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

AUGUST 2019

Code_Saturne documentation

Code_Saturne version 5.3.3 practical user's guide

contact: saturne-support@edf.fr

ABSTRACT

Code_Saturne is a system designed to solve the Navier-Stokes equations in the cases of 2D, 2D axisymmetric or 3D flows. Its main module is designed for the simulation of flows which may be steady or unsteady, laminar or turbulent, incompressible or potentially dilatable, isothermal or not. Scalars and turbulent fluctuations of scalars can be taken into account. The code includes specific modules, referred to as "specific physics", for the treatment of Lagrangian particle tracking, semi-transparent radiative transfer, gas combustion, pulverised coal combustion, electricity effects (Joule effect and electric arcs) and compressible flows. *Code_Saturne* relies on a finite volume discretisation and allows the use of various mesh types which may be hybrid (containing several kinds of elements) and may have structural non-conformities (hanging nodes).

The present document is a practical user's guide for *Code_Saturne* version 5.3.3. It is the result of the joint effort of all the members in the development team. It presents all the necessary elements to run a calculation with *Code_Saturne* version 5.3.3. It then lists all the variables of the code which may be useful for more advanced utilisation. The user subroutines of all the modules within the code are then documented. Eventually, for each key word and user-modifiable parameter in the code, their definition, allowed values, default values and conditions for use are given. These key words and parameters are grouped under headings based on their function. An alphabetical index list is also given at the end of the document for easier consultation.

Code_Saturne is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version. *Code_Saturne* is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

TABLE OF CONTENTS

1 Introduction . . . 9
2 Quick start . . . 10
2.1 How to use the Doxygen documentation? . . . 10
2.2 Running a calculation . . . 10
2.3 Troubleshooting . . . 11
3 Practical information about *Code_Saturne* . . . 12
3.1 System Environment for *Code_Saturne* . . . 12
3.1.1 Preliminary settings . . . 12
3.1.2 Configuration file . . . 12
3.1.3 Standard directory hierarchy . . . 13
3.1.4 *Code_Saturne* Solver library files . . . 15
3.2 Setting up and running a calculation . . . 15
3.2.1 Step by step calculation . . . 15
3.2.2 Temporary execution directory . . . 17
3.2.3 Execution modes . . . 17
3.2.4 Environment variables . . . 18
3.2.5 Interactive modification of selected parameters . . . 19
3.3 Case preparer . . . 20
3.4 Supported mesh and post-processing output formats . . . 20
3.4.1 Formats supported for input . . . 21
3.4.2 Formats supported for input or output . . . 23
3.4.3 Formats supported for output only . . . 27
3.4.4 Meshing tools and associated formats . . . 27
3.4.5 Meshing remarks . . . 27
3.5 Preprocessor command line options . . . 27
3.6 Solver command line options . . . 28
3.7 Launch scripts . . . 29
3.8 Graphical User Interface . . . 29
3.9 User subroutines . . . 31
3.9.1 Preliminary comments . . . 31
3.9.2 Example routines . . . 31
3.9.3 Main variables . . . 31
3.9.4 Using selection criteria in user subroutines . . . 40
3.10 Face and cell mesh-defined properties and selection . . . 42
4 Importing and preprocessing meshes . . . 44

4.1 Preprocessor options . . . 44
4.1.1 Mesh selection . . . 45
4.1.2 Post-processing output . . . 45
4.1.3 Element orientation correction . . . 45
4.2 Environment variables . . . 45
4.2.1 System environment variables . . . 46
4.3 Optional functionality . . . 46
4.4 General remarks . . . 46
4.5 Files passed to the Solver . . . 47
4.6 Mesh preprocessing . . . 47
4.6.1 Joining of non-conforming meshes . . . 47
4.6.2 Periodicity . . . 48
4.6.3 Parameters for conforming or non-conforming mesh joinings . . . 48
4.6.4 Parameters for periodicity . . . 51
4.6.5 Modification of the mesh geometry . . . 51
4.7 Mesh smoothing utilities . . . 52
4.7.1 Fix by feature . . . 52
4.7.2 Warped faces smoother . . . 52
5 Partitioning for parallel runs . . . 52
5.1 Partitioning stages . . . 53
5.2 Partitioner choice . . . 53
5.3 Effect of periodicity . . . 53
6 Basic modelling setup . . . 54
6.1 Initialisation of the main parameters . . . 54
6.2 Selection of mesh inputs: `cs_user_mesh_input` . . . 58
6.3 Non-default variables initialisation . . . 59
6.4 Manage boundary conditions . . . 63
6.4.1 Coding of standard boundary conditions . . . 65
6.4.2 Coding of non-standard boundary conditions . . . 67
6.4.3 Checking of the boundary conditions . . . 69
6.4.4 Sorting of the boundary faces . . . 69
6.4.5 Boundary conditions with LES . . . 70
6.5 Manage the variable physical properties . . . 74
6.5.1 Basic variable physical properties . . . 74
6.5.2 Modification of the turbulent viscosity . . . 76
6.5.3 Modification of the variable C of the dynamic LES model . . . 76
6.6 User source terms . . . 77

6.6.1 In Navier-Stokes . . . 78
6.6.2 For k and ε . . . 78
6.6.3 For R_{ij} and ε . . . 79
6.6.4 For φ and $\overline{f}$ . . . 79
6.6.5 For k and ω . . . 79
6.6.6 For $\tilde{\nu}_t$ . . . 80
6.6.7 For user scalars . . . 80
6.7 Pressure drops (head losses) and porosity . . . 80
6.7.1 Head losses . . . 80
6.7.2 Porosity . . . 81
6.8 Management of the mass sources . . . 81
6.9 User law editor of the GUI . . . 82
6.10 Modification of the variables at the end of a time step . . . 84
7 Advanced modelling setup . . . 85
7.1 Use of a specific physics . . . 85
7.2 Pulverised coal and gas combustion module . . . 90
7.2.1 Boundary conditions . . . 92
7.2.2 Initialisation of the options of the variables . . . 95
7.3 Heavy fuel oil combustion module . . . 98
7.3.1 Initialisation of transported variables . . . 98
7.3.2 Boundary conditions . . . 98
7.4 Radiative thermal transfers in semi-transparent gray media . . . 99
7.4.1 Initialisation of the radiation main parameters . . . 99
7.4.2 Radiative transfers boundary conditions . . . 99
7.4.3 Absorption coefficient of the medium, boundary conditions for the luminance and calculation of the net radiative flux . . . 102
7.5 Conjugate heat transfer . . . 103
7.5.1 Thermal module in a 1D wall . . . 103
7.5.2 Fluid-Thermal coupling with SYRTHES . . . 103
7.6 Particle-tracking (Lagrangian) Module . . . 105
7.6.1 General information . . . 105
7.6.2 Activating the particle-tracking module . . . 105
7.6.3 Basic guidelines for standard simulations . . . 105
7.6.4 Prescribing the main modelling parameters (GUI and/or `cs_user_lagr_model`) 106
7.6.5 Prescribing particle boundary conditions (GUI and/or `uslag2`) . . . 107
7.6.6 Advanced particle-tracking set-up . . . 107
7.7 Compressible module . . . 109

7.7.1 Initialisation of the options of the variables . . . 109
7.7.2 Management of the boundary conditions . . . 110
7.7.3 Initialisation of the variables . . . 110
7.7.4 Management of variable physical properties . . . 110
7.8 Management of the electric arcs module . . . 111
7.8.1 Activating the electric arcs module . . . 111
7.8.2 Initialisation of the variables . . . 111
7.8.3 Variable physical properties . . . 111
7.8.4 Boundary conditions . . . 112
7.8.5 Initialisation of the variable options . . . 113
7.8.6 *EnSight* output . . . 113
7.9 *Code_Saturne-Code_Saturne* coupling . . . 113
7.10 Fluid-Structure external coupling . . . 114
7.11 ALE module . . . 114
7.11.1 Initialisation of the options . . . 114
7.11.2 Mesh velocity boundary conditions . . . 115
7.11.3 Modification of the mesh viscosity . . . 116
7.11.4 Fluid - Structure internal coupling . . . 116
7.12 Management of the structure property . . . 117
7.13 Management of the atmospheric module . . . 118
7.13.1 Directory structure . . . 118
7.13.2 The atmospheric mesh features . . . 118
7.13.3 Atmospheric flow model and steady/unsteady algorithm . . . 119
7.13.4 Physical properties . . . 119
7.13.5 Boundary and initial conditions . . . 119
7.13.6 User subroutines . . . 121
7.13.7 Physical models . . . 122
7.13.8 Atmospheric main variables . . . 123
7.13.9 Recommendations . . . 123
7.14 Cavitation module . . . 124
8 Keyword list . . . 130
8.1 Input-output . . . 130
8.1.1 "Calculation" files . . . 131
8.1.2 Post-processing for *EnSight* or other tools . . . 131
8.1.3 Chronological records of the variables on specific points . . . 131
8.1.4 Time averages . . . 131
8.1.5 Others . . . 131

8.2 Numerical options . . . 132
8.2.1 Calculation management . . . 132
8.2.2 Scalar unknowns . . . 132
8.2.3 Definition of the equations . . . 132
8.2.4 Definition of the time advancement . . . 132
8.2.5 Turbulence . . . 133
8.2.6 Time scheme . . . 133
8.2.7 Gradient reconstruction . . . 134
8.2.8 Solution of the linear systems . . . 134
8.2.9 Convective scheme . . . 134
8.2.10 Pressure-continuity step . . . 134
8.2.11 Error estimators for Navier-Stokes . . . 134
8.2.12 Calculation of the distance to the wall . . . 136
8.2.13 Others . . . 136
8.3 Numerical, physical and modelling parameters . . . 136
8.3.1 Numeric parameters . . . 136
8.3.2 Physical parameters . . . 136
8.3.3 Physical variables . . . 136
8.3.4 Modelling parameters . . . 136
8.4 ALE . . . 137
8.5 Thermal radiative transfers: global settings . . . 137
8.6 Electric module (Joule effect and electric arcs): specificities . . . 137
8.7 Compressible module: specificities . . . 137
9 Bibliography . . . 138
Index of the main variables and keywords . . . 140

1 Introduction

Code_Saturne is an application designed to solve the Navier-Stokes equations in the cases of 2D, 2D axi-symmetric and 3D flows. Its main module is designed for the simulation of flows which may be steady or unsteady, laminar or turbulent, incompressible or potentially dilatable, isothermal or not. Scalars and turbulent fluctuations of scalars can be taken into account. The code includes specific modules, referred to as "specific physics", for the treatment of Lagrangian particle tracking, semi-transparent radiative transfer, gas combustion, pulverised coal combustion, electricity effects (Joule effect and electric arcs) and compressible flows.

Code_Saturne is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version. *Code_Saturne* is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.[1]

Code_Saturne relies on a finite volume discretisation and allows the use of various mesh types which may be hybrid (containing several kinds of elements) and may have structural non-conformities (hanging nodes).

Code_Saturne is composed of two main elements and an optional GUI, as shown on Figure 1:

- the Solver module is the numerical solver
- the Preprocessor module is in charge of mesh import

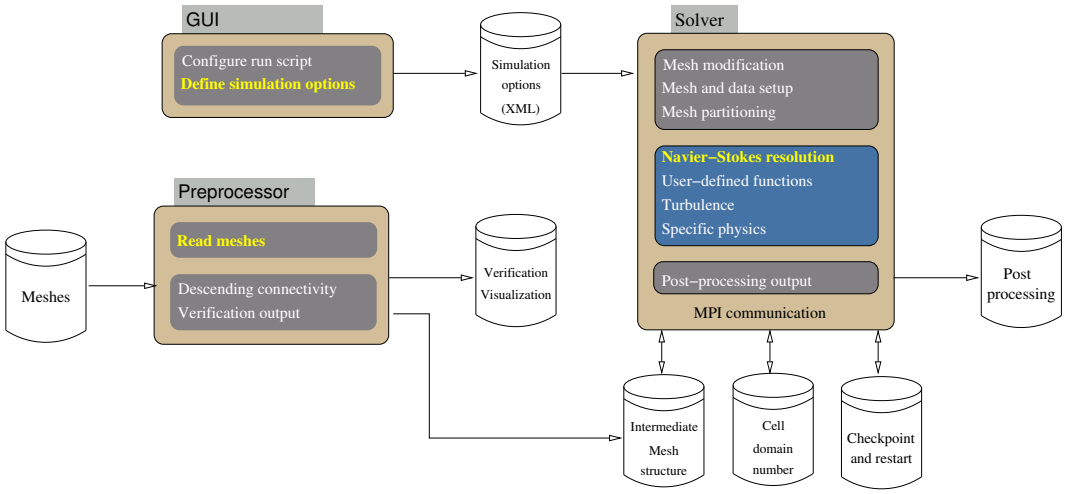

Figure 1: *Code_Saturne* elements

Code_Saturne also relies on the PLE (Parallel Location and Exchange) library (developed by the same team, under LGPL license) for the management of code coupling; this library can also be used independently.

This document is a practical user guide for *Code_Saturne* version 5.3.3. It is the result of the joint effort of all the members in the development team.

This document provides practical information for the usage of *Code_Saturne*. For more details about the algorithms and their numerical implementation, please refer to the reports [1], [4] and [10], and to the theoretical documentation [11].

[1] You should have received a copy of the GNU General Public License along with *Code_Saturne*; if not, write to the Free Software Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA

The latest updated version of this document is available on-line with the version of *Code_Saturne* and accessible through the command `code_saturne info --guide theory`.

This document first presents all the necessary elements to run a calculation with *Code_Saturne* version 5.3.3. It then lists all the variables of the code which may be useful for more advanced users. The user subroutines of all the modules within the code are then documented. Eventually, for each keyword and user-modifiable parameter in the code, their definition, allowed values, default values and conditions for use are given. These keywords and parameters are grouped under headings based on their function. An alphabetical index is also given at the end of the document for easier reference.

2 Quick start

2.1 How to use the Doxygen documentation?

In addition to the present user guide, a complete `Doxygen` documentation automatically generated from the code is available with *Code_Saturne*. It can provide various informations about the implementation such as details on variables used throughout the code kernel and the user subroutines. It also provides an easily explorable set of user subroutine examples and Fortran-C naming references for quantities linked to the mesh or the physical fields.

One can access the `Doxygen` main page through this link or from a terminal by typing the following command: `code_saturne info --guide theory`.

On the front page, several tabs are available :

- **Modules**: list of all the *Code_Saturne* modules,
- **Data structures**: list of all the *Code_Saturne* structures,
- **Files**: list of all the source files with a brief description of their purpose,
- **User examples**: provides various examples of how to use user subroutines,
- **Variables and structures references**: helps users implementing user C functions, Fortran subroutines or developing inside the code kernel.

In any case, the **search bar** can be used to look for a specific keyword which can be a function, a variable, a structure, a type, etc.

2.2 Running a calculation

We assume in this section that the user has at his disposal the calculation data file (calculation set up) or already prepared it following for instance the step-by-step guidance provided in *Code_Saturne* tutorial. The steps described below are intended to provide the user a way to run quickly on a workstation a calculation through the Graphical User Interface (GUI).

The first thing to do before running *Code_Saturne* is to define an alias to the `code_saturne` script (see §3.1.1), for example:

```
alias cs='${prefix}/bin/code_saturne'.
```

When using the *bash* shell, a completion file may be sourced so as to allow for syntax auto-completion:

```
source ${prefix}/etc/bash_completion.d/code_saturne'.
```

The second thing is to prepare the computation directories. For instance, the study directory `T_JUNCTION`, containing a single calculation directory CASE1, will be created by typing the command (see §3.3):

```
code_saturne create -s T_JUNCTION
```

The mesh files should be copied in the directory `MESH` (though they may also be selected from another directory, see §3.2.1), and the Fortran user files necessary for the calculation in the directory `CASE1/SRC`. Finally, the calculation data file `case_name.xml` read by the GUI should be copied to the directory `CASE1/DATA`. Once these steps completed, the user should go in the directory `CASE1/DATA` and type de command line `./SaturneGUI case_name.xml` to load the calculation file into the interface. A window similar to Figure2 will appear. Click on the heading "Calculation management", select the heading "Prepare batch calculation", see Figure 3. After having chosen the number of processors, press "start calculation" to run the calculation.

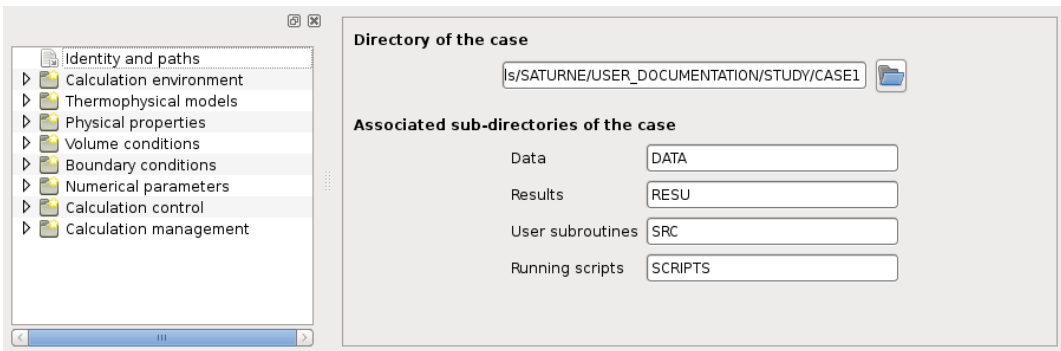

Figure 2: Identity and paths

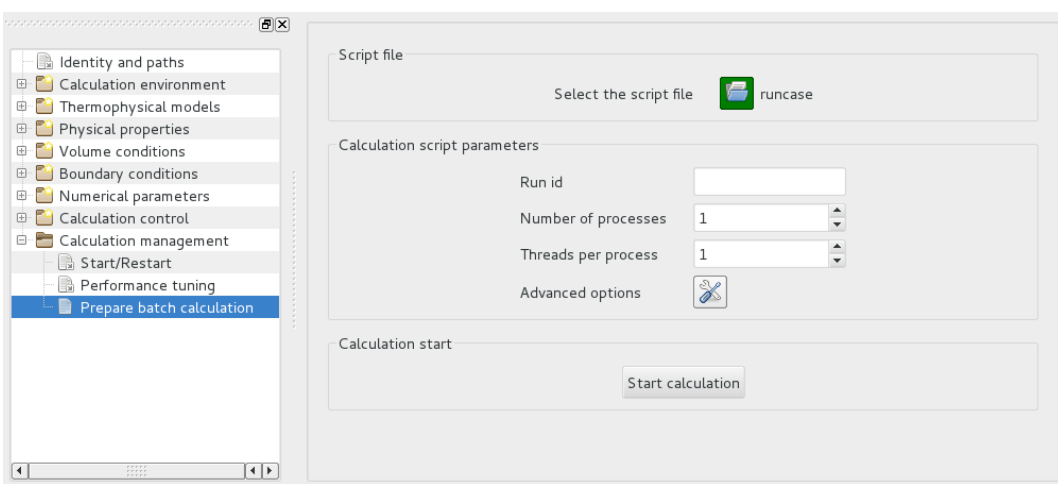

Figure 3: Prepare execution

If no problem arises, the simulation results can be found in the directory `CASE1/RESU` and be read directly by *ParaView* or *EnSight* in `CASE1/RESU/<YYYYMMDD-hhmm>/postprocessing`. Calculation history can be found in the file `<YYYYMMDD-hhmm>/listing`.

