OpenFLUID

Software Environment
for Modelling Fluxes in Landscapes

Quick reference manual

OpenFLUID-Engine 1.5.0

LISAH
Laboratoire d'étude des Interactions
Sol - Agrosystème - Hydrosystème

Fabre, J.C.,
and the modelling
group at LISAH

Contents

Foreword

This quick reference manual will help you to prepare and run simulations using OpenFLUID-engine. It will not explain the concepts behind the software, nor the scientific approaches in landscape fluxes modelling and simulation.

Typographic conventions

The "to note" informations are emphasized like this:

Note: "to note" information ...

The source code examples are emphasized like this:

example of source code

```
Source code, with grey background and fixed size font
```

The "warning" informations are emphasized like this:

Warning: "warning" informations ...

Chapter 1

Usage information

The OpenFLUID-Engine application is available on Linux, Windows and MacOSX platforms. We encourage you to use OpenFLUID-Engine program on Linux platform as it is the development and usually used platform.

1.1 Installation

On linux platforms, the OpenFLUID-Engine software is available as distribution packages (deb, rpm) or archive files (tar.gz, tar.bz2). The recommanded way to install it is to use packages for your Linux distribution. If you want to use archive files, you have to unarchive the software according to the directory tree.
Once installed, the `openfluid-engine` command should be available. You can check it by running the command `openfluid-engine --help` or `openfluid-engine --version` in your favorite terminal. You are now ready to run your first simulation.

1.2 Input dataset

Before running the simulation, the input dataset must be built. An OpenFLUID-Engine input dataset includes different informations, defined in one or many files:

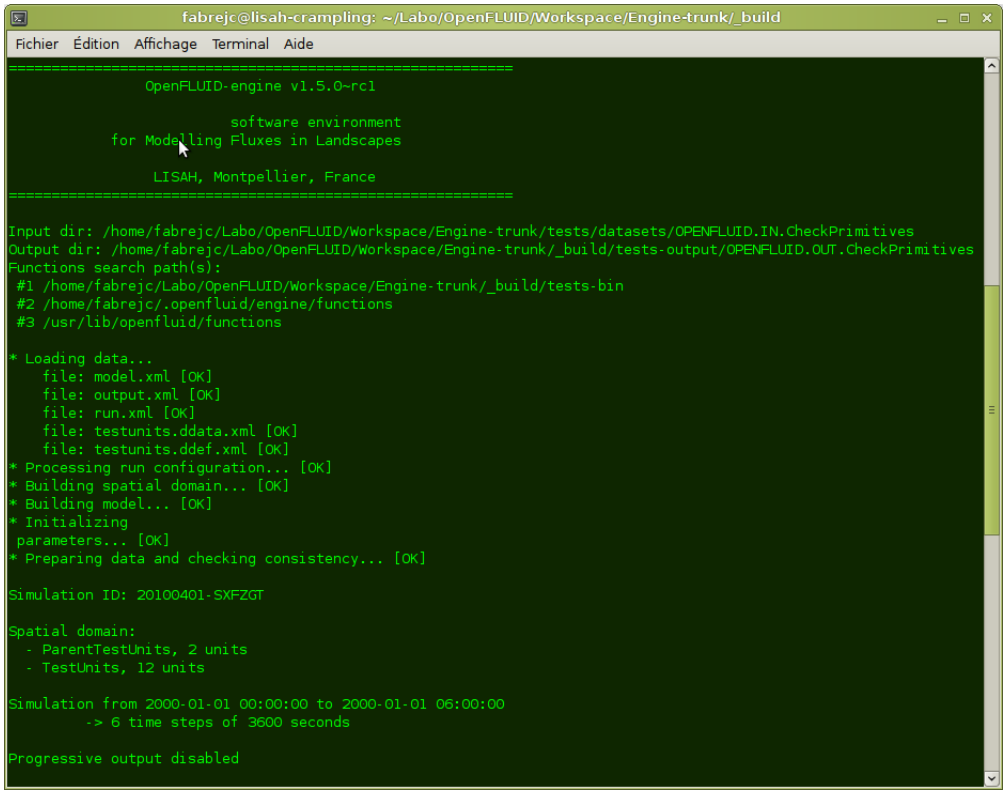
- the spatial domain definition
- the flux model definition
- the spatial domain input data
- the discrete events
- the run configuration
- the outputs configuration

All files must be placed into any directory that can be reached by the engine. The default searched directory is a directory named `.openfluid/engine/OPENFLUID.IN` and located into the user home directory (the user home directory may vary, depending on the used operating system). This directory is not automatically created, it should be created by hand. If you prefer to place your dataset in another directory, you can specify it using command line options passed to the engine (`-i` or `--input-dir`).

In order to build these files, we encouraged you to use a good text editor or, better, an XML editor. You can also use custom scripts or macros in specialized sotware, such as spreadsheets or Geographic Information Systems (GIS), to generate automatically the input dataset.

1.3 Run the simulation

To run the simulation, if the dataset is located in the default searched directory, simply run the command `openfluid-engine` in your favorite terminal. To specify a different input dataset directory, use the `-i` or `--input-dir` command line option.



1.4 Explore the results

The results are stored in files, gathered by spatial unit. In each files, the values for variables are stored as columns, each row corresponfing to a data exchange time step (represented as a date and time). The format of the files depends on the configuration of outputs, set through the `run.xml` file. The default output directory is a directory named `.openfluid/engine/OPENFLUID.OUT` and located into the user home directory (the user home directory may vary, depending on the used operating system). If you prefer to store your outputs into another directory, you can specify it using command line options passed to the engine (`-o` or `--output-dir`).

In order to process the results of your simulations, we encourage you to use software environments such as R, Scilab or Octave, spreadsheets such as OpenOffice Calc, GIS such as GRASS or QGIS.

1.5 Buddies

Buddies are small tools that help scientific developers in order to complete the modelling and/or development works. They are usable from the command line, using the `--buddyhelp`, `--buddy` and `--buddyopts` options. Four buddies are available:

- `func2doc`
- `newfunc`
- `newdata`
- `convert`

Options are given to buddies through a comma-separated list of `key=value` arguments, using the `--buddyopts` command line option.

General usage is:
```
openfluid-engine -buddy buddyname -buddyopts abuddyopt=avalue,anotherbuddyopt=anothervalue
```

1.5.1 func2doc

The `func2doc` buddy extracts scientific information from the source code of simulation functions. It uses the function signature and LaTeX-formatted text placed between the `<func2doc>` and `</func2doc>` tags (usually into C++ comments). From these sources of information, it builds a LaTeX document which could be compiled into a PDF document and/or HTML pages.
The `func2doc` buddy can also use information from an optional sub-directory named `doc`, located in the same directory as the input source file. The information in the `doc` subdirectory should be linked to the information from the source code using LaTeX `\input` command. The `func2doc` buddy is available on UNIX only systems (Linux, MacOSX).

Required options:

`inputcpp`	path for cpp file to parse
`outputdir`	path for generated files

Other options:

`html`	set to 1 in order to generate documentation as HTML files
`pdf`	set to 1 in order to generate documentation as PDF file
`tplfile`	path to template file

Usage example:
```
openfluid-engine -buddy func2doc -buddyopts inputcpp=/path/to/cppfile.cpp, outputdir=/path/to/out
```

1.5.2 newfunc

The `newfunc` buddy generate a skeleton source code of a simulation function, using given options.

Required options:

`cppclass`	C++ class name of the function
`funcid`	ID of the function

Other options:

`authoremail`	email(s) of the author(s) of the function
`authorname`	name(s) of the author(s) of the function
`outputdir`	path for generated files

Usage example:
```
openfluid-engine -buddy newfunc -buddyopts funcid=domain.subdomain.process.method,
outputdir=/path/to/outputdir
```

1.5.3 newdata

The `newdata` buddy generate a skeleton dataset.

Required options:

`outputdir`	Output directory for generated dataset

Usage example:
```
openfluid-engine -buddy newdata -buddyopts outputdir=/path/to/outputdir
```

1.5.4 convert

The `convert` buddy converts a dataset from a specific version format to another one. Currently, conversion is possible from 1.3.x format to 1.4.x format, and from 1.4.x format to 1.5.x format

Required options:

`convmode`	Conversion mode. Available modes are: `13_14`, `14_15`
`inputdir`	Input directory for dataset to convert
`outputdir`	Output directory for converted dataset

Usage example:
```
openfluid-engine -buddy convert -buddyopts convmode=13_14, inputdir=/path/to/inputdir,outputdir=/
```

Chapter 2

FluidX file(s) format

This part of the manual describes the FluidX file(s) format. Refer to the "Usage" part of this manual to run the simulation.

2.1 Overview

The FluidX file format is an XML based format for OpenFLUID input file(s). The OpenFLUID input information can be provided by a one or many files using the FluidX format.
Whatever the input information is put into one or many files, the following sections must be defined in the input file set:

- The *model* section defined by the `<model>` tag
- The *spatial domain* section defined by the `<domain>` tag
- The *run configuration* section defined by the `<run>` tag
- The *outputs configuration* section defined by the `<output>` tag

The order of the sections is not significant. All of these sections must be inclosed into an *openfluid* section defined by the `<openfluid>` tag.

summary view of the XML tree for FluidX files

```
<openfluid>

  <model>
    <!-- here is the model definition -->
  </model>

  <domain>
    <!-- here is the spatial domain definition, associated data and events -->
  </domain>

  <output>
    <!-- here is the output configuration -->
  </output>

  <run>
    <!-- here is the run configuration -->
  </run>

</openfluid>
```

2.2 Sections

2.2.1 Model

The flux model is defined by an ordered set of data generators and simulations functions that will be plugged to the OpenFLUID-Engine kernel. It defines the model for the simulation. It can also include a global parameters section which applies to all simulation functions and generators. The global parameters may be overridden by local parameters of simulation functions or generators.
The flux model must be defined in a section delimited by the `<model>` tag, and must be structured following these rules:

- Inside the `<model>` tag, there must be a set of `<function>`, `<generator>` and `<gparams>` tags
- Each `<function>` tag must bring a `fileID` attribute giving the identifier of the simulation function; the value of the `fileID` attribute must match the file name (without extension) of a reachable and pluggable simulation function.
- Each `<function>` tag may include zero to many `<param>` tags giving parameters to the function. Each `<param>` tag must bring a `name` attribute giving the name of the parameter and a `value` attribute giving the value of the parameter. These parameters can be scalar or vector of integer values, floating point values, string values. In case of vector, the values of the vector are separated by a ; (semicolon).
- Each `<generator>` tag must bring a `varname` attribute giving the name of the produced variable, a `unitclass` attribute giving the unit class of the produced variable, a `method` attribute giving the method used to produce the variable (`fixed` for constant value, `random` for random value in a range, `interp` for an interpolated value from given data series). An optional `<varsize>` attribute can be set in order to produce a vector variable instead of a scalar variable.
- Each `<generator>` tag may include zero to many `<param>` tags giving parameters to the generator. Each `<param>` tag must bring a `name` attribute giving the name of the parameter and a `value` attribute giving the value of the parameter.
- A generator using the `fixed` method must provide a param named `fixedvalue` for the value to produce.
- A generator using the `random` method must provide a param named `min` and a param named `max` delimiting the random range for the value to produce.
- A generator using the `interp` method must provide a param named `sources` giving the data sources filename and a param named `distribution` giving the distribution filename for the value to produce (see appendix).
- Each `<gparams>` tag may include zero to many `<param>` tags giving the global parameters. Each `<param>` tag must bring a `name` attribute giving the name of the parameter and a `value` attribute giving the value of the parameter.

example

```
<?xml version="1.0" standalone="yes"?>
<openfluid>
  <model>

    <gparams>
      <param name="gparam1" value="100" />
      <param name="gparam2" value="0.1" />
```

```
    </gparams>

    <function fileID="example.functionA" />

    <generator varname="example.generator.fixed" unitclass="EU1" method="fixed" varsize="11">
      <param name="fixedvalue" value="20" />
    </generator>

    <generator varname="example.generator.random" unitclass="EU2" method="random">
      <param name="min" value="20.53" />
      <param name="max" value="50" />
    </generator>

    <function fileID="example.functionB">
      <param name="param1" value="strvalue" />
      <param name="param2" value="1.1" />
      <param name="gparam1" value="50" />
    </function>

  </model>
</openfluid>
```

Warning: There must be only one model definition in the input dataset.

Warning: The order of the simulation functions and data generators in the `<model>` section is very important : the same order will be used for execution on the same time step

2.2.2 Spatial domain

Definition and relationships

The spatial domain is defined by a set of spatial units that are connected each others. These spatial units are defined by a numerical identifier (ID) and a class. They also include information about the processing order of the unit in the class. Each unit can be connected to zero or many other units from the same or a different unit class.
The spatial domain definition must be defined in a section delimited by the `<definition>` tag, which is a sub-section of the `domain` tag, and must be structured following these rules:

- Inside the `<definition>` tag, there must be a set of `<unit>` tags
- Each `<unit>` tag must bring an `ID` attribute giving the identifier of the unit, a `class` attribute giving the class of the unit, a `pcsorder` attribute giving the process order in the class of the unit
- Each `<unit>` tag may include zero or many `<to>` tags giving the outgoing connections to other units. Each `<to>` tag must bring an `ID` attribute giving the identifier of the connected unit and a `class` attribute giving the class of the connected unit
- Each `<unit>` tag may include zero or many `<childof>` tags giving the parent units. Each `<childof>` tag must bring an `ID` attribute giving the identifier of the parent unit and a `class` attribute giving the class of the parent unit

example

```
<?xml version="1.0" standalone="yes"?>
<openfluid>
  <domain>
```

```
    <definition>

      <unit class="PU" ID="1" pcsorder="1" />

      <unit class="EU1" ID="3" pcsorder="1">
        <to class="EU1" ID="11" />
        <childof class="PU" ID="1" />
      </unit>

      <unit class="EU1" ID="11" pcsorder="3">
        <to class="EU2" ID="2" />
      </unit>

      <unit class="EU2" ID="2" pcsorder="1" />

    </definition>
  </domain>
</openfluid>
```

Input data

The spatial domain input data are static data brought by units, usually properties and initial conditions for each unit.
The spatial domain input data must be defined in a section delimited by the `<inputdata>` tag, which is a sub-section of the `domain` tag, and must be structured following these rules:

- The `<inputdata>` tag must bring a `unitclass` attribute giving the unit class to which input data must be attached, and a `colorder` attribute giving the order of the contained column-formatted data
- Inside the `<inputdata>` tag, there must be the input data as row-column text. As a rule, the first column is the ID of the unit in the class given through the the `unitclass` attribute of `<inputdata>` tag, the following columns are values following the column order given through the `colorder` attribute of the `<inputdata>` tag. Values for the data can be real, integer or string.

example

```
<?xml version="1.0" standalone="yes"?>
<openfluid>
  <domain>

    <inputdata unitclass="EU1" colorder="indataA">
      3 1.1
      11 7.5
    </inputdata>

    <inputdata unitclass="EU2" colorder="indataB1;indataB3">
      2 18 STRVALX
    </inputdata>

  </domain>
</openfluid>
```

Note: Old inputdata format, with `<columns>` and `<data>` tags are still useable. However, you are encouraged to use the new FluidX file format.

Discrete events

The discrete events are events occuring on units, and that can be processed by simulation functions. The spatial events must be defined in a section delimited by the `<calendar>` tag, which is a subsection of the `domain` tag, and must be structured following these rules:

- Inside the `<calendar>` tag, there must be a set of `<event>` tags
- Each `<event>` tag must bring a `unitID` and a `unitclass` attribute giving the unit on which occurs the event, a `date` attribute giving the date and time of the event. The date format must be "YYYY-MM-DD hh:mm:ss". The `<event>` tag may bring a `name` attribute and a a `category` attribute, but they are actually ignored.
- Each `<event>` tag may include zero to many `<info>` tags.
- Each `<info>` tag give information about the event and must bring a `key` attribute giving the name (the "key") of the info, and a `value` attribute giving the value for this key.

example

```
<?xml version="1.0" standalone="yes"?>
<openfluid>
  <domain>
    <calendar>

      <event unitclass="EU1" unitID="11" date="1999-12-31 23:59:59">
        <info key="when" value="before" />
        <info key="where" value="1" />
        <info key="var1" value="1.13" />
        <info key="var2" value="EADGBE" />
      </event>
      <event unitclass="EU2" unitID="3" date="2000-02-05 12:37:51">
        <info key="var3" value="152.27" />
        <info key="var4" value="XYZ" />
      </event>
      <event unitclass="EU1" unitID="11" date="2000-02-25 12:00:00">
        <info key="var1" value="1.15" />
        <info key="var2" value="EADG" />
      </event>

    </calendar>
  </domain>
</openfluid>
```

2.2.3 Run configuration

The configuration of the simulation gives the simulation period, the data exchange time step, and the optionnal progressive output parameters.
The run configuration must be defined in a section delimited by the `<run>` tag, and must be structured following these rules:

- Inside the `<run>` tag, there must be a `<deltat>` tag giving the data exchange time step (in seconds)
- Inside the `<run>` tag, there must be a `<period>` tag giving the simulation period.
- The `<period>` tag must bring a `begin` and an `end` attributes, giving the dates of the beginning and the end of the simulation period. The dates formats for these attributes must be `YYYY-MM-DD hh:mm:ss`

- Inside the `<run>` tag, there may be a `<progressout>` tag giving the configuration for the progressive output mode. This `<progressout>` tag must bring a `packet` attribute giving the size (in number of time steps) of the saved packets, and a `keep` attribute giving the number of time steps kept in memory.

example

```
<?xml version="1.0" standalone="yes"?>
<openfluid>
  <run>

    <deltat>3600</deltat>
    <period begin="2000-01-01 00:00:00" end="2000-03-27 01:12:37" />

    <progressout packet="10" keep="2" />

  </run>
</openfluid>
```

2.2.4 Outputs configuration

The configuration of the simulation outputs gives the description of the saved results.
The outputs configuration must be defined in a section delimited by the `<output>` tag, and must be structured following these rules:

- Inside the `<output>` tag, there must be one to many `<files>` tags, defining files formats for saved data.
- These `<files>` tags must bring a `colsep` attribute defining the separator strings between columns, a `dtformat` attribute defining the date time format used (it could be `6cols`, `iso` or user defined using strftime() format whis is described in the appendix part of this document), a `commentchar` attribute defining the string prefixing lines of comments in output files.
- Inside the `<files>` tags, there must be one to many `<set>` tags. Each `<set>` tag will lead to a set of files.
- Each `<set>` tag must bring a `name` attribute defining the name of the set (this will be used as a suffix for generated output files), a `unitsclass` attribute and a `unitsIDs` attribute defining the processed units, a `vars` attribute defining the processed variables. It may also bring an a `precision` attribute giving the number of significant digits for the values in the outputs files. The IDs for the `unitsIDs` attribute are semicolon-separated, the wildcard character ('*') can be used to include all units IDs for the given class. The variables names for the `vars` attribute are semicolon-separated, the wildcard character ('*') can be used to include all variables for the given class. The value for the `precision` attribute must be ≥ 0. If not provided, the default value for the precision is 5.

example

```
<?xml version="1.0" standalone="yes"?>
<openfluid>
  <output>

    <files colsep=" " dtformat="%Y %m %d %H %M %S" commentchar="%">
      <set name="testRS" unitsclass="RS" unitsIDs="51;232" vars="*" />
      <set name="full" unitsclass="SU" unitsIDs="*" vars="*" precision="7"/>
    </files>
```

```
  </output>
</openfluid>
```

Chapter 3

Appendix

3.1 Command line options

`-a, --auto-output-dir`	generate automatic results output directory
`-b, --buddy <arg>`	run specified OpenFLUID buddy
`--buddyhelp <arg>`	display help message for specified OpenFLUID buddy
`--buddyopts <arg>`	set options for specified OpenFLUID buddy
`-c, --clean-output-dir`	clean results output directory by removing existing files
`-f, --functions-list`	list available functions (do not run the simulation)
`-h, --help`	display help message
`-i, --input-dir <arg>`	set dataset input directory
`-o, --output-dir <arg>`	set results output directory
`-p, --functions-paths <arg>`	add extra functions research paths
`-q, --quiet`	quiet display during simulation run
`-r, --functions-report`	print a report of available functions, with details (do not run the simulation)
`-s, --no-simreport`	do not generate simulation report
`--show-paths`	print the used paths (do not run the simulation)
`-u, --matching-functions-report <arg>`	print a report of functions matching the given wildcard-based pattern (do not run the simulation)
`-v, --verbose`	verbose display during simulation
`--version`	get version (do not run the simulation)
`-x, --xml-functions-report`	print a report of available functions in xml format, with details (do not run the simulation)
`-z, --no-result`	do not write results files

3.2 Environment variables

The `openfluid-engine` program takes into account the following environment variables (if they are set):

- `OPENFLUID_FUNCS_PATH`: extra search paths for OpenFLUID-Engine simulation functions. The path are separated by colon on UNIX systems, and by semicolon on Windows systems.
- `OPENFLUID_INSTALL_PREFIX`: overrides automatic detection of install path, useful on Windows systems.

3.3 Date-time formats used in outputs configuration

The output.xml file can use the ANSI strftime() standards formats for date time, through a format string. The format string consists of zero or more conversion specifications and ordinary characters. A conversion specification consists of a % character and a terminating conversion character that determines the conversion specification's behaviour. All ordinary characters (including the terminating null byte) are copied unchanged into the array.

For example, the nineteenth of April, two-thousand seven, at eleven hours, ten minutes and twenty-five seconds formatted using different format strings:

- "%d/%m/%Y %H:%M:%S" will give "19/04/2007 10:11:25"
- "%Y-%m-%d %H.%M" will give "2007-04-19 10.11"
- "%Y\t%m\t%d\t%H\t%M\t%S" will give "2007 04 19 10 11 25"

List of available conversion specifications:

Format	Description
%a	locale's abbreviated weekday name.
%A	locale's full weekday name.
%b	locale's abbreviated month name.
%B	locale's full month name.
%c	locale's appropriate date and time representation.
%C	century number (the year divided by 100 and truncated to an integer) as a decimal number [00-99].
%d	day of the month as a decimal number [01,31].
%D	same as %m/%d/%y.
%e	day of the month as a decimal number [1,31]; a single digit is preceded by a space.
%h	same as %b.
%H	hour (24-hour clock) as a decimal number [00,23].
%I	hour (12-hour clock) as a decimal number [01,12].
%j	day of the year as a decimal number [001,366].
%m	month as a decimal number [01,12].
%M	minute as a decimal number [00,59].
%n	is replaced by a newline character.
%p	locale's equivalent of either a.m. or p.m.
%r	time in a.m. and p.m. notation; in the POSIX locale this is equivalent to %I:%M:%S %p.
%R	time in 24 hour notation (%H:%M).
%S	second as a decimal number [00,61].
%t	is replaced by a tab character.
%T	time (%H:%M:%S).
%u	weekday as a decimal number [1,7], with 1 representing Monday.
%U	week number of the year (Sunday as the first day of the week) as a decimal number [00,53].
%V	week number of the year (Monday as the first day of the week) as a decimal number [01,53]. If the week containing 1 January has four or more days in the new year, then it is considered week 1. Otherwise, it is the last week of the previous year, and the next week is week 1.
%w	weekday as a decimal number [0,6], with 0 representing Sunday.
%W	week number of the year (Monday as the first day of the week) as a decimal number [00,53]. All days in a new year preceding the first Monday are considered to be in week 0.
%x	locale's appropriate date representation.
%X	locale's appropriate time representation.
%y	year without century as a decimal number [00,99].
%Y	year with century as a decimal number.
%Z	timezone name or abbreviation, or by no bytes if no timezone information exists.
%%	character %.

3.4 Example of an input dataset as a single FluidX file

```
<?xml version="1.0" standalone="yes"?>
<openfluid>

  <model>
    <gparams>
      <param name="gparam1" value="100" />
      <param name="gparam2" value="0.1" />
```

```
  </gparams>
  <function fileID="example.functionA" />
  <generator varname="example.generator.fixed" unitclass="EU1"
             method="fixed" varsize="11">
    <param name="fixedvalue" value="20" />
  </generator>
  <generator varname="example.generator.random" unitclass="EU2"
             method="random">
    <param name="min" value="20.53" />
    <param name="max" value="50" />
  </generator>
  <function fileID="example.functionB">
    <param name="param1" value="strvalue" />
    <param name="param2" value="1.1" />
    <param name="gparam1" value="50" />
  </function>
</model>


<domain>

  <definition>
    <unit class="PU" ID="1" pcsorder="1" />
    <unit class="EU1" ID="3" pcsorder="1">
      <to class="EU1" ID="11" />
      <childof class="PU" ID="1" />
    </unit>
    <unit class="EU1" ID="11" pcsorder="3">
      <to class="EU2" ID="2" />
    </unit>
    <unit class="EU2" ID="2" pcsorder="1" />
  </definition>

  <inputdata unitclass="EU1" colorder="indataA">
    3 1.1
    11 7.5
  </inputdata>

  <inputdata unitclass="EU2" colorder="indataB1;indataB3">
    2 18 STRVALX
  </inputdata>

  <calendar>
    <event unitclass="EU1" unitID="11" date="1999-12-31 23:59:59">
      <info key="when" value="before" />
      <info key="where" value="1" />
      <info key="var1" value="1.13" />
      <info key="var2" value="EADGBE" />
    </event>
    <event unitclass="EU2" unitID="3" date="2000-02-05 12:37:51">
      <info key="var3" value="152.27" />
      <info key="var4" value="XYZ" />
    </event>
    <event unitclass="EU1" unitID="11" date="2000-02-25 12:00:00">
      <info key="var1" value="1.15" />
      <info key="var2" value="EADG" />
    </event>
  </calendar>

</domain>


<run>
  <deltat>3600</deltat>
```

```
    <period begin="2000-01-01 00:00:00" end="2000-03-27 01:12:37" />
    <progressout packet="10" keep="2" />
  </run>


  <output>
    <files colsep=" " dtformat="%Y %m %d %H %M %S" commentchar="%">
      <set name="testRS" unitsclass="RS" unitsIDs="51;232" vars="*" />
      <set name="full" unitsclass="SU" unitsIDs="*" vars="*" precision="7" />
    </files>
  </output>

</openfluid>
```

3.5 File formats for `interp` data generator

3.5.1 Sources

The sources file format is an XML based format which defines a list of sources files associated to an unique ID.
The sources must be defined in a section delimited by the `<datasources>` tag, inside an `<openfluid>` tag and must be structured following these rules:

- Inside the `<datasources>` tag, there must be a set of `<filesource>` tags
- Each `<filesource>` tag must bring an `ID` attribute giving the identifier of source, and an `file` attribute giving the name of the file containing the source of data. The files must be placed in the input directory of the simulation.

example of a sources list file

```
<?xml version="1.0" standalone="yes"?>
<openfluid>

 <datasources>
    <filesource ID="1" file="source1.dat" />
    <filesource ID="2" file="source2.dat" />
  </datasources>

</openfluid>
```

An associated source data file is a seven columns text file, containing a serie of values in time. The six first columns are the date using the following format `YYYY MM DD HH MM SS`. The 7^{th} column is the value itself.

example of a source data file

```
1999 12 31 12 00 00      -1.0
1999 12 31 23 00 00      -5.0
2000 01 01 00 30 00      -15.0
2000 01 01 00 40 00      -5.0
2000 01 01 01 30 00      -15.0
```

3.5.2 Distribution

A distribution file is a two column file associating a unit ID (1^{st}column) to a source ID (2^{nd}column).

example of distribution file

```
2 2
3 1
4 2
5 1
```

3.6 Useful links

3.6.1 OpenFLUID project

- OpenFLUID web site : http://www.umr-lisah.fr/openfluid/
- OpenFLUID web community : http://www.umr-lisah.fr/openfluid/community/
- OpenFLUID on SourceSup (software forge): https://sourcesup.cru.fr/projects/openfluid/

3.6.2 External tools

- Geany : http://www.geany.org/
- Gnuplot : http://www.gnuplot.info/
- GRASS GIS : http://grass.itc.it/
- jEdit : http://www.jedit.org/
- Octave : http://www.gnu.org/software/octave/
- QGIS : http://www.qgis.org/
- R : http://www.r-project.org/
- Scilab : http://www.scilab.org/