VisNow module creation documentation

This documentation is the description of creating plugins with external module libraries for VisNow. As VisNow is the application written in Java programming language, so implementation of plugins and modules requires programming in JavaSE. It is recommended to use Java source in version 1.6.

VisNow modules - basics

In the most basic approach, from the programming point of view, VisNow module is a Java package containing two main parts:

- Module schema description
- Main module class

It is recommended to follow Java guidelines for package naming policy while creating VisNow modules. The good practice is to name packages with the inverted hierarchy of institution and project. For example VisNow packages created by ICM at the University of Warsaw in Poland are named **pl.edu.icm.visnow**. Usually VisNow module packages are stored in **lib** subtree. Building your own modules library can be stored in a package e.g. **your.institution.visnow.lib**.

Module schema description – module.xml

Module schema description is created via a XML file named **module.xml**. This file is responsible for definition of the following elements:

- Main class of this module
- Module name
- Description of the module
- Input ports definition
- Output port definition

The structure of module.xml file is as follows:

```
<module name="my module name" class="ModuleMainClass">
    <inputs>
      Input definitions goes here
    </inputs>
    <outputs>
      Output definitions goes here
    </outputs>
    <description value="my module short description"/>
</module>
```

Module name and class are required, other parts (inputs, outputs, description) are optional, as not every module has certain ports. Module name is the name visible in the library tree and on module box in workspace. Module class is the main module class inside the same package (see Main module class section in this document). Inputs section is responsible for definition of input ports. Outputs section is responsible for definition of output ports. Description tag is responsible for short description of the module visible as a tooltip in VisNow library tree.

Input port is defined by **<input>** tag with the following information:

- port name
- data type
- port modifiers

Port name is the text name of the port visible in VisNow workspace and connections description, and referenced in main module class code. Data type defines the type of data this port represents (currently on of: VNField, VNRegularField, VNIrregularField, GeometryObject, see *Data format and API documentation*). Port modifiers set additional parameters of this port.
Inside input port definition there is also a place to define so called **acceptors**. Acceptors are schematic descriptions of restrictions for data that this port accepts. Multiple acceptors are combined with OR operator, so the acceptable data must fulfill one of the given acceptors.

The structure of <input> tag is as follows:

```
<input name="input_name" type="data_type_class" modifiers="port_modifiers">
      Acceptor definitions and description goes here
</input>
```

Output port is defined by **<output>** tag with the following information:

- port name
- data type

Port name is the text name of the port visible in VisNow workspace and connections description, and referenced in main module class code. Data type defines the type of data this port represents (currently on of: VNField, VNRegularField, VNIrregularField, GeometryObject, see *Data format and API documentation*). Port modifiers set additional parameters of this port.
Inside output port definition there is also a place to define so called **schema**. Schema are schematic descriptions of data that this port produces. Multiple schema are combined with OR operator, so the produced data will fulfill one of the given schemas. Schemas are used against acceptors to build modules network when no data are present.
There is one special type of output port that is defined by **<geometryOutput/>** tag. This is the default geometry output port used in visualization modules (see main module class section).

The structure of <output> tag is as follows:

```
<output name="output_name" type="data_type_class">
      Schema definitions and description goes here
</output>
```

The following list gives a brief summary of XML tags used in module.xml:

- **<module>** - root tag to define module with tag parameters:

    - **name** – a text name of the module (as seen in library tree panel and workspace in VisNow), required
    - **class** – a text name of the mmain module class (the class must be placed in the same package), required
- **<inputs>** - tag for grouping input ports' definitions
- **<input>** - tag for single input port definition
    - **name** – a text name of the input port (referenced in module class code), required
    - **type** – class representing port data type, required
        - pl.edu.icm.visnow.lib.types.VNField
        - pl.edu.icm.visnow.lib.types.VNRegularField
        - pl.edu.icm.visnow.lib.types.VNIrregularField
        - pl.edu.icm.visnow.lib.types.VNGeometryObject
        - java.lang.Float, Double, Byte, Short, Integer, Long, Boolean, Object
    - **modifiers** – port parameters separated with :, required
        - NECESSARY – sets this port necessary (without this port connected module will not run)
        - TRIGGERING – sets if new data on this port starts module computation, it should be set in most cases
    - **maxConnections** – number of maximum allowed connections to this port, integer value or -1 for infinity, by default it is set to 1, optional
- **<outputs>** - tag for grouping output ports' definitions
- **<output>** - tag for single output port definition
    - **name** – a text name of the input port (referenced in module class code), required
    - **type** – class representing port data type, required
        - pl.edu.icm.visnow.lib.types.VNField
        - pl.edu.icm.visnow.lib.types.VNRegularField
        - pl.edu.icm.visnow.lib.types.VNIrregularField
        - pl.edu.icm.visnow.lib.types.VNGeometryObject
        - java.lang.Float, Double, Byte, Short, Integer, Long, Boolean, Object
- **<geometryOutput/>** - tag representing special default geometry output port used in visualization modules
- **<description>** - tag for short description of module (inside <module></module> tags), input port (inside <input></input> tags) or output port (inside <output></output> tags), used in tooltips of VisNow, optional
    - **value** - parameter with description text, required
- **<acceptor>** - tag defining an input port acceptor (inside <input></input> tags)
- **<schema>** - tag defining an output port schema (inside <output></output> tags)