2.3 Troubleshooting

If the calculation does not run properly, the user is advised to check the following points in `CASE1/RESU/<YYYYMMDD-hhmm>`:

- if the calculation stops in the pre-processor, the user should check for error messages in the file

preprocessor*.log.

- if the problem is related to boundary conditions, the user should visualise the file error.ensight with *EnSight* or *ParaView*,
- if the calculation stops in the *Code_Saturne* core, the user should look for messages at the end of the files listing and error*. In addition, the user can track the following keywords in the listing; these are specific error signals:
  - SIGFPE: a floating point exception occurred. It happens when there is a division by 0, when the calculation did not converge, or when a real number reached a value over 10^{300}. Depending on the architecture *Code_Saturne* is running on, this type of exception may be caught or ignored.
  - SIGSEGV: a memory error such as a segmentation violation occurred. An array may have exceeded its allocated memory size and a memory location in use was overwritten.

  In order to easily find the problem, it is also advised to use a debug version of *Code_Saturne* (see the installation documentation) in combination with the use of the valgrind tool (if it is installed). The use of valgrind can be specified in the GUI in the advanced options of the item "Prepare batch calculation" under the heading "Calculation management" or without the GUI, in the cs_user_scripts.py file (this file can be found in DATA/REFERENCE and should be copied in DATA, see §3.2.1).

3 Practical information about *Code_Saturne*

3.1 System Environment for *Code_Saturne*

3.1.1 Preliminary settings

In order to use *Code_Saturne*, the user should define the following alias (in their .bashrc, or equivalent, or .alias file, depending on the environment):

```
alias cs='${install_directory}/bin/code_saturne'
```

where install_directory is the base directory where *Code_Saturne* and its components have been installed[2].

This step may be skipped if ${install_directory} is in a standard location (such as /usr or /usr/local.

3.1.2 Configuration file

A configuration file for *Code_Saturne* is available in ${install_directory}/etc. This file can be useful as a post-install step for computing environments using a batch system, for separate front-end and compute systems (such as Blue Gene systems), or for coupling with SYRTHES 4 or *Code_Aster* (see the installation documentation for more details).

A user may define a local configuration, by copying ${install_directory}/etc/code_saturne.cfg (if present) or ${install_directory}/etc/code_saturne.cfg.template to $HOME/.code_saturne.cfg, then uncomment and define the applicable sections.

Note that this user configuration file's settings usually apply to all installed *Code_Saturne* versions.

Two options in the .code_saturne.cfg file could be useful for the user:

[2] Without this step, using the absolute path is still possible

- Set the temporary directory (see §3.2.2 for more details on the temporary execution directory).
- Set the mesh database directory: it is possible to indicate a path where meshes are stored. In this case, the GUI will propose this directory automatically for mesh selection. Without the GUI, it is then possible to fill in the `cs_user_scripts.py` file (see §3.2.1) with the name of the desired mesh of the database directory and the code will find it automatically (be careful if you have the same name for a mesh in the database directory and in the `MESH` directory, the mesh in `MESH` will be used).

3.1.3 Standard directory hierarchy

The standard architecture for the simulation studies is:

An optional study directory containing:

- A directory `MESH` containing the mesh(es) necessary for the study
- A directory `POST` for the potential post-processing scripts (not used directly by the code)
- One or several calculation directories

Every calculation directory contains:

- A directory `SRC` for the potential user subroutines necessary for the calculation
- A directory `DATA` for the calculation data (data file from the interface, input profiles, thermo-chemical data, ...), the user script and the XML file.
- A directory `SCRIPTS` for the launch script
- A directory `RESU` for the results
  To improve the calculation traceability, the files and directories sent to `RESU` after a calculation are placed in a subdirectory named after that run's "id", which is by default based on the run date and time, using the format: `YYYYMMDD-hhmm`. It is also possible to force a specific run id, using the `--id` option of `code_saturne run`.

In the standard cases, `RESU/<run_id>` contains a `postprocessing` directory with the post-processing (visualization) files, a `restart` directory for the calculation restart files, a `monitoring` directory for the files of chronological record of the results at specific locations (probes),
`preprocessor.log` and `listing` files reporting the Preprocessor and the Solver execution. All files from the `DATA` directory not in subdirectories are also copied. For a tracing of the modifications in prior calculations, the user-subroutines used in a calculation are stored in a `src_saturne` subdirectory. The data files (such as the XML Interface data file and thermo-chemical data files) and launch script are also copied into the results directory. `compil.log` and `summary` are respectively reports of the compilation stage and general information on the calculation (type of machine, user, version of the code, ...).

When running, the code may use additional files or directories inside its execution directory, set by the execution script, which include a `mesh_input` file or directory, as well as a `restart` directory (which is a link or copy of a previous run's `checkpoint` directory), as well as a `run_solver.sh` script.

For coupled calculations, whether with *Code_Saturne* itself or SYRTHES, each coupled calculation domain is defined by its own directory (bearing the same name as the domain), but results are placed in a `RESU_COUPLING` directory, with a subdirectory for each run, itself containing one subdirectory per coupled domain. Coupled cases are run through the standard the `code_saturne run` command, but require a coupling parameters file (`coupling_parameters.py`) specified using the `--coupling option`. The run command must be called from the toplevel (`STUDY`) directory, so an additional `STUDY/runcase`

Below are typical contents of a case directory CASE1 in a study STUDY

`STUDY/CASE1/DATA:`	***Code_Saturne* data**
`SaturneGUI`	Graphical User Interface launch script
`study.xml`	Graphical User Interface parameter file
`REFERENCE`	Example of user scripts and meteorological or thermochemical date files (used with the specific physics modules)
`STUDY/CASE1/SRC:`	***Code_Saturne* user subroutines**
`REFERENCE`	Available user subroutines
`EXAMPLES`	Examples of user subroutines
`cs_user_boundary_conditions.f90`	User subroutines used for the present calculation
`cs_user_parameters.f90`	
`STUDY/CASE1/RESU/`*`YYYYMMDD-hhmm:`*	**Results** for the calculation YYYYMMDD-hhmm
`postprocessing`	Directory containing the *Code_Saturne* post-processing output in the *EnSight*, MED, or CGNS format (both volume and boundary);
`src_saturne`	copy of the *Code_Saturne* user subroutines used for the calculation
`monitoring`	Directory containing the chronological records for *Code_Saturne*
`checkpoint`	Directory containing the *Code_Saturne* restart files
`compile.log`	Compilation log
`study.xml`	Graphical User Interface parameter file used for the calculation
`runcase`	Copy of the launch script used for the calculation
`preprocessor.log`	Execution report for the *Code_Saturne* Preprocessor
`listing`	Execution report for the Solver module of *Code_Saturne*
`summary`	General information (machine, user, version, ...)
`STUDY/CASE1/SCRIPTS:`	**Launch script**
`runcase`	Launch script (which may contain batch system keywords)

launch script is used in this case. Note that case-local scripts (such as `STUDY/CASE1/SCRIPTS/runcase`) are still used by the master script to determine which parameter file to use.

So in the coupled case, calculation results would not be placed in `STUDY/CASE1/RESU/`*`YYYYMMDD-hhmm`*, but in `STUDY/RESU_COUPLING/`*`YYYYMMDD-hhmm`*`/CASE1`, with the `summary` file being directly placed in `STUDY/RESU_COUPLING/`*`YYYYMMDD-hhmm`* (as it references all coupled domains).

Below are typical additional contents with a coupled SYRTHES case SOLID1 in a study STUDY

`STUDY/runcase`	Coupled launch script
`STUDY/coupling_parameters.py`	Coupled launch parameters
`STUDY/SOLID1/DATA:`	**SYRTHES data**
`syrthes_data.syd`	SYRTHES data file
`syrthes.py`	SYRTHES script
`usr_examples`	SYRTHES user subroutine examples
`STUDY/RESU_COUPLING/`*`YYYYMMDD-hhmm`*`/SOLID1:`	**results (file names defined in `syrthes.env`)**
`src`	SYRTHES user subroutines used in the calculation
`compile.log`	SYRTHES compilation report
`listsyr`	Execution log
`geoms`	SYRTHES solid geometry file
`histos1`	SYRTHES chronological records at specified monitoring points
`resus1`	SYRTHES calculation restart file (1 time step)
`resusc1`	SYRTHES chronological solid post-processing file (may be transformed into the *EnSight* or *MED* format with the *syrthes4ensight* or *syrthes4med30* utility)

3.1.4 *Code_Saturne* Solver library files

Information about the content of the *Code_Saturne* base directories is given below. It is not of vital interest for the user, but given only as general information. Indeed, the case preparer command `code_saturne create` automatically extracts the necessary files and prepares the launch script without the user having to go directly into the *Code_Saturne* base directories (see §3.3). The `code_saturne info` command gives direct access to the most needed information (especially the user's and theory guides and the Doxygen documentation) without the user having to look for them in the *Code_Saturne* directories.

The subdirectories {`install_directory`}`/lib` and {`install_directory`}`/bin` contain the libraries and compiled executables respectively.

The data files (for instance thermochemical data) are located in the directory `data`.

The user subroutines are available in the directory `src/user`, with examples in `src/user_examples`. The case preparer command `code_saturne create` copies all these files in the user directories `SRC/REFERENCE` and `SRC/EXAMPLES` during the case preparation.

The directory `bin` contains an example of the launch script, the compilation parameter files and various utility programs.

3.2 Setting up and running a calculation

3.2.1 Step by step calculation

This paragraph summarises the different steps which are necessary to prepare and run a standard case:

- Check the version of *Code_Saturne* set for use in the environment variables (`code_saturne info --version`). If it does not correspond to the desired version, update the user profile or aliases to get the required version, logging out of the session and in again if necessary (cf. §3.1.1).
- Prepare the different directories using the `code_saturne create` command (see §3.3).
- It is recommended to place the mesh(es) in the directory `MESH`, but they may be selected from other directories, either with the Graphical User Interface (GUI) or the `cs_user_scripts.py` file (see below). Make sure they are in a format compliant with *Code_Saturne* (see §3.4.5). There can be several meshes in case of mesh joining or coupling with SYRTHES[3].
- Go to the directory `DATA` and launch the GUI using the command `./SaturneGUI`.
- If not using the GUI, copy the `DATA/REFERENCE/cs_user_scripts.py` file to `DATA` and edit it, so that the correct run options and paths may be set. For advanced uses, this file may also be used in conjunction with the GUI. Just as with user Fortran subroutines below, settings defined in this file have priority over those defined in the GUI.
- Place the necessary user subroutines in the directory `SRC` (see §3.9). When not using the Interface, some subroutines are compulsory.

  For all physics:

  compulsory without Graphical User Interface:

  - `usipph` (in `cs_user_parameters.f90`) to specify the turbulence and temperature models
  - `usipsu` (in `cs_user_parameters.f90`) to define most user parameters
  - `cs_user_boundary_conditions` to manage the boundary conditions

  very useful without Graphical User Interface:

[3] SYRTHES 4 uses meshes composed of 4-node tetrahedra

- `cs_user_model.c` (in `cs_user_parameters.c`) to define user scalars (species)
- `usipes` (in `cs_user_parameters.f90`) to define monitoring points and additional parameters for results outputs

very useful:

- `usphyv` (in `cs_user_physical_properties.f90`) to manage variable physical properties (fluid density, viscosity ...)
- `cs_user_initialization` to manage the non-standard initialisations

For the "gas combustion" specific physics:

compulsory without Graphical User Interface:

- `usppmo` (in `cs_user_parameters.f90`) to select a specific physics module and combustion model

very useful:

- `cs_user_combustion` (in `cs_user_parameters.f90`), depending on the selected combustion model, to specify the calculation options for the variables corresponding to combustion model

For the "pulverized fuel combustion" specific physics:

compulsory without Graphical User Interface:

- `usppmo` (in `cs_user_parameters.f90`) to select the specific physics module

very useful:

- `cs_user_combustion` (in `cs_user_parameters.f90`) to specify the calculation options for the variables corresponding to pulverized fuel combustion

or `cs_user_combustion`

For the "heavy fuel combustion" specific physics:
(not accessible through the Graphical User Interface in version 5.3.3)

compulsory:

- `usppmo` (in `cs_user_parameters.f90`) to select the specific physics module
- `cs_user_combustion` (in `cs_user_parameters.f90`) to specify the calculation options for the variables corresponding to heavy fuel combustion

For the "atmospheric module" specific physics:

compulsory without Graphical User Interface:

- `usppmo` (in `cs_user_parameters.f90`) to select the specific physics module

very useful:

- `usati1` (in `cs_user_parameters.f90`) to manage the reading of the meteo file
- `usadtvor usatsoil` (in `cs_user_atmospheric_model.f90`) to manage the options to the specific physics

For the "electric module" specific physics (Joule effect and electric arcs):

compulsory without Graphical User Interface:

- `usppmo` (in `cs_user_parameters.f90`) to select the specific physics module
- `cs_user_initialization` to initialise the enthalpy in case of Joule effect
- `cs_user_physical_properties.c` to define the physical properties in case of Joule effect

very useful:

- `cs_user_model` and `cs_user_parameters` (in `cs_user_parameters.c`) to manage the options related to the variables corresponding to the electric module

For the "Lagrangian module" (dispersed phase):
(the continuous phase is managed in the same way as for a case of standard physics)

compulsory without Graphical User Interface:

- `cs_user_lagr_model` to manage the calculation conditions
- `cs_user_lagr_boundary_conditions` to manage the boundary conditions for the dispersed phase

For the "compressible module":

compulsory without Graphical User Interface:

- `usppmo` (in `cs_user_parameters.f90`) to select the specific physics module

very useful:

- `uscfx1` and `uscfx2` (in `cs_user_parameters.f90`) to manage the calculation parameters
- `usphyv` (in `cs_user_physical_properties` to manage the variable physical properties

A comprehensive list of the user subroutines and their instructions for use are given in §3.9.

- If necessary, place in the directory `DATA` the different external data (input profiles, thermochemical data files, ...)
- Prepare the launch script `runcase`, directly or through the Graphical Interface (see §3.7), or prepare the `DATA/cs_user_scripts.py` file.
- Run the calculation and analyse the results
- If necessary, purge the temporary files (in `RESU/<run_id>` or `<scratch>/<run_id>` directory) (see §3.2.2).

3.2.2 Temporary execution directory

During a calculation, *Code_Saturne* may use a temporary directory for the compilation and the execution if such a "scratch" directory is defined in the GUI, by setting the `CS_SCRATCHDIR` environment variable, or in the `code_saturne.cfg` file. In this case, it is only at the end of the compilation that the result files are only copied at the end in the directory `RESU`. This is recommended if the compute environment includes different file-systems, some better suited to data storage, others to intensive I/O. If this is not the case, there is no point in running in a scratch directory rather than the results directory, as this incurs additional file copies.

If the environment variable `CS_SCRATCHDIR` is defined, its value has priority over that defined in the preference file so if necessary, it is possible to define a setting specific to a given run using this mechanism.

WARNING: in case of an error, the temporary directories are not deleted after a calculation, so that they may be used for debugging. They may then accumulate and may hinder the correct operation of the machine.

It is therefore essential to remove them regularly.

3.2.3 Execution modes

As explained before, *Code_Saturne* is composed of two main modules, the Preprocessor and the Solver. The Preprocessor reads the meshes. The resulting data is transferred to the Solver through specific files, named `mesh_input`, or placed in a directory of that name when multiple meshes are imported. In a standard calculation, the files are not copied from the temporary execution directory to the results directory, as they have no interest for data analysis, and are considered "internal" files, whose format or contents is not guaranteed not to change between *Code_Saturne* versions.

Yet, the Preprocessor does not run in parallel and may require a large amount of memory. The launch scripts therefore allows specifically choosing which modules to run, either through the GUI or through the `cs_user_scripts.py` file:

If a mesh_input file or directory is defined (which may be either a mesh_input from a previous Preprocessor run or a mesh_output from a previous solver run), the script will copy or link it to the execution directory, and the Preprocessor will not be rerun.

If `domain.exec_kernel = False`, the Solver will not be run. This is useful when only the mesh import stage is required.

In a similar manner, the Solver accepts several command-line options relative to execution mode, notably `domain.solver_args = '--preprocess'` or `'--quality'`, restricting the run to the preprocessing stages, or preprocessing stages augmented by mesh quality criteria computation. Whenever the preprocessing stages defined lead to an effective mesh modification, a `mesh_output` file is produced, which can be used directly as an input for a successive calculation.

The GUI presents the range of options in the form of four execution modes:

- **mesh import**: the Preprocessor is run to transform one or more meshes into an internal `mesh_input` file (or directory in case of multiple meshes).
- **mesh preprocessing**: the Solver is run in preprocessing mode, so as to handle all mesh modification operations, such as joining, periodicity, smoothing, *etc.* If a `mesh_input` file or directory is provided, it is used directly; otherwise, mesh import is run first.
- **mesh quality criteria**: similar to preprocessing, with the addition of mesh quality criteria computation, and post-processing output of those criteria. Some additional mesh consistency checks are also run.
- **standard**: this includes preprocessing, followed by a standard computation.

Note that to allow preprocessing in multiple passes, all defined preprocessing operations are run even on previously preprocessed meshes. In most cases, those will not produce additional changes (such as joining already joined meshes), but in the case of mesh smoothing, they might lead to small changes. So when using a previously preprocessed mesh it is recommended not to define any preprocessing operations, so as to skip the preprocessing stage.

It is encouraged to separate the preprocessing and calculation runs, as this not only speeds up calculations, but also ensures that the mesh is identical, regardless of the architecture or number of processors it is run on. Indeed, when running the same pre-processing stages such as mesh joining on a different machine or a different number of processors, very minor floating-point truncation errors may lead to very slightly different preprocessed meshes.

