Developer documentation of FullSWOF_UI

Version 1.01.01 (2015-03-17)

2012-07-19

Creation of a configuration tree

The architecture of FullSWOF_UI is designed to allow an easy modification of the number and type of entry parameters used by FullSWOF, and, at the same time, to allow the user to choose among various configurations. The configurations available today are for FullSWOF_1D and FullSWOF_2D. To ensure a backward-compatibility, it is recommended to create a configuration for each major version of FullSWOF instead of modifying the 1D and 2D existing configurations.

Create a configuration

All the classes required for the creation of a configuration are located in the package *model*. For more information, look at the documentation of these classes.

A configuration is a tree, made of objects of type *Node*. While the creation of a node is simple, to describe a configuration fully often require numerous lines of code. For the clarity of the code, it is recommended to create a configuration using a static method located in a class completely dedicated to the creation of this object. This method will return the tree root, to which had been attached the other nodes. Put this class in the package *model.definition* which already contains the classes dedicated to the building of the 1D and 2D configurations.

Every visible item of the configuration panel match one node. This is valid for the parameters themselves (*ExternalNode*) but also for the parameter groups (*InternalNode*) such as the various tabs of the parametrization panel or the groups of parameters on a single page (for example the group « left limit » located in the tab « Limits »).

Creation of the root and internal nodes

Start by creating the root tree, with the type *RootNode*. It is to this object that it should be pointed to make reference to the whole tree.

```
Node root = new RootNode(String name, String description);
```

Be careful that the name of this none is the one of the configuration, and is sometimes used by the code, for example to open a previously-created file. Be careful not to change it or to use a name already existing for another configuration.

Then create the internal nodes and add them to the parent node:

```
Node node1 = new InternalNode(String name);
```

```
root.addNode(node1);
```

The first level of internal nodes will be the tabs in the parameterization panel, and the lower levels will be displayed as frames in the page. There is no limit to the number of imbricated levels.

Creation of the parameters

Once the parameter groups have been created, the parameters have to be added. All the parameters come from the abstract class *ExternalNode*. For a detailed description of each type of parameter, look at the Doxygen documentation of the class *ExternalNode* and of the classes that origin from it. Here is an abstract of the most useful classes:

- *FieldParameter*: text field accepting any value
- *FileParameter*: allow to specify the path to a file
- *IntegerParameter*: Integer number (with the possibility to define a range of accepted values)
- *FloatParameter*: Float number (with the possibility to define a range of accepted values)
- *MultipleChoiceParameter*: Parameter for a finite set of possible values (displayed as a drop-down menu). To add values, use the command addPossibleValue(String name, String value)

The other classes extending *ExternalNode* are parameters with specific behaviors (such as the suffix of the output directory) or external nodes which are not parameters (such as the nodes used to create entry files).

Dependencies

For the value of a node to activate a reaction on other nodes, it is required to create a dependency (abstract class *Dependency*). A dependency alwas has a “master” node, a “slave” node and a target value. If the value of the master node is equal to the target value, a change is carried out on the slave node. This tool is quite useful with the parameters having multiple choices. Hence, a specific choice can deactivate other parameters or change their value. However, the dependencies can be used for every type of parameter.

The following example, allow to deactivate the parameter node2 if the parameter node1 has the value « 4 ». Creating the object is enough to make active the dependency.

```
new DisablingDependency(node1, node2, "4");
```

The current code define three types of dependencies:

- *DisablingDependency :* allow to deactivate the slave node for a specific value of the master node. Be careful that the slave node is not re-activated automatically if the value of the master node is modified afterward. To have such behavior, you have to use *EnablingDependency* of the other possible values of the master node.
- *EnablingDependency :* allow the opposite behavior.
- *SettingDependency :* modify the value of the target parameter when the main parameter has a specific value.

Activate the configuration

Once the configuration has been created, it must be added to the list of used configurations. They are inside the class *io.Procedures*, as a variable (table) named *AVAILABLE_CONFIGURATIONS*. Add your configuration to this table as a call to the creation function of the configuration. This is the root of the tree configuration that should be returned by this function.

The created class and the *io.Procedures class must me compiled* (command javac), and the created files .class added to the jar archive. To each package matches a directory where the file .class should be put. As for the whole project, all the new classes should be compiled using Java 6 (or above).

If you prefer to recompile the whole project, you must use the export tool of Eclipse (File > Export > Runnable jar file). This is one of the only tools that allow for the creation of a jar containing other jar, which is not allowed by the usual command java -jar.

Localization of parameters

To keep the multi-language feature of FullSWOF_UI, it is better not to write directly the names and descriptions of the parameters directly into the class. It is recommended to create a localization file for your configuration and to put it in its own directory (see below).

To access to this localization, create a *ResourceBundle* with (this example suppose that the default localization file is in the folder */l10n/myconfig/* and is named *Config.properties*):

```
ResourceBundle messages = ResourceBundle.getBundle("l10n.myconfig.Config", Start.currentLocale);
```

This command will convert the messages in the designated language if it is available. Never specify the country code in the recovery method of *ResourceBundle*.

Then, instead of writing directly the name of the parameter in the class, write:

```
messages.getString("key");
```

And your localization file should include the line:

```
key = Name of parameter
```

The only exception to this rule is the name of the root configuration (*RootNode*) which should be written directly into the class and be different from the other configuration names.

For more details, look at the Oracle documentation for the class *ResourceBundle*.

Localization

FullSWOF_UI is fully internationalized, i.e. the user messages are not written directly into the code. The software load on launch the messages from a file, which allow to choose easily the language. The localization is defined as giving a file containing the translations for the messages in a specific language. The interface is currently localized in French and English. Adding of a new localization can be done without recompiling the jar archive. Adding the localization files is enough to get the new language displayed as a new choice in the preference interface.

The localization files are only text files with a list of pair key/value. To create a localization file, copy the content of the default file (see file naming) and replace the values with the suitable translation. The key should remain identical. If a key is missing from your file, the interface with use the corresponding value of the default file.

Some words (for example the answers “Yes”, “No” and “Cancel” of the dialog boxes) are not defined by the localization files but by the Java virtual machine. The virtual machine installed on the user's computer may not have the translation for these words in the selected localization. In this case, these words will be displayed in language different from the one chosen for the interface (the default language of the JVM).

File naming

The localization files are distributed withing several folders located in the directory */l10n* of the jar archive. In each folder, there are several files with a name always using the pattern name_CountryCode.extension (e.g. *UIMessages_fr.properties*). For a localization file to be taken into account, it must follow this pattern. The contry code is a two-letter code defined by the ISO 639 standard.

Moreover, each folder has a file without country code. It is the default localization. If a translation key is not available in one of the created localization file, the key located inside the default file will be used instead.

The different localization files

The localization files are distributed inside several directories. Each directory matches one type of file and contains the different localizations available for this file type.

- */l10n/ui* contains the files *UIMessages*. They are the main messages of the interface. If you want to add a localization, you need to create a file *UIMessages* with the suitable country code. The existence of this file while allow the display of the language in the preferences. Please, note that this file, as the others, can be empty or incomplete. In such case, the interface will use the translations of the default file.

- */l10n/config* contains the localization files specific to each configuration of parameters (a directory for each configuration: FullSWOF_1D, FullSWOF_2D...). We recommend to create a new directory for each defined configuration instead of re-use the file for other configuration, even if numerous keys are identical.

- */l10n/manual* contains the user manual displayed by the interface. Unlike the other files, it is an HTML file wich content is displayed in a window of the software.