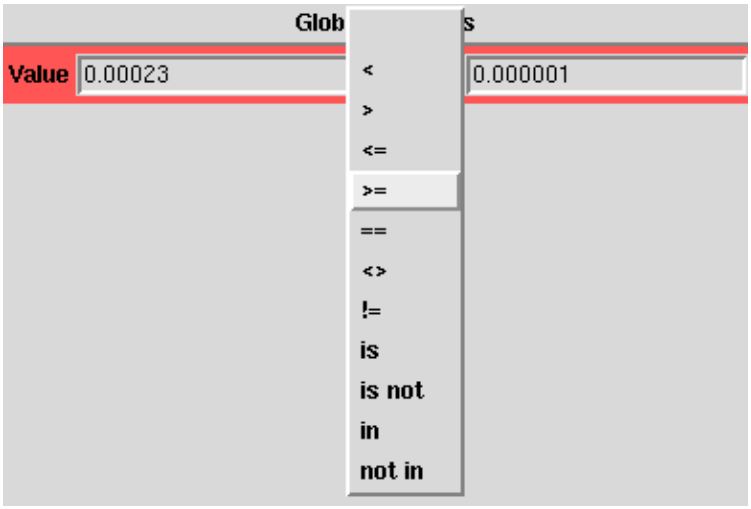
If you fire now manually the first action of the sequence, all downstream actions are fired as soon as all input ports have been provided with data.
Let's now build a two path computationnal sequence :



Fire first action : when its execution if finished, data is sent downstream to both connected "Solver" modules which start **simultaneously**. For each action execution, the MapEditor create an independant thread, thus allowing several actions from paralel pathes to execute simultaneously.
Be aware that the standard generated FactoryActions are not yet multi−threaded in this version, so that some FactoryActions will not be able to take advantage of that.

Execution control with conditions

You may set conditions on the global values, by pressing left mouse key on the button which is on the right of the global value entry field :



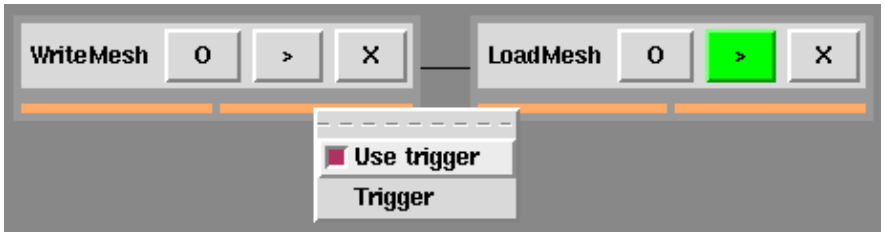
While the condition is verified, the test appears surrounded in green :

If as a result of any action changing the parameter value, the test is not any more verified, it will appears surrounded in red, ***and the execution mode is automaticazlly switched back to "Manual fire"*** thus stopping the process (except by manual continuation or analysis) :

Triggers

There are some cases, when you'll want to sequence two actions without having any data to be flushed from one to the other (ex: start a plotting software when the output file has been generated). You will then need a way for upstream action to signal downstream action that it should start.
Triggers are designed to solve this problem.
You may connect any output port to an input trigger port, especially an output trigger if you've nothing else to send downstream. For the trigger to be active, the "Use trigger"

checkbox should be activated :



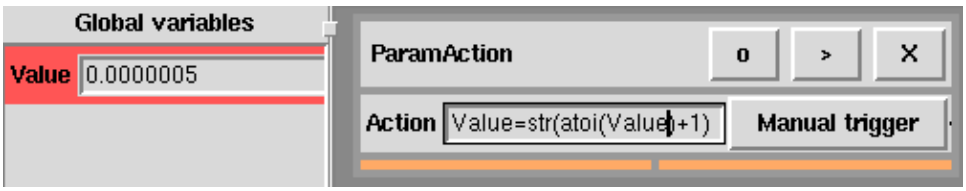
Be aware that once the input trigger is activated, it may receive any kind of data which it will not try to understand, but will need to have data available as any other port, for the action to start.