Below is the example code of module.xml for Isosurface module and module representation in workspace panel:

```
<?xml version="1.0" encoding="UTF-8"?>
<module name="isosurface" class="Isosurface">
  <inputs>
    <input name="inField" type="pl.edu.icm.visnow.lib.types.VNField"
                                          modifiers="NECESSARY:TRIGGERING">
      <description value="Input for data field to extract isosurface"/>
      <acceptor>
```

```
            <param name="NSPACE" value="3"/>
            <param name="REGULAR" value="true"/>
            <param name="NDIMS" value="3"/>
            <param name="DATA_VECLEN" value="1"/>
        </acceptor>
        <acceptor>
            <param name="NSPACE" value="3"/>
            <param name="IRREGULAR" value="true"/>
            <param name="DATA_VECLEN" value="1"/>
            <param name="CELLS_3D" value="true"/>
        </acceptor>
      </input>
      <input name="threshold" type="java.lang.Float"
                                    modifiers="TRIGGERING">
        <description value="Input for isolevel value"/>
      </input>
    </inputs>
    <outputs>
      <output name="isosurfaceField"
                     type="pl.edu.icm.visnow.lib.types.VNIrregularField">
        <description value="Output for surface field"/>
        <schema>
          <param name="NSPACE" value="3"/>
          <param name="IRREGULAR" value="true"/>
          <param name="CELLS_TRIANGLE" value="true"/>
        </schema>
      </output>
      <geometryOutput/>
    </outputs>
    <description value="Maps volumetric data creating surface of constant
                                                                  data values."/>
</module>
```

Main module class

Main module class is the Java class that is instantiated when the module is created in VisNow. On the most general level it extends the **pl.edu.icm.visnow.engine.core.ModuleCore** class. This class gives all the administrative interfaces to properly work in VisNow network environment. However, from the programmers point of view, direct extending of this class is very low level module programming and it is rarely required. More convenient path for main module class is to use a *visualization module* - **pl.edu.icm.visnow.lib.templates.visualization.modules.OutFieldVisualizationModule** class.

But for both paths the main module class has to have:

- a field: **public static InputEgg[] inputEggs = null;**
- a field: **public static OutputEgg[] inputEggs = null;**
- overridden method **public void onActive() {}**

The first two fields are required for static descriptors of ports based on module.xml, while onActive() method is the main computational method that is executed when module is started.

Common module API (for both **ModuleCore** and **OutFieldVisualizationModule** classes):

- Methods to override
  - **public void onActive()**
    - executed on module start in separate module compute thread
  - **public void onInputAttach(LinkFace link)**
    - executed when a connection is made on input port
    - **link** variable has information on the created connection
  - **public void onInputDetach(LinkFace link)**
    - executed when a connection is removed from input port
    - **link** variable has information on the deleted connection
  - **public void onOutputAttach(LinkFace link)**
    - executed when a connection is made on output port
    - **link** variable has information on the created connection
  - **public void onOutputDetach(LinkFace link)**
    - executed when a connection is removed from output port
    - **link** variable has information on the deleted connection
  - **public void onDelete()**
    - executed after module was deleted
  - **public boolean isGenerator()**
    - should return true is a module is a starting entry point in network graph (i.e. reader or data creator) and has no input ports
  - **public boolean isViewer()**
    - should return true is a module is a viewer module, being always the final presentation module and has no output ports
- Methods to use
  - **String getName()**
    - Returns the name of the current module.
  - **void startAction()**
    - Non-blocking method used to start a module. It notifies network engine that this module needs to recalculate. As a consequence **onActive()** method will be executed and the modules connected to output ports also started. ***WARNING!!*** When onActive() method is executed directly (not by startAction()) a network will not start, so the below modules might not run. Try to avoid calling onActive() directly!
  - **Vector<Object> getInputValues(String name)**
    - A method returning all data objects that are available on the input port. The **"name"** parameter required is the input port name defined in module.xml. The returned vector objects' type will be the same as input port data type and can be locally casted.
  - **Object getInputFirstValue(String name)**
    - A method returning first data object that is available on the input port. The **"name"** parameter required is the input port name defined in module.xml. The returned object's type will be the same as input port data type and can be locally casted.
  - **boolean setOutputValue(String name, Object value)**
    - A method setting data to the output port. The **"name"** parameter required is the output port name defined in module.xml. The **"value"** parameter is the

object to be set to the output and it has to be the same type as output port data type. The returned Boolean is the success info.

