THALES RESEARCH & TECHNOLOGY
CET System Engineering / Simulation

JNI ACTOR HELP

or

‘ How call an existing native C function into a Ptolemy II model using the JNI Actor and its add-in ’

Index

Table of figures

How activate the add-in

By launching **$PTII/bin/vergil –jni** or **./bin/vergil.bat -jni**, you activate the TRT Add-in for Ptolemy II, which allows to use C functions in Ptolemy II models in the dedicated JNI Actor.

You can find this JNI Actor in the Thalès library, so that you can drag and drop it into a Ptolemy II model.

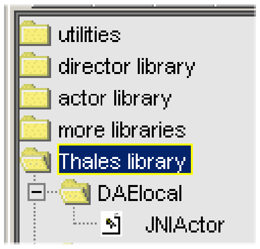


Fig. 1 : The JNI Actor in the Thalès library

Once you have dropped the JNI Actor into the model, it needs to be configured.

How configure the JNI Actor

A JNI Actor is configured for one function, which will be executed at each firing.

You can configure several JNI Actor to call different functions or copy and paste one you've configured to use the same function several times.

So before using it, you need to specify the function, first by clicking right on the JNI Actor, and by choosing "Configure".

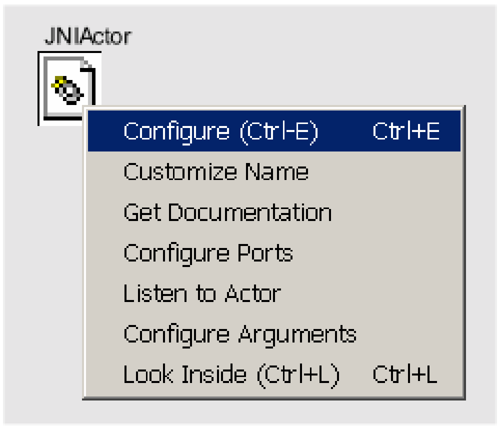
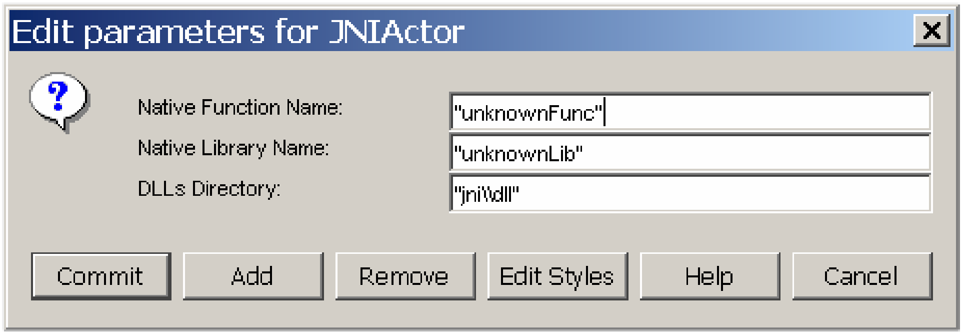


Fig. 2 : The contextual menu for a JNI Actor

The first parameters for a JNI Actor are :
- The name of the existing C function.

- The name of the existing shared or dynamic library, containing the function.
- The directory where is this existing library. (warning : you have to use \\ as file separator). By default, the directory is set to $PTII/jni/dll



Native Function Name: "unknownFunc"
Native Library Name: "unknownLib"
DLLs Directory: "jni\\dll"

Fig. 3 : The frame to edit the global parameters of the JNI Actor

How configure the arguments

Once global parameters have been set, the arguments of the function have to be described. The “Configure Arguments” button allows to graphically do it

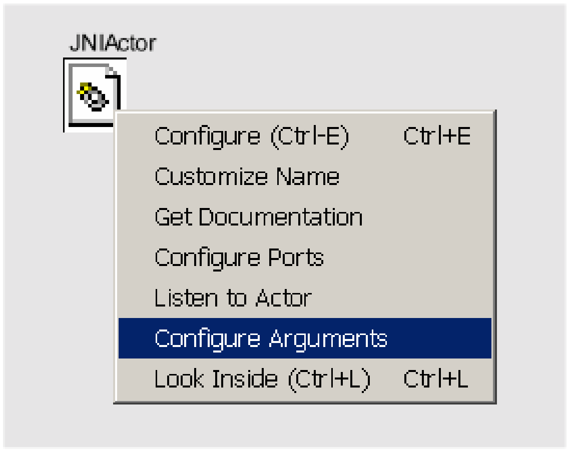


Fig. 4 : The ‘Configure Arguments button in the contextual menu of JNI Actors

The first time the button is pressed, the following menu appear :