Parameter actions

Parameter actions are ***Python* affectations concerning global parameters**.
In some cases, you'll want to modify a **global** parameter without to have a specific action doing it (ex: increment a global iteration loop counter). You will then need a way to do it directly from the MapEditor.
Parameter actions are designed to solve this problem.
They should be fired via triggers, and will accept any *Python* expression of the form :

```
<global variable name>= function of (<global variable name>)
```

Be aware that you should for the moment strictly respect *Python* syntax, and *Python* is a dynamically but strictly typed language. This makes expressions difficult and will be improved in next version.
Here is an example for incrementing a global iteration counter named "Value":

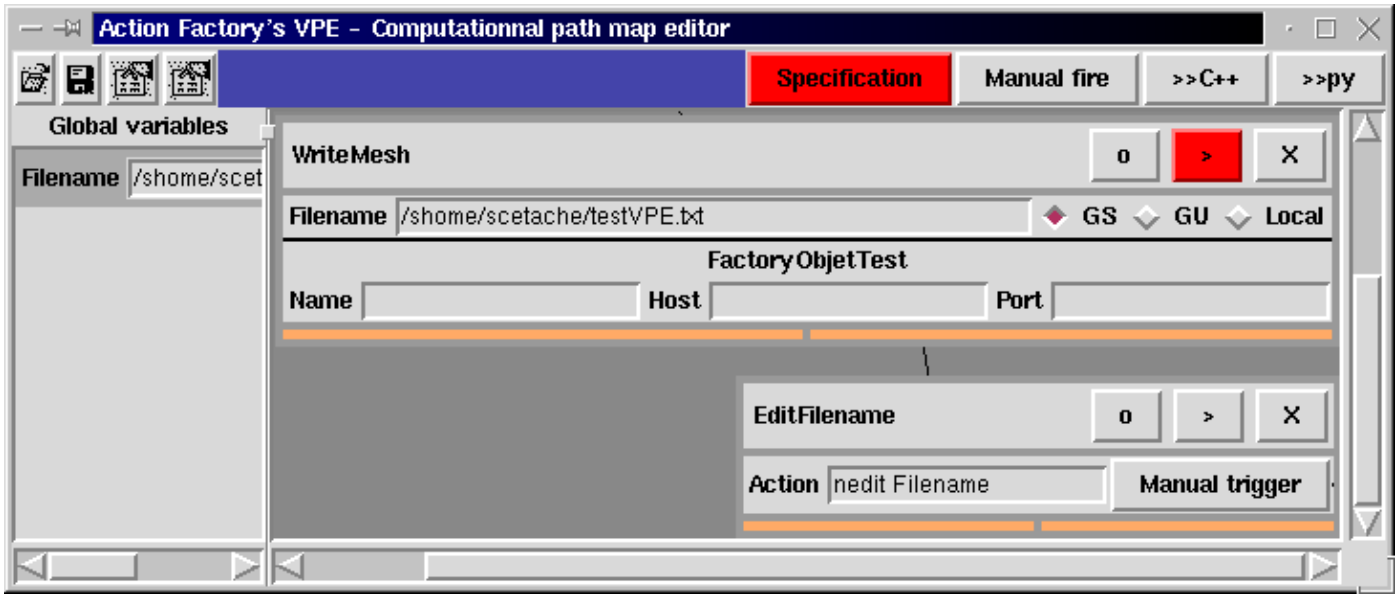


The "Manual trigger" button allow you to test your syntax by running the action without flushing any output.
You may customize a parameter actions name by pressing right mouse key on its name.

