HowTo setup a jETI Service Provider (jETI Server)

07.06.2006

Welcome to jETI

This document should give you a short overview on how to install a jETI Server to provide services for remote usage within the jABC. We show the few configuration steps necessary to start providing services, and demonstrate them on an example.

FB 4 UNIVERSITÄT DORTMUND

Prerequisites: To begin installing a jETI Server you would need to have some things already installed on your machine. The following table shows up what you would need to be able to start over.

Required	min. Version
A Unix based OS (like Linux, Solaris, etc.)	any
Java Software Development Kit	1.5.0_06
Apache Ant	any
Bash shell	any

Installation: The jETI Service Provider comes as a ziped archive which you only need to unzip to a directory of your choice. After that, the server is ready to run. Anything else needed is include within the archive.

Configuration: After installing the server you have to change some configuration files to your needs. There are three files which need your attention.

- *config/jeti.properties*
- *sibcreator/etc/sibcreator.properties*

jeti.properties

Variable	Description
jeti.toolxml	The location of the tools.xml file
jeti.session.folder	The relative folder where the toolsever shall store session data (with ending /)
jeti.session.timeout	Session timeout in milliseconds
jeti.session.checkinterval	Time between each check for timed out sessions
jeti.connector.soap.warfile	The relative WAR file of the soap connector
jeti.connector.soap.port	The port the soap connector shall use
jeti.connector.sepp.port	The port used by the sepp connector (used by JavaME jETI clients)
jeti.configurator.port	The port where you could reach the HTML configurator

Variable	Description
jeti.log4j.propertyfile	The location of the configuration file used by the logger

sibcreator.properties

Variable	Description
sibCreator.templateDir	The location of the template used for generating SIBs
sibCreator.logfileName	The location of the log file
sibCreator.etiToolList	Location of the tools.xml
sibCreator.etiServer	The URL of the jETI Executor Webservice
sibCreator.velocityOutputPath	The relative location to store the generated sibs

Running the server: After configuring the server you can start by executing the *toolserver.sh* script within the main directory of the server. Be sure to set your path to JAVA_HOME correctly within the script file before launching the server. The script is able to handle three Parameters which are shown in the following table.

Parameter	Description
run	Starts the service provider in foreground mode (use CTRL-C to stop)
start	Starts the service provider in background mode
stop	Stops service providers running in background mode

After starting the server you need to setup your services for providing them online. At this point, jETI only supports services that can be executed at the commandline of your system. There is no support for applications using a GUI yet. Anyway, there is a way to call any Java Class which is able to be instanced at the server site (see section *Configuring services by hand* for detailed information).

Configuring services using the HTML Configurator: The easiest way to configure your services would be using the HTML configurator which is a graphical user interface to edit your *tools.xml* where the service descriptions are stored. You can access the interface by launching a web browser and opening the URL of the service provider by using the configurator port defined inside your jeti.properties (default is 8081). The HTML Configurator shows up giving you a list of all services currently installed (see figure 1 of the appendix). By selecting the checkboxes in front of each service you can enable or disable the availability of the tools. Behind each service you will find links to edit or delete a tool or watch a detailed description. To create a new service just click on the “Create new tool“ button at the bottom of the site and the create/edit page shows up (see figure 2 of the appendix). There you will find options to define the name of your service (containing no spaces!), a description and the commandline call and parameters. The expert mode is only needed if you want to change settings of the internal executing environment, e.g. to call a Java Class instead of a command line service, which we will not discuss at this point (see section *Configuring services by hand* for detailed information).

The following table gives an overview of the possible parameter settings you can change within the HTML Configurator.

Parameter type	Class type	Description
Optional	String, Integer, Float, Input file, Outpu file	An optional parameter editable inside the generated SIB
Required	String, Integer, Float, Input file, Outpu file	A required parameter editable inside the generated SIB

Parameter type	Class type	Description
Optional with default value	String, Integer, Float, Input file, Outpu file	An optional parameter with a predefined value editable inside the generated SIB
Static	String, Integer, Float, Input file, Outpu file	A static value which is not editable inside the generated SIB.

To change the type of a parameter to a class not supported by default you also need to switch to the expert mode mentioned above by clicking the relating button at the overview page of the HTML Configurator. To define the commandline call to your service, simply add a static parameter containing the execution call. As there are no spaces allowed within parameter definitons you will have to define one parameter per word of the call. For instance if your service is invoked by calling *java -jar servic.jar* you will have to define three static parameters containing one single word of the commandline call each. The order of the paramters can be switched by clicking on the arrows displayed within the parameter overview.