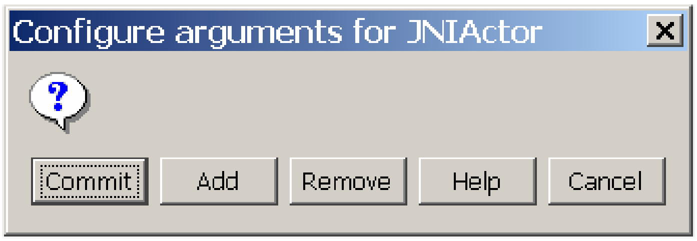


Fig. 5 : The frame to configure the arguments of the called function

Add an argument by clicking on the “Add” button. The ‘Add argument frame’ appears.

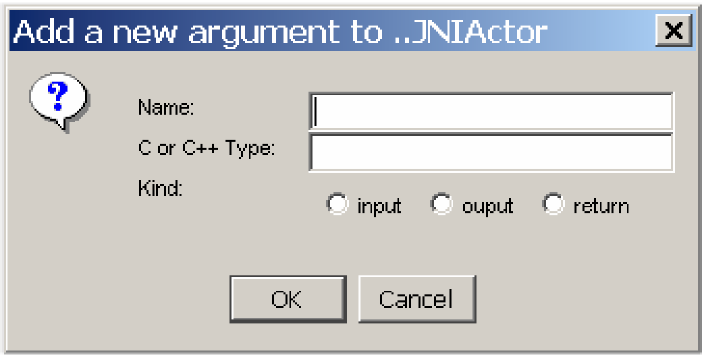
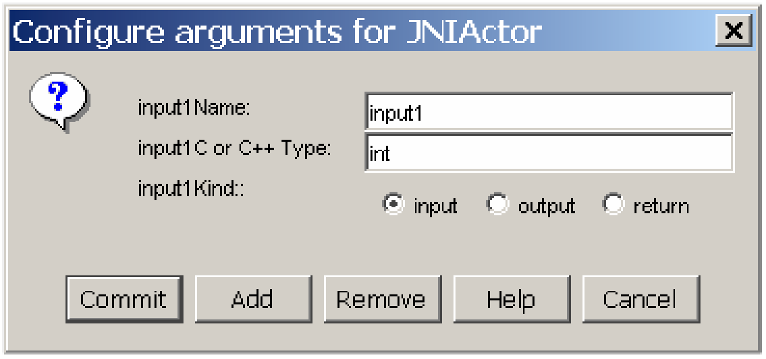


Fig. 6 : The frame to add an new argument

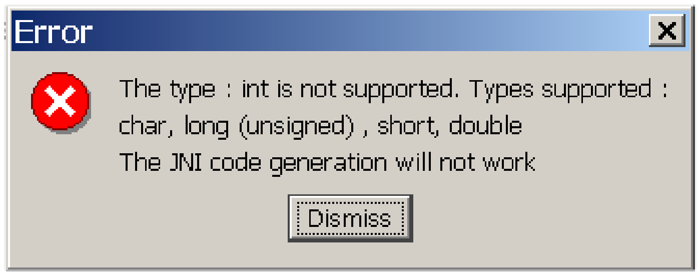
Then, you have to set a name, a native type and a kind for the current argument, and then press OK. The complete list of arguments for this Actor is then displayed.



input1 Name: input1
input1 C or C++ Type: int
input1 Kind:: [✓ input] output return

Fig. 7 : The frame to configure the arguments with one argument

By clicking "Commit", the arguments are added and checked. If the type is not supported, or if the kind is not correct, an Error is returned :



The type : int is not supported. Types supported : char, long (unsigned) , short, double
The JNI code generation will not work

Fig. 8 : Error with a basic type not supported

Types supported

The basic native types supported are :
`char, long (unsigned) , short, double`
The one dimension arrays are also supported
`char[], long[] (unsigned) , short[], double[]`
The corresponding Java types are : `boolean, int, char, double`
There is no type checking for the moment.

Persistence of the JNI parameters

The configuration of JNI Actors is saved in the MoML file. The arguments are property of JNI Actors, saved in the format :

```
<property name="argumentName" class="jni.Argument" value="true,false,false,long[]">
    </property>
```

, where the three booleans represent the kind of input, output and return, and the last token the native type of the argument.

When the code generation has been done one time, there is no use to redo it.

How makes the JNI Actor executable

Once you've configured the JNI Actor and all the fields of its arguments, you're ready to generated the interface code which will allow to execute the JNI Actor.

Use for this the "JNI / Generate C Interface" button.