Command actions

Command actions are ***system* commands which parameters may contain global parameters**.
In some cases, you'll want to run a system command with one or several

global parameter as arguments, without to have a specific action doing it (ex: launch an editor on your output listing, or a plotting software on your generated output file). You will then need a way to do it directly from the MapEditor.
Command actions are designed to solve this problem.
They should be fired via triggers, and will accept any system expression including global variable which value will be replaced before execution by their current value.
Here is an example for editing automatically an output file :



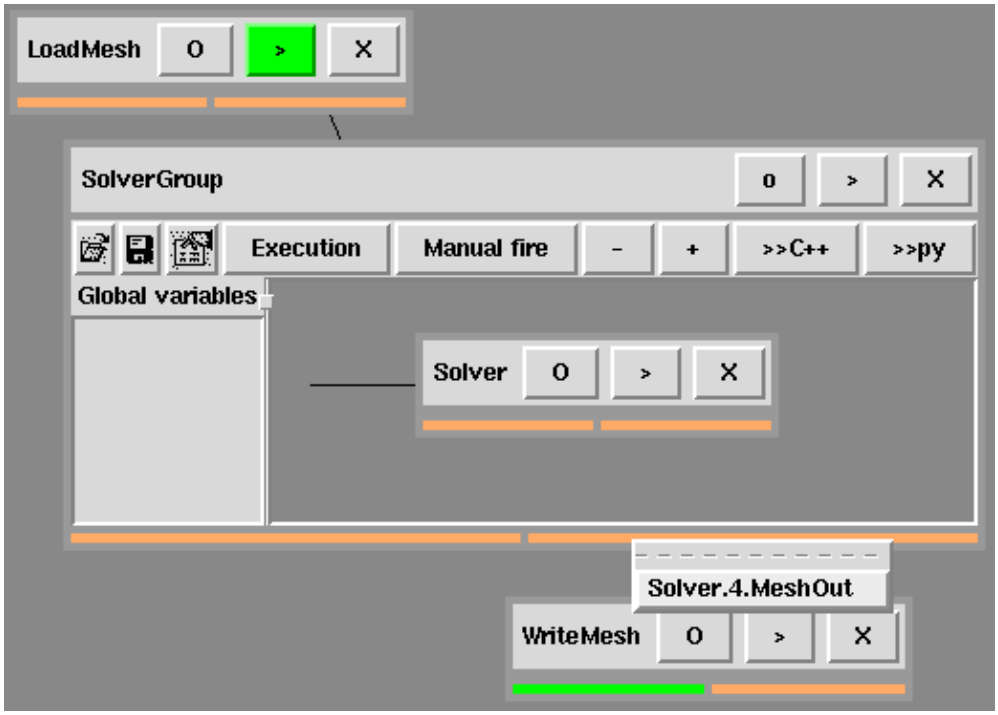
The "Manual trigger" button allow you to test your syntax by running the action without flushing any output. You may customize a command actions name by pressing right mouse key on its name.

Groups

Groups are a way to wrap several actions and make them appear as a single one. It may be seen (and it is as a matter of fact !) as an Action Box containing itself a sub−map editor.
When opened, the Group shows it's internal map editor in which all of the previously described behavior are available.
The only two specific things about group are :

- First the size grow ("+") and size reduce ("−") buttons, on which you should push to adapt opened group size (sorry, it's yet still a little ugly and you cat set size independantly in both directions : there are many improvements planned around that...).
  They are located in the internal map editor toolbar :

- Group ports are the replication of internal Action Boxes ports. To establish connection from an outer Action box to an inner action box, connect the outer action ports to the group ports corresponding to the inner action you want to connect with :



Please be aware that group handling is in alpha development in this version, and especially the Load/Save is still buggy when dealing with groups.
So experiment with it and provide feedback, but don't rely on them ! Thanks.

Python and *C/C++* static generation of a hard wired process controller from a MapEditor's map is still at prototype version for the moment, so the corresponding buttons remain inactive:

Comments, bugs, fixes to *t_chevalier@libertysurf.fr*.

