Geomop 1.1.0 - reference guide

Geomop is a collection of supporting tools for the Flow123d simulator of transport processes in the fractured porous media.
Currently it consists of following tools:

Layers - preparation of layered computational geometry from the GIS data
Model Editor - editor of the main Flow123d input file in YAML format
Job Panel - running and management of job on distributed computational resources
Analysis - arrangement of complex scenarios (without GUI yet)

1 Installation

Installation packages are available at http://geomop.nti.tul.cz/packages/1.1.0.
Standard graphical instalator is available for Windows OS. To run the installation execute file geomop_1.1.0_x86.exe and go through installation wizard. Individual tools are executable by their shortcuts or by batch files. Batch files are in default installation located in directory: c:\Users\UserName\AppData\Local\GeoMop\bin\.

Known issue in Win 10:
Terminal windows are displayed within the installation, in case of accidental click on the terminal, the installation process is suspended, you must press the ESC key to continue.

2 Layers

The Layers module provides a tool to prepare layered computational geometries and meshes for hydrology models. Provided features include:

- import of GIS data
- approximation of point grids by B-spline surfaces, these surfaces can define layer interfaces
- 2d vector editor of horizontal structuring of individual layers
- editor of regions (used to set model parameters and boundary conditions later on)
- meshing via. GMSH meshing SW.

The Layers module is still in the early development phase although its stability is much better compared to the version 1.0.0. In order to diminish inconveniences associated with possible crash the autosave function is implemented. The data are saved about every second into a backup file associated with the current file. In the case of crash, the user is asked for recovery when the original file is opened. If recovery is rejected the backup file is deleted.

2.1 Overview

When creating a new file or starting the application you are asked for determine extent of your initial view. This can be given either by providing a grid file with definition of the terrain, or by selecting an initial GIS shapefile, or by providing the extent manually. Alternatively one can open an existing geometry file.
The application layout is depicted on Figure 1. The key components are:
Topology Editor - specialized 2d vector editor of the horizontal structuring of a single layer. Extent of the terrain grid is marked by a gray quadrilateral.
Layer Panel - scheme of vertical structuring into layers (strata). Allows editing of vertical structuring and setting association of layer interfaces with surfaces configured in the surface panel.
Region Panel - setting computational regions to the polygons and line segments in the topology editor window. Editor of region properties.
Surface Panel - interpolation of point grids into B-spline surfaces that can be used for interfaces.
Background Panel - allows to set GIS layers to display for one or more *Shape files* opened through `File -> Add shape file …`

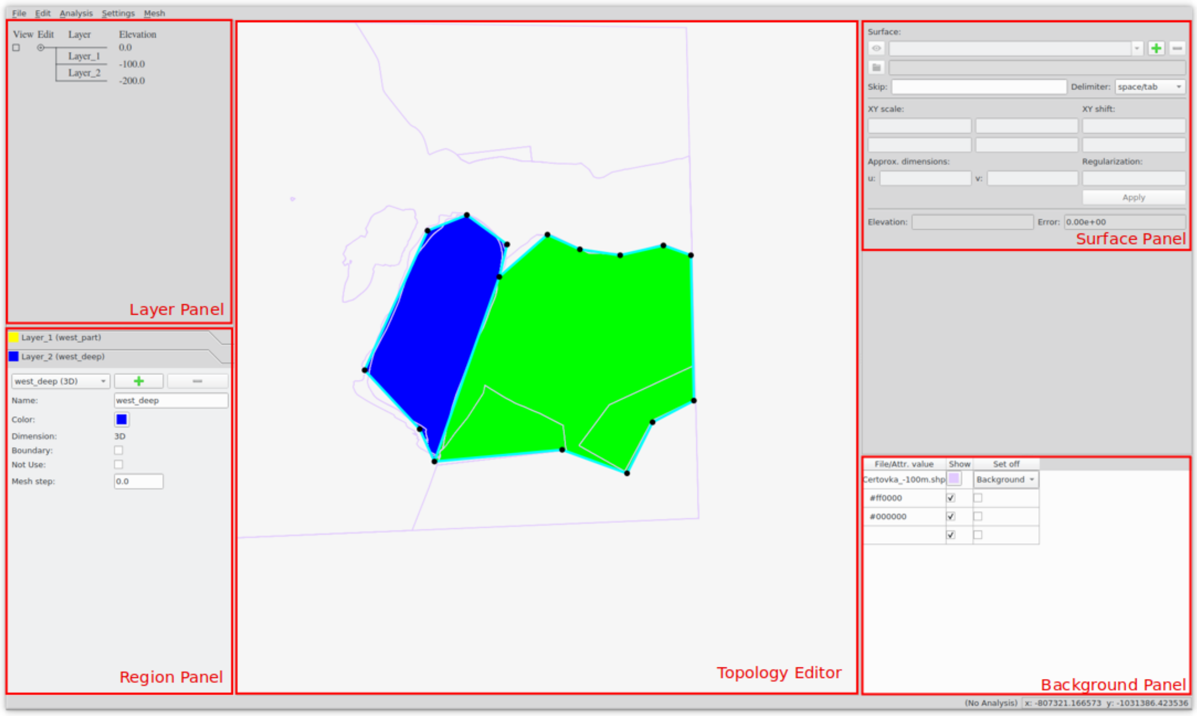


Fig. 1: Layers editor - application layout.

2.2 Layered geometry

Whole 3d geometry consists of individual geological layers. Layers can be of two types: *Fracture layer* - a surface layer representing a horizontal fracture and *Stratum layer* - a volume layer. The stratum layers are delimited by top and bottom *interface* while the fracture

layers directly lays on a single interface. Single interface is either given by constant elevation or by an interpolated B-spline surface.
Horizontal structuring of layers is called *Topology. Layer block is a* continuous block of layers sharing a same topology. Topology consists of nodes, segments(lines) and polygons. For stratum layers these objects corresponds to wells, fractures, and bulk regions in the 3d geometry. For fracture layers the segments and polygons corresponds to wells and fractures in the 3d geometry. Wells, fractures and bulks across the layers can be grouped into regions with the same meaning as in the Flow123d input file. All horizontal interfaces and vertical interfaces given by segments are captured in the resulting mesh.

2.3 Layers panel

Layers panel displays vertical order of the layers from the top to the bottom. Layers are organized into layer blocks sharing the same topology. Layers in the common block are marked by the vertical line at the left part of the layer panel. Interfaces between stratum layers are depicted by horizontal lines ended with depth of the interface taken in the center of the domain.
Click on a layer opens a context menu allowing to:

- add interface - split the layer into two logical layers, new layer is created
- rename the layer
- remove the layer

Click on the depth allows to:

- set the surface of the interface (see Surface Dialog section)
- set/remove fracture layer on the interface
- Prepend or append the layer for the first and last interface, respectively.