To define relations between parameter you are able to add so called *unions* which simply means that you define a “container“ containing a set of parameters. For instance, if an optional parameter within a union is not defined by the user within the generated SIB later on, the whole set of parameters inside this union is ignored during execution of the service. This is usefull when calling a commandline service handling optional parameters. If a value for this paramter is set the call is made invoking the paramter and the value, else neither is done (e.g. *convert -rotate 90*), if inside a union.

After defining and enabling your services you can click on the download link to the left of the HTML Configurator site, getting to the download page (see figure 3 of the appendix). Do not forget to save your *tools.xml* configuration by clicking on the relating button at the bottom of the overview site of the HTML Configurator. Otherwise your changes to the services are not activated!

Once your services are defined and enabled, you are ready to generate the SIBs to be used within the jABC by the clients developing workflows and service chains. At the download page you can invoke the generation of SIBs to use within the jABC by clicking on the download button at the bottom of the site. After the generation is finished you will get a JAR file containing all the SIBs of your enabled services. Simply import this JAR within your jABC project and use your remote jETI tools as if they were locally available. If there is an error during generation it will be presented at the top of the download page.

Configuring services by hand: Expert users may also configure their services by directly editing the *tools.xml* file. The following instructions assume that you are familiar to XML and know how to handle those files.

<etitoolserver></etitoolserver>

This is the surrounding root tag of the whole XML document. It has no parameters.

<tool></tool>

This is the surrounding tag of each service to be defined whithin the *tools.xml*. It has options *name* defining the name of the service, *active* defining the status of the service, *class* setting up the class to be instanced by the server and *method* defining the method to be called. If you want to call a commandline application as a jETI service you have to specify the specific runtime class and method fitting the requirements of your operating system and special parameters defining the commandline instructions. The default would be *de.unido.ls5.eti.executor.RuntimeUnix* calling method *exec*.

In general you are able to remotely call any method of any class of Java objects on the server. The order of the parameters within the *tool.xml* file defines the signature of this method the jETI Server is looking for via Reflection API.

<parameter></parameter>

This is the surrounding tag of each parameter. It has options *name* defining the name of the parameter, *class* specifying the type of the parameter, *value* giving an initial fixed value, *default* giving an initial default value, *required* specifying whether the parameter has to be set or not, *description* giving detailed information of the parameter and *classArgument* to test if the parameter fullfills special conditions (for instance if a number has a value between 0 and 1 or if a String contains only characters). To use the *classArgument* options the parameter class has to implement two methods *setClassArgument(String s)* and *validateClassArgument()*. If *validateClassArgument()* returns an error, an exception has to be thrown and delegated to the client With all those options to be set there is the possibility of defining different types of parameters as shown by the table below.

Required	Value	Default	Type
false	not set	not set	optional
false	set	not set	static
false	not set	set	optional with default
false	set	set	not def.
true	not set	not set	required
true	set	not set	not def.
true	not set	set	not def.
true	set	set	not Def.

The meaning of the parameter types is defined as follows.

Type	Description
Optional	If the parameter is send by the client, it is instanced by the server, otherwise not.
Optional with default value	If the parameter is send by the client, it is instanced by the server, otherwise it is set to the default value.
Required	Parameter has to be send by the client. If not a RequiredParameterNotSedException is thrown and send to the client.
Static	Parameter with a static value. Could not be changed by the client.

<array> </array>

To freely define the number of parameters of a type you can give arrays as parameters of a method. The array tag has an option *class* defining the type of elements of the array to be taken. For instance the method *exec* included within the *RuntimeUnix* class as mentioned above takes an array of objects as a parameter. To define which values are included within an array the tag has to include different parameter tags. It is important to keep the *class* options of arrays and parameters consistent.

<union> </union>

The union tag combines several parameters to a group of parameters depending on each other. For instance, if a union contains a required parameter which is not send by the client as it has to be by definition, the whole union is ignored. A union has no options to be specified.

Defining own classes for input- and output files

To write own classes relating to input- and output files, those classes have to inherit *de.unido.ls5.eti.executor.FileReference* because of internal methods handling relative and absolute filenames within session at the server. To get the SIB Creator to know if the file is an input or output parameter the class names have to begin with “Input“ or “Output“.

Example

An example *tools.xml* defining a service rotating an image by using the *convert* tool of the ImageMagic package is shown in figure 4 of the appendix.