- **boolean setProgress(float progress)**
  - Sets current progress value represented by progress bar when the module is running. The set value should be from 0-1 range and represent the progress of module computation task. ***WARNING!!*** Do not set progress too often as it will strongly decrease performance.
- **float getProgress()**
  - Returns current progress value if set by setProgress() method. The returned value should be from 0-1 range and represents the progress of module computation task.
- **void setPanel(Component component)**
  - Sets the Swing or AWT component that is representing the module GUI visible in main VisNow window in module panel and responsible for module parameters steering. ***WARNING!!*** Do not use this method directly in **OutFieldVisualizationModule**. Use it only explicitly by **setPanel(ui)** in the constructor to set the default visualization GUI as module GUI. If you wish to add parameters steering GUI (called computation tab) use the method **ui.addComputeGUI(JPanel gui)**.

Module API for **OutFieldVisualizationModule** only:

- Fields to use
  - **Field outfield**
    - Use this field as a default output field result that is then automatically mapped with presentation tab parameters to the default geometry output, by the use of **prepareOutputGeometry()** and **show()** methods.
  - **RegularField outRegularField**
    - Use this field as a convenient variable to work with output regular field. Usually used to cast outField.
  - **IrregularField outIrregularField**
    - Use this field as a convenient variable to work with output irregular field. Usually used to cast outField.
- Methods to use
  - **void prepareOutputGeometry()**
    - Call this method after creating a result field and setting outField variable. It creates a default geometry for presentation of outfield. Usually called in the end of **onActive()** method. Almost always followed by **show()** method call.
  - **void show()**
    - Call this method after **prepareOutputGeometry()**. It adds the created geometry to the output object to be shown in the connected viewer. Usually called in the end of **onActive()** method. Almost always right after **prepareOutputGeometry ()** method call.

VisNow modules – the next step

As the basic module could suffice the module schema and main module class only it rarely happens in practice. Usually, in full functionality, the module model consists of:

- Module schema description
- Module class
- Parameters class
- Graphical User Interface class
- Compute core class
- Additional classes and resources

It is not required to use this scheme, however, we recommend such division of module elements.

Parameter class

Parameters class is the class that extends pl.edu.icm.visnow.engine.core.Parameters class and is responsible for holding all internal module parameters that are set via GUI and used for computations (e.g. selected data component, float value of some factor, dimension of data). It can be of course implemented without extending Parameters class, however, this class will be in future releases used to save and restore parameters so it is advisable to use it. In main module class there is a field "parameters" that should be set in the module constructor with appropriate instance of parameters class.
The extending class (usually called Params) has to set a static field:

```
private static ParameterEgg[] eggs = new ParameterEgg[] {
        Parameter definitions goes here
};
```

that defines parameters used by the module. Each definition of the parameter is declared as follows:

```
new ParameterEgg<param_class>("param_name", param_type, param_value)
```

where:

- *param_class* – is the class of the parameter value (e.g. Float, Integer, int[])
- *param_name* – is the string name of the parameter used in all referenced setters and getters inside this class
- *param_type* – is one of three values ParameterType.independent (meaning that this parameter does not depend on input data, e.g. some arbitral value), ParameterType.dependent (meaning that this parameter does depend on input data, e.g. selected component) or ParameterType.filename (meaning this is the path to the file)
- *param_value* – is the object of *param_value* type being the default value. ***WARNING!!*** Avoid setting objects other than simple numeric types in the static context as the object will be shared across several modules of the same type. Use **null** instead and set the value in Params class constructor.

Each parameter can be then referenced (read or written) by methods:

- void setValue(String name, Object value) – to set value of parameter given by "name"
- Object getValue(String name) – to get the current value of parameter given by "name"

Additionally the good practice is to not call these methods from outside Params class but to define directly setters and getters for each parameter with proper calling and returning types and proper action starting.
There are two types of events in Parameters class:

- ParameterChange event
    - Fired every time setValue method is called
    - Uses **pl.edu.icm.visnow.engine.core.ParameterChangeListener** interface to catch events with **parameterChanged(String name)** method.
    - Listeners can be added with **addParameterChangelistener(ParameterChangeListener listener)** method
    - Listeners can be removed with **removeParameterChangelistener(ParameterChangeListener listener)** method
- StateChange event
    - Fired when **fireStateChanged()** method is called
    - Uses **javax.swing.event.ChangeListener** interface to catch events with **stateChanged(ChangeEvent e)** method.
    - Listeners can be added with **addChangeListener(ChangeListener listener)** method
    - Listeners can be removed with r**emoveChangeListener(ChangeListener listener)** method