Gotchas:

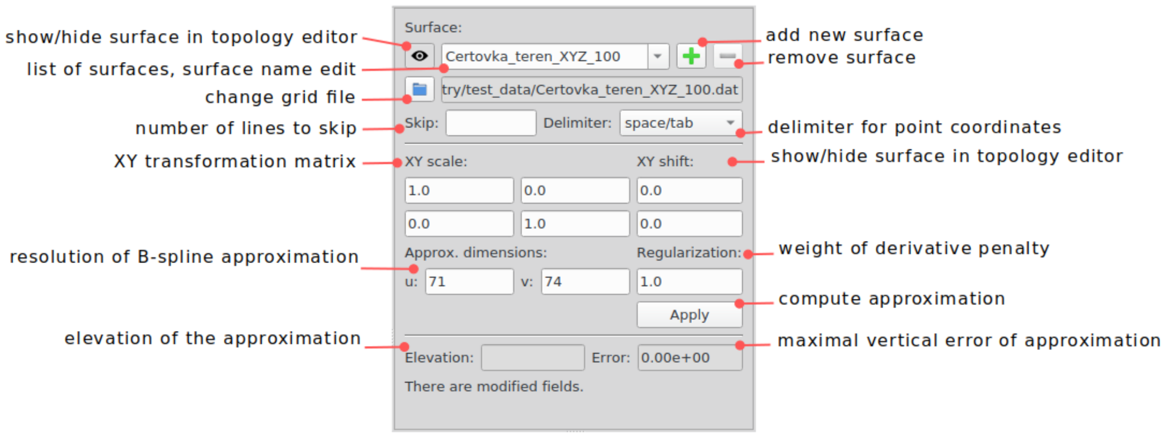
- Work with more layer blocks with different topologies is not very stable yet.
- Missing support for non-compatible layers (used for non-compatible fractures and wells and for selecting groups of elements in resulting mesh for various post-processing)
- No support for inclined vertical interfaces. Can not move with nodes on individual interfaces.
- No graphical support for the grid transform.

created in the surface panel. A single surface can be vertically scaled and shifted and used for several interfaces.

2.3.1 Surface Dialog

One can set the surface of an interface either given by a constant elevation or given by the B-spline surface created in the Surface Panel from a terrain points grid. In the second case user can also specify vertical scaling and shift that is used only for particular interface. This way single approximation can be used for several interfaces.

2.4 Surface Panel


show/hide surface in topology editor
list of surfaces, surface name edit
change grid file
number of lines to skip
XY transformation matrix
resolution of B-spline approximation
elevation of the approximation
add new surface
remove surface
delimiter for point coordinates
show/hide surface in topology editor
weight of derivative penalty
compute approximation
maximal vertical error of approximation
Surface:
Certovka_teren_XYZ_100
try/test_data/Certovka_teren_XYZ_100.dat
Skip:
Delimiter: space/tab
XY scale:
XY shift:
1.0
0.0
0.0
0.0
1.0
0.0
Approx. dimensions:
Regularization:
u: 71
v: 74
1.0
Apply
Elevation:
Error: 0.00e+00
There are modified fields.

Surface panel serves to create a B-spline approximation from a given external file with grid of points. The new surface can be added by the ‘plus’ icon, the grid file must be selected through the standard ‘Open file’ dialog. The grid file of a created surface can be reloaded or changed by the ‘change grid file’ icon on the left. Name of the surface is automatically derived from the file name, but can be freely edited. The grid file is a text file with single point per line given by X, Y, Z coordinates separated by a selected delimiter. One can also specify number of lines from the beginning of the file to skip.
After grid points are loaded the minimum area bounding rectangle is computed and showed in the topology editor window. User can specify transformation matrix and shift for XY part of the points. Quality of the approximation is determined by the number of the intervals used in the ‘u’ and ‘v direction. Further the regularization parameter can be used to smooth out the approximation. The approximation is constructed after the ‘Apply’ button is pressed, which also store the approximation permanently into the surface list.

2.5 Region Panel

Region panel have a tab for every layer. On a tab you can select actual region from the list of all regions, create a new region and edit the actual region. Region properties are:
Name - name of the region (leading dot for the boundary regions is added during meshing).
Color - used to highlight objects of the selected layer belonging to the region
Dimension - topological dimension, set when region is created.
Boundary - indicator for boundary regions.
Not-used - indicator for regions excluded from the geometry. Three predefined None regions for every dimension are always “not-used”. These are default regions for new topology objects if no actual region is selected.

Gotchas:

- No support for sets of regions (composed regions).

2.6 GIS data display

One or more ESR shape files (*.shp) can be loaded via `File -> Add shape file …`
GIS objects to display (or highlight) can be filtered by individual attributes. Every layers have individual color.

Gotchas:

- no coloring per attribute is supported
- no data export from the GIS to topology is supported
- no support for displaying bitmaps or grid data implemented

2.7 Topology editor

Main vector editor window serves for drawing horizontal structure of the geological layers. The GIS layers selected in the GIS panel are displayed at background. The extent of the terrain grid is displayed at background as a light gray quadrilateral.

Move view

wheel - zoom in / out
right click + drag - move the canvas

Editing operations (left mouse button)

left click - start drawing a polyline, Esc - drop the moving segment and stop the line drawing
ctrl + left click - create single node, stop the line drawing
left click + drag - moving a node or a segment with all connected segments

Selection and region operations (right mouse button)

right click - select an object (node, segment, polygon)
shift + right click - add object to selection
ctrl + right click - set region of the object to the actual region (if dimension match)
ctrl + shift + right click - set actual region according to the object
alt + ctrl + shift click - set region of corresponding object in every layer within the same block to the actual region selected at the tab of every layer.
alt + shift + shift click - set actual region of every layer according to the object of that layer

Selected object can be deleted through `Edit -> Delete`
Undo and Redo operations are accessible through `Edit -> Undo/Redo`

Gotchas:

- Undo and Redo are not very reliable. Save often.
- Missing hot-keys (delete, undo, redo) and not very practical mapping of operations.

2.8 Meshing

Meshing of created geometry is performed through main menu action: `Mesh -> Make mesh …` which opens the meshing dialog. User can optionally specify the mesh step that is used to scale mesh step specification for individual regions. Meshing process is started by pressing the 'start' button. Standard output of the meshing process done by GMSH is displayed in the window. Premature termination can be enforced by the 'kill button.

The meshing process consists of three phases: geometry construction, meshing and region assignment.

