OpenFLUID

Software Environment for Modelling Fluxes in Landscapes

Quick reference manual

OpenFLUID-Engine 1.4.2

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:

"to note" information with blue background...

The source code examples are emphasized like this:

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

The "warning" informations are emphasized like this:

"warning" informations with red background...

Chapter 1

File formats

This part of this manual describes the file formats. Refer to the "Usage" part of this manual to run the simulation.

1.1 Spatial domain definition (*.ddef.xml)

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 preocessing 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.
This information is defined through XML files that must end with the suffix `.ddef.xml`. All the files in the dataset named using this suffix will be read and considered as spatial domain definition files, and must be structured following these rules:

- These files are XML files
- The root tag must be `<openfluid>`
- Inside the `<openfluid>` tag, there must be a `<domain>` tag
- Inside the `<domain>` tag, there must be a `<definition>` tag
- 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 `<class>`
- 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

```
<?xml version="1.0" standalone="yes"?>
<openfluid>
  <domain>
    <definition>
      <unit class="SU" ID="1" pcsorder="1">
        <to class="SU" ID="2" />
      </unit>
      <unit class="SU" ID="2" pcsorder="2">
        <to class="RS" ID="1" />
      </unit>
      <unit class="SU" ID="3" pcsorder="1">
        <to class="RS" ID="2" />
      </unit>
      <unit class="RS" ID="1" pcsorder="1">
        <to class="RS" ID="2" />
      </unit>
      <unit class="RS" ID="2" pcsorder="1">
      </unit>
    </definition>
  </domain>
</openfluid>
```

1.2 Flux model definition (model.xml)

The flux model is defined by an ordered set of simulations functions that will be plugged to the OpenFLUID-Engine kernel. It defines the model for the simulation.
The flux model must be defined in a file named `model.xml`, and must be structured following these rules:

- The `model.xml` file is an XML file
- The root tag must be `<openfluid>`
- Inside the `<openfluid>` tag, there must be a `<model>` tag
- Inside the `<domain>` tag, there must be a set of `<function>` tags
- Each `<function>` tag must bring a `<fileID>` attr
- ibute 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).

The order of the simulation functions in the `model.xml` is very important : the same order will be used for executions on the same time step

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

      <function fileID="water.atm-surf.rain-su.files" />

      <function fileID="water.surf-uz.runoff-infiltration.mseytoux" >
        <param name="resstep" value="0.000005" />
      </function>

      <function fileID="water.surf.transfer-su.hayami">
        <param name="maxsteps" value="100" />
        <param name="meancel" value="0.045" />
        <param name="meansigma" value="500" />
      </function>

  </model>
</openfluid>
```

1.3 Spatial domain input data (*.ddata.xml)

The spatial domain input data are static data brought by units, usually properties and initial conditions for each unit.
This information is defined through XML files that must end with the suffix `.ddata.xml`. All the files in the dataset named using this suffix will be read and considered as spatial domain input data files, and must be structured following these rules:

- These files are XML files
- The root tag must be `<openfluid>`
- Inside the `<openfluid>` tag, there must be a `<domain>` tag
- Inside the `<domain>` tag, there must be one (and only one) `<inputdata>` tag
- The `<inputdata>` tag must bring a `<unitclass>` attribute giving the unit class to which input data must be attached
- Inside the `<inputdata>` tag, there must be one (and only one) `<columns>` tag
- The `<columns>` tag must bring a `<order>` attribute defining the order of the columns in the `<data>` tag.
- Inside the `<inputdata>` tag, there must be one (and only one) `<data>` tag containing 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 `<inputdata>` tag, the following columns are values following the column order given through the `<columns>` tag.

```
<?xml version="1.0" standalone="yes"?>
<openfluid>
  <domain>
    <inputdata unitclass="SU">
      <columns order="ks;hc;betaMS;thetares;thetasat;nmanning" />
      <data>

1 0.000001 0.1 1.3 0.02 0.36 0.05
2 0.000001 0.1 1.3 0.02 0.36 0.05
3 0.000001 0.1 1.3 0.02 0.36 0.05
4 0.000001 0.1 1.3 0.02 0.36 0.05
5 0.000001 0.1 1.3 0.02 0.36 0.05
6 0.000001 0.1 1.3 0.02 0.36 0.05
7 0.000001 0.1 1.3 0.02 0.36 0.05

      </data>
    </inputdata>
  </domain>
</openfluid>
```

1.4 Discrete events (*.events.xml)

The discrete events are events occuring on units, and that can be processed by simulation functions. They are defined through calendars in XML files that must end with the suffix `.events.xml`. All the files in the dataset named using this suffix will be read and considered as spatial domain input data files, and must be structured following these rules:

- These files are XML files
- The root tag must be `<openfluid>`
- Inside the `<openfluid>` tag, there must be a `<calendar>` tag
- 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.

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

    <event name="" category="test" unitclass="SU" unitID="1" 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 name="" category="test" unitclass="RS" unitID="1" date="1999-12-01 12:00:00">
      <info key="when" value="before"/>
      <info key="where" value="1"/>
      <info key="var3" value="152.27"/>
      <info key="var4" value="XYZ"/>
    </event>
    <event name="" category="test" unitclass="SU" unitID="2" date="1999-12-01 12:00:00">
      <info key="when" value="before"/>
      <info key="where" value="7"/>
      <info key="var1" value="1.15"/>
      <info key="var2" value="EADG"/>
    </event>

  </calendar>
</openfluid>
```

1.5 Run configuration(run.xml)

The configuration of the simulation gives the simulation period, the data exchange time step, and the optionnal progressive output parameters.
The configuration of the simulation must be defined in a file named `run.xml`, and must be structured following these rules:

- The `run.xml` file is an XML file
- The root tag must be `<openfluid>`
- Inside the `<openfluid>` tag, there must be a `<run>` tag
- 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.

```
<?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>
```

1.6 Outputs configuration(output.xml)

The configuration of the simulation outputs gives the description of the saved results.
The configuration of the outputs must be defined in a file named `output.xml`, and must be structured following these rules:

- The `run.xml` file is an XML file
- The root tag must be `<openfluid>`
- Inside the `<openfluid>` tag, there must be a `<output>` tag
- 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), 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. 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.

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

    <!-- dtformat can be predefined (6cols,iso) or using the strftime() format, default is iso -->
    <!-- colsep default is \t -->
    <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="*" />
    </files>

  </output>
</openfluid>
```

Chapter 2

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. The following instructions mainly applies to Linux platforms.

2.1 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 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.

2.2 Input dataset

Before running the simulation, the input dataset must be built. An OpenFLUID-Engine input dataset includes different informations, shared into many files:

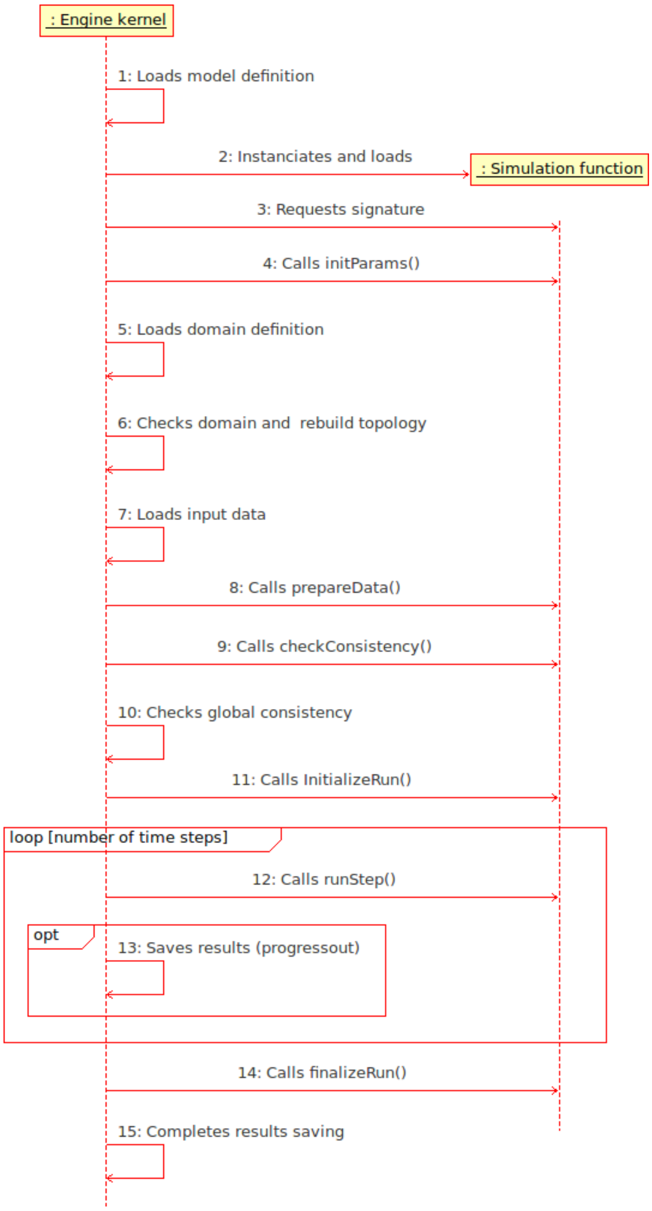
- the spatial domain definition
- the flux model definition
- the spatial domain input data
- the discrete events
- the run configuration
- the outputs configuration

All these 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.

2.3 Engine sequence

The following sequence diagram describes the stage-by-stage execution of an OpenFLUID-Engine-engine simulation. The kernel is the main application, and the simulation function represents each simulation function used in the model definition.



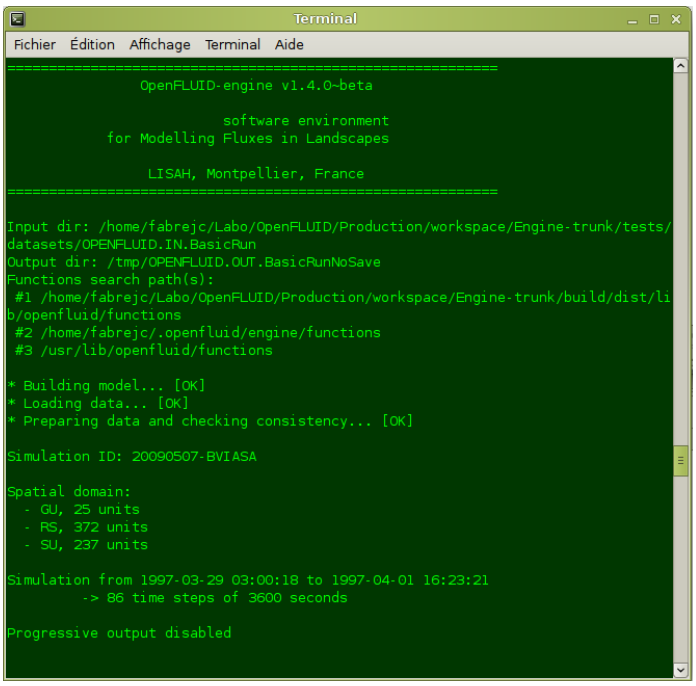
Stages description:

1. the kernel loads the model definition (from the `model.xml` file)

2. the kernel loads and instanciates the simulation functions, according to the model definition
3. the kernel requests the signature of each simulation function
4. the kernel runs the initParams() method of each simulation function
5. the kernel loads the domain definition (from the `*.ddef.xml` files)
6. the kernel check the domain definition consistency and rebuild the domain topology
7. the kernel loads the domain input data (from the `*.ddata.xml` files)
8. the kernel runs the prepareData() method of each simulation function
9. the kernel runs the checkConsistency() method of each simulation function
10. the kernel checks the global consistency (model + domain definition + domain input data)
11. the kernel runs the initializeRun() method of each simulation function
12. at every time step, the kernel runs the runStep() method of each simulation function
13. at every time step, if the progressive output of results is enabled and if the current time step is a "progressive output time step", the kernel saves a packet of data and frees memory
14. the kernel runs the finalizeRun() method of each simulation function
15. the kernel completes the save of results (all results if progressive output is disabled, the remaining results if progressive output is enabled)

2.4 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.



2.5 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 save your outputs in 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.

2.6 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
```

2.6.1 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.

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/outputdir,pdf=1
```

2.6.2 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
```

2.6.3 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
```

2.6.4 convert

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

Required options:

`convmode`	Conversion mode. Available modes are: `13_14`
`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=/path/to/outputdir
```

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
`--no-varname-check`	do not check variable name against nomenclature
`-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 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:

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

3.3 Useful links

3.3.1 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/

3.3.2 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/