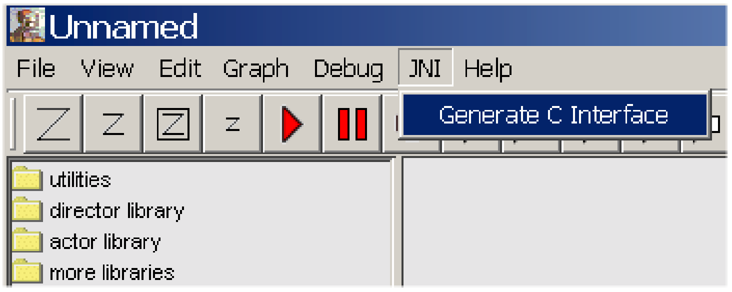
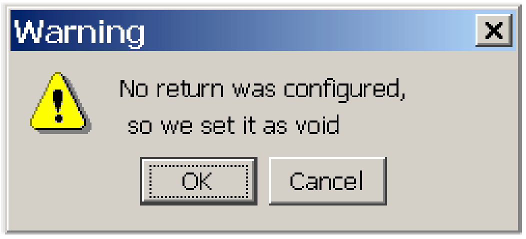


Fig. 9 : The JNI add-in Menu

If there is no return argument configured for the JNI Actor, then a default is automatically created and its type is set as void.



Warning

No return was configured,
so we set it as void

Fig. 10 : If no argument of kind return has been set

Once the button pressed, all the farmer ports are deleted, and a new port on the JNI Actor(s) is generated for each argument. The actor is renamed as "<library name>I<function name>"

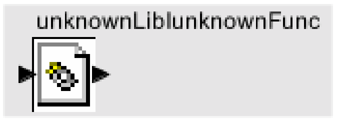


Fig. 11 : A JNI Actor after the interface generation

If the argument is an in-output, then two ports are generated for this argument.

The last step consists in opening the project file for Visual Studio (.dsp) which has been generated in the folder “ $PTII / jni / jni<existing library name> /”, and then in building the interface library (shortcut ‘F7’ or in the menu “Build/ Build jni<existing library name>.dll”).

Then the model under Ptolemy II is ready to be executed without having to restart Ptolemy.

What is actually done

Indeed, JNI Actors are working with classes which are auto-generated from the configuration made. This classes contain the loading of the auto-generated native interface library, which will call the existing native library. They have a fire method which is called during the fire method of their corresponding JNI Actor, and which calls the existing native function.

For each JNI Actor with a different function these is a class generated and compiled. The native part has to be manually compiled, from the visual project.

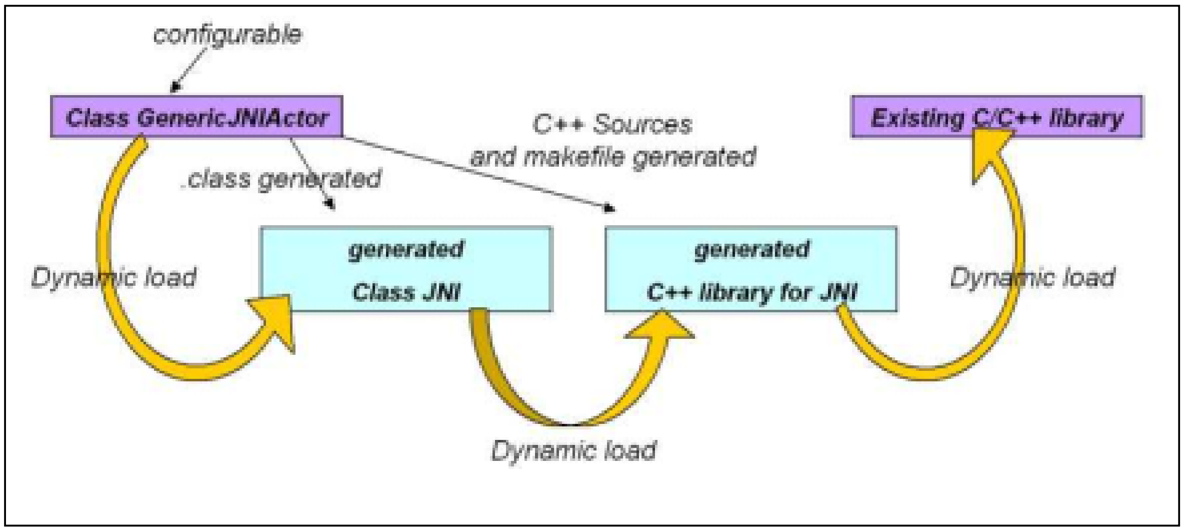


Fig. 12 : Principle of the add-in