Geometry construction
Geometry construction is done by the GeoMop itself. Individual layers and interfaces are glued together forming a 3d geometry in the BREP format.

1. Up to three different topologies are merged for every interface (layer above, fracture, layer below).
2. Planar points, segments, and polygons are projected to the B-spline surface approximation associated with the interface forming the vertices, edges, and faces in the 3d space.
3. For a fracture layer, these objects directly forms principal geometry entities.
4. For a stratum layer, the corresponding points, lines and polygons for the top and bottom interface are extruded into vertical edges, faces, and volumes, respectively.
5. The mesh step is assigned to every vertex X as a minimum of the mesh steps of the adjacent regions. These include the region possibly assigned with the vertex X as well as all regions having X on their boundary.
6. The geometry is saved in the BREP format, the mapping of the vertices to the mesh steps are saved into the main geo-file.

Mesh construction
The meshing of the geometry is performed by the GMSH software in version 3.0.

1. The geo-file (*.geo) forming the main GMSH input is created. It contains a reference to the BREP file and mapping of vertices to the mesh steps.
2. The GMSH software is called and meshing is performed for all vertices, edges, faces and volumes regardless of their regions (i.e. including the NONE region and regions with 'Not in use.' flag).

Region assignment
The raw mesh is post-processed by the GeoMop.

1. Regions are mapped to the mesh elements through the ID numbers of the geometry primitives.
2. Elements with None region or regions 'not in use' are removed.
3. The regions are introduced as the *physical groups* (see documentation of the GMSH mesh format). For boundary regions the names are prefixed by the dot as required by the Flow123d.

Gotchas:

- No mean to visualize the geometry or the mesh within the GeoMop. GMSH can be used.

3 Model Editor

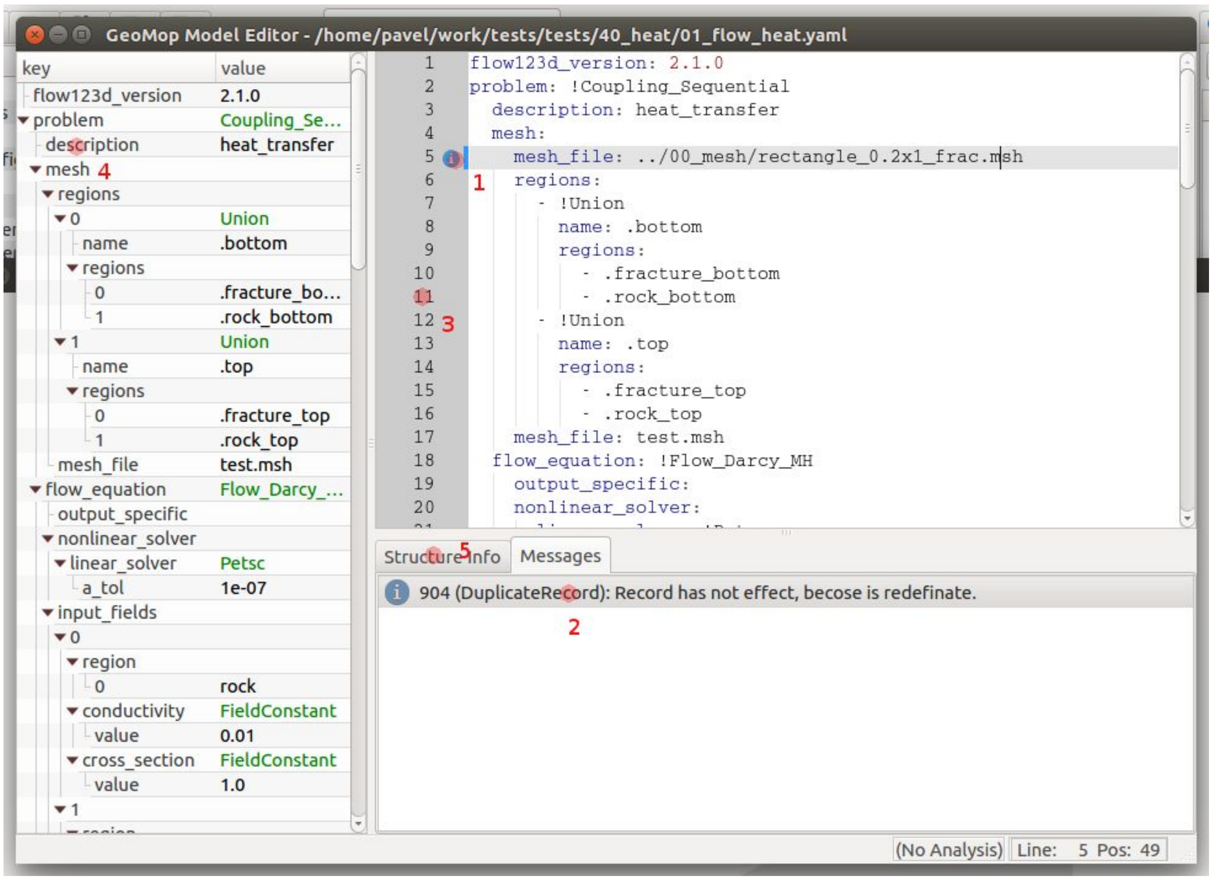
3.1 Overview

Model Editor is a specialized editor of the YAML configuration files of the Flow123d simulator.
Main features of the editor:

- validation of the configuration files (see: Messages tab)
- context-sensitive help widget describing the data types (see: Documentation tab)
- visual representation of the whole data structure (see: Tree window)
- autocompletion of keys and values while typing
- import of old-format (CON) configuration files and conversion between format of different versions of Flow123d
- data structure transformations of the edited files
- usual text editor functions

Structure of the input YAML file for the simulator Flow123d is described in a machine readable form that allows basic validation of the input, but contains the documentation of the input structure as well. Using the format description the Model Editor adopts to various versions of the simulator and is also able to do semi-automatic conversion of various versions.

The application consist of three windows. On the left there is the *Tree window* displaying the content of the file in the tree structure. On the right-top there is the *Text editor window* and finally on the right-bottom there is either the *Notification tab* reporting errors in the input file, or the *Documentation tab* with context dependent documentation of the file format. Basic layout is on Figure 2.