[Home]

Tutorial

[home]

Start the FactoryEditor :

Define factory basics

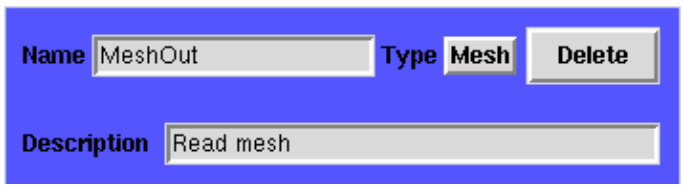
[manual]
Set factory name and description :

Define used types :

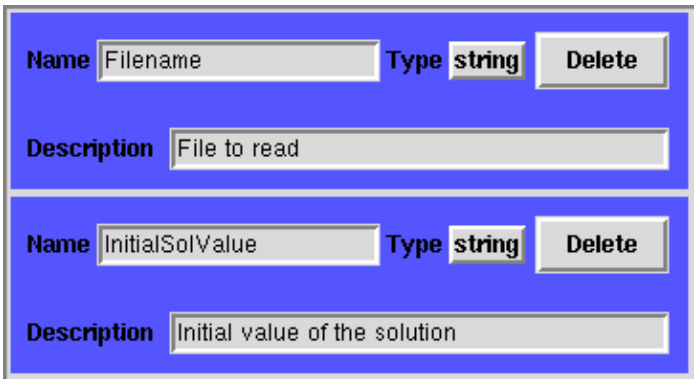
Add action "LoadMesh"

[manual]
Add the action and set its name and description :

There are no input ports for this action.
Define output ports:



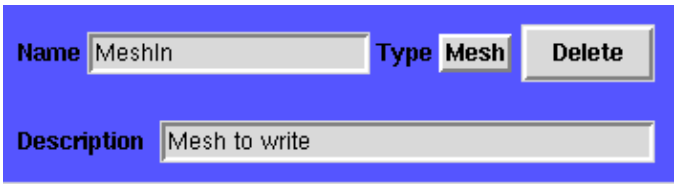
Define parameters:



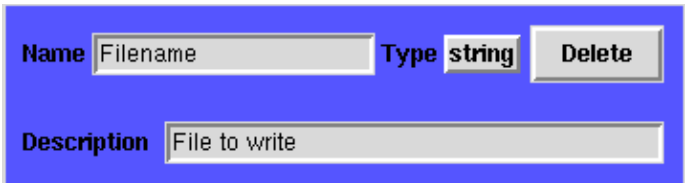
Add action "WriteMesh"

Add the action and set its name and description :

Define input ports :



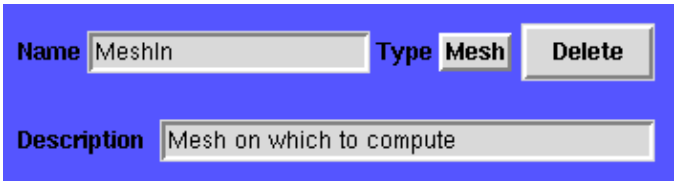
There are no output ports.
Define parameters:



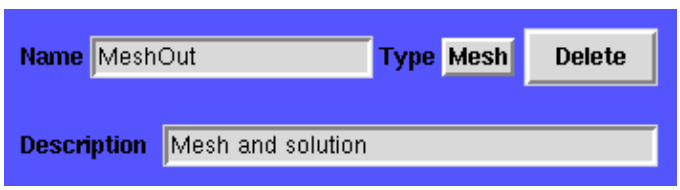
Add action "Solver"

Add the action and set its name and description :

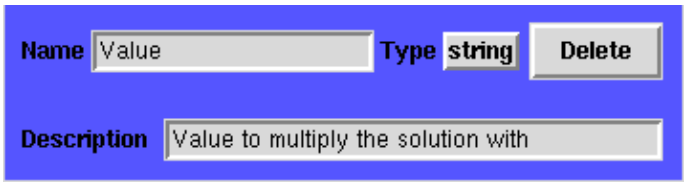
Define input ports :



Define output ports:



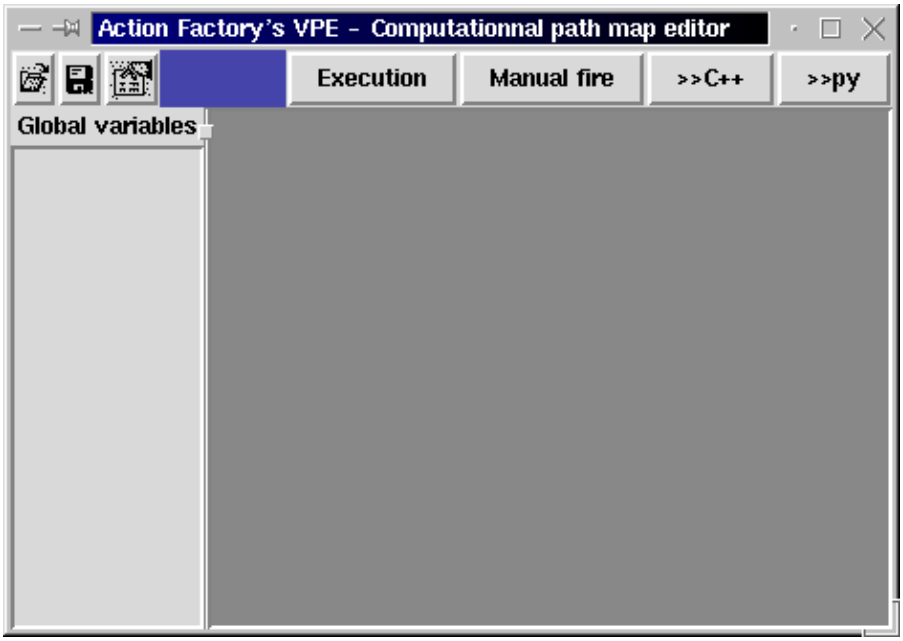
Define parameters:



Save your factory

[manual]
Save the factory in file "Test.factory.xml".

Start the MapEditor :

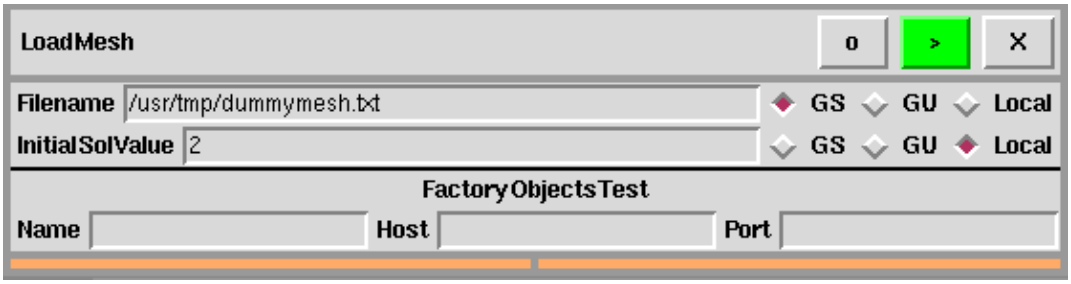


Load factory in MapEditor

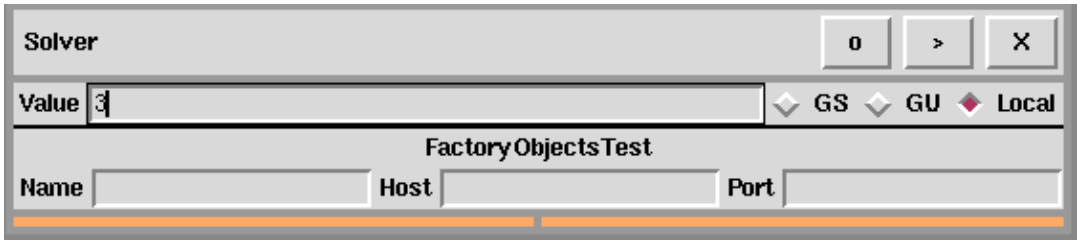
[manual]
Switch MapEditor to "Specification" mode, then load your factory descrription from "Test.factory.xml".