Note also that mesh partitioning is done directly by the Solver. Depending on the partitioning algorithm used, a partition map (`partition_output/domain_number_*`) may be output, allowing the use of the same partitioning in future calculations. By default, this file is output when using graph-based partitioners, which may use randomization and do not guarantee a reproducible output, and is not output when using a deterministic space-filling curve based partitioning.

If the code was built only with a serial partitioning library, graph-based partitioning may best be run in a serial pre-processing stage. In some cases, serial partitioning might also provide better partitioning quality than parallel partitioning, so if both are available, comparing the performance of the code may be worthwhile, at least for calculations expected to run for many iterations.

3.2.4 Environment variables

Setting a few environment variables specific to *Code_Saturne* allows modifying its default behaviour. The environment variables used by *Code_Saturne* are described here:

`CS_SCRATCHDIR`

Allows defining the execution directory (see §3.2.2), overriding the default path or settings from the global or user `code_saturne.cfg`.

`CS_MPIEXEC_OPTIONS`

This variable allows defining extra arguments to be passed to the MPI execution command by the run scripts. If this option is defined, it will have priority over the value defined in the preference file (or by computed defaults), so if necessary, it is possible to define a setting specific to a given run using this mechanism. This may be useful when tuning the installation to a given machine, for example experimenting MPI mapping and "bind to core" features.

3.2.5 Interactive modification of selected parameters

During a calculation, it is possible to change the limit time step number (`ntmabs`) specified through the GUI or in `cs_user_parameters.f90`. To do so, a file named `control_file` must be placed in the execution directory (see §3.2.2). The existence of this file is checked at the beginning of each time step.

To change the maximum number of time steps, this file must contain a line indicating the value of the new limit number of time steps.
If this new limit has already been reached, *Code_Saturne* will stop properly at the end of the current time step (the results and restart files will be written correctly).
This procedure allows the user to stop a calculation in a clean and interactive way whenever they wish.

The `control_file` may also contain a few other commands, allowing the user to force checkpointing or postprocessing at a given time step or physical time, or to force an update of log and `listing` files. The following commands are available (using the common notations <> to indicate a required argument, [] to indicate an optional argument).

`max_time_step`
`max_time_value`
`max_wall_time`
	<time_step_number>
	<time_value>
	<wall_time>

`checkpoint_time_step`
`checkpoint_time_value`
`checkpoint_wall_time`
`checkpoint_time_step_interval`
`checkpoint_time_value_interval`
`checkpoint_wall_time_interval`
	<time_step_number>
	<time_value>
	<wall_clock_time>
	<time_step_interval>
	<time_interval>
	<wall_time_interval>

`control_file_wtime_interval`	<wall_time_interval>
`flush`	[time_step_number]
`postprocess_time_step` `postprocess_time_value`	<time_step_number> [writer_id] <time_step_value> [writer_id]
`time_step_limit`	<time_step_count>

The `time_step_limit` differs from the `max_time_step` command, in the sense that it allows reducing the maximum number of time steps, but not increasing it. Also, in the case of a restart, it refers to the number of additional time steps, not to the number of absolute time steps.

Note that for the `postprocess_time_*` options, the last argument (`writer_id` is optional. If not defined, or 0, postprocessing is activated for all writers; if specified, only the writer with the specified id is affected. Also, postprocessing output by one ore more writers at a future time step may be cancelled using the negative value of that time step.

For the `flush` option, the time step is also optional. If not specified, logs and time plots are updated at the beginning of the next time step. Also, if the `control_file` is empty (such as when created by the `touch control_file` command on Unix/Linux systems, a `flush` request for the next time step.

Multiple entries may be defined in this file, with one line per entry.

3.3 Case preparer

The case preparer command `code_saturne create` automatically creates a study directory according to the typical architecture and copies and pre-fills an example of calculation launch script.

The syntax of `code_saturne create` is as follows:

`code_saturne create --study STUDY CASE_NAME1 CASE_NAME2...`
creates a study directory `STUDY` with case subdirectories `CASE_NAME1` and `CASE_NAME2`... If no case name is given, a default case directory called `CASE1` is created.

`code_saturne create --case Flow3 --case Flow4`
executed in the directory `STUDY` adds the case directories `Flow3` and `Flow4`. Whenever multiple cases are created simultaneously, it is assumed they may be coupled, so toplevel `runcase` and `coupling_parameters.py` files and `RESU_COUPLING` directory are also created.

In the directory `DATA`, the `code_saturne create` command places a subdirectory `REFERENCE` containing examples of thermochemical data files used for pulverised coal combustion, gas combustion, electric arcs, or a meteo profile. The file to be used for the calculation must be copied directly in the `DATA` directory and its name may either be unchanged, or be referenced using the GUI or using the `usppmo` subroutine in `cs_user_parameters.f90`. As a rule of thumb, all files in `DATA` except for `SaturneGUI` are copied, but subdirectories are not.
The `code_saturne create` command also places in the directory `DATA` the launch script for the Graphical User Interface: `SaturneGUI`.

In the directory `SRC`, the `code_saturne create` command creates a subdirectory `REFERENCE` containing all the available user subroutines, and the subdirectory `EXAMPLES` containing examples of user subroutines. Only the user subroutines placed directly under the directory `SRC` will be considered. The others will be ignored.

In the directory `SCRIPTS`, the `code_saturne create` command copies an example of the launch script: `runcase`. The XML file may be specified in the script (see §3.7), and using the GUI sets it automatically.

3.4 Supported mesh and post-processing output formats

Code_Saturne supports multiple mesh formats, all of these having been requested at some time by users or projects based on their meshing or post-processing tools. All of these formats have advantages and disadvantages (in terms of simplicity, functionality, longevity, and popularity) when compared to each other. The following formats are currently supported by *Code_Saturne*:

- SIMAIL (NOPO)
- I-deas universal
- MED
- CGNS
- EnSight 6
- EnSight Gold
- GAMBIT neutral
- Gmsh
- STAR-CCM+
- Catalyst (co-processing)

These formats are described in greater detail in the following sections. Unless a specific option is used, the Preprocessor determines the mesh format directly from the file suffix: *".case"* for EnSight (6 or Gold), *".ccm"* for STAR-CCM+, *".cgns"* for CGNS, *".des"* for SIMAIL, *".med"* for MED, *".msh"* for Gmsh, *".neu"* for GAMBIT neutral, *".unv"* for I-deas universal.

Note that the preprocessor can read gzipped mesh files directly (for Formats other than MED or CGNS, which use specific external libraries) on most machines.

3.4.1 Formats supported for input

3.4.1.1 NOPO/SIMAIL (INRIA/Distene)

This format is output by SIMAIL, which was used heavily at EDF until a few years ago. *Code_Saturne* does not currently handle cylindrical or spherical coordinates, but it seems that SIMAIL always outputs meshes in Cartesian coordinates, even if points have been defined in another system. Most "classical" element types are usable, except for pyramids.

Note that depending on the architecture on which a file was produced by SIMAIL[4], it may not be directly readable by SIMAIL on a different machine, while this is not a problem for the Preprocessor, which automatically detects the byte ordering and the 32/64 bit variant and adjusts accordingly.

Default extension:	`.des`
File type:	semi-portable "Fortran" binary (IEEE integer and floating-point numbers on 4 or 8 bytes, depending on 32 or 64 bit SIMAIL version, bytes also ordered based on the architecture)
Surface elements:	triangles, quadrangles (+ volume element face references)
Volume elements:	tetrahedra, prisms, hexahedra
Zone selection:	element face references and volume sub-domains (interpreted as numbered groups)
Compatibility:	all files of this type as long as the coordinate system used is Cartesian and not cylindrical or spherical
Documentation:	Simail user documentation and release notes or MODULEF documentation: http://www-rocq.inria.fr/modulef Especially: http://www-rocq.inria.fr/modulef/Doc/FR/Guide2-14/node49.html

3.4.1.2 I-deas universal file

This format was very popular in the 1990's and early 2000's, and though the I-deas tool has not focused on the CFD (or even meshing) market since many years, it is handled (at least in part) by many tools, and may be considered as a major "legacy" format. It may contain many different datasets, relative to CAD, meshing, materials, calculation results, or part representation. Most of these datasets are ignored by *Code_Saturne*, and only those relative to vertex, element, group, and coordinate system definitions are handled.

This format's definition evolves with I-deas versions, albeit in a limited manner: some datasets are declared obsolete, and are replaced by others, but the definition of a given dataset type is never modified. Element and Vertex definitions have not changed for many years, but group definitions have gone through several dataset variants through the same period, usually adding minor additional group types not relevant to meshing. If one were to read a file generated with a more recent version of I-deas for which this definitions would have changed with no update in the Preprocessor, as the new dataset would be unknown, it would simply be ignored.

[4] "little endian" on Intel or AMD processors, or "big endian" on most others, and starting with SIMAIL 7, 32-bit or 64-bit integer and floating-point numbers depending on architecture

Note that this is a text format. Most element types are handled, except for pyramids.

Default extension:	`.unv`
File type:	text
Surface elements:	triangles, quadrangles
Volume elements:	tetrahedra, prisms, hexahedra
Zone selection:	colors (always) and named groups
Compatibility:	I-deas (*Master Series* 5 to 9, *NX Series* 10 to 12) at least
Documentation: https://docs.plm.automation.siemens.com/tdoc/nx/10/nx_help/#uid:index_advanced:xid602249:id625716:id625821	Online I-deas NX Series documentation, and

3.4.1.3 GAMBIT neutral

This format may be produced by Ansys FLUENT's GAMBIT meshing tool. As this tool does not export meshes to other formats directly handled by the Preprocessor (though FLUENT itself may export files to the CGNS or I-deas universal formats), it was deemed useful to enable the Preprocessor to directly read files in GAMBIT neutral format.

Note that this is a text format. "Classical" element types are usable.

Default extension:	`.neu`
File type:	text
Surface elements:	triangles, quadrangles
Volume elements:	tetrahedra, pyramids, prisms, hexahedra
Zone selection:	boundary conditions for faces, element groups for cells (interpreted as named groups)
Documentation:	GAMBIT on-line documentation

3.4.1.4 EnSight 6

This format is used for output by the Harpoon meshing tool, developed by Sharc Ltd (also the distributor of EnSight for the United Kingdom). This format may represent all "classical" element types.

Designed for post processing, it does not explicitly handle the definition of surface patches or volume zones, but allows the use of many *parts* (i.e. groups of elements) which use a common vertex list. A possible convention (used at least by Harpoon) is to add surface elements to the volume mesh, using one *part* per group. The volume mesh may also be separated into several *parts* so as to identify different zones. As *part* names may contain up to 80 characters, we do not transform them into groups (whose names could be unwieldy), so we simply convert their numbers to group names.

Also note that files produced by Harpoon may contain badly oriented prisms, so the Preprocessor orientation correction option (`--reorient`) may must be used. Meshes built by this tool also contain hanging nodes, with non-conforming elements sharing some vertices. Mesh joining must thus also be used, and is not activated automatically, as the user may prefer to specify which surfaces should be joined, and which ones should not (*i.e.* to conserve thin walls).

Default extension:	`.case`
File type:	text file (extension *.case*), and text, binary, or Fortran binary file with (*.geo* extension), describing integers and floats in the IEEE format, using 32 bits
Surface elements:	triangles, quadrangles
Volume elements:	tetrahedra, pyramids, prisms, hexahedra
Zone selection:	part numbers interpreted as numbered groups
Compatibility:	All files of this type
Documentation:	on-line documentation, also available at: www3.ensight.com/EnSight10_Docs/UserManual.pdf

3.4.1.5 Gmsh

This format is used by the free Gmsh tool. This tool has both meshing and post-processing functionality, but *Code_Saturne* only imports the meshes.

Note that some meshes produced by Gmsh man contain some badly oriented elements, so the Preprocessor's `-reorient` option may be necessary.

The Preprocessor handles versions 1 and 2 of this array. In version 1, two labels are associated with each element: the first defines the element's physical entity number, the second defines its elementary entity number. Using version 2, it is possible to associate an arbitrary number of labels with each element, but files produced by Gmsh use 2 labels, with the same meanings as with version 1.

The decision was taken to convert physical entity numbers to groups. It is possible to build a mesh using Gmsh without defining any physical entities (in which case all elements will belong to the same group, but the Gmsh documentation clearly says that geometric entities are to be used so as to group elementary entities having similar "physical" meanings.

To obtain distinct groups with a mesh generated by Gmsh, it is thus necessary for the user to define physical entities. This requires an extra step, but allows for fine-grained control over the groups associated with the mesh, while using only elementary entities could lead to a high number of groups.

Default extension:	`.msh`
File type:	text or binary file
Surface elements:	triangles, quadrangles
Volume elements:	tetrahedra, pyramids, prisms, hexahedra
Zone selection:	physical entity numbers interpreted as numbered groups
Compatibility:	all files of this type
Documentation:	included documentation, also available at: http://www.geuz.org/gmsh

3.4.2 Formats supported for input or output

3.4.2.1 EnSight Gold

This format may represent all "classical" element types, as well as arbitrary polygons and convex polyhedra.

This format evolves slightly from one EnSight version to another, keeping backwards compatibility. For example, polygons could not be used in the same *part* as other element types prior to version 7.4, which removed this restriction and added support for polyhedra. Version 7.6 added support for material type definitions.

This format offers many possibilities not used by *Code_Saturne*, such as defining values on part of a mesh only (using "undefined" marker values or partial values), assigning materials to elements, defining rigid motion, or defining per-processor mesh parts with ghost cells for parallel runs. Note

that some libraries allowing direct EnSight Gold support do not necessarily support the whole format specification. Especially, VTK does not support material types. Also, both EnSight Gold (8.2 and above) and VTK allow for automatic distribution, reducing the usefulness of pre-distributed meshes with per-processor files.

Note than when using ParaView, if multiple parts (i.e. meshes) are present in a give case, using the "Extract Blocks" filter is required to separate those parts and obtain a proper visualization, unless the `separate_meshes` writer option is used. The VisIt software does not seem to handle multiple parts in an EnSight case, so different meshes must be assigned to different *writers*, or the `separate_meshes` option must be used (see §**??**) when using this tool.

This format may be used as an input format, similar to EnSight 6. Compared to the latter, each *part* has its own coordinates and vertex connectivity; hence as a convention, we consider that surface or volume zones may only be considered to be part of the same mesh if the file defines vertex IDs (which we consider to be unique vertex labels). In this case, *part* numbers are interpreted as group names. Without vertex IDs, only one part is read, and no groups are assigned.

Default extension:	directory {*case_name*}`.ensight`, containing a file with the `.case` extension
File type:	multiple binary or text files
Surface elements:	triangles, quadrangles, polygons
Volume elements:	tetrahedra, pyramids, prisms, hexahedra, convex polyhedra
Zone selection:	possibility of defining element materials (not used), or interpret part number as group name if vertex IDs are given
Compatibility:	files readable by EnSight 7.4 to 10.0, as well as tools based on the VTK library, especially ParaView (`http://www.paraview.org`)
Documentation:	online documentation, also available at: `www3.ensight.com/EnSight10_Docs/UserManual.pdf`

3.4.2.2 MED

Initially defined by EDF R&D, this format (*Modèle d'échanges de Données*, or *Model for Exchange of Data*) has been defined and maintained through a MED working group comprising members of EDF R&D and CEA. This is the reference format for the *S*ALOME environment. This format is quite complete, allowing the definition of all "classical" element types, in nodal or descending connectivity. It may handle polygonal faces and polyhedral cells, as well as the definition of structured meshes.

This format, which requires a library also depending on the free HDF5 library, allows both for reading and writing meshes with their attributes ("families" of group combinations), as well as handling calculation data, with the possibility (unused by *Code_Saturne*) of defining variables only on a subset ("profile") of a mesh.

The MED library is available under a LGPL license, and is even packaged in some Linux distributions (at least Debian and Ubuntu). *Code_Saturne* requires at least MED 3.0.2, which in turn requires HDF5 1.8. This format is upwards-compatible with MED 2.3, so old files in that version of the format may be read, though not output.

Default extension:	`.med`
File type:	portable binary, based on the HDF5 library (http://www.hdfgroup.org/HDF5/index.html)
Surface elements:	triangles, quadrangles, simple polygons
Volume elements:	tetrahedra, pyramids, prisms, hexahedra, simple polyhedra
Zone selection:	element families (*i.e.* colors and groups)
Input compatibility:	MED 2.3, 3.0 to 3.3 (only unstructured nodal connectivity is supported)
Output compatibility:	MED 3.0 and above
Documentation:	on-line documentation. Download link at http://files.salome-platform.org/Salome/other/med-3.3.1.tar.gz

3.4.2.3 CGNS

Promoted by organizations including the AIAA, NASA, Boeing Commercial, ANSYS, Airbus, ONERA, SAFRAN, ANSYS, Pointwise, Inc., Numeca, and others, this format(*CFD General Notation System*) is quite well established in the world of CFD. The concept is similar to that of MED, with a bigger emphasis on normalization of variable names or calculation information, and even richer possibilities. Contrary to MED, the first version of this format was limited to multi-bloc structured meshes, unstructured meshes having been added in CGNS 2.

Slightly older than MED, this library was free from the start, with a good English documentation, and is thus much better known. It is more focused on CFD, where MED is more generic. A certain number of tools accompany the CGNS distribution, including a mesh visualizer (which does not handle polygonal faces although the format defines them), and an interpolation tool.

Code_Saturne should be able to read almost any mesh written in this format, though meshes with over-set interfaces may not be usable for a calculation (calculations with over-set interfaces may be possible in the context of coupling *Code_Saturne* with itself but with two separate meshes). Other (abutting) interfaces are not handled automatically (as there are at least 3 or 4 ways of defining them, and some mesh tools do not export them[5]), so the user is simply informed of their existence in the Preprocessor's log file, with a suggestion to use an appropriate conformal joining option. Structured zones are converted to unstructured zones immediately after being read.

Boundary condition information is interpreted as groups with the same name. The format does not yet provide for selection of volume elements, as only boundary conditions are defined in the model (and can be assigned to faces in the case of unstructured meshes, or vertices in any case). Note that boundary conditions defined at vertices are not ignored by the Preprocessor, but are assigned to the faces of which all vertices bear the same condition.[6]

The Preprocessor also has the capability of building additional volume or surface groups, based on the mesh sections to which cells or faces belong. This may be activated using a sub-option of the mesh selection, and allows obtaining zone selection information from meshes that do not have explicit boundary condition information but that are subdivided in appropriate zones or sections (which depends on the tool used to build the mesh).