GeoMop Model Editor - /home/pavel/work/tests/tests/40_heat/01_flow_heat.yaml

key	value
flow123d_version	2.1.0
problem	Coupling_Se...
description	heat_transfer
mesh	
regions	
0	Union
name	.bottom
regions	
0	.fracture_bo...
1	.rock_bottom
1	Union
name	.top
regions	
0	.fracture_top
1	.rock_top
mesh_file	test.msh
flow_equation	Flow_Darcy_...
output_specific	
nonlinear_solver	
linear_solver	Petsc
a_tol	1e-07
input_fields	
0	
region	
0	rock
conductivity	FieldConstant
value	0.01
cross_section	FieldConstant
value	1.0
1	

```
flow123d_version: 2.1.0
problem: !Coupling_Sequential
  description: heat_transfer
  mesh:
    mesh_file: ../00_mesh/rectangle_0.2x1_frac.msh
    regions:
      - !Union
        name: .bottom
        regions:
          - .fracture_bottom
          - .rock_bottom
      - !Union
        name: .top
        regions:
          - .fracture_top
          - .rock_top
    mesh_file: test.msh
  flow_equation: !Flow_Darcy_MH
    output_specific:
    nonlinear_solver:
```

Structure Info | Messages

904 (DuplicateRecord): Record has not effect, becose is redefinate.

(No Analysis) | Line: 5 Pos: 49

Figure 2: Layout of the application, interplay of the Tree, Editor and Messages.

3.2 Text editor window

Displays the input YAML file and allows its free editing. The editor offers a context-sensitive completion feature that can be activated by pressing Ctrl+Space. Automatic completion can be turned on through `Settings -> Options ⋯`
Click to the line number on the left part of the editor window (see Figure 2/3) finds and highlights corresponding place in the tree window.
Content of the window is continuously validated against format of the Flow123d given by the ‘flow123d_version’ key. Errors are marked next to the line numbers (see Figure 2/1). Click to the error mark opens the Messages tab and highlights corresponding error message.

3.3 Tree window

The tree window displays the structure of the data in the input file as it is seen by Flow123d after automatic conversions. Three conversions are applied: conversion from single Record key to the whole record, conversion a value to the single element list, and transposition, i.e. conversion of records of lists to a list of records. See Flow123d documentation for the details. The tree window does not allow any edit operations.
Click to an element of the tree (see Figure 2/4) highlights corresponding part of the input file.

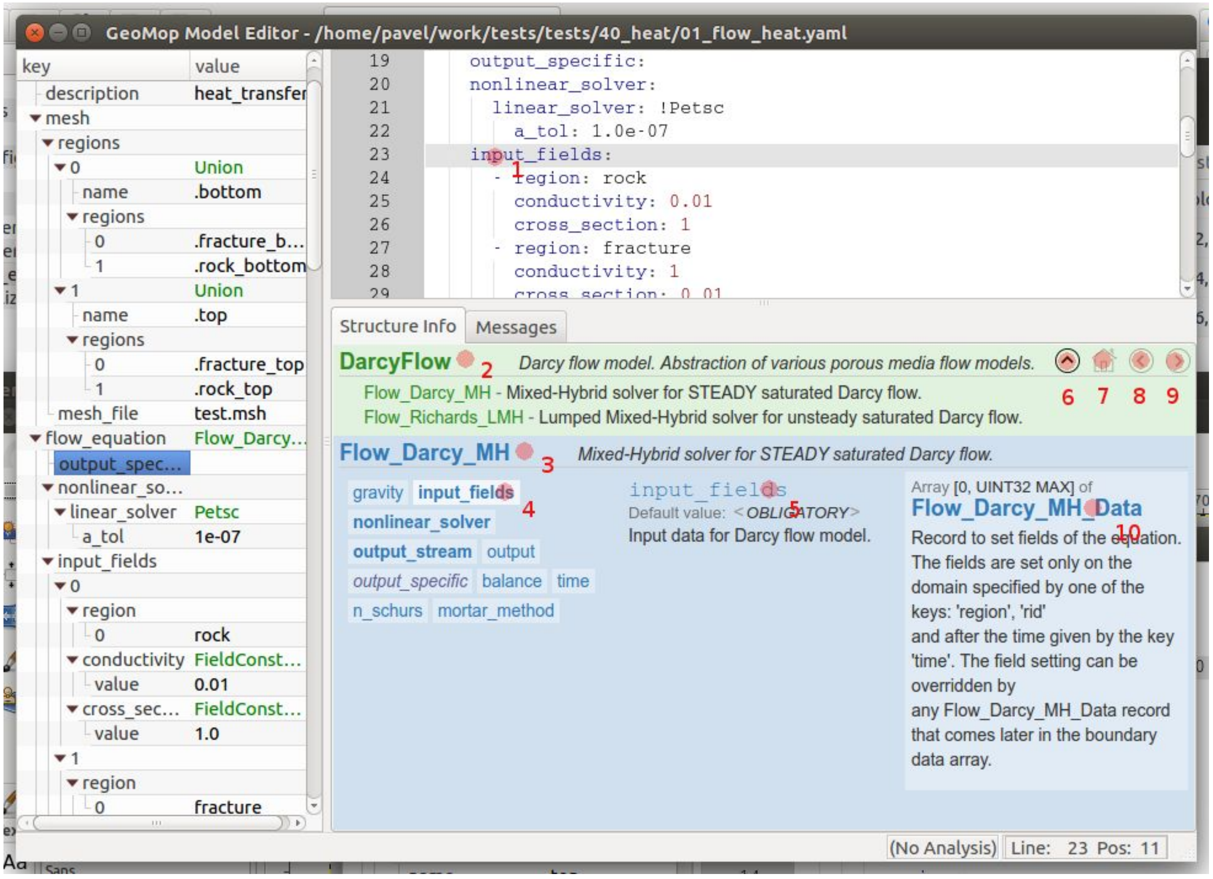
3.4 Messages tab

The messages tab reports errors (see Figure 2/2) in the YAML input when compared to the Flow123d format. Four types of messages are reported:

- **fatal error** - error that prevents further file parsing, e.g. wrong indentation
- **error** - local error in validation, e.g. missing obligatory key, unknown value of a selection
- **warning** - possible error, e.g. unknown key which is ignored, but may be a misspell
- **info** - warning of law importance, e.g. duplicate key

Click on the message highlights corresponding place in the Text editor window.

3.5 Documentation tab (Structure Info)



GeoMop Model Editor - /home/pavel/work/tests/tests/40_heat/01_flow_heat.yaml

```
key                 value
 description        heat_transfer
▾mesh
  ▾regions
    ▾0              Union
       name         .bottom
      ▾regions
         0          .fracture_b...
         1          .rock_bottom
    ▾1              Union
       name         .top
      ▾regions
         0          .fracture_top
         1          .rock_top
   mesh_file        test.msh
▾flow_equation      Flow_Darcy...
   output_spec...
  ▾nonlinear_so...
    ▾linear_solver  Petsc
       a_tol        1e-07
  ▾input_fields
    ▾0
      ▾region
         0          rock
      ▾conductivity FieldConst...
         value      0.01
      ▾cross_sec... FieldConst...
         value      1.0
    ▾1
      ▾region
         0          fracture
```

```
19      output_specific:
20      nonlinear_solver:
21        linear_solver: !Petsc
22          a_tol: 1.0e-07
23      input_fields:
24        - region: rock
25          conductivity: 0.01
26          cross_section: 1
27        - region: fracture
28          conductivity: 1
29          cross_section: 0.01
```

Structure Info | Messages

DarcyFlow *Darcy flow model. Abstraction of various porous media flow models.*

Flow_Darcy_MH - Mixed-Hybrid solver for STEADY saturated Darcy flow.
Flow_Richards_LMH - Lumped Mixed-Hybrid solver for unsteady saturated Darcy flow.