Create "LoadMesh", "Solver" and "WriteMesh" action boxes

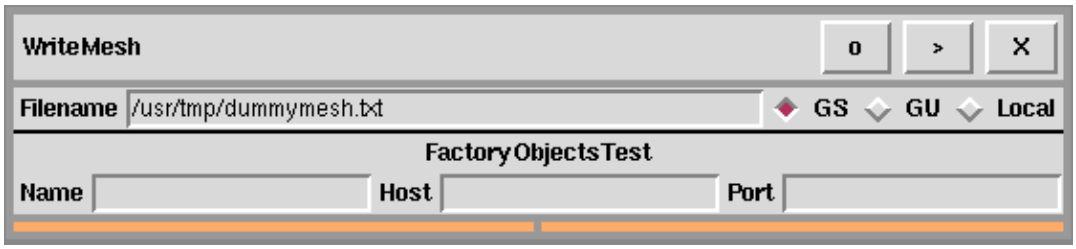
[manual]
Create a "LoadMesh" action box, and set it's internal parameters. Set the "Filename" parameter as global shared :



Create a "Solver" action box, and set it's internal parameters :



Create a "WriteMesh" action box. Set the "Filename" parameter as global shared to fix the same Filename value as in LoadMesh :

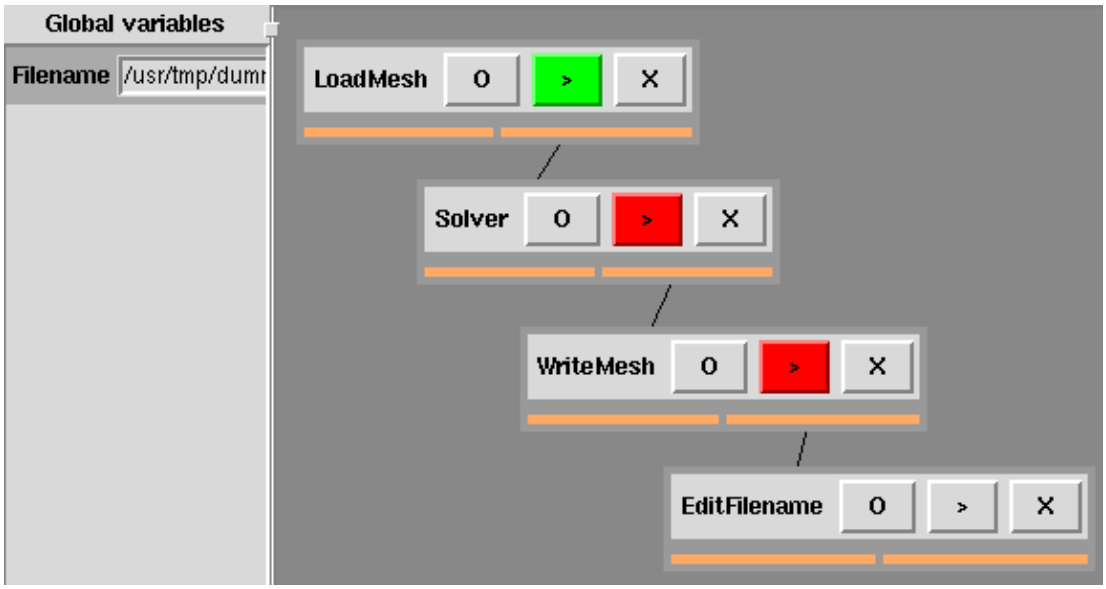


Create command box

[manual]
Create a command action box, name it "EditFilename" and set it's internal ordrer to an editor command ('xterm −e vi Filename' should work almost anywhere):

Establish connexions

[manual]
Connect **LoadMesh** output port **MeshOut** to **Solver** input port **MeshIn**.
Connect **Solver** output port **MeshOut** to **WriteMesh** input port **MeshIn**.
Connect **WriteMesh** output port **Trigger** to **EditFilename** input port.



On **WriteMesh** output port, activate the **UseTrigger** checkbox.

Run simulation in specification mode, save your map

[manual]
Switch to "Auto Fire" mode, run your specification and look at the listing to check if it behaves as you wish it should.
Then save your map in "Test.map".