When outputting to CGNS, an unstructured connectivity is used for the calculation domain, with no face joining information or face boundary condition information.[7]

Though many tools support CGNS, that support was often quite disappointing, at least for unstructured meshes, until recent years. Some editors seem to use different means to mark zones to associate

[5]For example, ICEM CFD can join non-conforming meshes, but it exports joining surfaces as simple boundary faces with user-defined boundary conditions.

[6]If one of a face's vertices does not bear a boundary condition, that condition is not transferred to the face.

[7]Older versions of the documentation specified that a field must be defined on all elements of a zone, so that adding faces on which to base boundary conditions to a volume mesh would have required also defining volume fields on these faces. More recent versions of the documentation make it clear that a field must be defined on all elements of maximum dimension in a zone, not on all elements.

with boundary conditions than the ones recommended in the CGNS documentation, and some behaviours are worse. Also, many readers do not allow the user to choose between multiple CGNS bases (meshes in the *Code_Saturne* sense), so when outputting to CGNS, it may be necessary to output each post-processing mesh using a separate output.

Default extension:	`.cgns`
File type:	portable binary (uses the ADF library specific to CGNS, or HDF5)
Surface elements:	triangles, quadrangles, simple polygons
Volume elements:	tetrahedra, pyramids, prisms, hexahedra, simple polyhedra
Zone selection:	Surface zone selection using boundary conditions, no volume zone selection, but the Preprocessor allows creation of groups associated to zones or sections in the mesh using mesh selection sub-options
Input compatibility:	CGNS 2.5 or CGNS 3.1 and above
Output compatibility:	CGNS 3.1 and above
Documentation:	See CGNS site: http://www.cgns.org

3.4.2.4 STAR-CCM+

This polyhedral format is the current CD-Adapco (SIEMENS) format, and is based on CD-Adapco's libccmio, which is based on ADF (the low-level file format used by CGNS prior to the shift to HDF5). libccmio comes with a version of ADF modified for performance, but also works with a standard version from CGNS.

Currently, geometric entity numbers are converted to numbered groups, with the corresponding names printed to the Preprocessor log. Depending on whether the names were generated automatically or set by the user, it would be preferable to use the original group names rather than base their names on their numbers.

This format may also be used for output, though its limitations make this a less general solution than other output formats: only 3D meshes are handled, though values can be output on boundary face regions (which may not overlap). As such, to ensure consistency, output using this format is limited as follows:

- output of the full volume mesh and cell or vertex data on that mesh is handled normally.
- output of the full surface mesh and per face data on that mesh handled normally, only if output of the full volume mesh to this format is also enabled. It is ignored otherwise.
- output of sub-meshes or meshes built during the preprocessing stage and all other data is ignored.

As such, this formal may be useful for interoperability of data with a CCMIO-based tool-chain, but simultaneously using another output format to visualize possible error output is recommended.

The CCMIO library is distributed by CD-Adapco to its clients upon demand.

Use of the CGNS format should be preferred to this format when possible, and CGNS output is available in Star-CCM+ since version 12.06 at least.

Default extension:	`.ccm`
File type:	binary file using modified ADF library.
Surface elements:	polygons
Volume elements:	polyhedra
Zone selection:	named face and cell sets (interpreted as numbered groups, with names appearing in log)
Compatibility:	all files of this type?
Documentation:	documentation and source code provided by CD-Adapco

3.4.3 Formats supported for output only

3.4.3.1 Catalyst

This is not a "true" output format in the sense that output is not written directly to file, but is exported to the Catalyst co-processor. In turn, this co-processor will execute operations based on a special ParaView Python script, and directly generate output such as images or movies.

Co-processing scripts may be generated under ParaView 4.2 or above, using initial output in another format (such as EnSight Gold). With ParaView 4.2 to 5.4, this required activating the CoProcessing plugin. With ParaView 5.5, a "Generate Script" item can be found directly under the "Catalyst" menubar item.

A *Code_Saturne* postprocessing writer will try to read a script named `<writer_name>.py`, which should be places in a case's `DATA` directory. Using ParaView 5.5 or above, in the "Name Simulation Inputs" stage of the Catalyst script generator, the "simulation name" field should be set to the same name as the script (i.e. *writer_name*).

Note that this output is heavily dependent on ParaView. Some operations may work very well, while other, similar operations may fail.

Default extension:	not applicable
File type:	co-processing
Surface elements:	triangles, quadrangles, polygons
Volume elements:	tetrahedra, pyramids, prisms, hexahedra, convex polyhedra
Compatibility:	Catalyst from ParaView 4.2 or above (version 5.4 or above recommended)
Documentation:	online documentation and Wiki, at: http://paraview.org/Wiki/Main_Page

3.4.4 Meshing tools and associated formats

Most often, the choice of a mesh format is linked to the choice of a meshing tool. Still, some tools allow exporting a mesh under several formats handled by *Code_Saturne*. This is the case of FLUENT and ICEM CFD, which can export meshes to both the I-deas universal and CGNS formats (FLUENT's GAMBIT is also able to export to I-deas universal format).

Traditionally, users exported files to the I-deas universal format, but it does not handle pyramid elements, which are often used by these tools to transition from hexahedral to tetrahedral cells in the case of hybrid meshes. The user is encouraged to export to CGNS, which does not have this limitation.

Tools related to the SALOME platform should preferably use SALOME's native MED format.

3.4.5 Meshing remarks

WARNING: Some turbulence models ($k-\varepsilon$, $R_{ij}-\varepsilon$ SSG, ...) used in *Code_Saturne* are "High-Reynolds" models. Therefore the size of the cells neighbouring the wall must be greater than the thickness of the viscous sub-layer (at the wall, $y^+ > 2.5$ is required, and $30 < y^+ < 100$ is preferable). If the mesh does not match this constraint, the results may be false (particularly if thermal phenomena are involved). For more details on these constraints, see the keyword `iturb`.

3.5 Preprocessor command line options

The main options are:

- `--help`: provides a summary of the different command line options

- `<mesh>`: the last argument is used to specify the name of the mesh file. The launch script automatically calls the Preprocessor for every mesh in the `MESHES[]` list specified by the user.
- `--reorient`: attempts to re-orient badly-oriented cells if necessary to compensate for mesh-generation software whose output does not conform to the format specifications.

3.6 Solver command line options

In the standard cases, the compilation of *Code_Saturne* and its execution are entirely controlled by the launch script. The potential command line options are passed through user modifiable variables at the beginning of the `cs_user_scripts.py` file (this file may be copied from the `DATA/REFERENCE` to the `DATA` and edited). This way, the user only has to fill these variables and doesn't need to search deep in the script for the Solver command line. For more advanced usage, the main options are described below:

- `--app-name`: specifies the application name. This is useful only in the case of code coupling, where the application name is used to distinguish between different code instances launched together.
- `--mpi`: specifies that the calculation is running with MPI communications. The number of processors used will be determined automatically by the Solver. With most MPI implementations, the code will detect the presence of an MPI environment automatically, and this option is redundant. It is only kept for the rare case in which the MPI environment might not be detected.
- `--preprocess`: triggers the preprocessing-only mode. The code may run without any Interface parameter file or any user subroutine. Only the initial operations such as mesh joining and modification are executed.
- `-q` or `--quality`: triggers the verification mode. The code may run without any Interface parameter file or any user subroutine. This mode includes the preprocessing stages, and adds elementary tests:
  - the quality criteria of the mesh are calculated (non-orthogonality angles, internal faces offset, ...) and corresponding visualizable post-processing output is generated.
  - a few additional mesh consistency tests are run.
- `--benchmark`: triggers the benchmark mode, for a timing of elementary operations on the machine. A secondary option `--mpitrace` can be added. It is to be activated when the benchmark mode is used in association with an MPI trace utility. It restricts the elementary operations to those implying MPI communications and does only one of each elementary operation, to avoid overfilling the MPI trace report.
  This command is to be placed in the
  texttdomain.solver_args variable in the `cs_user_scripts.py` file to be added automatically to the Solver command line.
- `--log n`: specifies the destination of the output for a single-processor calculation or for the processor of rank 0 in a parallel calculation.
  `n=0`: output directed towards the standard output
  `n=1`: output redirected towards a file `listing` (default behaviour)
  This option can be specified in the `domain.logging_args` field of the user script.
- `--logp n`: specifies the destination of the output for the processors of rank 1 to $N-1$ in a calculation in parallel on N processors (*i.e.* the redirection of all but the first processor).
  `n=-1`: no output for the processors of rank 1 to $N-1$ (default behaviour).

`n=0`: no redirection. Every processor will write to the standard output. This might be useful in case a debugger is used, with separate terminals for each processor.

`n=1`: one file for the output of each processor. The output of the processors of rank 1 to $N-1$ are directed to the files `listing_r0001` to `listing_r`$N-1$. This option can be specified in the `domain.logging_args` field of the user script.

- `-p <filename>` or `--param <filename>`: specifies the name of the GUI parameter file to use for the calculation.
  The value of `<filename>` is to be defined by the `--param` option of `code_saturne run`, either directly or in the standard `runcase` script (the file will be searched for in the `data` directory, though an absolute path name may also be defined).

- `-h` or `--help`: displays a summary of the different command line options.

3.7 Launch scripts

The case preparer command `code_saturne create` places an example of launch script, `runcase`, in the `SCRIPTS` directory. This script is quite minimalist and is known to work on every architecture *Code_Saturne* has been tested on. If a batch system is available, this script will contain options for batch submission. The script will then contain a line setting the proper `PYTHONPATH` variable for *Code_Saturne* to run. Finally, it simply contains the `code_saturne run` command, possible with a `--param` option when a parameters file defined by the GUI is used. Other options recognized by `code_saturne run` may be added.

In the case of a coupled calculation, this script also exists, and may be used for preprocessing stages, but an additional `runcase` and accompanying `coupling_parameters.py` file is added in the directory above the coupled case directories, and may be used to define the list of coupled cases, as well as global options, such as MPI options of the temporary execution directory.

When not using the GUI, or if additional options must be accessed, the `cs_user_scripts.py` file may be copied from the `DATA/REFERENCE` to the `DATA` and edited. This file contains several Python functions:

- `define_domain_parameter_file` allows defining the choice of a parameters file produced by the GUI. This is generally not useful, as the parameters file may be directly defined in `runcase`, or passed as an option to `code_saturne run`, but could be useful when running more complex parametric scripts, and is provided for the sake of completeness.

- `define_domain_parameters` allows defining most parameters relative to case execution for the current domain, including advanced options not accessible through the GUI. This function is the most important one in the user scripts file, and contains descriptions of the various options. Note that in most examples, setting of options is preceded by a `if domain.param == None:` line, ensuring the settings are only active if no GUI-defined parameters file is present. This is used to prevent accidental override of parameters defined by the GUI: parameters defined through the user script have priority over the GUI parameters file, so if both are used, these tests may be removed for parameters which should be defined through user scripts.

3.8 Graphical User Interface

A Graphical User Interface is available with *Code_Saturne*. This Interface creates or reads an XML file according to a specific *Code_Saturne* schema which is then interpreted by the code.

In version 5.3.3, the Graphical Interface manages calculation parameters, standard initialisation values and boundary conditions for standard physics, pulverised fuel combustion, gas combustion, atmospheric flows, Lagrangian module, electrical model, compressible model and radiative transfers (user subroutines can still be completed though).

The Interface is optional. Every data that can be specified through the Interface can also be specified in the user subroutines. In case of conflict, all calculation parameters, initialisation value or boundary condition set directly in the user subroutines will prevail over what is defined by the Interface. However, it is no longer necessary to redefine everything in the user subroutines. Only what was not set or could not be set using the Graphical Interface should be specified.

WARNING: There are some limitations to the changes that can be made between the Interface and the user routines. In particular, it is not possible to specify a certain number of solved variables in the Interface and change it in the user routines (for example, it is not possible to specify the use of a $k-\varepsilon$ model in the Interface and change it to $R_{ij}-\varepsilon$ in `cs_user_parameters.f90`, or to define additional scalars in `cs_user_parameters.f90` with respect to the Interface). Also, all boundaries should be referenced in the Interface, even if the associated conditions are intended to be modified in `cs_user_boundary_conditions`, and their nature (entry, outlet, wall[8], symmetry) should not be changed.

For example, in order to set the boundary conditions of a calculation corresponding to a channel flow with a given inlet velocity profile, one should:
- set the boundary conditions corresponding to the wall and the output using the Graphical Interface
- set a dummy boundary condition for the inlet (uniform velocity for instance) - set the proper velocity profile at inlet in `cs_user_boundary_conditions`. The wall and output areas must not appear in `cs_user_boundary_conditions`. The dummy velocity entered in the Interface will not be taken into account.

The Graphical User Interface is launched with the `./SaturneGUI` command in the directory `DATA`. The first step is then to load an existing parameter file (in order to modify it) or to open a new one. The headings to be filled for a standard calculation are the following:

- Identity and paths: definition of the calculation directories (STUDY, CASE, DATA, SRC, SCRIPTS, MESH).
- Calculation environment: definition of the mesh file(s), stand-alone execution of the Preprocessor module (used by the Interface to get the groups of the boundary faces).
- Thermophysical models: physical model, ALE mobile mesh features, turbulence model, thermal model, coupling with SYRTHES.
- Additional scalars: definition, initialisation of the scalars, and physical characteristics.
- Physical properties: reference pressure, fluid characteristics, gravity. It is also possible to write user laws for the density, the viscosity, the specific heat and the thermal conductivity in the interface through the use of a formulae interpreter.
- Volume conditions: initialisation of the variables, and definition of the zones where to apply head losses or source terms.
- Boundary conditions: definition of the boundary conditions for each variable. The colors of the boundary faces may be read directly from a "preprocessor.log*" files created by the Preprocessor or a "listing" file from a previous Solver run.
- Numerical parameters: number and type of time step, advanced parameters for the numerical solution of the equations.
- Calculation control: parameters concerning the time averages, time step, location of the probes where some variables will be monitored over time, definition of the frequency of the outputs in the calculation listing and in the chronological records and of the EnSight outputs. The item *Profiles* allows to save, with a given frequency, 1D profiles on an axis defined from two points provided by the user.

[8] Smooth and rough walls are considered to have the same nature

- Calculation management: management of the calculation restarts, updating of the launch script (temporary execution directory, parallel computing, user data or result files, ...) and interactive launch of the calculation.

The *Code_Saturne* tutorial [14] offers a step-by-step guidance to the setting up of some simple calculations with the *Code_Saturne* Interface.

To launch *Code_Saturne* using an XML parameter file, the name of the file must be given using the `--param` option of `code_saturne run` in the launch script (see §3.7). When the launch script is edited from the Interface (Calculation management → Prepare batch analysis), this option is set automatically.

3.9 User subroutines

3.9.1 Preliminary comments

The user can run the calculations with or without an interface, with or without the user subroutines. Without interface, some user subroutines are needed (see §3.2.1). With interface, all the user subroutines are optional.

The parameters can be read in the interface and then in the user subroutines. In the case that a parameter is specified in the interface and in a user subroutine, it is the value in the user subroutine that is taken into account. For this reason, all the examples of user subroutines are placed in the `EXAMPLES` directory by the case setup `code_saturne create` (and available subroutines in the directory `REFERENCE`).

3.9.2 Example routines

Some user subroutines may be used for many different user definitions. As including enough examples in those subroutines would make them very difficult to read, these routines provided as templates only, with separate examples in a case's `EXAMPLES` subdirectory of its `SRC` directory.

Example file names are defined by inserting the name of the matching example in the file name. For example, a basic example for `cs_user_boundary_conditions.f90` is provided in `cs_user_boundary_conditions-base.f90`, while an example dedicated to atmospheric flows is provided in cs_user_boundary_conditions-atmospheric.f90.

The user is encouraged to check what examples are available, and to study those that are relevant to a given setup.

Template user subroutines contain three sections the user may define, marked by the following strings:

- `INSERT_VARIABLE_DEFINITIONS_HERE`
- `INSERT_ADDITIONAL_INITIALIZATION_CODE_HERE`
- `INSERT_MAIN_CODE_HERE`

Comparing template and example files with a graphical file comparison tool should help the user highlights the matching sections from the examples, so it is recommended as good practice for those not already very familiar with those user subroutines.

3.9.3 Main variables

This section presents a non-exhaustive list of the main variables that may be encountered by the user. Most of them should not be modified by the user. They are calculated automatically from the data. However it may be useful to know what they represent. Developers can also refer to [11].

These variables are listed in the alphabetical index at the end of this document (see § 8).

The type of each variable is given: integer [i], real number [r], integer array [ia], real array [ra].

For a further detailed list of variables, one can refer to the dedicated `Doxygen` documentation.

3.9.3.1 Array sizes

For array sizes, please refer to the following `Doxygen` documentation:

- Mesh dimensions,
- General variable array dimensions,
- Specific variable array dimensions.

3.9.3.2 Geometric variables

The main geometric variables are available in most of the subroutines and directly accessible through arrays defined in the `mesh` module (i.e. `use mesh`). For further details, please refer to the following `Doxygen` documentation.

3.9.3.3 Physical variables

Almost all physical variables[9] can be accessed via the `cs_field` API and are available in all the subroutines as fields (either through their name or their id). The previous system, which used multi-dimensional arrays, has been progressively replaced by the `cs_field` API.

For a thorough description of the user management of all physical variables as well as the corresponding syntaxes between the `cs_field` API (both in C and Fortran) and the previous system, please refer to the dedicated `Doxygen` documentation.

Note that local arrays of values of physical variables, retrieved via the `cs_field` API, follow a naming convention, fully described at this page of the `Doxygen` documentation. It is highly recommended to follow this convention to ease the comprehension.

<u>About the solved variables</u>

The indexes allowing marking out the different solved variables (from 1 to `nvar`) are integers available in a "module" called `numvar`.

For example, `ipr` refers to the variable "pressure".

The list of integers referring to solved variables can be accessed through the following `Doxygen` documentation. These variable index-numbers can be used to retrieve the corresponding field indices (for instance, `ivarfl(ipr)` is the field index for the pressure), but also for some arrays of variable associated options (for instance, `visls0(itempk)` is the viscosity of the temperature).

To access the main solved variables, please refer to the following `Doxygen` documentation.

Concerning the solved scalar variables (apart from the variables pressure, k, ε, R_{ij}, ω, φ, $\overline{f}$, α, ν_t), the following is very important:

- The designation "scalar" refers to scalar variables which are solution of an advection equation, apart from the variables of the turbulence model (k, ε, R_{ij}, ω, φ, $\overline{f}$, α, ν_t): for instance the temperature, scalars which may be passive or not, "user" or not. The mean value of the square of the fluctuations of a "scalar" is a "scalar", too. The scalars may be divided into two groups:

[9] except some of the properties defined at the cell centers

`nscaus` "user" scalars and `nscapp` "specific physics" scalars, with `nscal=nscaus+nscapp`. `nscal` must be less than or equal to `nscamx`.