Flow_Darcy_MH *Mixed-Hybrid solver for STEADY saturated Darcy flow.*

gravity **input_fields** **nonlinear_solver** **output_stream** output *output_specific* balance time n_schurs mortar_method

input_fields
Default value: <*OBLIGATORY*>
Input data for Darcy flow model.

Array [0, UINT32 MAX] of
Flow_Darcy_MH_Data
Record to set fields of the equation. The fields are set only on the domain specified by one of the keys: 'region', 'rid' and after the time given by the key 'time'. The field setting can be overridden by any Flow_Darcy_MH_Data record that comes later in the boundary data array.

(No Analysis) Line: 23 Pos: 11

Figure 3: Structure of the Documentation (Structure Info) tab with and its relation to the Text editor window.

The documentation tab (entitled ‘Structure Info’ in Figure 3) contains a context dependent documentation of the Flow123d format for current position in the Text editor window.
For example for the selected line 23 (see Figure 3/1) the documentation tab displays documentation of the record ‘Flow_Darcy_MH’ (see Figure 3/3) containing the active key ‘input_fields’. Since the record ‘Flow_Darcy_MH’ is an instance of the abstract ‘DarcyFlow’ the documentation of the abstract is displayed as well containing links to all existing implementation records (see Figure 3/2).
For actual record, the list of its keys is located on the left (see Figure 3/4) with actual key highlighted by the white background. Different types of keys are distinguished by the font:

- **bold** for obligatory keys

- *italic* for optional keys
- normal for keys with a default value

Text description of the key type is also shown in tooltips while hovering over the key.

The documentation of the actual key is in the center (see Figure 3/5) and documentation of the type of the key value is on the right (see Figure 3/10).
Different types are distinguished by colors: green for abstract, blue for record and violet for scalar types (Integer, Double, String, Selection). Arrays of a type use the same color as the type of their elements.

Navigation over the whole documentation is possible through:

- list of implementations of an Abstract (see Figure 3/2)
- keys in the left part (see Figure 3/4)
- record key types on the right (see Figure 3/10)
- up icon (see Figure 3/6) - to display parent record of the actual record
- home icon (see Figure 3/7) - to navigate to the documentation of the actual element in the Text editor window
- previous icon (see Figure 3/8) - go back in the history
- next icon (see Figure 3/9) - go forward in the history

3.6 Format conversion

To convert an old-style format CON file, first open the file by navigating to `File -> Import File …`
Next go to `Settings -> Transformation …` and select your desired version.

4 Analysis

The analysis module provides Python classes to compose and execute complex modeling calculations from simpler tools called *actions.* In order to pass the data between actions we use *data type* classes to describe the structure of the data. Conversion of the data structures between actions is performed by *convertors.* The complete pipeline of the actions interconnected by convertors forms the *analysis.* The analysis can be executed as a batch job. The validation of the pipeline is performed before the job is queued for the bath processing. When executed via a *multi job* (see Job Panel documentation), the pipeline can execute actions in parallel as a child batch jobs. Currently the analysis can be composed only by a Python script, however the design assumes creation of a GUI for this task.

4.1 Data types

Data type classes in the analysis are used for a type checking and data transfer between the actions. The basic data types are: Bool, Int, Float, String, Enum. The composite data types include:

Struct is like the Python's dictionary where the keys are string-only. A single instance of Struct have fixed set of keys and types of corresponding values.

```
      # type definition
      a = Struct(x=Int(), y=String())
```

Tuple is an array of a fixed length with values of the prescribed (but possibly different) types. It is like the Struct type, where natural numbers (from zero) are used as keys.

```
      # type definition
      a = Tuple(Int(), String())
```

Ensamble is a set of values with given common type and without any prescribed order. This implies that individual elements of an ensamble can be processed in parallel.

```
      # type definition
      a = Ensemble(Int())
```

Sequence is like the Ensamble but with defined order of elements.

```
      # type definition
      a = Sequence(Int())
```

4.2 Converters

It serves only to merge, select and reorganize data, does not make any calculations. It is primarily used in action connectors, but one converter can also be used inside another converter, especially for Ensemble and Sequence operations.

Reformatting. Basic tool of conversion is creation of the new structure through extraction from the input data structures.

Example:

```
      Converter(Struct(a=Input(0).c, b=Input(1)[2]))
```

Make a structure from two inputs setting key 'a' to the key 'c' of the struct on input 0 and setting key 'b' to the second element of the tuple on the input 1.

Each. Is used to apply the converter to each item of Ensemble. The converter, which is the 'each' parameter, must have just one Input(0) input.

Example:

```
# from Ensamble of Structs, make Ensamble of Tuples
Converter(Input(0).each(Adapter(Tuple(Input(0).a, Input(0).b))))
```

Select. The ‘select’ operation extract elements of Ensamble or Sequence satisfying a given *Predicate*.

Example:

```
# Extract elements smaller than 3.
Converter(Input(0).select(Predicate(Input(0) < 3)))
```

Sort. Sort Ensemble or Sequence according to a given value and producing a Sequence. Sorting value is given by a converter that returns a scalar (or, more generally, a comparable type) which is called *Selector.*

Example:

```
# Sort an Ensamble of structs by the value of the key ‘b’
Converter(Input(0).sort(Selector(Input(0).b)))
```

4.3 Actions

One action is a specific computational or graphic operation. The specific details of the performed operations can be affected by the action configuration. Each action has a single Input Slot and single Output Slot. The input and output data type is fixed by the action type and its configuration. The configuration of an action can affect the data type of the input and output slot. The configuration can not be result of another action, its specification is part of the workflow definition.