Graphical user interface class

GUI class (usually called GUI) is an extension of Component or most likely **javax.swing.JPanel** class. It is meant to be the controls panel to steer module parameters from VisNow GUI. It is should be set correctly in main module class and is then visible in the module panel area in main VisNow window. Usually it contains Java Swing components to represent and manipulate parameter values. It is recommended to use, besides standard Swing/AWT components, also some common widgets that are already available in VisNow library (e.g. component chooser, float slider, extended integer slider). See packages **pl.edu.icm.visnow.gui**, **pl.edu.icm.visnow.gui.widgets** and **pl.edu.icm.visnow.lib.gui** for more details.
WARNING!! It is very important while implementing GUI to properly solve the problem of loop refreshing. One of the most common issues for beginners is to get into infinite GUI refreshing and module running loops: click on widget to change value, fire widget action, set new value to module parameters, run module with new parameters, refresh GUI, set new parameter value to widget, fire widget action, and so on. The simplest way to avoid it is to use Boolean flags for silent refresh or using parameters active flag.

Compute core class

To keep main module class clean, especially for modules with very complex calculations, it is advisable to move all calculations to different class – usually called **Core**. This class is then called from onActive() with proper parameters and input data, and produces output data.

Additional classes and resources

If required, keep any other classes and resources that are exclusive to this module inside the module package.

Plugins – external module libraries

VisNow, from version 1.1, enables external plugins to be loaded to the application to add module libraries without modifying VisNow source code. A plugin that is readable for VisNow is actually a JAR file containing:

- VisNow modules in packages
- Library files

Library files are XML files that from library tree, as visible in VIsNow library panel. By default, when no plugins are loaded, library tree contains only one library – "internal". When external plugin is loaded additional libraries are added to the library tree. The library file must be present in the root directory of JAR file. When plugin is loaded VisNow checks if a proper library XML file is present:

- **simple_library.xml** – for VisNow Simple
- **extended_library.xml** – for VisNow Pro

Depending on VisNow version a respective file is loaded and library and modules listed inside are loaded. By default both files can be identical. If developer wants to provide different modules for Simple and Pro versions in the library it can be defined by different library files.

Library.xml file for definition of library tree structure is as follows:

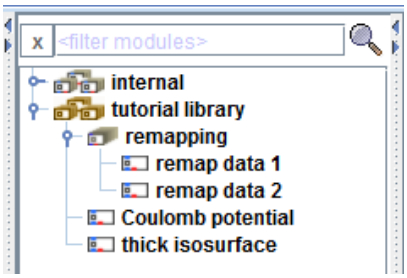
```
<library name="my external library">
    <core package="your.institution.visnow.lib.module1"/>
    <core package="your.institution.visnow.lib.module2"/>
    <folder name="my folder">
        <core package="your.institution.visnow.lib.module3"/>
    </folder>
</library>
```

XML tags used in the library file are:

- **<library>** - root tag to define library with tag parameters:
  - **name** – a text name of the library (as seen in library tree panel in VisNow), required
- **<folder>** - tag for grouping modules in subfolders/branches of library tree
  - **name** – a text name of the folder (as seen in library tree panel in VisNow), required
  - **open** – sets if the folder is extended or collapsed by default (value "yes" or "no"), optional
  - **autosort** – sets if folder lists modules in alphabetical sorting or as listed in XML (value "yes" or "no"), optional
- **<core>** - tag defining a module
  - **package** – a name of the module package, required

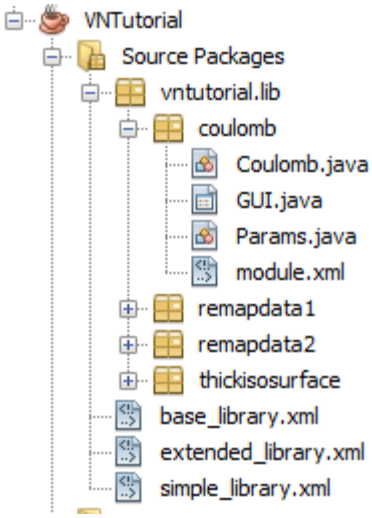
Below is the example library.xml file and its representation in running VisNow library tree.

```
<library name="tutorial library">
    <folder name="remapping">
        <core package="vntutorial.lib.remapdata1"/>
        <core package="vntutorial.lib.remapdata2"/>
    </folder>
    <core package="vntutorial.lib.coulomb"/>
    <core package="vntutorial.lib.thickisosurface"/>
</library>
```



Names of modules visible in the library tree are defined in module.xml file.

The example below shows the most usual structure of plugin:



where each package in vntutorial.lib represents a single module, and coulomb package/module has module.xml file with module structure description, Coulomb.java file with main module class, GUI.java file with GUI class and Params.java file with parameters class. Library XML files are defined in the root folder.