- The `j`th user scalar is, in the whole list of the `nscal` scalars, the scalar number `j`. In the list of the `nvar` solved variables, it corresponds to the variable number `isca(j)`.
- The `j`th scalar related to a specific physics is, in the whole list of the `nscal` scalars, the scalar number `iscapp(j)`. In the list of the `nvar` solved variables, it corresponds to the variable number `isca(iscapp(j))`.
- Apart from specific physics, the temperature (or the enthalpy) is the scalar number `iscalt` in the list of the `nscal` scalars. It corresponds to the variable number `isca(iscalt)`. if there is no thermal scalar, `iscalt` is equal to -1.
- A "user" scalar number `j` may represent the mean of the square of the fluctuations of a scalar `k` (*i.e.* the average $\overline{\varphi'\varphi'}$ for a fluctuating scalar φ). This can be made either *via* the interface or by declaring that scalar using `cs_parameters_add_variable_variance` in `cs_user_parameters.c` (if the scalar in question is not a "user" scalar, the selection is made automatically). For instance, if `j` and `k` are "user" scalars, the variable φ corresponding to `k` is the variable number `isca(k)=isca(iscavr(j))`.[10].

About the physical properties at the cell centers

To access the physical properties, please refer to the following Doxygen documentation. Some index numbers are also described in the physical properties numbering Doxygen documentation.

NOTE: VARIABLE PHYSICAL PROPERTIES

Some physical properties such as specific heat or diffusivity are often constant (choice made by the user). In that case, in order to limit the necessary memory, these properties are stored as a simple real number rather than in a domain-sized array of reals.

- This is the case for the specific heat C_p.
  - If C_p is constant, it can be specified in the interface or by indicating `icp=0` in `cs_user_parameters.f90`, and the property will be stored in the real number `cp0`.
  - If C_p is variable, it can be specified in the interface or by indicating `icp=1` in `cs_user_parameters.f90`. The code will then modify this value to make `icp` refer to the effective property field id corresponding to the specific heat, in a way which is transparent for the user. For each cell `iel`, the value of C_p can then be defined in `usphyv` in an array which pointer can be retrieved by calling `field_get_val_s(icp, cpro_cp)`.
- This is the same for the diffusivity K of each scalar `iscal`.
  - If k is constant, it can be specified in the interface or by calling `field_set_key_int(ivarfl(isca(iscal)), kivisl, -1)` in `cs_user_parameters.f90`, (in `usipsu`) and the property will be stored in the real number `visls0(iscal)`.
  - If k is variable, it can be specified in the interface or by calling `field_set_key_int(ivarfl(isca(iscal)), kivisl, 0)` in `cs_user_parameters.f90`, (in `usipsu`). The code will then modify this key value to make it refer to the effective field id corresponding to the diffusivity of the scalar `iscal`, in a way which is transparent for the user. For each cell `iel`, the value of k is then given in `usphyv` and stored in the field whose id is given by calling `field_set_key_int(ivarfl(isca(iscal)), kivisl, ...)`.

[10] It is really $\overline{\varphi'\varphi'}$, and not $\sqrt{\overline{\varphi'\varphi'}}$

Two other variables, `hbord` and `tbord`, should be noted here, although they are relatively local (they appear only in the treatment of the boundary conditions) and are used only by developers.

`hbord(nfabor)` [ra]: Array of the exchange coefficient for temperature (or enthalpy) at the boundary faces. The table is allocated only if `isvhb` is set to 1 in the subroutine `tridim` (which is note a user subroutine), which is done automatically, but only if the coupling with SYRTHES or the 1D thermal wall module are activated..

`tbord(nfabor)` [ra]: Temperature (or enthalpy) at the boundary faces[11]. The table is allocated only if `isvtb` is set to 1 in the subroutine `tridim` (which is note a user subroutine), which is done automatically but only if the coupling with SYRTHES or the 1D thermal wall module are activated..

Tables `hbord` and `tbord` are of size `nfabor`, although they concern only the wall boundary faces.

3.9.3.4 Variables related to the numerical methods

The main numerical variables and "pointers" are described in the `Doxygen` documentation below.

Boundary conditions

- `ifmfbr` and `isympa` arrays.
- `itrifb`, `itypfb` and `uetbor` arrays.

Distance to the wall

- `dispar` and `yplpar` arrays.

Pressure drops and porosity

- `icepdc`, `ckupdc` and `porosi` arrays as well as `ncepdc`ncepdc.

Mass sources

- `icetsm`, `itypsm` and `smacel` arrays as well as `ncetsm`.

Wall 1D thermal module

`nfpt1d` [i]: Number of boundary faces which are coupled with a wall 1D thermal module. See the user subroutine `cs_user_1d_wall_thermal.c`.

`ifpt1d` [ia]: Array allowing marking out the numbers of the `nfpt1d` boundary faces which are coupled with a wall 1D thermal module. The numbers of these boundary faces are given by `ifpt1d(ii)`, with 1⩽`ii`⩽`nfpt1d`. See the user subroutine `cs_user_1d_wall_thermal.c`.

[11] It is the physical temperature at the boundary faces, not the boundary condition for temperature. See [11] for more details on boundary conditions

`nppt1d` [ia]: Number of discretisation cells in the 1D wall for the `nfpt1d` boundary faces which are coupled with a 1D wall thermal module. The number of cells for these boundary faces is given by `nppt1d(ii)`, with 1⩽`ii`⩽`nfpt1d`. See the user subroutine `cs_user_1d_wall_thermal.c`.

`eppt1d` [ia]: Thickness of the 1D wall for the `nfpt1d` boundary faces which are coupled with a 1D wall thermal module. The wall thickness for these boundary faces is therefore given by `eppt1d(ii)`, with 1⩽`ii`⩽`nfpt1d`. See the user subroutine `cs_user_1d_wall_thermal.c`.

Others

`dt(ncelet)` [ra]: Value of the time step.

`ifmcel(ncelet)` [ia]: Family number of the elements. See note 1.

`s2kw(ncelet)` [ra]: Square of the norm of the deviatoric part of the deformation rate tensor ($S^2 = 2S^D_{ij}S^D_{ij}$). This array is defined only with the $k-\omega$ (SST) turbulence model.

`divukw` [ia]: Divergence of the velocity. More precisely it is the trace of the velocity gradient (and not a finite volume divergence term). In the cell `iel`, $div(\underline{u})$ is given by `divukw(iel1)`. This array is defined only with the $k-\omega$ SST turbulence model (because in this case it may be calculated at the same time as S^2)..

Note: boundary conditions

The **gradient** boundary conditions in *Code_Saturne* boil down to determine a value for the current variable Y at the boundary faces f_b, that is to say Y_{f_b}, value expressed as a function of $Y_{I'}$, value of Y in I', projection of the center of the adjacent cell on the straight line perpendicular to the boundary face and crossing its center:

$$Y_{f_b} = A^g_{f_b} + B^g_{f_b} Y_{I'}. \tag{1}$$

For a face `ifac`, the pair of coefficients $A^g_{f_b}$, $B^g_{f_b}$ is may be accessed using the `field_get_coefa_s` and `field_get_coefb_s` functions, replacing `s` with `v` for a vector.

The **flux** boundary conditions in *Code_Saturne* boil down to determine the value of the diffusive flux of the current variable Y at the boundary faces f_b, that is to say $D_{ib}\left(K_{f_b}, Y\right)$, value expressed as a function of $Y_{I'}$, value of Y in I', projection of the center of the adjacent cell on the straight line perpendicular to the boundary face and crossing its center:

$$D_{ib}\left(K_{f_b}, Y\right) = A^f_{f_b} + B^f_{f_b} Y_{I'}. \tag{2}$$

For a face `ifac`, the pair of coefficients $A^f_{f_b}$, $B^f_{f_b}$ may be accessed using the `field_get_coefaf_s` and `field_get_coefbf_s` functions, replacing `s` with `v` for a vector.

The **divergence** boundary conditions in *Code_Saturne* boil down to determine a value for the current variable Y (mainly the Reynolds stress components, the divergence $\underline{\mathrm{div}}\left(\underline{\underline{R}}\right)$ used in the calculation of the momentum equation) at the boundary faces f_b, that is to say Y_{f_b}, value expressed as a function of $Y_{I'}$, value of Y in I', projection of the center of the adjacent cell on the straight line perpendicular to the boundary face and crossing its center:

$$Y_{f_b} = A^d_{f_b} + B^d_{f_b} Y_{I'}. \tag{3}$$

For a face `ifac`, the pair of coefficients $A^d_{f_b}$, $B^d_{f_b}$ may be accessed using the `field_get_coefad_s` and `field_get_coefbd_s` functions, replacing `s` with `v` for a vector.

3.9.3.5 User arrays

Modules containing user arrays accessible from all user subroutines may be defined in the `user_modules.f90` file. This file is compiled before any other Fortran user file, to ensure modules may be accessed in other user subroutines using the `use <module>` construct. It may contain any routines or variables the user needs, and contains no predefined routines or variables (i.e. the only specificity of this file is that a file with this name is compiled before all others).

3.9.3.6 Parallelism and periodicity

Parallelism is based on domain partitioning: each processor is assigned a part of the domain, and data for cells on parallel boundaries is duplicated on neighbouring processors in corresponding "ghost", or "halo" cells (both terms are used interchangeably). Values in these cells may be accessed just the same as values in regular cells. Communication is only required when cell values are modified using values from neighbouring cells, as the values in the "halo" can not be computed correctly (since the halo does not have access to all its neighbours), so halo values must be updated by copying values from the corresponding cells on the neighbouring processor.

Compared to other tools using a similar system, a specificity of *Code_Saturne* is the separation of the halo in two parts: a standard part, containing cells shared through faces on parallel boundaries, and an extended part, containing cells shared through vertices, which is used mainly for least squares gradient reconstruction using an extended neighbourhood. Most updates need only to operate on the standard halo, requiring less data communication than those on the extended halos.

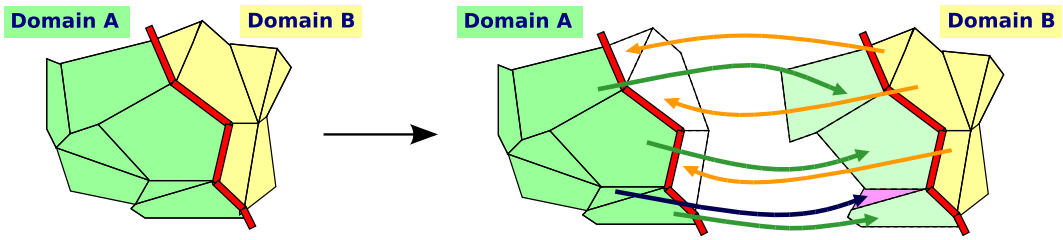

Figure 4: Parallel domain partitioning: halos

Periodicity is handled using the same halo structures as parallelism, with an additional treatment for vector and coordinate values: updating coordinates requires applying the periodic transformation to the copied values, and in the case of rotation, updating vector and tensor values also requires applying the rotation transformation. Ghost cells may be parallel, periodic, or both. The example of a pump combining parallelism and periodicity is given in Figure 5. In this example, all periodic boundaries match with boundaries on the same domain, so halos are either parallel or periodic.

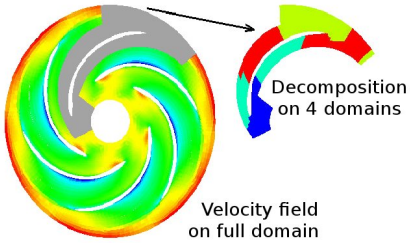

Figure 5: Combined parallelism and periodicity

Activation

Parallelism is activated by means of the GUI or of the launch scripts in the standard cases:

- On clusters with batch systems, the launching of a parallel run requires to complete the batch cards located in the beginning of `runcase` script, and set the number of MPI processes, or the numbers of physical nodes and processors per node (`ppn`) wanted. This can be done through the Graphical Interface or by editing the `runcase` file directly. The number of processors defined here will override the number defined through the GUI in a non-batch environment (so that studies defined on one environment may be migrated to larger compute resources easily), but it may be overridden by the `define_case_parameters` function from the `cs_user_scripts.py` file, or by setting the `n_procs_weight`, `n_procs_min`, and `n_procs_max` parameters for the different domains defined in `coupling_parameters.py`.
- On clusters with unsupported batch systems, `runcase` file may have to be modified manually. Please do not hesitate to contact the *Code_Saturne* support (saturne-support@edf.fr) so that these modifications can be added to the standard launch script to make it more general.
- A parallel calculation may be stopped in the same manner as a sequential one using the file `control_file` (see paragraph 3.2.5).
- The standard elements of information displayed in the listing (marked out with `'v '` for the min/max values of the variables), `'c '` for the data concerning the convergence and `'a '` for the values before clipping) are global values for the whole domain and not related to each processor.

User subroutines

The user can check in a subroutine

- that the presence of periodicity is tested with the variable `iperio` (=1 if periodicity is activated);
- that the presence of rotation periodicities is tested with the variable `iperot` (number of rotation periodicities);
- that running of a calculation in parallel is tested for with the variable `irangp` (`irangp` is worth -1 in the case of a non-parallel calculation and $p-1$ in the case of a parallel calculation, p being the number of the current processor)

Attention must be paid to the coding of the user subroutines. If conventional subroutines like `cs_user_parameters.f90` or `cs_user_boundary_conditions` usually do not cause any problem, some kind of developments are more complicated. The most usual cases are dealt with below.
Examples are given for the subroutine `cs_user_extra_operations`.

- **Access to information related to neighbouring cells in parallel and periodic cases**.
  When periodicity or parallelism are brought into use, some cells of the mesh become physically distant from their neighbours. Concerning parallelism, the calculation domain is split and distributed between the processors: a cell located at the "boundary" of a given processor may have neighbours on different processors.
  In the same way, in case of periodicity, the neighbouring cells of cells adjacent to a periodic face are generally distant.
  When data concerning neighbouring cells are required for the calculation, they must first be searched on the other processors or on the other edge of periodic frontiers. In order to ease the manipulation of these data, they are stored temporarily in virtual cells called "halo" cells, as can be seen in Figure 4. It is in particular the case when the following operations are made on a variable A:
  - calculation of the gradient of A (use of the subroutine `grdcel`);
  - calculation of an internal face value from the values of A in the neighbouring cells (use of `ifacel`).

The variable A must be exchanged before these operations can be made: to allow it, the subroutine `synsca` may be called.

- **Global operations in parallel mode**.
  In parallel mode, the user must pay attention when performing global operations. The following list is not exhaustive:

  - calculation of extreme values on the domain (for instance, minimum and maximum of some calculation values);
  - test of the existence of a certain value (for instance, do faces of a certain color exist?);
  - verification of a condition on the domain (for instance, is a given flow value reached somewhere?);
  - counting out of entities (for instance, how many cells have pressure drops?);
  - global sum (for instance, calculation of a mass flow or the total mass of a pollutant).

  The user may refer to the different examples present in the directory `EXAMPLES` in the `cs_user_extra_operations-parallel_operations.f90` file. Care should be taken with the fact that the boundaries between subdomains consist of **internal** faces shared between two processors (these are indeed internal faces, even if they are located at a "processor boundary"). They should not be counted twice (once per processor) during global operations using internal faces (for instance, counting the internal faces per processor and summing all the obtained numbers drives into over-evaluating the number of internal faces of the initial mesh).

- **Writing operations that should be made on one processor only in parallel mode**.
  In parallel mode, the user must pay attention during the writing of pieces of information. Writing to the "listing" can be done simply by using the `nfecra` logical unit (each processor will write to its own "listing" file): use `write(nfecra, ....`
  If the user wants an operation to be done by only one processor (for example, open or write a file), the associated instructions must be included inside a test on the value of `irangp` (generally it is the processor 0 which realises these actions, and we want the subroutine to work in non-parallel mode, too: `if (irangp.le.0) then ...`).

Some notes about periodicity

Note that periodic faces are not part of the domain boundary: periodicity is interpreted as a "geometric" condition rather than a classical boundary condition.

Some particular points should be reminded:

- Periodicity can also work when the periodic boundaries are meshed differently (periodicity of non-conforming faces), *except* for the case of a 180 degree rotation periodicity with faces coupled on the rotation axis.
- rotation periodicity is incompatible with
  - semi-transparent radiation,
  - reinforced velocity-pressure coupling (`ipucou=1`).
- although it has not been the case so far, potential problems might be met in the case of rotation periodicity with the $R_{ij}-\varepsilon$ (LRR) model. They would come from the way of taking into account the orthotropic viscosity (however, this term usually has a low influence).

3.9.3.7 Variables saved to allow calculation restarts

The directory `checkpoint` contains:

- `main`: main restart file,

- `auxiliary`: auxiliary restart file (see `ileaux`, `iecaux`),
- `radiative_transfer`: restart file for the radiation module,
- `lagrangian`: main restart file for the Lagrangian module,
- `lagrangian_stats`: auxiliary restart file for the Lagrangian module (mainly for the statistics),
- `1dwall_module`: restart file for the 1D wall thermal module,
- `vortex`: restart file for the vortex method (see `ivrtex`).

The main restart file contains the values in every cell of the mesh for pressure, velocity, turbulence variables and all the scalars (user scalars et specific physics scalars. Its content is sufficient for a calculation restart, but the complete continuity of the solution at restart is not ensured[12].