4.3.1 Generator actions

VariableGenerator The data passed in the configuration, provides on the output.
Example:

```
var = Struct(x1=Float(1.0), x2=Float(2.0))
gen = VariableGenerator(Variable=var)
```

RangeGenerator Generator for generation parallel lists.
Example:

```
items = [
    {'name':'a', 'value':1, 'step':0.1, 'n_plus':1,
     'n_minus':1,'exponential':False},
    {'name':'b', 'value':10, 'step':1, 'n_plus':2,
     'n_minus':3,'exponential':False}
]
```

```
gen = RangeGenerator(Items=items, AllCases=False)
```

4.3.2 Parametrized actions

The common denominator of these actions is that they call non-trivial external programs and assume the definition of the data of these programs in specific contexts, the input of actions can only affect the declared parameters in the input data (files) of each program.

Flow123dAction
Action calls the Flow123d simulator. Configuration includes a YAML file and a computing mesh.
Example:

```
gen = VariableGenerator(Variable=Struct(par=Float(1.2)))
flow = Flow123dAction(Inputs=[gen], YAMLFile="test.yaml")
```

FunctionAction
This action compute some mathematical expressions on input data and return results on output.
Example:

```
var = Struct(x1=Float(1.0), x2=Float(2.0), x3=Float(math.pi/2))
gen = VariableGenerator(Variable=var)
fun = FunctionAction(
    Inputs=[gen],
    Params=["x1", "x2", "x3"],
    Expressions=["y1 = 2 * x1 + 3 * x2", "y2 = sin(x3)"])
```

4.3.3 Wrapper actions

ForEach
The action configuration includes a workflow with IN input type and OUT output type. The workflow is executed on all input elements of type Ensemble(IN), resulting in output of type Ensemble(OUT).
Example:

```
var = Ensemble(Float(), Float(1.0), Float(2.0), Float(3.0))
gen = VariableGenerator(Variable=var)
w = Workflow()
f = FunctionAction(
    Inputs=[w.input()],
    Params=["x"],
    Expressions=["y = 2 * x"]
)
w.set_config(
    OutputAction=f,
    InputAction=f
)
for_each = ForEach(
    Inputs=[gen],
    WrappedAction=w
)
```

Calibration
Calibrating a generic model for given data, ie identifying model parameters so that the result of the model was in some sense the closest measured data.
Example:

```
gen = VariableGenerator(
    Variable=(
        Struct(
            observations=Struct(
                y1=Float(1.0),
                y2=Float(5.0)
            )
        )
    )
)
w = Workflow()
f = FunctionAction(
    Inputs=[
        w.input()
    ],
    Params=["x1", "x2"],
    Expressions=["y1 = 2 * x1 + 2", "y2 = 2 * x2 + 3"]
)
w.set_config(
    OutputAction=f,
    InputAction=f
)
cal = Calibration(
    Inputs=[
        gen
    ],
    WrappedAction=w,
    Parameters=[
        CalibrationParameter(
            name="x1",
            group="test",
            bounds=(-1e+10, 1e+10),
            init_value=1.0
        ),
        CalibrationParameter(
            name="x2",
            group="test",
            bounds=(-1e+10, 1e+10),
            init_value=1.0
        )
    ],
    Observations=[
        CalibrationObservation(
            name="y1",
            group="tunnel",
            weight=1.0
        ),
        CalibrationObservation(
            name="y2",
            group="tunnel",
            weight=1.0
        )
    ],
    AlgorithmParameters=[
        CalibrationAlgorithmParameter(
            group="test",
            diff_inc_rel=0.01,
            diff_inc_abs=0.0
        )
    ],
    TerminationCriteria=CalibrationTerminationCriteria(
        n_max_steps=100
    ),
    MinimizationMethod="SLSQP",
    BoundsType=CalibrationBoundsType.hard
)
```

4.3.4 Special actions

Workflow
Encapsulates the entire workflow into a new user-defined action.
Example:

```
      w = Workflow()
      f = FunctionAction(
          Inputs=[w.input()],
          Params=["x1", "x2"],
          Expressions=["y1 = 2 * x1 + 2", "y2 = 2 * x2 + 3"]
```

```
    )
    w.set_config(
        OutputAction=f,
        InputAction=f
    )
```

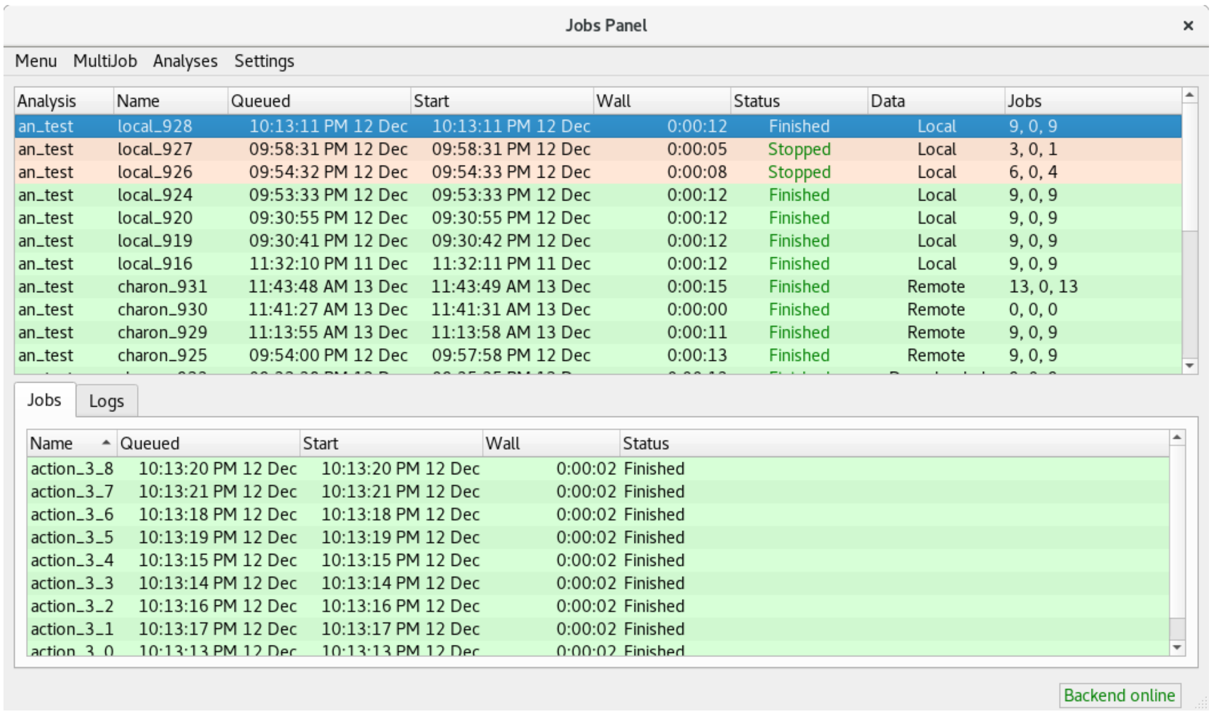
Pipeline
Pipeline groups all of other actions. It has not any input action and least one output action. Output action define action contained in pipeline. Output action would be actions, theirs results will be downloaded.
Example:

```
    var = Struct(x1=Float(1.0), x2=Float(2.0))
    gen = VariableGenerator(Variable=var)
    print_action = PrintDTTAction(
        Inputs=[gen],
        OutputFile="output.txt")
    pipeline = Pipeline(
        ResultActions=[print_action]
    )
```

5 Job Panel

The Job Panel application simplifies execution of calculations on remote computing resources. It allows to setup and organise SSH connection to several clusters, configure PBS settings, execute the jobs, monitor the status of executed jobs and it takes care of data upload and download. Directly executed jobs are called *multijobs* since they execute an analysis, which in turn can call other jobs to perform individual actions.