Appendix

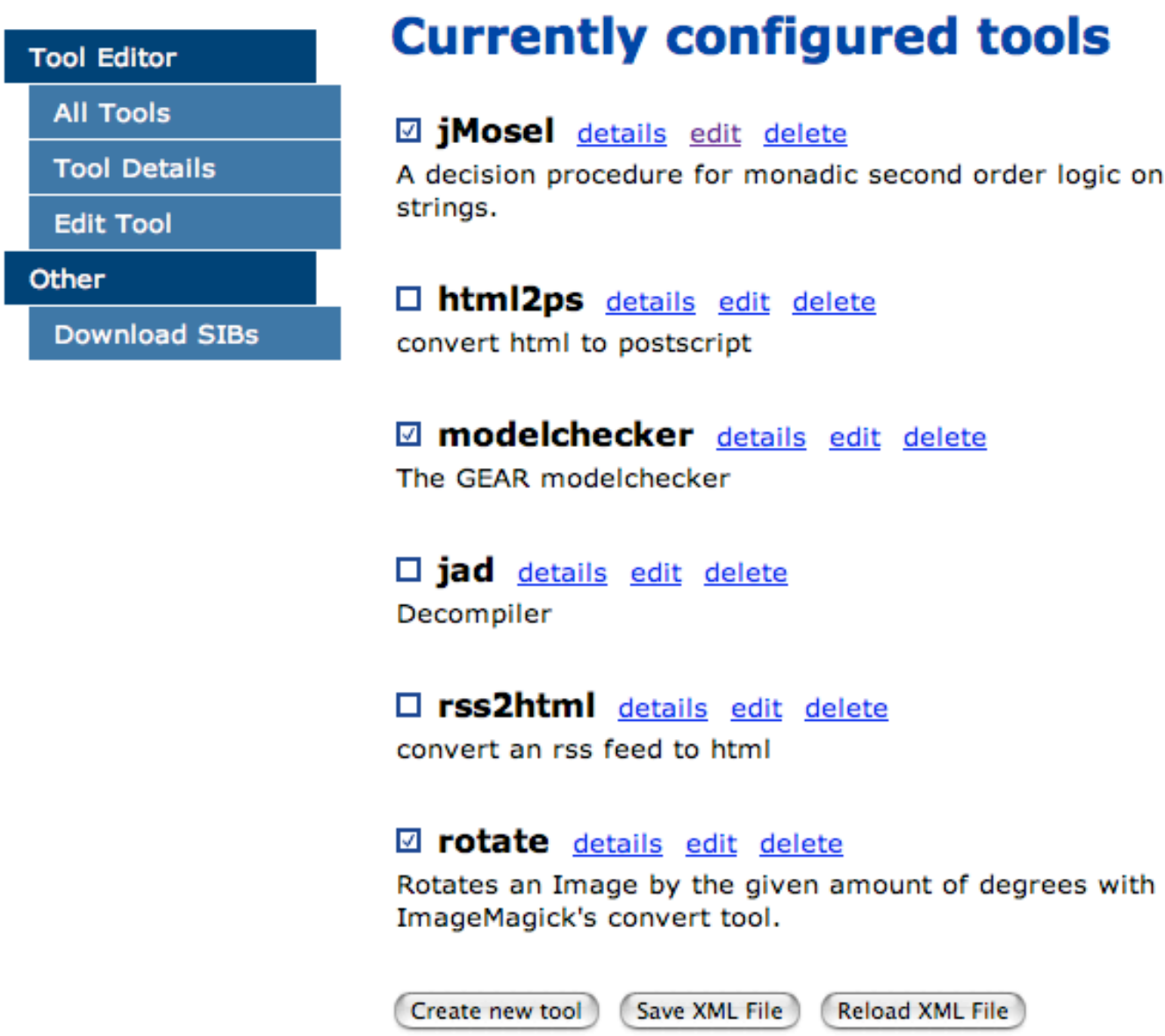

Figure 1 - The HTML Configurator tools overview page

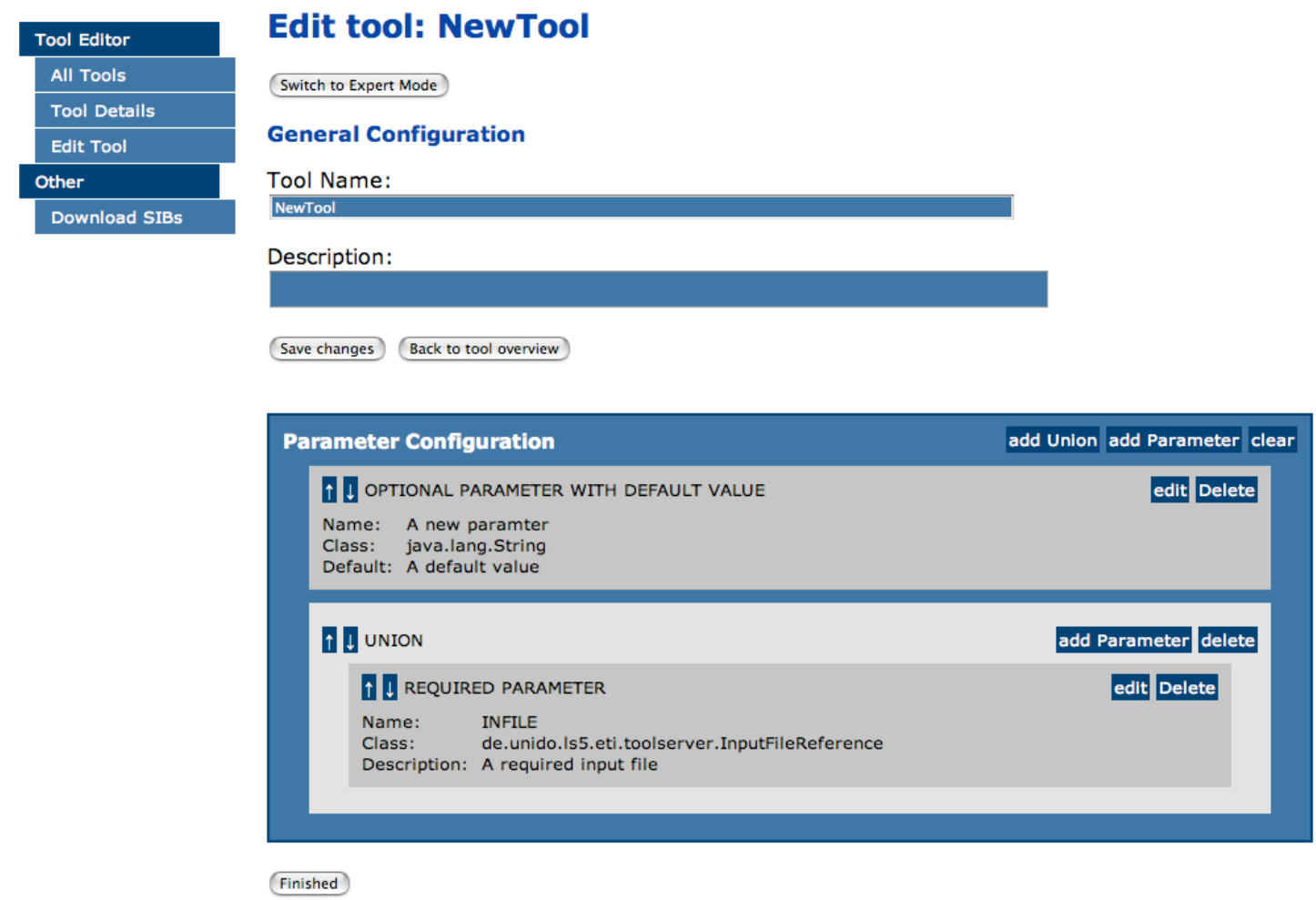

Figure 2 - The HTML Configurator tool creation/editing page

Figure 3 - The HTML Configurator SIBs download page

```
<etitoolserver>
     <tool name='rotate' active='true' class='de.unido.ls5.eti.executor.RuntimeUnix'
method='exec'>
          <description>
               Rotates an Image by the given amount of degrees with ImageMagick's con-
vert tool.
          </description>
          <array class='java.lang.Object'>
               <parameter class='java.lang.String' value='convert' />
               <union>
                    <parameter class='java.lang.String' value='-rotate' />
                    <parameter name='ROTATEANGLE' class='java.lang.Integer' default='90'
description='Amount of degrees to rotate, default is 90' />
               </union>
               <union>
                    <parameter class='java.lang.String' value='-radial-blur' />
                    <parameter name='BLURANGLE' class='java.lang.Integer' requi-
red='true' description='adds a radial blur to the image' />
               </union>
               <parameter name='INFILE'
class='de.unido.ls5.eti.executor.InputFileReference' classargument='gif' requi-
red='true' description='File to read from' />
               <parameter name='OUTFILE'
class='de.unido.ls5.eti.executor.OutputFileReference' required='true' description='File
to write rotated image to' />
          </array>
     </tool>
</etitoolserver>
```

Figure 4 - A tools.xml example