The auxiliary restart file completes the main restart file to ensure solution continuity in the case of a calculation restart. If the code cannot find one or several pieces of data required for the calculation restart in the auxiliary restart file, default values are then used. This allows in particular to run calculation restarts even if the number of faces has been modified (for instance in case of modification of the mesh merging or of periodicity conditions[13]). More precisely, the auxiliary restart file contains the following data:

- type and value of the time step, turbulence model,
- density value at the cells and boundary faces, if it is variable,
- values at the cells of the other variable physical properties, when they are extrapolated in time (molecular dynamic viscosity, turbulent or sub-grid scale viscosity, specific heat, scalar diffusivity). The specific heat is stored automatically for the Joule effect (in case the user should need it at restart to calculate the temperature from the enthalpy before the new specific heat has been estimated),
- time step value at the cells, if it is variable,
- mass flow value at the internal and boundary faces (at the last time step, and also at the previous time step if required by the time scheme),
- boundary conditions,
- values at the cells of the source terms when they are extrapolated in time,
- number of time-averages, and values at the cells of the associated cumulated values,
- for each cell, distance to the wall when it is required (and index-number of the nearest boundary face, depending on `icdpar`),
- values at the cells of the external forces in balance with a part of the pressure (hydrostatic, in general),
- for the D3P gas combustion model: massic enthalpies and temperatures at entry, type of boundary zones and entry indicators,
- for the EBU gas combustion model: temperature of the fresh gas, constant mixing rate (for the models without mixing rate transport), types of boundary zones, entry indicators, temperatures and mixing rates at entry,

[12] In other words, a restart calculation of n time steps following a calculation of m time steps will not yield strictly the same results as a direct calculation on m+n time steps, whereas it is the case when the auxiliary file is used

[13] Imposing a periodicity changes boundary faces into internal faces

- for the LWC gas combustion model: the boundaries of the probability density functions for enthalpy and mixing rate, types of boundary zones, entry indicators, temperatures and mixing rates at entry,
- for the pulverised coal combustion: coal density, types of boundary zones, variables `ientat`, `ientcp`, `inmoxy`, `timpat`, `x20` (in case of coupling with the Lagrangian module, `iencp` and `x20` are not saved),
- for the pulverised fuel combustion: types of boundary zones, variables `ientat`, `ientfl`, `inmoxy`, `timpat`, `qimpat` , `qimpfl`,
- for the electric module: the tuned potential difference `dpot` and, for the electric arcs module, the tuning coefficient `coejou` (when the boundary conditions are tuned), the Joule source term for the enthalpy (when the Joule effect is activated) and the Laplace forces (with the electric arc module).

It should be noted that, if the auxiliary restart file is read, it is possible to run calculation restarts with relaxation of the density[14](when it is variable), because this variable is stored in the restart file. On the other hand, it is generally not possible to do the same with the other physical properties (they are stored in the restart file only when they are extrapolated in time, or with the Joule effect for the specific heat).

Apart from `vortex` which has a different structure and is always in text format, all the restart files are binary files. Nonetheless, they may be dumped or compared using the `cs_io_dump` tool.

In the case of parallel calculations, it should be noted that all the processors will write their restart data in the same files. Hence, for instance, there will always be one and only one `main` file, whatever the number of processors used. The data in the file are written according to the initial full domain ids for the cells, faces and nodes. This allows in particular to restart using p processors a calculation begun with n processors, or to make the restart files independent of any mesh renumbering that may be carried out in each domain.

WARNING: if the mesh is composed of several files, the order in which they appear in the launch script or in the Graphical Interface must not be modified in case of a calculation restart[15].

NOTE: when joining of faces or periodicity is used, two nodes closer than a certain (small) tolerance will be merged. Hence, due to numerical truncation errors, two different machines may yield different results. This might change the number of faces in the global domain[16] *and make restart files incompatible. Should that problem arise when making a calculation restart on a different architecture, the solution is to ignore the* ***`auxiliary`*** *file and use only the* ***`main`*** *file, by setting* ***`ileaux = 0`*** *in* ***`cs_user_parameters.f90`***

3.9.4 Using selection criteria in user subroutines

In order to use selection criteria (cf. §3.10) in Fortran user subroutines, a collection of utility subroutines is provided. The aim is to define a subset of the mesh, for example:

- boundary regions (cf. `cs_user_boundary_conditions`, `usalcl`, `cs_user_radiative_transfer_bcs.f90`, `cs_user_lagr_boundary_conditions`, ...),
- volume initialization (cf. `cs_user_initialization`, ...),
- head-loss region (cf. `cs_user_head_losses.f90`),
- source terms region (cf. `cs_user_source_terms`),

[14]Such a relaxation only makes sense for a steady calculation

[15]When uncertain, the user can check the saved copy of the launch script in the `RESU` directory, or the head of the `preprocessor*.log` files, which repeat the command lines passed to the Preprocessor module

[16]The number of cells will not be modified, it is always the sum of the number of cells of the different meshes

- advanced post-processing (cf. `cs_user_postprocess.c`, `cs_user_extra_operations`, ...),

This section explains how to define surface or volume sections, in the form of lists `lstelt` of `nlelt` elements (internal faces, boundary faces or cells). For each type of element, the user calls the appropriate Fortran subroutine: `getfbr` for boundary faces, `getfac` for internal faces and `getcel` for cells. All of these take the three following arguments:

- the character string which contains the selection criterion (see some examples below),
- the returned number of elements `nlelt`,
- the returned list of elements `lstelt`.

Several examples of possible selections are given here:

- `call getfbr('Face_1, Face_2', nlelt, lstelt)` to select boundary faces in groups Face_1 or Face_2,
- `call getfac('4', nlelt, lstelt)` to select internal faces of color 4,
- `call getfac('not(4)', nlelt, lstelt)` to select internal faces which have a different color than 4,
- `call getfac('4 to 8', nlelt, lstelt)` to internal faces with color between 4 and 8 internal faces,
- `call getcel('1 or 2', nlelt, lstelt)` to select cells with colors 1 or 2,
- `call getfbr('1 and y > 0', nlelt, lstelt)` to select boundary faces of color 1 which have the coordinate $Y > 0$,
- `call getfac('normal[1, 0, 0, 0.0001]', nlelt, lstelt)` to select internal faces which have a normal direction to the vector (1,0,0),
- `call getcel('all[]', nlelt, lstelt)` to select all cells.

The user may then use a loop on the selected elements.
For instance, in the subroutine `cs_user_boundary_conditions` used to impose boundary conditions, let us consider the boundary faces of color number 2 and which have the coordinate $X <= 0.01$ (so that `call getfbr('2 and x <= 0.01', nlelt,lstelt)`); we can do a loop (`do ilelt = 1, nlelt`) and obtain `ifac = lstelt(ilelt)`.

More examples are available in the `doxygen` documentation.

NOTE: LEGACY METHOD USING EXPLICIT FAMILIES AND PROPERTIES

The selection method for user subroutines by prior versions of *Code_Saturne* is still available, though it may be removed in future versions. This method was better adapted to working with colors than with groups, and is explained here:

From *Code_Saturne*'s point of view, all the references to mesh entities (boundary faces and volume elements) correspond to a number (color number or negative of group number) associated with the entity. An entity may have several references (for instance, one entity may have one color and belong to several groups). In *Code_Saturne*, these references may be designated as "properties".
The mesh entities are gathered in equivalence classes on the base of their properties. These equivalence classes are called "families". All the entities of one family have the same properties. In order to know the properties (in particular the color) of an entity (a boundary face for example), the user must first determine the family to which it belongs.

For instance, let's consider a mesh whose boundary faces have all been given one color (for example using SIMAIL). The family of the boundary face `ifac` is `ifml=ifmfbr(ifac)`. The first (and only) property of this family is the color `icoul`, obtained for the face `ifac` with `icoul=iprfml(ifml,1)`. In order to know the property number corresponding to a group, the utility function `numgrp(nomgrp, lngnom)` (with a name `nomgrp` of the type `character*` and its length `lngnom` of the type `integer`) can be used.

3.10 Face and cell mesh-defined properties and selection

The mesh entities may be referenced by the user during the mesh creation. These references may then be used to mark out some mesh entities according to the need (specification of boundary conditions, pressure drop zones, ...). The references are generally of one of the two following types:

- color. A color is an integer possibly associated with boundary faces and volume elements by the mesh generator. Depending on the tool, this concept may have different names, which *Code_Saturne* interprets as colors. Most tools allow only one color per face or element.
  - I-deas uses a color number with a default of 7 (green) for elements, be they volume elements or boundary "surface coating" elements. Color 11 (red) is used for for vertices, but vertex properties are ignored by *Code_Saturne*.
  - SIMAIL uses the equivalent notions of "reference" for element faces, and "subdomain" for volume elements. By default, element faces are assigned no reference (0), and volume elements domain 1.
  - Gmsh uses "physical property" numbers.
  - EnSight has no similar notion, but if several parts are present in an EnSight 6 file, or several parts are present *and* vertex ids are given in an Ensight Gold file, the part number is interpreted as a color number by the Preprocessor.
  - The MED 2.3 model allowed integer "attributes", though many tools working with this format ignored those and only handle groups.
- groups. Named "groups" of mesh entities may also be used with many mesh generators or formats. In some cases, a given cell or face may belong to multiple groups (as some tools allow new groups to be defined by boolean operations on existing groups). In *Code_Saturne*, every group is assigned a group number (base on alphabetical ordering of groups).
  - I-deas assigns a group number with each group, but by default, this number is just a counter. Only the group name is considered by *Code_Saturne* (so that elements belonging to two groups with identical names and different numbers are considered as belonging to the same group).
  - CGNS allows both for named boundary conditions and mesh sections. If present, boundary condition names are interpreted as group names, and groups may also be defined based on element section or zone names using additional Preprocessor options (`-grp-cel` or `-grp-fac` followed by `section` or `zone`).
  - Using the MED format, it is preferable to use "groups" than colors, as many tools ignore the latter.

Selection criteria may be defined in a similar fashion whether using the GUI or in user subroutines. Typically, a selection criteria is simply a string containing the required color numbers or group names, possibly combined using boolean expressions. Simple geometric criteria are also possible.

A few examples are given below:

```
ENTRY
1 or 7
all[]
```

```
3.1 >= z >= -2 or not (15 or entry)
range[04, 13, attribute]
sphere[0, 0, 0, 2] and (not no_group[])
```

Strings such as group names containing white-space or having names similar to reserved operators may be protected using "escape characters".[17] More complex examples of strings with protected strings are given here:

```
"First entry" or Wall\ or\ sym
entry or \plane or "noone's output"
```

The following operators and syntaxes are allowed (fully capitalized versions of keywords are also allowed, but mixed upper-case/lower-case versions are not):

escape characters	
protect next character only:	\
protect string:	'*string*' "*string*"
basic operators	
priority:	( )
not:	not ! !=
and:	and & &&
or:	or | || , ;
xor:	xor ^
general functions	
select all:	all[]
entities having no group or color:	no_group[]
select a range of groups or colors:	range[*first*, *last*]
	range[*first*, *last*, group]
	range[*first*, *last*, attribute]

For the range operator, *first* and *last* values are inclusive. For attribute (color) numbers, natural integer value ordering is used, while for group names, alphabetical ordering is used. Note also that in the bizarre (not recommended) case in which a mesh would contain for example both a color number 15 and a group named "15", using `range[15, 15, group]` or `range[15, 15, attribute]` could be used to distinguish the two.

Geometric functions are also available. The coordinates considered are those of the cell or face centres. Normals are of course usable only for face selections, not cell selections.

[17] Note that for defining a string in Fortran, double quotes are easier to use, as they do not conflict with Fortran's single quotes delimiting a string. In C, the converse is true. Also, in C, to define a string such as `\plane`, the string `\\plane` must be used, as the first \ character is used by the compiler itself. Using the GUI, either notation is easy.

geometric functions

face normals:	`normal[`x`, `y`, `z`, `$epsilon$`]`
	`normal[`x`, `y`, `z`, epsilon = `$epsilon$`]`
plane, $ax + by + cz + d = 0$ form:	`plane[`a`, `b`, `c`, `d`, `$epsilon$`]`
	`plane[`a`, `b`, `c`, `d`, epsilon = `$epsilon$`]`
	`plane[`a`, `b`, `c`, `d`, inside]`
	`plane[`a`, `b`, `c`, `d`, outside]`
plane, normal + point in plane form:	`plane[`n_x`, `n_y`, `n_z`, `x`, `y`, `z`, `$epsilon$`]`
	`plane[`n_x`, `n_y`, `n_z`, `x`, `y`, `z`, epsilon = `$epsilon$`]`
	`plane[`n_x`, `n_y`, `n_z`, `x`, `y`, `z`, inside]`
	`plane[`n_x`, `n_y`, `n_z`, `x`, `y`, `z`, outside]`
box, extents form:	`box[`x_{min}`, `y_{min}`, `z_{min}`, `x_{max}`, `y_{max}`, `z_{max}`]`
box, origin + axes form:	`box[`x_0`, `y_0`, `z_0`, $dx_1$`, `$dy_1$`, `$dz_1$`, `$dx_2$`, `$dy_2$`, `$dz_2$`, `$dx_3$`, `$dy_3$`, `$dz_3$`]`
cylinder:	`cylinder[`x_0`, `y_0`, `z_0`, `x_1`, `y_1`, `z_1`, `$radius$`]`
sphere:	`sphere[`x_c`, `y_c`, `z_c`, `$radius$`]`
inequalities:	`>`, `<`, `>=`, `<=` associated with `x`, `y`, `z` or `X`, `Y`, `Z` keywords and coordinate value; x_{min} `<= x <` x_{max} type syntax is allowed.

In the current version of *Code_Saturne*, all selection criteria used are maintained in a list, so that re-interpreting a criterion already encountered (such as at the previous time step) is avoided. Lists of entities corresponding to a criteria containing no geometric functions are also saved in a compact manner, so re-using a previously used selection should be very fast. For criteria containing geometric functions, the full list of corresponding entities is not maintained, so each entity must be compared to the criterion at each time step. Heavy use of many selection criteria containing geometric functions may thus lead to reduced performance.

4 Importing and preprocessing meshes

Importing meshes is done by the Preprocessor module, while and preprocessing is done mainly by the code Solver (except for element orientation checking, which is done by the Preprocessor).

The Preprocessor module of *Code_Saturne* reads the mesh file(s) (under any supported format) and translates the necessary information into a Solver input file.

When multiple meshes are used, the Preprocessor is called once per mesh, and each resulting output is added in a `mesh_input` directory (instead of a single `mesh_input` file).

The executable of the Preprocessor module is `cs_preprocess`, and is normally called through the run script, so it is not in standard paths (it is at `<prefix>/libexec/code_saturne/cs_preprocess`). Its most useful options and sub-options are described briefly here. To obtain a complete and up-to-date list of options and environment variables, use the following command: `cs_preprocess -h` or `cs_preprocess --help`. Many options, such as this one, accept a short and a long version.

Nonetheless, it may be useful to call the Preprocessor manually in certain situations, especially for frequent verification when building a mesh, so its use is described here. Verification may also be done using the GUI or the mesh quality check mode of the general run script.

The Preprocessor is controlled using command-line arguments. A few environment variables allow advanced users to modify some behaviours or to obtain a trace of memory management.

4.1 Preprocessor options

Main choices are done using command-line options. For example:

```
cs_preprocess --num 2 fluid.med
```

means that we read the second mesh defined in the `fluid.med` file, while:

```
cs_preprocess --no-write --post-volume med fluid.msh
```

means that we read file `fluid.msh`, and do not produce a `mesh_input` file, but do output a `fluid.med` file (effectively converting a Gmsh file to a MED file).

4.1.1 Mesh selection

Any use of the preprocessor requires one mesh file (except for `cs_preprocess` and `cs_preprocess -h` which respectively print the version number and list of options). This file is selected as the last argument to `cs_preprocess`, and its format is usually automatically determined based on its extension (c.f. 3.4.1 page 21) but a `--format` option allows forcing the format choice of the selected file.

For formats allowing multiple meshes in a single file, the `--num` option followed by a strictly positive integer allows selection of a specific mesh; by default, the first mesh is selected.

For meshes in CGNS format, we may in addition use the `--grp-cel` or `--grp-fac` options, followed by the `section` or `zone` keywords, to define additional groups of cell or faces based on the organization of the mesh in sections or zones. The sub-options have no effect on meshes of other formats.

4.1.2 Post-processing output

By default, the Preprocessor does not generate any post-processor output. By adding `--post-volume [format]`, with the optional `format` argument being one of `ensight`, `med`, or `cgns` to the command-line arguments, the output of the volume mesh to the default or indicated format is provoked.

In case of errors, output of error visualization output is always produced, and by adding `--post-error [format]`, the format of that output may be selected (from one of `ensight`, `med`, or `cgns`, assuming MED and CGNS are available),

4.1.3 Element orientation correction

Correction of element orientation is possible and can be activated using the `--reorient` option.

Note that we cannot guarantee correction (or even detection) of a bad orientation in all cases. Not all local numbering possibilities of elements are tested, as we focus on "usual" numbering permutations. Moreover, the algorithms used may produce false positives or fail to find a correct renumbering in the case of highly non convex elements. In this case, nothing may be done short of modifying the mesh, as without a convexity hypothesis, it is not always possible to choose between two possible definitions starting from a point set.

With a post-processing option such as `--post-error` or, `--post-volume`, visualizable meshes of corrected elements as well as remaining badly oriented elements are generated.

4.2 Environment variables

Setting a few environment variables specific to the Preprocessor allows modifying its default behaviour. In general, if a given behaviour is modifiable through an environment variable rather than by a command-line option, it has little interest for a non-developer, or its modification is potentially hazardous. The environment variables used by the Preprocessor are described here:

`OMP_NUM_THREADS`

Deactivating OpenMP by setting `OMP_NUM_THREADS=1`.

`CS_PREPROCESS_MEM_LOG`

Allows defining a file name in which memory allocation, reallocation, and freeing is logged.

`CS_PREPROCESS_MIN_EDGE_LEN`

Under the indicated length (10^{-15} by default), an edge is considered to be degenerate and its vertices will be merged after the transformation to descending connectivity. Degenerate edges and faces will thus be removed. Hence, the post-processed element does not change, but the Solver may handle a prism where the preprocessor input contained a hexahedron with two identical vertex couples (and thus a face of zero surface). If the Preprocessor does not print any information relative to this type of correction, it means that it has not been necessary. To completely deactivate this automatic correction, a negative value may be assigned to this environment variable.

`CS_PREPROCESS_IGNORE_IDEAS_COO_SYS`

If this variable is defined and is a strictly positive integer, coordinate systems in I-deas universal format files will be ignored. The behaviour of the Preprocessor will thus be the same as that of versions 1.0 and 1.1. Note that in any case, non Cartesian coordinate systems are not handled yet.

`CS_RENUMBER`

Deactivating renumbering is possible by setting `CS_RENUMBER=off`.

4.2.1 System environment variables

Some system environment variables may also modify the behaviour of the Preprocessor. For example, if the Preprocessor was compiled with MED support on an architecture allowing shared (dynamic) libraries, the `LD_PRELOAD` environment variable may be used to define a "prioritary" path to load MED or HDF5 libraries, thus allowing the user to experiment with other versions of these libraries without recompiling the Preprocessor. To determine which shared libraries are used by an executable file, use the following command: `ldd` {`executable_path`}.