Layout of the user interface is depicted on Figure 5.1. List of all queued multijobs are at the top part, the bottom part displays informations about current multijob in two tabs: Jobs - queued child jobs, Logs - log of messages from processing.



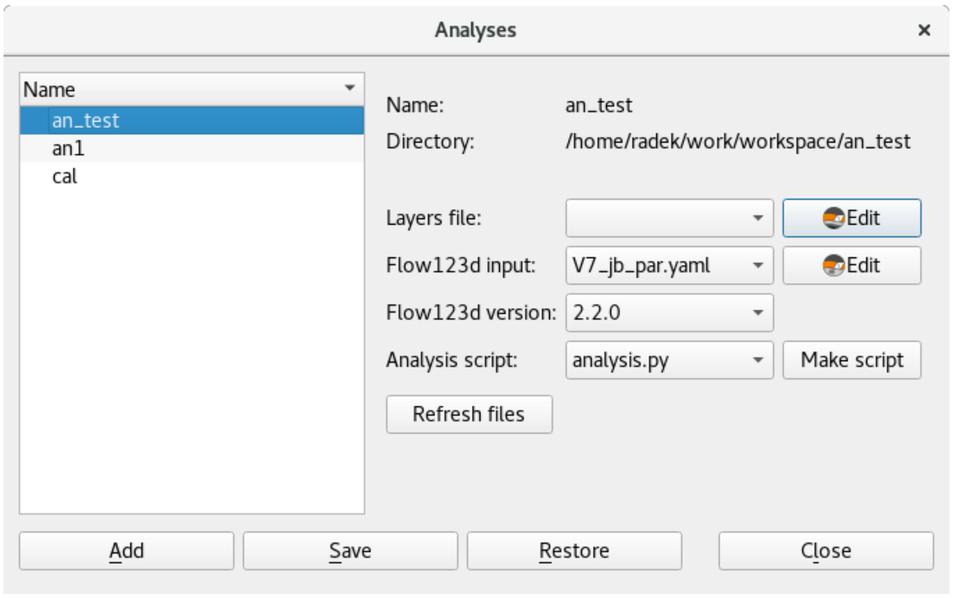
Analysis	Name	Queued	Start	Wall	Status	Data	Jobs
an_test	local_928	10:13:11 PM 12 Dec	10:13:11 PM 12 Dec	0:00:12	Finished	Local	9, 0, 9
an_test	local_927	09:58:31 PM 12 Dec	09:58:31 PM 12 Dec	0:00:05	Stopped	Local	3, 0, 1
an_test	local_926	09:54:32 PM 12 Dec	09:54:33 PM 12 Dec	0:00:08	Stopped	Local	6, 0, 4
an_test	local_924	09:53:33 PM 12 Dec	09:53:33 PM 12 Dec	0:00:12	Finished	Local	9, 0, 9
an_test	local_920	09:30:55 PM 12 Dec	09:30:55 PM 12 Dec	0:00:12	Finished	Local	9, 0, 9
an_test	local_919	09:30:41 PM 12 Dec	09:30:42 PM 12 Dec	0:00:12	Finished	Local	9, 0, 9
an_test	local_916	11:32:10 PM 11 Dec	11:32:11 PM 11 Dec	0:00:12	Finished	Local	9, 0, 9
an_test	charon_931	11:43:48 AM 13 Dec	11:43:49 AM 13 Dec	0:00:15	Finished	Remote	13, 0, 13
an_test	charon_930	11:41:27 AM 13 Dec	11:41:31 AM 13 Dec	0:00:00	Finished	Remote	0, 0, 0
an_test	charon_929	11:13:55 AM 13 Dec	11:13:58 AM 13 Dec	0:00:11	Finished	Remote	9, 0, 9
an_test	charon_925	09:54:00 PM 12 Dec	09:57:58 PM 12 Dec	0:00:13	Finished	Remote	9, 0, 9

Name	Queued	Start	Wall	Status
action_3_8	10:13:20 PM 12 Dec	10:13:20 PM 12 Dec	0:00:02	Finished
action_3_7	10:13:21 PM 12 Dec	10:13:21 PM 12 Dec	0:00:02	Finished
action_3_6	10:13:18 PM 12 Dec	10:13:18 PM 12 Dec	0:00:02	Finished
action_3_5	10:13:19 PM 12 Dec	10:13:19 PM 12 Dec	0:00:02	Finished
action_3_4	10:13:15 PM 12 Dec	10:13:15 PM 12 Dec	0:00:02	Finished
action_3_3	10:13:14 PM 12 Dec	10:13:14 PM 12 Dec	0:00:02	Finished
action_3_2	10:13:16 PM 12 Dec	10:13:16 PM 12 Dec	0:00:02	Finished
action_3_1	10:13:17 PM 12 Dec	10:13:17 PM 12 Dec	0:00:02	Finished
action_3_0	10:13:13 PM 12 Dec	10:13:13 PM 12 Dec	0:00:02	Finished

Figure 5.1: Layout of the Job Panel application

5.1 Analysis

Job Panel is able to run calculations in form of analysis. We can create and edit analysis using menu analysis. When analysis is created, directory with name of analysis is created in workspace. Dialog Analysis is depicted on Figure 5.2. Analysis computation is described in analysis script file. Simplest way to create script file is copy Flow123d input file to analysis directory, chose this file in combobox and click on ‘Make script’ button. It is possible to create own Analysis script file (see chapter 4).