At this stage, the user should give its specification files to the integrator :

- the "Test.factory.xml" file as a specification file

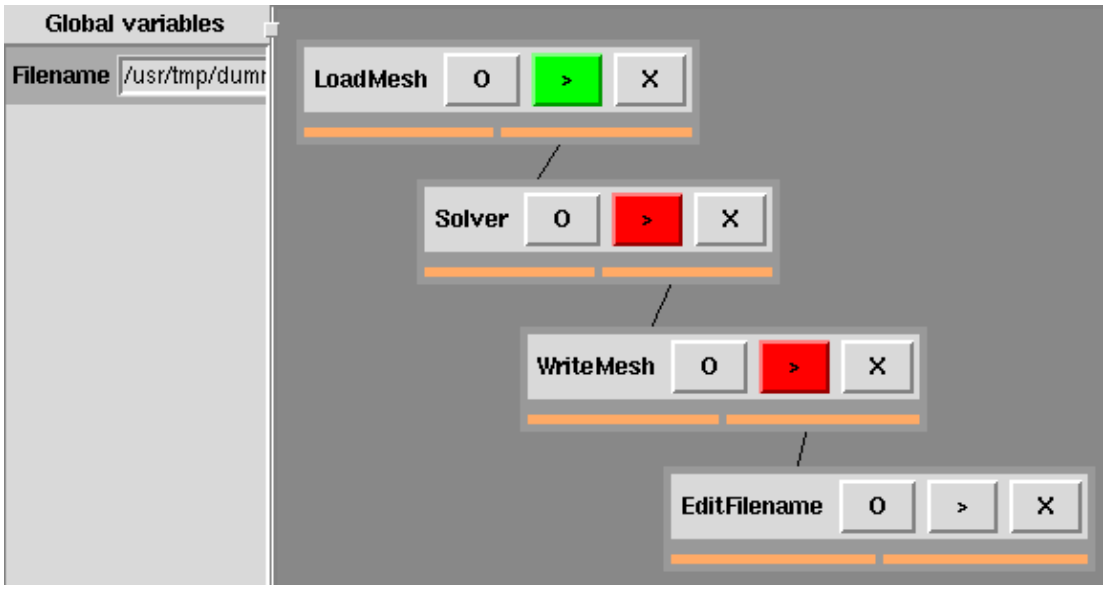
- the "Test.map" as a use case file

The user should keep its "Test.map" file, because it'll be directly used for steering the simulation once the integrator will have provided a proper implementation.

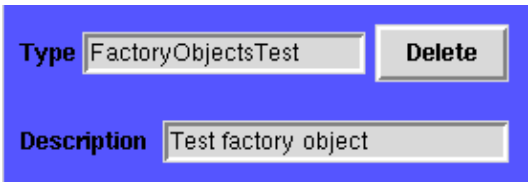
Specify servers and generate

[manual]

Launch the FactoryEditor, load the "Test.factory.xml" factory specification, and complete the "Server" tab of each action notebook with the name of the FactoryObjects (see FactoryACtion documentation) that will be needed to realize the implementation. For **each** "LoadMesh", "WriteMesh" and "Solver" action, specify you'll need the "FactoryObjectTest" server :



Save your updated factory description, and generate the C++ wrapper :



You should now switch to the FactoryAction tutorial to complete the implementation of the specified FactoryAction.
Let's assume we're back in user's role, with a finished implementation. Our integrator gave us a FactoryActionTest, and we started it.

Reload your computationnal path in execution mode

[manual]
Leave MapEditor in "Execution" mode, then load your factory description from the running FactoryActionTest.
Load your saved map "Test.map".

Choose execution platforms

[manual]
For **each** of the "LoadMesh", "WriteMesh" and "Solver" action boxes, you'll need to specify the host on which you'll want your simulation to run.
See with your integrator if he's set the automatic activation or not, and act accordingly (see manual).
Save your updated computationnal path in "Test.map".

Run the real simulation

[manual]
Switch to "Auto Fire" mode, and run your computation !

Comments, bugs, fixes to *t_chevalier@libertysurf.fr*.

[Home]

Copyright notice:

GNU GENERAL PUBLIC LICENSE
Version 2, June 1991

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software--to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

GNU GENERAL PUBLIC LICENSE
TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based

on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

a) Accompany it with the complete corresponding machine−readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine−readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not

signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty−free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS