

EDF R&D



FLUID DYNAMICS, POWER GENERATION AND ENVIRONMENT DEPARTMENT
SINGLE PHASE THERMAL-HYDRAULICS GROUP

6, QUAI WATIER
F-78401 CHATOU CEDEX

TEL: 33 1 30 87 75 40
FAX: 33 1 30 87 79 16

JANUARY 2019

Code_Saturne documentation

***Code_Saturne* version 5.3.1: studymanager**

contact: saturne-support@edf.fr



| | | |
|---------|---|--|
| EDF R&D | <i>Code_Saturne</i> version 5.3.1: studymanager | <i>Code_Saturne</i> documentation Page 1/ 21 |
|---------|---|--|

TABLE OF CONTENTS

| | | |
|----------|---|----|
| 1 | Introduction | 2 |
| 2 | Installation and prerequisites | 3 |
| 3 | Command line options | 3 |
| 4 | File of parameters | 5 |
| 4.1 | BEGIN AND END OF THE FILE OF PARAMETERS | 5 |
| 4.2 | CASE CREATION AND COMPILATION OF THE USER FILES | 5 |
| 4.3 | RUN CASES | 6 |
| 4.4 | COMPARE CHECKPOINT FILES | 7 |
| 4.5 | RUN EXTERNAL ADDITIONAL PREPROCESSING SCRIPTS WITH OPTIONS | 8 |
| 4.6 | RUN EXTERNAL ADDITIONAL POSTPROCESSING SCRIPTS WITH OPTIONS FOR A CASE | 9 |
| 4.7 | RUN EXTERNAL ADDITIONAL POSTPROCESSING SCRIPTS WITH OPTIONS FOR A STUDY | 10 |
| 4.8 | POST-PROCESSING: CURVES | 11 |
| 4.8.1 | DEFINE CURVES | 11 |
| 4.8.2 | DEFINE SUBSETS OF CURVES | 12 |
| 4.8.3 | DEFINE FIGURES | 13 |
| 4.8.4 | EXPERIMENTAL OR ANALYTICAL DATA | 14 |
| 4.8.5 | CURVES WITH ERROR BAR | 14 |
| 4.8.6 | MONITORING POINTS OR PROBES | 15 |
| 4.8.7 | MATPLOTLIB RAW COMMANDS | 15 |
| 4.9 | POST-PROCESSING: INPUT FILES | 16 |
| 5 | Output and restart | 17 |
| 6 | Tricks | 17 |

1 Introduction

STUDYMANAGER is a small framework to automate the launch of *Code_Saturne* computations and do some operations on new results.

The script needs a directory of previous *Code_Saturne* cases which are candidates to be duplicated. This directory is called **repository**. The duplication is done in a new directory which is called the **destination**.

For each duplicated case, STUDYMANAGER is able to compile the user files, to run the case, to compare the obtained checkpoint file with the previous one from the **repository**, and to plot curves in order to illustrate the computations.

For all these steps, STUDYMANAGER generate two reports, a global report which summarizes the status of each case, and a detailed report which gives the differences between the new results and the previous ones in the **repository**, and display the defined plots.

In the **repository**, previous results of computations are required only for checkpoint files comparison purpose. They can be also useful, if the user needs to run specific scripts.

2 Installation and prerequisites

STUDYMANAGER does not need a specific installation: the related files are installed with the other Python scripts of *Code_Saturne*. Nevertheless, additional prerequisites required are:

- `numpy`,
- `matplotlib`,

3 Command line options

The command line options can be found with the command: `code_saturne studymanager -h`.

- `-h`, `--help`: show the help message and exit
- `-f FILE`, `--file=FILE`: give the file of parameters for STUDYMANAGER. This file is mandatory, and therefore this option must be completed
- `-q`, `--quiet`: do not print status messages to stdout
- `-u`, `--update`: update installation paths in scripts (i.e. `SaturneGUI` and `runcase`) only in the repository, reinitialize XML files of parameters and compile
- `-x`, `--update-xml`: update only XML files in the repository
- `-t`, `--test-compile`: compile all cases
- `-r`, `--run`: run all cases
- `-n N_ITERATIONS`, `--n-iterations=N_ITERATIONS`: maximum number of iterations for cases of the study
- `-c`, `--compare`: compare checkpoint files between **repository** and **destination**
- `-d REFERENCE`, `--ref-dir=REFERENCE`: absolute reference directory to compare dest with
- `-p`, `--post`: postprocess results of computations
- `-m ADDRESS1 ADDRESS2 ...`, `--mail=ADDRESS1 ADDRESS2 ...`: addresses for sending the reports
- `-l LOG_FILE`, `--log=LOG_FILE`: name of studymanager log file (default value is 'studymanager.log')
- `-z`, `--disable-tex`: disable text rendering with \LaTeX when plotting with Matplotlib. It then uses Mathtext instead which is less complete
- `--rm`: remove all existing run directories in destination directory
- `--fow`: overwrite files in MESH and POST directories in destination directory
- `-s`, `--skip-pdflatex`: disable tex reports compilation with pdflatex
- `--fmt=DEFAULT_FMT`: set the global format for exporting matplotlib figure (default is pdf)
- `--repo=REPO_PATH`: force the path to the repository directory
- `--dest=DEST_PATH`: force the path to the destination directory
- `-g`, `--debug`: activate debugging mode

| | | |
|---------|---|---|
| EDF R&D | <i>Code_Saturne</i> version 5.3.1: studymanager | <i>Code_Saturne</i> documentation Page 4/21 |
|---------|---|---|

- `--with-tags=WITH_TAGS`: only process runs with all specified tags (separated by commas)
- `--without-tags=WITHOUT_TAGS`: exclude any run with one of specified tags (separated by commas)

Examples:

- duplicates all cases from the **repository** in the **destination**, compile all user files and exits;

```
$ code_saturne studymanager -f sample.xml
```

- as above, and run all cases if defined in `sample.xml`

```
$ code_saturne studymanager -f sample.xml -r
```

- as above, and compares all new checkpoint files with those from the **repository** if defined in `sample.xml`

```
$ code\_saturne smgr -f sample.xml -r -c
```

- as above, and plots results if defined in `sample.xml`

```
$ code_saturne smgr -f sample.xml -rcp
```

- as above, and send the two reports

```
$ code_saturne smgr -f sample.xml -r -c -p -m "dt@moulinsart.be \
dd@moulinsart.be"
```

- compares and plots results in the **destination** already computed

```
$ code_saturne smgr -f sample.xml -c -p
```

- compares and plots results in the **destination** already computed

```
$ code_saturne smgr -f sample.xml -c -p
```

- run cases tagged "coarse" (standing for coarse mesh for example) and "hr" (standing for high Reynolds for example) only for 2 time iterations in destination directory of path `../RUNS/RIBS` (RIBS will be created, RUNS already exists). The command is launched from inside the study directory, hence the repository which has to be the directory containing the original study is simply indicated by `..`

```
$ code_saturne smgr -f smgr_ribs.xml -r -n 2 --with-tags=coarse,hr \
--dest=../RUNS/RIBS --repo=..
```

Note:

The detailed report is generated only if the options `-c`, `--compare` or `-p`, `--post` is present in the command line.

4 File of parameters

The file of parameters is a XML formatted ascii file.

4.1 Begin and end of the file of parameters

This example shows the four mandatory first lines of the file of parameters.

```
<?xml version="1.0"?>
<studymanager>
  <repository>/home/dupond/codesaturne/MyRepository</repository>
  <destination>/home/dupond/codesaturne/MyDestination</destination>
```

The third and fourth lines correspond to the definition of the **repository** and **destination** directories. Inside the markups **<repository>** and **<destination>** the user must inform the related directories. If the **destination** does not exist, the directory is created.

The last line of the file of parameters must be:

```
</studymanager>
```

4.2 Case creation and compilation fo the user files

When STUDYMANAGER is launched, the file of parameters is parsed in order to known which studies and cases from the **repository** should be duplicated in the **destination**. The selection is done with the markups **<study>** and **<case>** as the following example:

```
<?xml version="1.0"?>
<studymanager>
  <repository>/home/dupond/codesaturne/MyRepository</repository>
  <destination>/home/dupond/codesaturne/MyDestination</destination>

  <study label="MyStudy1" status="on">
    <case label="Grid1" run_id="Grid1" status="on" compute="on" post="off"/>
    <case label="Grid2" run_id="Grid2" status="off" compute="on" post="off"/>
  </study>
  <study label="MyStudy2" status="off">
    <case label="k-eps" status="on" compute="on" post="off"/>
    <case label="Rij-eps" status="on" compute="on" post="off"/>
  </study>
</studymanager>
```

The attributes are:

- **label**: the name of the file of the script;
- **status**: must be equal to **on** or **off**, activate or deactivate the markup;
- **compute**: must be equal to **on** or **off**, activate or deactivate the computation of the case;
- **post**: must be equal to **on** or **off**, activate or deactivate the post-processing of the case;
- **run_id**: name of the run directory (sub-directory of RESU) in which the result is stored. This attribute is optional. If it is not set (or if set to **run_id=""**), an automatic value will be proposed by the code (usually based on current date and time).

- **tags**: possible tags distinguishing the run from the others in the same XML parameter file (ex.: `tags="coarse,high-reynolds"`).

Only the attributes **label**, **status**, **compute** and **post** are mandatory.

If the directory specified by the attribute **run_id** already exists, the computation is not performed again. For the post-processing step, the existing results are taken into account only if no error file is detected in the directory.

With the attribute **status**, a single case or a complete study can be switched off. In the above example, only the case **Grid1** of the study **MyStudy1** is going to be created.

After the creation of the directories in the **destination**, for each case, all user files are compiled. The STUDYMANAGER stops if a compilation error occurs: neither computation nor comparison nor plot will be performed, even if they are switched on.

Notes:

- During the duplication, every files are copied, except mesh files, for which a symbolic link is used.
- During the duplication, if a file already exists in the **destination**, this file is not overwritten by STUDYMANAGER by default.

4.3 Run cases

The computations are activated if the option **-r**, **--run** is present in the command line.

All cases described in the file of parameters with the attribute **compute="on"** are taken into account.

```
<?xml version="1.0"?>
<studymanager>
  <repository>/home/dupond/codesaturne/MyRepository</repository>
  <destination>/home/dupond/codesaturne/MyDestination</destination>

  <study label="MyStudy1" status="on">
    <case label="Grid1" status="on" compute="on" post="off"/>
    <case label="Grid2" status="on" compute="off" post="off"/>
  </study>
  <study label="MyStudy2" status="on">
    <case label="k-eps" status="on" compute="on" post="off"/>
    <case label="Rij-eps" status="on" compute="on" post="off"/>
  </study>
</studymanager>
```

After the computation, if no error occurs, the attribute **compute** is set to **"off"** in the copy of the file of parameters in the **destination**. It is allow to restart STUDYMANAGER without re-run successfull previous computations.

Note that it is allowed to run several times the same case in a given study. The case has to be repeated in the file of parameters:

```
<?xml version="1.0"?>
<studymanager>
  <repository>/home/dupond/codesaturne/MyRepository</repository>
  <destination>/home/dupond/codesaturne/MyDestination</destination>

  <study label="MyStudy1" status="on">
    <case label="CASE1" run_id="Grid1" status="on" compute="on" post="on">
      <prepro label="grid.py" args="-m grid1.med -p cas.xml" status="on"/>
    </case>
  </study>
```

```

    </case>
    <case label="CASE1" run_id="Grid2" status="on" compute="on" post="on"/>
      <prepro label="grid.py" args="-m grid2.med -p cas.xml" status="on"/>
    </case>
  </study>
</studymanager>

```

If nothing is done, the case is repeated without modifications. In order to modify the setup between two runs of the same case, an external script has to be used to change the related setup (see sections 4.5 and 6).

4.4 Compare checkpoint files

The comparison is activated if the option `-c`, `--compare` is present in the command line.

In order to compare two checkpoint files for a given case, a markup `<compare>` has to be added as child of the considered case. In the following exemple, a checkpoint file comparison is switched on for the case *Grid1* (for all variables, with the default threshold), whereas no comparison is planed for the case *Grid2*. The comparison is done by the external script `cs_io_dump` with the option `--diff`.

```

<study label='MyStudy1' status='on'>
  <case label='Grid1' status='on' compute="on" post="off">
    <compare dest="" repo="" status="on"/>
  </case>
  <case label='Grid2' status='on' compute="off" post="off"/>
</study>

```

The attributes are:

- **repo**: id of the results directory in the **repository** for example `repo="20110704-1116"`, if there is a single results directory in the RESU directory of the case, the id can be ommitted: `repo=""`;
- **dest**: id of the results directory in the **destination**:
 - if the id is not known already because the case has not yet run, just let the attribute empty `dest=""`, the value will be updated after the run step in the **destination** directory (see section 5);
 - if STUDYMANAGER is restarted without the run step (with the command line `code_saturne studymanager -f sample.xml -c` for example), the id of the results directory in the **destination** must be given (for example `dest="20110706-1523"`), but if there is a single results directory in the RESU directory of the case, the id can be ommitted: `dest=""`, the id will be completed automatically;
- **args**: additional options for the script `cs_io_dump`
 - ◇ `--section`: name of a particular variable;
 - ◇ `--threshold`: real value above which a difference is considered significant (default: $1e-30$ for all variables);
- **status**: must be equal to `on` or `off`: activate or deactivate the markup.

Only the attributes **repo**, **dest** and **status** are mandatory.

Several comparisons with different options are permitted:


```
<study label='MyStudy1' status='on'>
  <case label='Grid1' status='on' compute="on" post="off">
    <compare dest="" repo="" args="--section Pressure --threshold=1000" status="on"/>
    <compare dest="" repo="" args="--section VelocityX --threshold=1e-5" status="on"/>
    <compare dest="" repo="" args="--section VelocityY --threshold=1e-3" status="on"/>
  </case>
</study>
```

Comparisons results will be summarized in a table in the file `report_detailed.pdf` (see [5](#)):

| Variable Name | Diff. Max | Diff. Mean | Threshold |
|---------------|-----------|------------|-----------|
| VelocityX | 0.102701 | 0.00307058 | 1.0e-5 |
| VelocityY | 0.364351 | 0.00764912 | 1.0e-3 |

Alternatively, in order to compare all activated cases (status at on) listed in a STUDYMANAGER parameter file, a reference directory can be provided directly in the command line, as follows:

```
code_saturne studymanager -f sample.xml -c -d /scratch/***/reference_destination_directory.
```

4.5 Run external additional preprocessing scripts with options

The markup `<prepro>` has to be added as a child of the considered case.

```
<study label='STUDY' status='on'>
  <case label='CASE1' status='on' compute="on" post="on">
    <prepro label="mesh_coarse.py" args="-n 1" status="on"/>
  </case>
</study>
```

The attributes are:

- **label**: the name of the file of the considered script;
- **status**: must be equal to **on** or **off**: activate or deactivate the markup;
- **args**: additional options to pass to the script.

Only the attributes **label** and **status** are mandatory.

An additional option `"-c"` (or `"--case"`) is given by default with the path of the current case as argument (see example in section [6](#) for decoding options).

Note that all options must be processed by the script itself.

Several calls of the same script or to different scripts are permitted:

```
<study label="STUDY" status="on">
  <case label="CASE1" status="on" compute="on" post="on">
    <prepro label="script_pre1.py" args="-n 1" status="on"/>
    <prepro label="script_pre2.py" args="-n 2" status="on"/>
  </case>
</study>
```

All preprocessing scripts are first searched in the MESH directory from the current study in the **repository**. If a script is not found, it is searched in the directories of the current case. The main objective of running such external scripts is to create or modify meshes or to modify the current setup of the related case (see section 6).

4.6 Run external additional postprocessing scripts with options for a case

The launch of external scripts is activated if the option `-p`, `--post` is present in the command line. The markup `<script>` has to be added as a child of the considered case.

```
<study label='STUDY' status='on'>
  <case label='CASE1' status='on' compute='on' post='on'>
    <script label="script_post.py" args="-n 1" dest="" repo="20110216-2147" status="on"/>
  </case>
</study>
```

The attributes are:

- **label**: the name of the file of the considered script;
- **status**: must be equal to **on** or **off**: activate or deactivate the markup;
- **args**: the arguments to pass to the script;
- **repo** and **dest**: id of the results directory in the **repository** or in the **destination**;
 - if the id is not known already because the case has not yet run, just let the attribute empty **dest=""**, the value will be updated after the run step in the **destination** directory (see section 5);
 - if there is a single results directory in the RESU directory (either in the **repository** or in the **destination**) of the case, the id can be omitted: **repo=""** or **dest=""**, the id will be completed automatically.

If attributes **repo** and **dest** exist, their associated value will be passed to the script as arguments, with options `"-r"` and `"-d"` respectively.

Only the attributes **label** and **status** are mandatory.

Several calls of the same script or to different scripts are permitted:

```
<study label="STUDY" status="on">
  <case label="CASE1" status="on" compute="on" post="on">
    <script label="script_post.py" args="-n 1" status="on"/>
    <script label="script_post.py" args="-n 2" status="on"/>
    <script label="script_post.py" args="-n 3" status="on"/>
    <script label="another_script.py" status="on"/>
  </case>
</study>
```

All postprocessing scripts must be in the POST directory from the current study in the **repository**. The main objective of running external scripts is to create or modify results in order to plot them.

Example of script, which searches printed informations in the listing, note the function to process the passed command line arguments:

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import os, sys
import string
from optparse import OptionParser

def process_cmd_line(argv):
    """Processes the passed command line arguments."""
    parser = OptionParser(usage="usage: %prog [options]")

    parser.add_option("-r", "--repo", dest="repo", type="string",
                      help="Directory of the result in the repository")

    parser.add_option("-d", "--dest", dest="dest", type="string",
                      help="Directory of the result in the destination")

    (options, args) = parser.parse_args(argv)
    return options

def main(options):
    m = os.path.join(options.dest, "listing")
    f = open(m)
    lines = f.readlines()
    f.close()

    g = open(os.path.join(options.dest, "water_level.dat"), "w")
    g.write("# time, h_sim, h_th\n")
    for l in lines:
        if l.rfind("time, h_sim, h_th") == 0:
            d = l.split()
            g.write("%s %s %s\n" % (d[3], d[4], d[5]))
    g.close()

if __name__ == '__main__':
    options = process_cmd_line(sys.argv[1:])
    main(options)
```

4.7 Run external additional postprocessing scripts with options for a study

The launch of external scripts is activated if the option `-p`, `--post` is present in the command line.

The purpose of this functionality is to create new data based on several runs of cases, and to plot them (see section 4.8) or to insert them in the final detailed report (see section 4.9).

The markup `<postpro>` has to be added as a child of the considered study.

```
<study label='STUDY' status='on'>
  <case label='CASE1' status='on' compute='on' post='on'>
    <postpro label='Grid2.py' status='on' arg='-n 100'>
      <data file="profile.dat">
        <plot fig="1" xcol="1" ycol="2" legend="Grid level 2" fmt='b-p'>
        <plot fig="2" xcol="1" ycol="3" legend="Grid level 2" fmt='b-p'>
      </data>
    <input file="output.dat" dest="">
    </postpro>
  </study>
```

The attributes are:

- **label**: the name of the file of the considered script;
- **status**: must be equal to **on** or **off**: activate or deactivate the markup;
- **args**: the additional options to pass to the script;

Only the attributes **label** and **status** are mandatory.

The options given to the script in the command line are:

- **-s** or **--study**: label of the current study;
- **-c** or **--cases**: string which contains the list of the cases
- **-d** or **--directories**: string which contains the list of the directories of results.

Additional options can be pass to the script throught the attributes **args**.

Note that all options must be processed by the script itself.

Several calls of the same script or to different scripts are permitted.

4.8 Post-processing: curves

The post-processing is activated if the option **-p**, **--post** is present in the command line.

The following example shows the drawing of four curves (or plots, or 2D lines) from two files of data (which have the same name **profile.dat**). There are two subsets of curves (i.e. frames with axis and 2D lines), in a single figure. The figure will be saved on the disk in a **pdf** (default) or **png** format, in the **POST** directory of the related study in the **destination**. Each drawing of a single curve is defined as a markup child of a file of data inside a case. Subsets and figures are defined as markup children of **<study>**.

```
<study label='Study' status='on'>
  <case label='Grid1' status='on' compute="off" post="on">
    <data file="profile.dat" dest="">
      <plot fig="1" xcol="1" ycol="2" legend="Grid level 1" fmt='r-s' />
      <plot fig="2" xcol="1" ycol="3" legend="Grid level 1" fmt='r-s' />
    </data>
  </case>
  <case label='Grid2' status='on' compute="off" post="on">
    <data file="profile.dat" dest="">
      <plot fig="1" xcol="1" ycol="2" legend="Grid level 2" fmt='b-p' />
      <plot fig="2" xcol="1" ycol="3" legend="Grid level 2" fmt='b-p' />
    </data>
  </case>
  <subplot id="1" legstatus='on' legpos ='0.95 0.95' ylabel="U ($m/s$)" xlabel="Time ($s$)" />
  <subplot id="2" legstatus='on' legpos ='0.95 0.95' ylabel="U ($m/s$)" xlabel="Time ($s$)" />
  <figure name="velocity" idlist="1 2" figsize="(4,5)" format="png" />
</study>
```

4.8.1 Define curves

The curves of computational data are build from data files. These data must be ordered as column and the files should be in results directory in the **RESU** directory (either in the **repository** or in the **destination**). Commentaries are allowed in the file, the head of every commentary line must start with character **#**.

In the file of parameters, curves are defined with two markups: `<data>` and `<plot>`:

- `<data>`: child of markup `<case>`, defines a file of data;
 - `file`: name of the file of data
 - `repo` or `dest`: id of the results directory either in the **repository** or in the **destination**;
 - ⇒ if the id is not known already because the case has not yet run, just let the attribute empty `dest=""`, the value will be updated after the run step in the **destination** directory (see section 5);
 - ⇒ if there is a single results directory in the RESU directory (either in the **repository** or in the **destination**) of the case, the id can be omitted: `repo=""` or `dest=""`, the id will be completed automatically.

The attribute `file` is mandatory, and either `repo` or `dest` must be present (but not the both) even if it is empty.

- `<plot>`: child of markup `<data>`, defines a single curve; the attributes are:
 - `fig`: id of the subset of curves (i.e. markup `<subplot>`) where the current curve should be plotted;
 - `xcol`: number of the column in the file of data for the abscisse;
 - `ycol`: number of the column in the file of data for the ordinate;
 - `legend`: add a label to a curve;
 - `fmt`: format of the line, composed from a symbol, a color and a linestyle, for example `fmt="r--"` for a dashed red line;
 - `xplus`: real to add to all values of the column `xcol`;
 - `yplus`: real to add to all values of the column `ycol`;
 - `xfois`: real to multiply to all values of the column `xcol`;
 - `yfois`: real to multiply to all values of the column `ycol`;
 - `xerr` or `xerrp`: draw horizontal error bar (see section 4.8.5);
 - `yerr` or `yerrp`: draw vertical error bar (see section 4.8.5);
 - some standard options of 2D lines can be added, for example `markevery="2"` or `markersize="3.5"`. These options are summarized in the table 2. Note that the options which are string of characters must be overquoted likes this: `color="'g'"`.

The attributes `fig` and `ycol` are mandatory.

In case a column should undergo a transformation specified by the attributes `xfois`, `yfois`, `xplus`, `yplus`, scale operations take precedence over translation operations.

Details on 2D lines properties can be found in the `matplotlib` documentation. For more advanced options see section 4.8.7.

4.8.2 Define subsets of curves

A subset of curves is a frame with two axis, axis labels, legend, title and drawing of curves inside. Such subset is called subplot in the nomenclature of `matplotlib`.

`<subplot>`: child of markup `<study>`, defines a frame with severals curves; the attributes are:

- `id`: id of the subplot, should be an integer;
- `legstatus`: if "on" display the frame of the legend;

| Property | Value Type |
|------------------------|--|
| alpha | float (0.0 transparent through 1.0 opaque) |
| antialiased or aa | True or False |
| color or c | any matplotlib color |
| dash_capstyle | butt ; round ; projecting |
| dash_joinstyle | miter ; round ; bevel |
| dashes | sequence of on/off ink in points ex: dashes="(5,3)" |
| label | any string, same as legend |
| linestyle or ls | - ; -- ; -. ; : ; steps ; ... |
| linewidth or lw | float value in points |
| marker | + ; , ; . ; 1 ; 2 ; 3 ; 4 ; ... |
| markeredgecolor or mec | any matplotlib color |
| markeredgewidth or mew | float value in points |
| markerfacecolor or mfc | any matplotlib color |
| markersize or ms | float |
| markevery | None ; integer; (startind, stride) |
| solid_capstyle | butt ; round ; projecting |
| solid_joinstyle | miter ; round ; bevel |
| zorder | any number |

Table 2: Options authorized as attributes of the markup plot.

- **legpos**: sequence of the relative coordinates of the center of the legend, it is possible to draw the legend outside the axis;
- **title**: set title of the subplot;
- **xlabel**: set label for the x axis;
- **ylabel**: set label for the y axis;
- **xlim**: set range for the x axis;
- **ylim**: set range for the y axis.

The attributes **fig** and **ycol** are mandatory.

For more advanced options see section [4.8.7](#).

4.8.3 Define figures

Figure is a compound of subset of curves.

<figure>: child of markup **<study>**, defines a pictures with a layout of frames; the attributes are:

- **name**: name of the file to be written on the disk;
- **idlist**: list of the subplot to be displayed in the figure;
- **title**: add a title on the top of the figure;
- **nbrow**: impose a number of row of the layout of the subplots;
- **nbcol**: impose a number of column of the layout of the subplots;
- **format**: format of the file to be written on the disk, **"pdf"** (default) or **"png"** ¹;

¹Other format could be choosen (eps, ps, svg,...), but the pdf generation with pdflatex will failed.

- **figsize**: width x height in inches; defaults to (4,4);
- **dpi**: resolution; defaults to 200 if format is set to pdf; or to 800 if format is set to png; only customizable for png format.

The attributes **name** and **idlist** are mandatory.

4.8.4 Experimental or analytical data

A particular markup is provided for curves of experimental or analytical data: **<measurement>**; the attributes are:

- **file**: name of the file to be read on the disk;
- **path**: path of the directory where the file of data is. the path could be omitted (**path=""**), and in this case, the file will be searched recursively in the directories of the considered study.

The attributes **file** and **path** are mandatory.

In order to draw curves of experimental or analytical data, the markup **<measurement>** should be used with the markup **<plot>** as illustrated below:

```
<study label='MyStudy' status='on'>
  <measurement file='expl.dat' path=''>
    <plot fig='1' xcol='1' ycol='2' legend='U Experimental data' />
    <plot fig='2' xcol='3' ycol='4' legend='V Experimental data' />
  </measurement>
  <measurement file='exp2.dat' path=''>
    <plot fig='1' xcol='1' ycol='2' legend='U Experimental data' />
    <plot fig='2' xcol='1' ycol='3' legend='V Experimental data' />
  </measurement>
  <case label='Grid1' status='on' compute="off" post="on">
    <data file="profile.dat" dest="">
      <plot fig="1" xcol="1" ycol="2" legend="U computed" fmt='r-s' />
      <plot fig="2" xcol="1" ycol="3" legend="V computed" fmt='b-s' />
    </data>
  </case>
</study>
<subplot id="1" legstatus='on' ylabel="U ($m/s$)" xlabel= "$r$ ($m$)" legpos = '0.05 0.1' />
<subplot id="2" legstatus='off' ylabel="V ($m/s$)" xlabel= "$r$ ($m$)" />
<figure name="MyFigure" idlist="1 2" figsize="(4,4)" />
```

4.8.5 Curves with error bar

In order to draw horizontal and vertical error bars, it is possible to specify to the markup **<plot>** the attributes **xerr** and **yerr** respectively (or **xerrp** and **yerrp**). The value of these attributes could be:

- the number of the column, in the file of data, that contains the total absolute uncertainty spans:

```
<measurement file='axis.dat' path=''>
  <plot fig='1' xcol='1' ycol='3' legend='Experimental data' xerr='2' />
</measurement>
```

- the numbers of the two columns, in the file of data, that contain the absolute low spans and absolute high spans of uncertainty:

```
<data file='profile.dat' dest="">
  <plot fig='1' xcol='1' ycol='2' legend='computation' yerr='3 4' />
</data>
```

- a single real value equal to the percentage of uncertainty that should be applied to the considered data set:

```
<data file='profile.dat' dest="">
  <plot fig='1' xcol='1' ycol='2' legend='computation' yerrp='2.' />
</data>
```

4.8.6 Monitoring points or probes

A particular markup is provided for curves of probes data: `<probes>`; the attributes are:

- **file**: name of the file to be read on the disk;
- **fig**: id of the subset of curves (i.e. markup `<subplot>`) where the current curve should be plotted;
- **dest**: id of the results directory in the **destination**:
 - if the id is not known already because the case has not yet run, just let the attribute empty **dest=""**, the value will be updated after the run step in the **destination** directory (see section 5);
 - if STUDYMANAGER is restarted without the run step (with the command line `code_saturne studymanager -f sample.xml -c` for example), the id of the results directory in the **destination** must be given (for example **dest="20110706-1523"**), but if there is a single results directory in the RESU directory of the case, the id can be omitted: **dest=""**, the id will be completed automatically;

The attributes **file**, **fig** and **dest** are mandatory.

In order to draw curves of probes data, the markup `<probes>` should be used as a child of a markup `<case>` as illustrated below:

```
<study label='MyStudy' status='on'>
  <measurement file='expl.dat' path=''>
    <plot fig='1' xcol='1' ycol='2' legend='U Experimental data' />
  </measurement>
  <case label='Grid1' status='on' compute="off" post="on">
    <probes file="probes_U.dat" fig="2" dest="">
      <data file="profile.dat" dest="">
        <plot fig="1" xcol="1" ycol="2" legend="U computed" fmt='r-s' />
      </data>
    </case>
  </study>
  <subplot id="1" legstatus='on' ylabel="U ($m/s$)" xlabel= "$r$ ($m$)" legpos ='0.05 0.1' />
  <subplot id="2" legstatus='on' ylabel="U ($m/s$)" xlabel= "$time$ ($s$)" legpos ='0.05 0.1' />
  <figure title="Results" name="MyFigure" idlist="1" />
  <figure title="Grid1: probes for velocity" name="MyProbes" idlist="2" />
```

4.8.7 Matplotlib raw commands

The file of parameters allows to execute additional matplotlib commands (i.e Python commands), for curves (2D lines), or subplot, or figure. For every object drawn, **studymanager** associate a name to this object that can be reused in standard matplotlib commands. Therefore, children markup `<plt.command>` could be added to `<plot>`, `<subplot>` or `<figure>`.

It is possible to add commands with **Matlab style** or **Python style**. For the Matlab style, commands are called as methods of the module `plt`, and for Python style commands or called as methods of the instance of the graphical object.

Matlab style and Python style commands can be mixed.

- curves or 2D lines: when a curve is drawn, the associated name are `line` and `lines` (with `line = lines[0]`).

```
<plot fig="1" xcol="1" ycol="2" fmt='g^' legend="Simulated water level">
  <plt_command>plt.setp(line, color="blue")</plt_command>
  <plt_command>line.set_alpha(0.5)</plt_command>
</plot>
```

- subset of curves (subplot): for each subset, the associated name is `ax`:

```
<subplot id="1" legend='Yes' legpos ='0.2 0.95'>
  <plt_command>plt.grid(True)</plt_command>
  <plt_command>plt.xlim(0, 20)</plt_command>
  <plt_command>ax.set_ylim(1, 3)</plt_command>
  <plt_command>plt.xlabel(r"Time ($s$)", fontsize=8)</plt_command>
  <plt_command>ax.set_ylabel(r"Level ($m$)", fontsize=8)</plt_command>
  <plt_command>for l in ax.xaxis.get_ticklabels(): l.set_fontsize(8)</plt_command>
  <plt_command>for l in ax.yaxis.get_ticklabels(): l.set_fontsize(8)</plt_command>
  <plt_command>plt.axis([-0.05, 1.6, 0.0, 0.15])</plt_command>
  <plt_command>plt.xticks([-3, -2, -1, 0, 1])</plt_command>
</subplot>
```

4.9 Post-processing: input files

The post-processing is activated if the option `-p`, `--post` is present in the command line.

STUDYMANAGER is able to include files into the final detailed report. These files must be in the directory of results either in the **destination** or in the **repository**. The following example shows the inclusion of three files: `performance.log` and `setup.log` from the **destination**, and a `performance.log` from the **repository**:

```
<case label='Grid1' status='on' compute="on" post="on">
  <input dest="" file="performance.log"/>
  <input dest="" file="setup.log"/>
  <input repo="" file="performance.log"/>
</case>
```

Text files, L^AT_EXsource files, or graphical (PNG, JPEG, or PDF) files may be included.

In the file of parameters, input files are defined with markups `<input>` as children of a single markup `<case>`. The attributes of `<input>` are:

- `file`: name of the file to be included
- `repo` or `dest`: id of the results directory either in the **repository** or in the **destination**;
 - ⇒ if the id is not known already because the case has not yet run, just let the attribute empty `dest=""`, the value will be updated after the run step in the **destination** directory (see section 5);
 - ⇒ if there is a single results directory in the RESU directory (either in the **repository** or in the **destination**) of the case, the id can be omitted: `repo=""` or `dest=""`, the id will be completed automatically.

The attribute `file` is mandatory, and either `repo` or `dest` must be present (but not the both) even if it is empty.

5 Output and restart

STUDYMANAGER produces several files in the **destination** directory:

- **report.txt**: standard output of the script;
- **auto.vnv.log**: log of the code and the **pdflatex** compilation;
- **report_global.pdf**: summary of the compilation, run, comparison, and plot steps;
- **report_detailed.pdf**: details the comparison and display the plot;
- **sample.xml**: updated file of parameters, useful for restart the script if an error occurs.

After the computation of a case, if no error occurs, the attribute **compute** is set to "off" in the copy of the file of parameters in the **destination**. It is allow a restart of STUDYMANAGER without re-run successfull previous computations. In the same manner, all empty attributes **repo=""** and **dest=""** are completed in the updated file of parameters.

6 Tricks

- How to comment markups in the file of parameter ?

The opening and closing signs for commantaries are `<!--` and `-->`. In the following example, nothing from the study **MyStudy2** will be read:

```
<?xml version="1.0"?>
<studymanager>
  <repository>/home/dupond/codesaturne/MyRepository</repository>
  <destination>/home/dupond/codesaturne/MyDestination</destination>

  <study label="MyStudy1" status="on">
    <case label="Grid1" status="on" compute="on" post="on"/>
    <case label="Grid2" status="on" compute="off" post="on"/>
  </study>
  <!--
  <study label="MyStudy2" status="on">
    <case label="k-eps" status="on" compute="on" post="on"/>
    <case label="Rij-eps" status="on" compute="on" post="on"/>
  </study>
  -->
</studymanager>
```

- How to add text in a figure ?

It is possible to use raw commands:

```
<subplot id='301' ylabel ='Location ($m$)' title='Before jet -0.885' legstatus='off'>
  <plt_command>plt.text(-4.2, 0.113, 'jet')</plt_command>
  <plt_command>plt.text(-4.6, 0.11, r'$\downarrow$', fontsize=15)</plt_command>
</subplot>
```

- Adjust margins for layout of subplots in a figure.

You have to use the raw command **subplots_adjust**:

```
<subplot id="1" legend='Yes' legpos ='0.2 0.95'>
  <plt_command>plt.subplots_adjust(hspace=0.4, wspace=0.4, right=0.9,
    left=0.15, bottom=0.2, top=0.9)</plt_command>
</subplot>
```

- How to find a syntax error in the XML file ?

When there is a misprint in the file of parameters, STUDYMANAGER indicates the location of the error with the line and the column of the file:

```
my_case.xml file reading error.
```

```
This file is not in accordance with XML specifications.
```

```
The parsing syntax error is:
```

```
my_case.xml:86:12: not well-formed (invalid token)
```

- How to render less-than and greater-than signs in legends, titles or axis labels ?

The less-than < and greater-than > symbols are among the five predefined entities of the XML specification that represent special characters.

In order to have one of the five predefined entities rendered in any legend, title or axis label, use the string "&name;" . Refer to the following table 3 for the name of the character to be rendered:

| name | character | description |
|------|-----------|-----------------------|
| quot | " | double quotation mark |
| amp | & | ampersand |
| apos | ' | apostrophe |
| lt | < | less-than sign |
| gt | > | greater-than sign |

Table 3: Some predefined entities of XML specification.

For any of this predefined entities, the XML parser will first replace the string "&name;" by the character, which will then allow L^AT_EX(or Mathtext if L^AT_EXis disabled) to process it.

For example, in order to write " $\lambda < 1$ " in a legend, the following attribute will be used:

```
<plot fig="4" fmt="k--" legend="solution for $\lambda$ < 1" xcol="1" ycol="2"/>
```

- How to set a logarithmic scale ?

The following raw commands have to be used:

```
<subplot id="2" title="Grid convergence" xlabel="Number of cells" ylabel="Error (\%)">
  <plt_command>ax.set_xscale('log')</plt_command>
  <plt_command>ax.set_yscale('log')</plt_command>
</subplot>
```

- How to create a mesh automatically with SALOME ?

The following example shows how to create a mesh with a SALOME command file:

```
<study label="STUDY" status="on">
  <case label="CASE1" status="on" compute="on" post="on">
    <prepro label="salome.sh" args="-t -u my_mesh.py" status="on"/>
  </case>
</study>
```

with the script `salome.sh` (depending of the local installation of SALOME):

```
#!/bin/bash
```

```
export ROOT_SALOME=/home/salome/salome-640/Salome-V6_4_0-c7-v2
source /home/salome/salome-640/Salome-V6_4_0-c7-v2/salome_prerequisites_V6_4_0_appli.sh
source /home/salome/salome-640/Salome-V6_4_0-c7-v2/salome_modules_V6_4_0.sh
```

```
/home/salome/salome-640/appli_V6_4_0/bin/salome/runSalome $*
```

and the script of SALOME commands `my_mesh.py`:

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import geomp
import smesh

# create a box
box = geomp.MakeBox(0., 0., 0., 100., 200., 300.)
idbox = geomp.addToStudy(box, "box")

# create a mesh
tetra = smesh.Mesh(box, "MeshBox")

algo1D = tetra.Segment()
algo1D.NumberOfSegments(7)

algo2D = tetra.Triangle()
algo2D.MaxElementArea(800.)

algo3D = tetra.Tetrahedron(smesh.NETGEN)
algo3D.MaxElementVolume(900.)

# compute the mesh
tetra.Compute()

# export the mesh in a MED file
tetra.ExportMED("./my_mesh.med")
```

- How to carry out a grid convergence study ?

The following exemple shows how to carry out a grid convergence study by running the same case three times and changing the parameters between each run with the help of a prepro script.

Here the mesh, the maximum number of iterations, the reference time step and the number of processes are modified, before each run, by the script `prepro.py`.

The file of parameters is as follows:

```
<case compute="on" label="COUETTE" post="on" run_id="21_Theta_1" status="on">
  <prepro args="-m 21_Theta_1.med -p Couette.xml -n 4000 -a 1. -t 0.01024 -u 1"
    label="prepro.py" status="on"/prepro>
  <data dest="" file="profile.dat">
    <plot fig="5" fmt="r-+" legend="21 theta 1" markersize="5.5" xcol="1" ycol="5"/>
  </data>
</case>

<case compute="on" label="COUETTE" post="on" run_id="43_Theta_05" status="on">
  <prepro args="-m 43_Theta_05.med -p Couette.xml -n 8000 -a 0.5 -t 0.00512 -u 2"
    label="prepro.py" status="on"/prepro>
  <data dest="" file="profile.dat">
    <plot fig="5" fmt="b" legend="43 Theta 05" markersize="5.5" xcol="1" ycol="5"/>
  </data>
</case>

<case compute="on" label="COUETTE" post="on" run_id="86_Theta_025" status="on">
  <prepro args="-m 86_Theta_025.med -p Couette.xml -n 16000 -a 0.25 -t 0.00256 -u 4"
    label="prepro.py" status="on" /prepro>
  <data dest="" file="profile.dat">
    <plot fig="5" fmt="g" legend="86 Theta 025" markersize="5.5" xcol="1" ycol="5"/>
  </data>
</case>
```

Recall that the case attribute `run_id` should be given a different value for each run, while the `label` should stay the same and that the prepro script should be copied in the directory `MESH` of the study or in the directory `DATA` of the case.

The prepro script is given below. Note that it can be called inside the file of parameters without specifying a value for each option:

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
#-----
#
#-----

#-----
# Standard modules import
#-----

import os, sys
import string
from optparse import OptionParser

#-----

#-----
# Application modules import
#-----
from Pages.ScriptRunningModel import ScriptRunningModel

#-----

def process_cmd_line(argv):
    """Processes the passed command line arguments."""
    parser = OptionParser(usage="usage: %prog [options]")

    parser.add_option("-c", "--case", dest="case", type="string",
                      help="Directory of the current case")

    parser.add_option("-p", "--param", dest="param", type="string",
                      help="Name of the file of parameters")

    parser.add_option("-m", "--mesh", dest="mesh", type="string",
                      help="Name of the new mesh")

    parser.add_option("-n", "--iter-num", dest="iterationsNumber", type="int",
                      help="New iteration number")

    parser.add_option("-u", "--n-procs", dest="n_procs", type="int",
                      help="Number of processes (units)")

    parser.add_option("-t", "--time-step", dest="timeStep", type="float",
                      help="New time step")

    parser.add_option("-a", "--perio-angle", dest="rotationAngle", type="float",
                      help="Periodicity angle")

    (options, args) = parser.parse_args(argv)

    return options

#-----
```

```

def main(options):
    from cs_package import package
    from Base.XMLengine import Case
    from Base.XMLinitialize import XMLinit
    from Pages.SolutionDomainModel import SolutionDomainModel
    from Pages.TimeStepModel import TimeStepModel
    from Pages.SteadyManagementModel import SteadyManagementModel

    fp = os.path.join(options.case, "DATA", options.param)
    if os.path.isfile(fp):
        try:
            case = Case(package = package(), file_name = fp)
        except:
            print("Parameters file reading error.\n")
            print("This file is not in accordance with XML specifications.")
            sys.exit(1)

        case['xmlfile'] = fp
        case.xmlCleanAllBlank(case.xmlRootNode())
        XMLinit(case).initialize()

    if options.mesh:
        s = SolutionDomainModel(case)
        l = s.getMeshList()
        s.delMesh(l[0])
        s.addMesh((options.mesh, None))

    if options.rotationAngle:
        s.setRotationAngle(0, options.rotationAngle)

    if (options.iterationsNumber):
        s = SteadyManagementModel(case)
        t = TimeStepModel(case)
        if s.getSteadyFlowManagement() == 'on':
            s.setNbIter(options.iterationsNumber)
        else:
            t.setIterationsNumber(options.iterationsNumber)

    if (options.TimeStep):
        t = TimeStepModel(case)
        t.setTimeStep(options.TimeStep)

    if (options.n_procs):
        mdl = ScriptRunningModel(case)
        mdl.setString('n_procs', str(options.n_procs))

    case.xmlSaveDocument()

#-----

if __name__ == '__main__':
    options = process_cmd_line(sys.argv[1:])
    main(options)

#-----

```