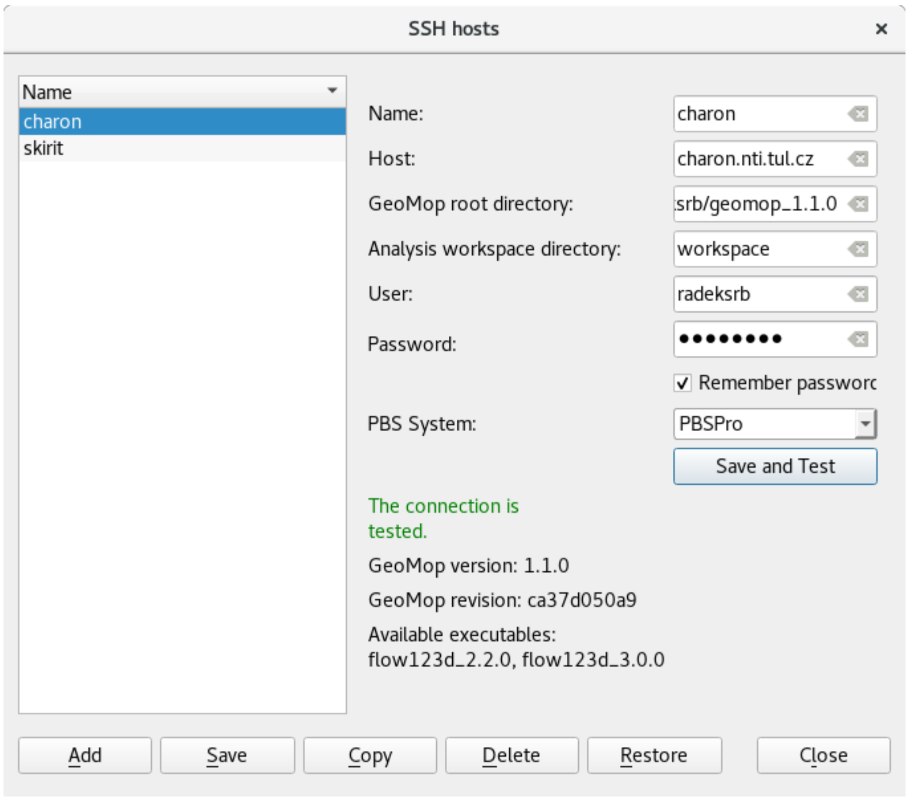


Name: an_test
Directory: /home/radek/work/workspace/an_test
Layers file:
Flow123d input: V7_jb_par.yaml
Flow123d version: 2.2.0
Analysis script: analysis.py

Figure 5.2: Analysis dialog

5.2 SSH Hosts

In order to compute on remote servers, it is need setup ssh connections to them. Dialog SSH hosts (Figure 5.3) is accessible from menu Settings. A single SSH connection consists of the address of IP of the remote host, credentials, type of PBS system and remote directories. GeoMop root directory is absolute path on remote server where GeoMop is installed. Analysis workspace directory is absolute or relative path on remote server where working data will be stored. Actual SSH connection must be tested using the ‘Save and Test’ button before use.



Name: charon
Host: charon.nti.tul.cz
GeoMop root directory: srb/geomop_1.1.0
Analysis workspace directory: workspace
User: radeksrb
Password: ●●●●●●●●
✓ Remember passworc
PBS System: PBSPro

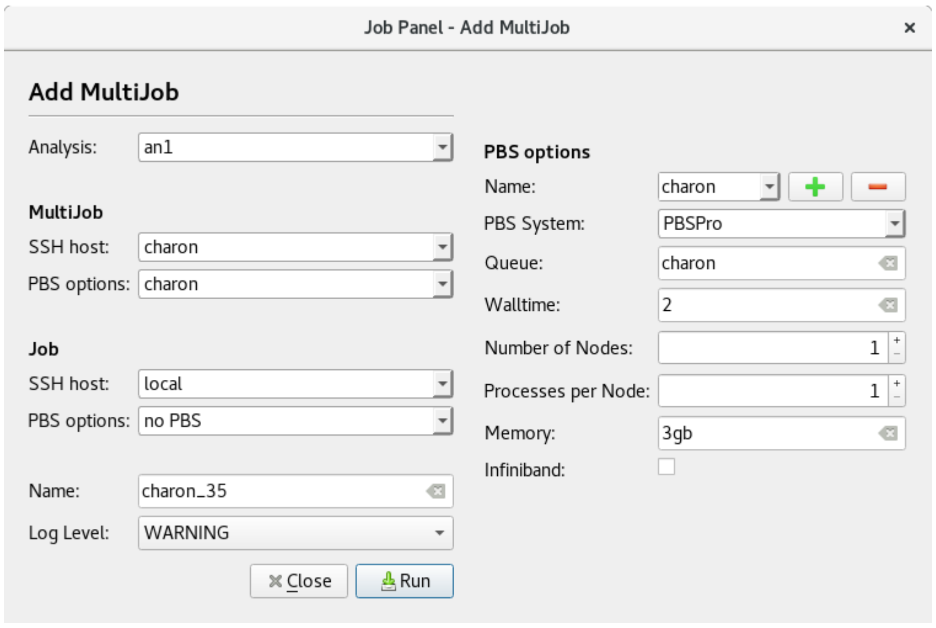
The connection is tested.
GeoMop version: 1.1.0
GeoMop revision: ca37d050a9
Available executables:
flow123d_2.2.0, flow123d_3.0.0

Figure 5.3: SSH hosts dialog

5.3 MultiJob

MultiJob represents single evaluation of particular analysis. MultiJob operations are accessible from menu MultiJob.

Add - is used to create new MultiJob. Figure 5.4 depict Add MultiJob dialog. On left side are MultiJob parameters and on right side is possible edit PBS options. During creation MultiJob is necessary chose analysis, that will be computed within the MultiJob. MultiJob SSH host specify where MultiJob will be running. Local means that MultiJob will be running on local computer. If target host has PBS system, it is possible to choose PBS options. Selected PBS option may be edited on right side of dialog. For Job (single task performed by MultiJob) can be selected SSH host and PBS options too. Currently is for Job supported only local host, it means that Job will be running on same machine as MultiJob. For each MultiJob is possible to choose its name and level of logging.


Job Panel - Add MultiJob

Add MultiJob

Analysis:	an1

MultiJob

SSH host:	charon
PBS options:	charon

Job

SSH host:	local
PBS options:	no PBS
Name:	charon_35
Log Level:	WARNING

PBS options

Name:	charon
PBS System:	PBSPro
Queue:	charon
Walltime:	2
Number of Nodes:	1
Processes per Node:	1
Memory:	3gb
Infiniband:	☐

Figure 5.4: Add MultiJob dialog

Reuse - same as Add except that new MultiJob is based on selected MultiJob. If it is possible some actions results are reused.
Delete Remote - deletes MultiJob’s data on remote server.
Delete MultiJob - deletes MultiJob’s data on remote server and local computer.
Stop - stops running MultiJob.
Download Whole MultiJob - downloads all MultiJob’s data from remote server to local computer.