4.3 Optional functionality

Some functions of the Preprocessor are based on external libraries, which may not always be available. It is thus possible to configure and compile the Preprocessor so as not to use these libraries. When running the Preprocessor, the supported options are printed. The following optional libraries may be used:

- CGNS library. In its absence, CGNS format support is deactivated.
- MED-file library. In its absence, MED format is simply deactivated.
- libCCMIO library. In its absence, CCM format is simply deactivated.
- Read compressed files using Zlib. With this option, it is possible to directly read mesh files compressed with a *gzip* type algorithm and bearing a *.gz* extension. This is limited to formats not already based on an external library (i.e. it is not usable with CGNS, MED, or CCM files), and has memory and CPU time overhead, but may be practical. Without this library, files must be uncompressed before use.

4.4 General remarks

Note that the Preprocessor is in general capable of reading all "classical" element types present in mesh files (triangles, quadrangles, tetrahedra, pyramids, prisms, and hexahedra). Quadratic or cubic elements are converted upon reading into their linear counterparts. Vertices referenced by no element

(isolated vertices or centres of higher-degree elements) are discarded. Meshes are read in the order defined by the user and are appended, vertex and element indices being incremented appropriately. [18]

At this stage, volume elements are sorted by type, and the fluid domain post-processing output is generated if required.

In general, groups assigned to vertices are ignored. selections are thus based on faces or cells. with tools such as SIMAIL, faces of volume elements may be referenced directly, while with I-deas or SALOME, a layer of surface elements bearing the required colors and groups must be added. Internally, the Preprocessor always considers that a layer of surface elements is added (i.e. when reading a SIMAIL mesh, additional faces are generated to bear cell face colors. When building the $faces \rightarrow cells$ connectivity, all faces with the same topology are merged: the initial presence of two layers of identical surface elements belonging to different groups would thus lead to a calculation mesh with faces belonging to two groups).

4.5 Files passed to the Solver

Data passed to the Solver by the Preprocessor is transmitted using a binary file, using "big endian" data representation, named `mesh_input` (or contained in a directory of that name).

When using the Preprocessor for mesh verification, data for the Solver is not always needed. In this case, the `--no-write` option may avoid creating a Preprocessor output file.

4.6 Mesh preprocessing

4.6.1 Joining of non-conforming meshes

Conforming joining of possibly non-conforming meshes may be done by the solver, and defined either using the Graphical User Interface (GUI) or the `cs_user_join` user function. In the GUI, the user must add entries in the "Face joining" section of the "Meshes" tab in the item "Calculation environment → Meshes selection". The user may specify faces to be joined, and can also modify basic joining parameters, see Figure6. For a simple mesh, it is rarely useful to specify strict face selection criteria, as joining is sufficiently automated to detect which faces may actually be joined. For a more complex mesh, or a mesh with thin walls which we want to avoid transforming into interior faces, it is recommended to filter boundary faces that may be joined by using face selection criteria. This has the additional advantage of reducing the number of faces to test for in the intersection/overlap search, and thus reduced to the time required by the joining algorithm.

One may also modify tolerance criteria using 2 options:

`fraction` r	assigns value r (where $0 < r < 0.49$) to the maximum intersection distance multiplier (0.1 by default). The maximum intersection distance for a given vertex is based on the length of the shortest incident edge, multiplied by r. The maximum intersection at a given point along an edge is interpolated from that at its vertices, as shown on the left of Figure 7;
`plane` c	assigns the maximum angle between normals for two faces to be considered coplanar (25° by default); this parameter is used in the second stage of the algorithm, to reconstruct conforming faces, as shown on the right of figure 7.

In practice, we are sometimes led to increase the maximum intersection distance multiplier to 0.2 or even 0.3 when joining curved surfaces, so that all intersection are detected. As this influences merging of vertices and thus simplification of reconstructed faces, but also deformation of "lateral" faces, it is recommended only to modify it if necessary. As for the `plane` parameter, its use has only been necessary on a few meshes up to now, and always in the sense of reducing the tolerance so that face reconstruction does not try to generate faces from initial faces on different surfaces.

[18] Possible entity labels are not maintained, as they would probably not be unique when appending multiple meshes.

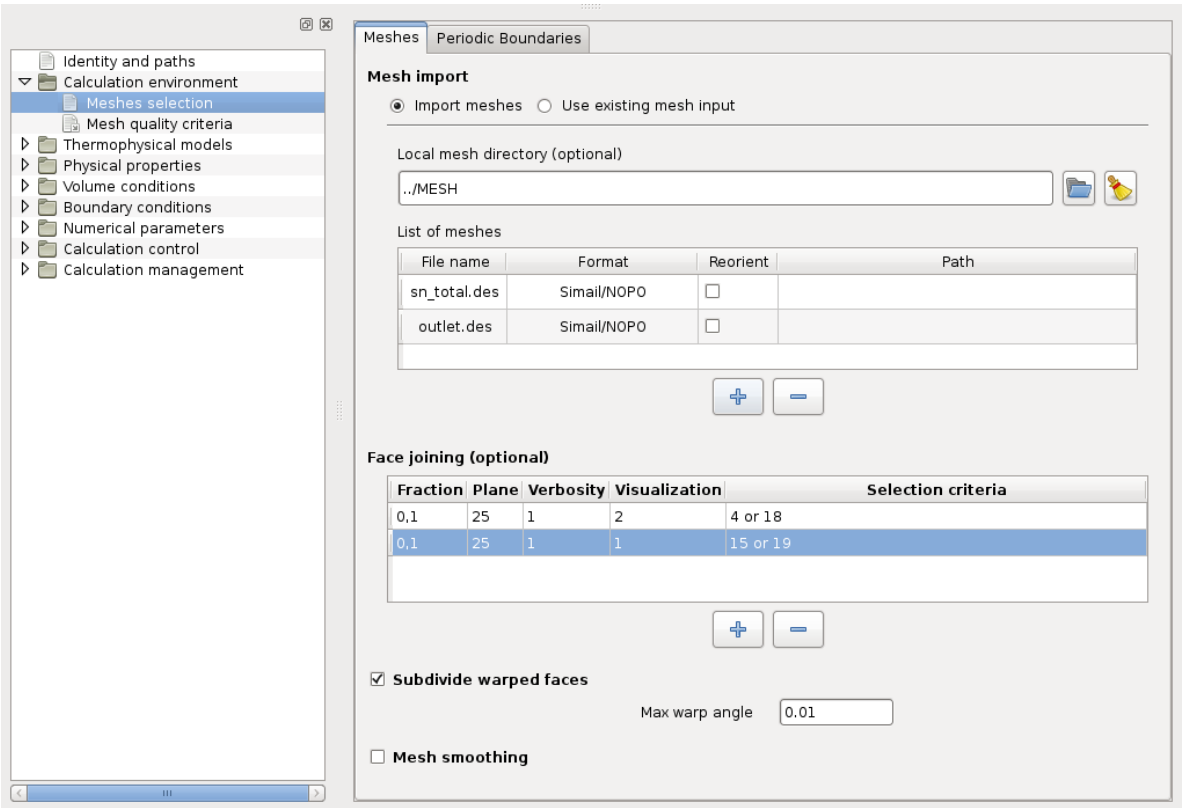

Figure 6: Conformal or non-conformal joining

4.6.2 Periodicity

Handling of periodicity is based on an extension of conforming joining, as shown on Figure 8. It is thus not necessary for the periodic faces to be conforming (though it usually leads to better mesh quality). All options relative to conforming joining of non-conforming faces also apply to periodicity. Note also that once pre-processed, 2 periodic faces have the same orientation (possibly adjusted by periodicity of rotation).

This operation can also be performed by the solver and specified either using the GUI or the `cs_user_periodicity` function.

As with joining, it is recommended to filter boundary faces to process using a selection criterion. As many periodicities may be built as desired, as long as boundary faces are present. Once a periodicity is handled, faces having periodic matches do not appear as boundary faces, but as interior faces, and are thus not available anymore for other periodicities.

4.6.3 Parameters for conforming or non-conforming mesh joinings

The setting of these parameters is done in the user subroutine `cs_user_join` (called once). The user can specify the parameters used for the joining of different meshes. Below is given the list of the standard parameters which can me modified:

- `fract`: the initial tolerance radius associated to each vertex is equal to the length of the shortest incident edge, multiplied by this fraction,
- `plane`: when subdividing faces, 2 faces are considered coplanar and may be joined if the angle

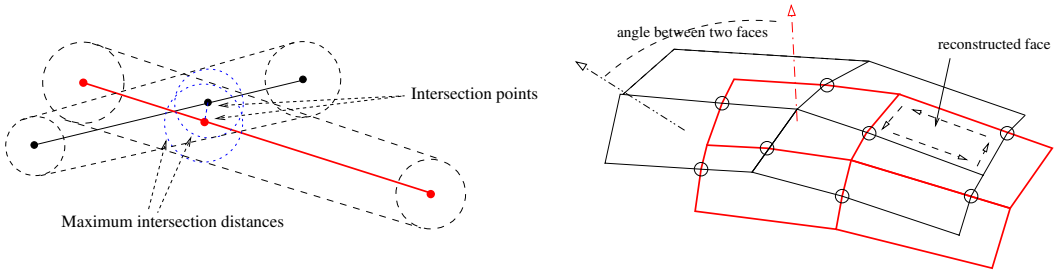

Figure 7: Maximum intersection tolerance and faces normal angle

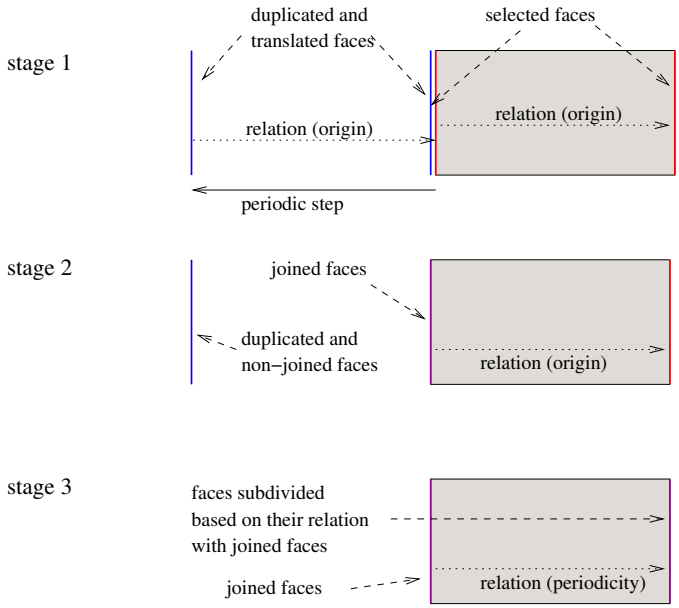

Figure 8: Matching of periodic faces

between their unit normals (in degrees) does not exceed this parameter,

- `iwarnj`: the associated verbosity level (debug level if over 3).

In the call of the function `cs_join_add`, a selection criteria for mesh faces to be joined is specified.

The call to the function '`cs_join_set_advanced_param`' allows defining parameters not available through the GUI.

The list of advanced modifiable parameters is given below:

- `mtf`: a merge tolerance factor, used to locally modify the tolerance associated to each vertex before the merge step. Depending on its value, four scenarios are possible:
  - → if $mtf = 0$, no vertex merge
  - → if $mtf < 1$, the vertex merge is more strict. It may increase the number of tolerance reduction and therefore define smaller subset of vertices to merge together but it can drive to loose intersections.
  - → if $mtf = 1$, no change occurs
  - → if $mtf > 1$, the vertex merge is less strict. The subset of vertices able to merge is greater.
- `pmf`: a pre-merge factor. This parameter is used to define a limit under which two vertices are merged before the merge step,
- `tcm`: a tolerance computation mode. If its value is:
  - → 1 (default), the tolerance is the minimal edge length related to a vertex, multiplied by a fraction.
  - → 2, the tolerance is computed like for 1 with, in addition, the multiplication by a coefficient equal to the maximum between $sin(e1)$ and $sin(e2)$; where $e1$ and $e2$ are two edges sharing the same vertex V for which we want to compute the tolerance.
  - → 11, it is the same as 1 but taking into account only the selected faces.
  - → 12, it is the same as 2 but taking into account only the selected faces.
- `icm`: the intersection computation mode. If its value is:
  - → 1 (default), the original algorithm is used. Care should be taken to clip the intersection on an extremity.
  - → 2, a new intersection algorithm is used. Caution should be used to avoid clipping the intersection on an extremity.
- `maxbrk`: defines the maximum number of equivalence breaks which is enabled during the merge step,
- `maxsf`: defines the maximum number of sub-faces used when splitting a selected face

The following are advanced parameters used in the search algorithm for face intersections between selected faces (octree structure). They are useful in case of memory limitation:

- `tml`: the tree maximum level is the deepest level reachable during the tree building,
- `tmb`: the tree maximum boxes is the maximum number of bounding boxes (BB) which can be linked to a leaf of the tree (not necessary true for the deepest level),
- `tmr`: the tree maximum ratio. The building of the tree structure stops when the number of bounding boxes is superior to the product of `tmr` with the number of faces to locate. This is an efficient parameter to reduce memory consumption.

Examples of mesh modification are given in the following `doxygen` documentation.

4.6.4 Parameters for periodicity

Periodicities can be set directly in the Graphical User Interface (GUI) or using the user function `cs_user_periodicity` (called once during the calculation initialisation). In both cases, the user can choose between 3 types of periodicities: translation, rotation, or mixed (see Figure9). Then specific parameters must be set.

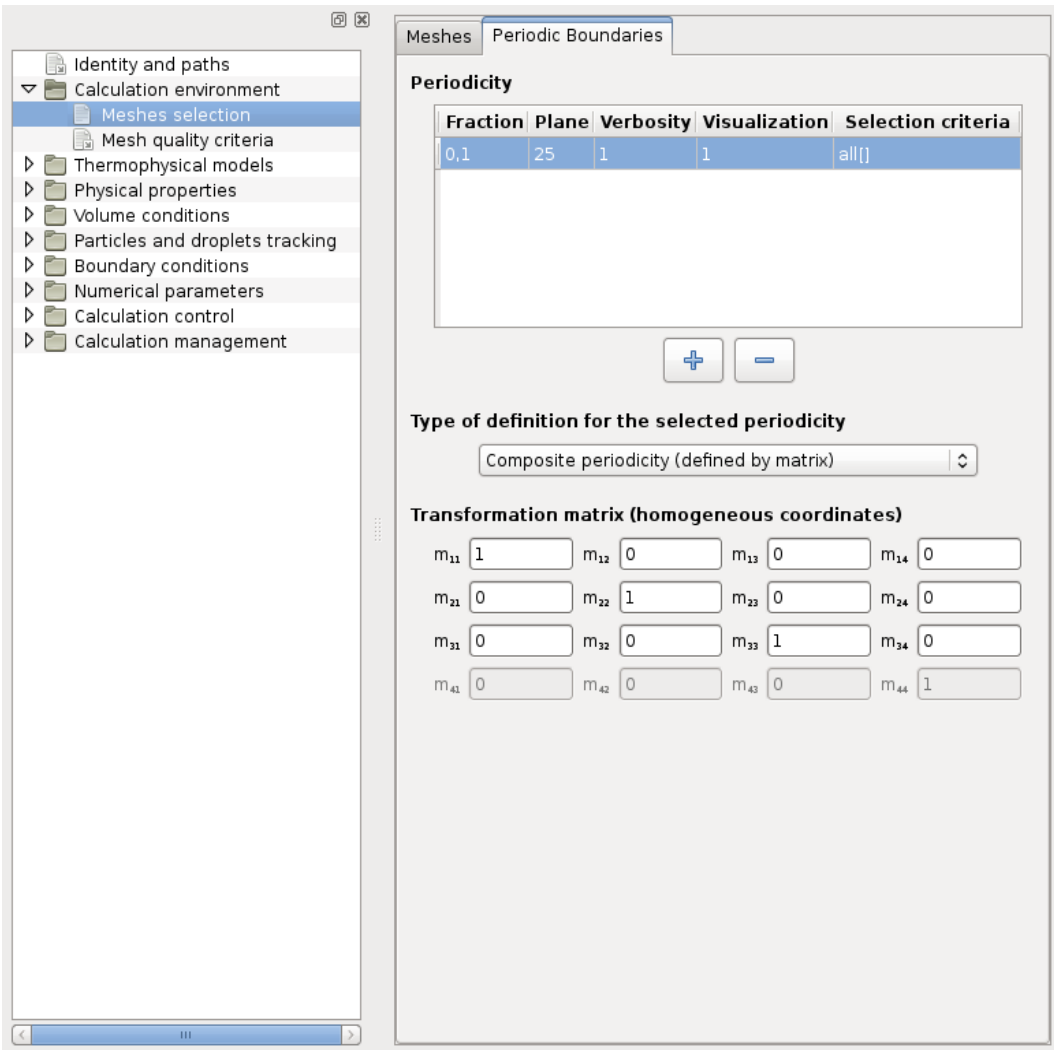

Figure 9: Periodicity

As periodicity is an extension of mesh joining, all parameters (whether basic or advanced) available for mesh joining are also applicable for periodicity, in addition to the parameters defining the periodicity transformation.

4.6.5 Modification of the mesh geometry

Functions called only once during the calculation initialisation.

The user function `cs_user_mesh_input` allows a detailed selection of imported meshes read, reading files multiple times, applying geometric transformations, and renaming groups.

The user function `cs_user_mesh_modify` may be used for advanced modification of the main `cs_mesh_t`

structure.

WARNING: Caution must be exercised when using this function along with periodicity. Indeed, the periodicity parameters are not updated accordingly, meaning that the periodicity may not be valid after mesh vertex coordinates have changed. It is particularly true when one rescales the mesh. Rescaling should thus be done in a separate run, before defining periodicity.

The user function `cs_user_mesh_thinwall` allows insertion of thin walls in the calculation mesh. Currently, this function simply transforms the selected internal faces into boundary faces, on which boundary conditions can (and must) be applied.

Faces on each side of a thin wall will share the same vertices, so post-processing of the main volume mesh may not show the inserted walls, though they will appear in the main boundary output mesh.

4.7 Mesh smoothing utilities

Function called once only during the calculation initialisation.

The smoothing utilities may be useful when the calculation mesh has local defects. The principle of smoothers is to mitigate the local defects by averaging the mesh quality. This procedure can help for calculation robustness or/and results quality.

The user function `cs_user_mesh_smoothe` allows to use different smoothing functions detailed below.

WARNING 1: Caution must be exercised when using this function along with periodicity. Indeed, the periodicity parameters are not currently updated accordingly, meaning that the periodicity may be unadapted after one changes the mesh vertex coordinates. It is particularly true when one rescales the mesh. Rescaling should thus be done in a separate run, before defining periodicity.

WARNING 2: Caution must be exercised when using smoothing utilities because the geometry may be modified. In order to preserve geometry, the function `cs_mesh_smoother_fix_by_feature` *allows to fix by a feature angle criterion the mobility of boundary vertices.*

4.7.1 Fix by feature

The vertex normals are defined by the average of the normals of the faces sharing the vertex. The feature angle between a vertex and one of its adjacent faces is defined by the angle between the vertex normal and the face normal.

This function sets a vertex if one of its feature angles is less than $cos(\theta)$ where θ is the maximum feature angle (in degrees) defined by the user. In fact, if $\theta = 0°$ all boundary vertices will be fixed, and if $\theta = 90°$ all boundary vertices will be free.

Fixing all boundary vertices ensures the geometry is preserved, but reduces the smoothing algorithm's effectiveness.

4.7.2 Warped faces smoother

The function `cs_mesh_smoother_unwarp` allows reducing face warping in the calculation mesh.

Be aware that, in some cases, this algorithm may degrade other mesh quality criteria.

5 Partitioning for parallel runs

Graph partitioning (using one of the optional METIS or SCOTCH libraries) is done by the Solver. Unless explicitly deactivated, this stage produces one or several "cell → domain" distribution files, named `domain_number_p` for a partitioning on p sub-domains, which may be read when starting a subsequent computation so as to avoid re-running that stage. These files are placed in a directory

named `partition_output`.

The Solver redistributes data read in `mesh_input` based on the associated (re-read or computed) partitioning, so there is no need to run any prior script when running on a different number of processors, although a previous partitioning may optionally be re-used.

Without a graph-based partitioning library, or based on the user's choice, the Solver will use a built-in partitioning using a space-filling curve (Z or Hilbert curve) technique. This usually leads to partitionings of lower quality than with graph partitioning, but parallel performance remains reasonable.

Partitioning options may be defined using the GUI or by calling the appropriate functions in the `cs_user_partition_options` user function.

5.1 Partitioning stages

Partitioning is always done just after reading the mesh, unless a partitioning input file is available, in which case the partitioning replaces this stage.

When a mesh modification implying a change of cell connectivity graph is expected, the mesh may be repartitioned after the pre-processing stage, prior to calculation. By default, re-partitioning is only done if the partitioning algorithm chosen for that stage is expected to produce different results due to the connectivity change. This is the case for graph-based algorithms such as those of METIS or SCOTCH, when mesh joining is defined, or additional periodic matching is defined (and the algorithm is not configured to ignore periodicity information).

There are thus two possible partitioning stages:

- `CS_PARTITION_FOR_PREPROCESS`, which is optional, and occurs just after reading the mesh.
- `CS_PARTITION_MAIN`, which occurs just after reading the mesh if it is the only stage, or after mesh preprocessing (and before computation), if the partitioning for preprocessing stage is activated.

The number of partitioning stages is determined automatically based on information provided through `cs_partition_set_preprocess_hints()`, but repartitioning may also be forced or inhibited using the `cs_partition_set_preprocess()` user function.

5.2 Partitioner choice

If the Solver has been configured with both PT-SCOTCH or SCOTCH and PARMETIS or METIS libraries, PT-SCOTCH will be used by default[19], but the user may force the selection of another partitioning type using either the GUI or user routines.

In addition to graph-based partitionings, a space-filling curve based algorithm is available, using either a Morton (Z) or Peano-Hilbert curve, in the computation domain's bounding box or bounding curve.

When partitioning for preprocessing, a space-filling curve is used, unless forced by calling `cs_partition_set_algorithm()` with the appropriate algorithm choice for the `CS_PARTITION_FOR_PREPROCESS` stage.

5.3 Effect of periodicity

By default, face periodicity relations are taken into account when building the "cell → cell" connectivity graph used for partitioning. This allows better partitioning optimization, but increases the probability of having groups of cells at opposite sides of the domain in a same sub-domain. This is not an issue for standard calculations, but may degrade performance of search algorithms based on bounding boxes. It is thus possible to ignore periodicity when partitioning a mesh.

[19] Though PARMETIS will be chosen before serial SCOTCH in a parallel run

Also, partitioning using a space-filling curve ignores periodicity.

Note that nothing guarantees that a graph partitioner will not place disjoint cells in the same subdomain independently of this option, but this behaviour is rare.

6 Basic modelling setup

6.1 Initialisation of the main parameters

This operation is done in the Graphical User Interface (GUI) or by using the user subroutines in `cs_user_parameters.f90`. In the GUI, the initialisation is performed by filling the parameters displayed in Figure10 to 27. If the option 'Mobile mesh' is activated in Figure 11, please see Section 7.11.4 for more details. The headings filled for the initialisation of the main parameters are the followings:

- Thermophysical model options: Steady or unsteady algorithm, specific physics, ALE mobile mesh, turbulence model, thermal model and species transport (definition of the scalars and their variances), see Figure 10 to Figure 14. If a thermal scalar, temperature or enthalpy, is selected, two other headings on conjugate heat transfer and radiative transfers can be filled in (see Figure 13).
- Physical properties: reference pressure, velocity and length, fluid properties (density, viscosity, thermal conductivity, specific heat and scalar diffusivity), gravity, see Figure 15 to Figure 17. If non-constant values are used for the fluid properties, and if the GUI is not used, the `cs_user_physical_properties` file must be used, see § 6.5.1.
- Volume conditions: definition of volume regions (for initialisation, head losses and source terms, see § 6.6 and § 6.7), initialisation of the variables (including scalars), Coriolis source term, see Figure 18 and Figure 19.
- Boundary conditions: definition and parametrisation of boundary conditions for all variables (including scalars). If the GUI is not used, the `cs_user_boundary_conditions` file must be used, see § 6.4.
- Numerical parameters: number and type of time steps, and advanced parameters for the numerical solution of the equations, see Figure 20 to Figure 22.
- Calculation control: parameters related to the time averages, the locations of the probes where some variables will be monitored over time (if the GUI is not used, this information is specified in § 6.3), the definition of the frequency of the outputs in the calculation listing, the post-processing output writer frequency and format options, and the post-processing output meshes and variables selection, see Figure 23, Figure 24, Figure 25, and Figure 26. The item "Profiles" allows to save, with a frequency defined by the user, 1D profiles on a parametric curve define by its equation, see Figure27.

With the GUI, the subroutine `cs_user_parameters.f90` is only used to modify high-level parameters which can not be managed by the interface. Without the GUI, this subroutine is compulsory and some of the headings must be completed (see §3.2.1). `cs_user_parameters.f90` is used to indicate the value of different calculation basic parameters: constant and uniform physical values, parameters of numerical schemes, input-output management...
It is called only during the calculation initialisation.

For more details about the different parameters, please refer to the key word list (§8).

`cs_user_parameters.f90` is in fact constituted of 4 separate subroutines: `usipph`, `usppmo`, `usipsu` and `usipes`. Each one controls various specific parameters. The keywords which are not featured in the supplied example can be provided by the user in `SRC/REFERENCE/base`; in this case, understanding of the comments is required to add the keywords in the appropriate subroutine, it will

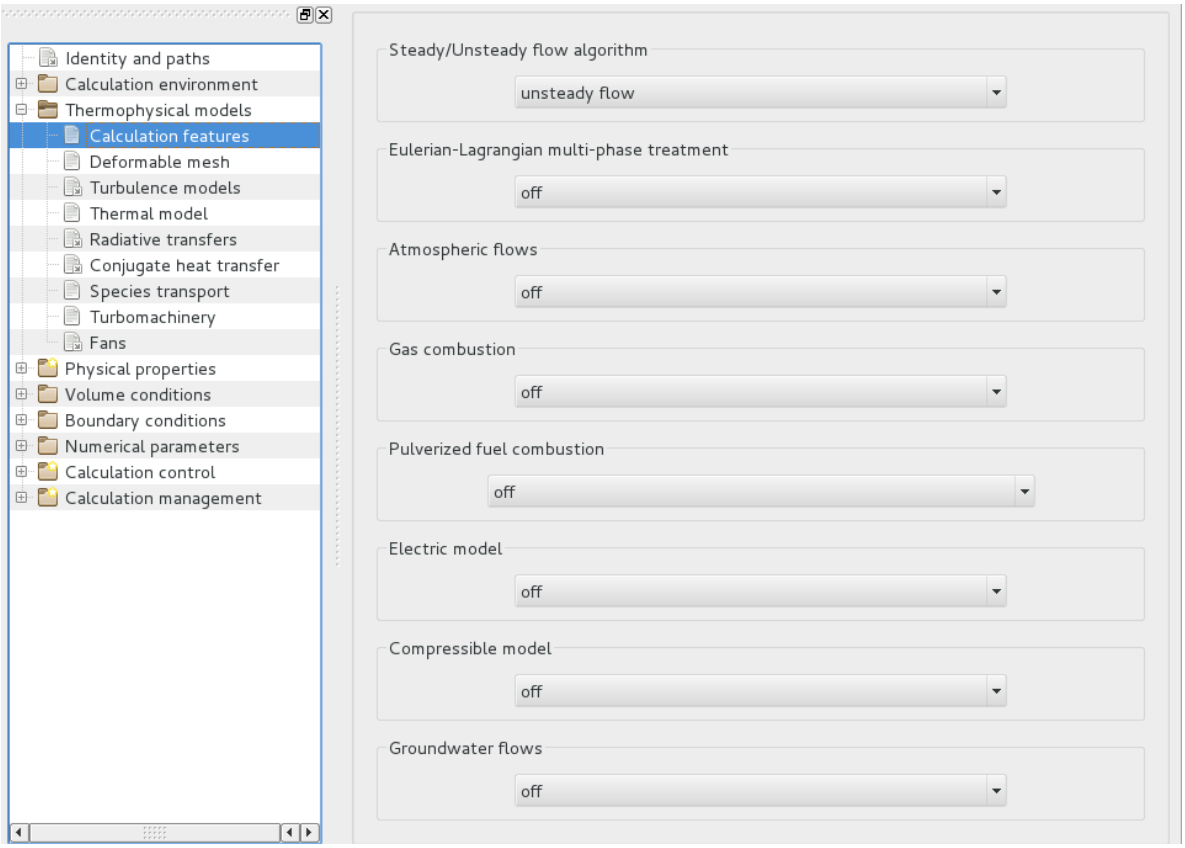

Figure 10: Calculation features options

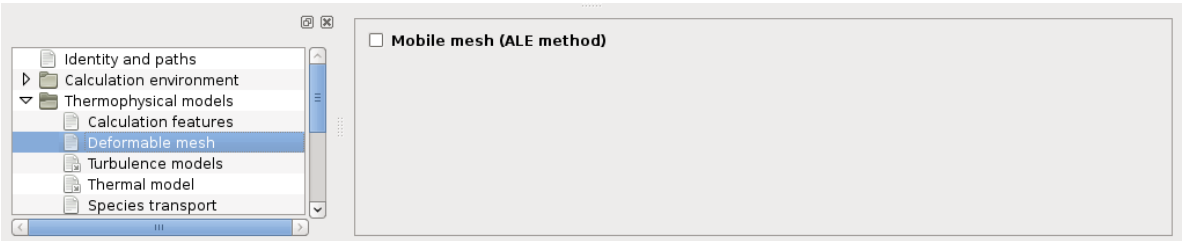

Figure 11: Mobile mesh option

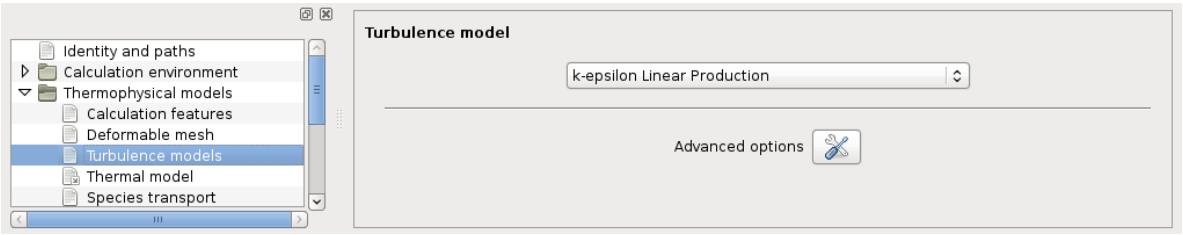

Figure 12: Turbulence model selection

ensure that the value has been well defined. The modifiable parameters in each of the subroutines of `cs_user_parameters.f90` are:

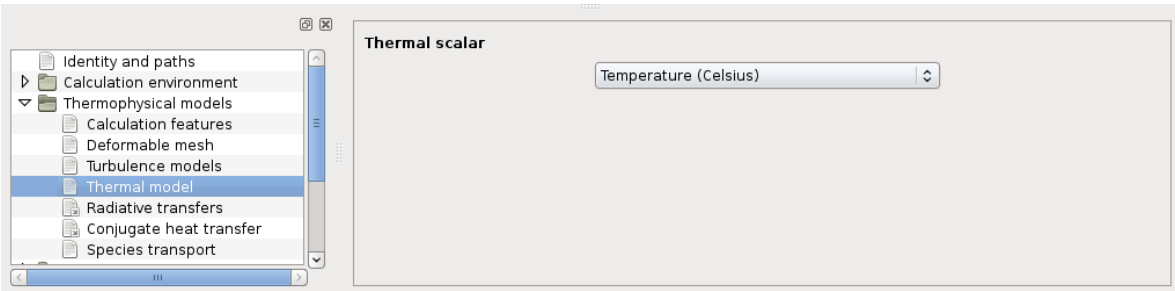

Figure 13: Thermal scalar selection

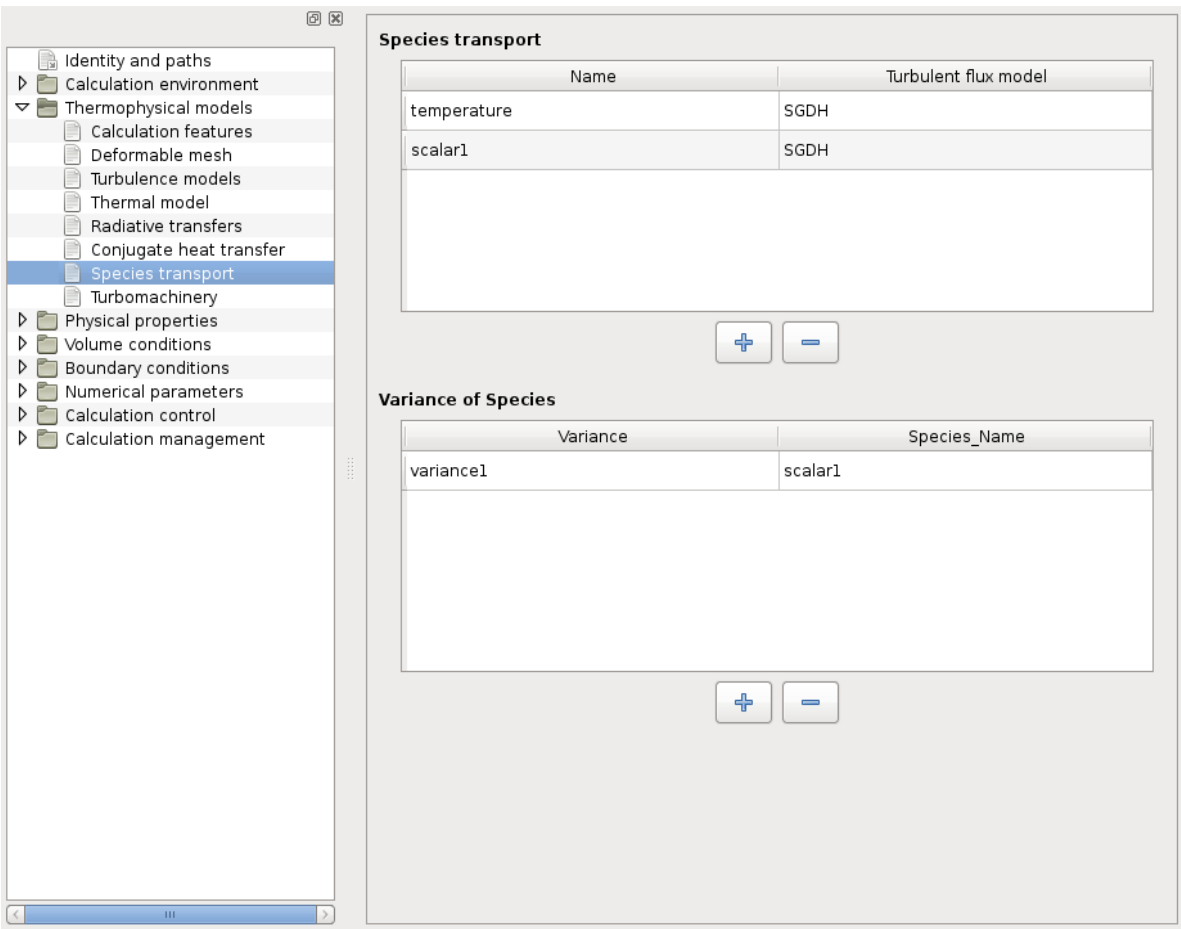

Figure 14: Definition of the transported species/scalars

- `usipph`: `iturb`, `itherm` and `icavit` (don't modify these parameters anywhere else)
- `usppmo`: activation of specific physical models.
- `usipsu`: physical parameters of the calculation (thermal scalar, physical properties, ...), numerical parameters (time steps, number of iterations, ...), definition of the time averages.
- `usipes`: post-processing output parameters (periodicity, variable names, probe positions, ...)

For more details on the different parameters, see the list of keywords (§ 8). The names of the keywords

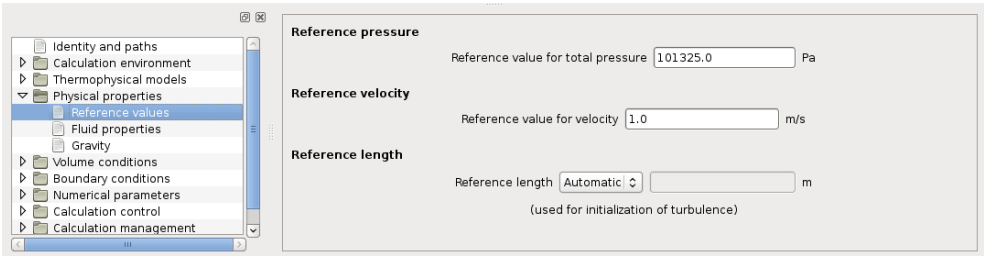

Figure 15: Setting of the reference values for pressure, velocity and length

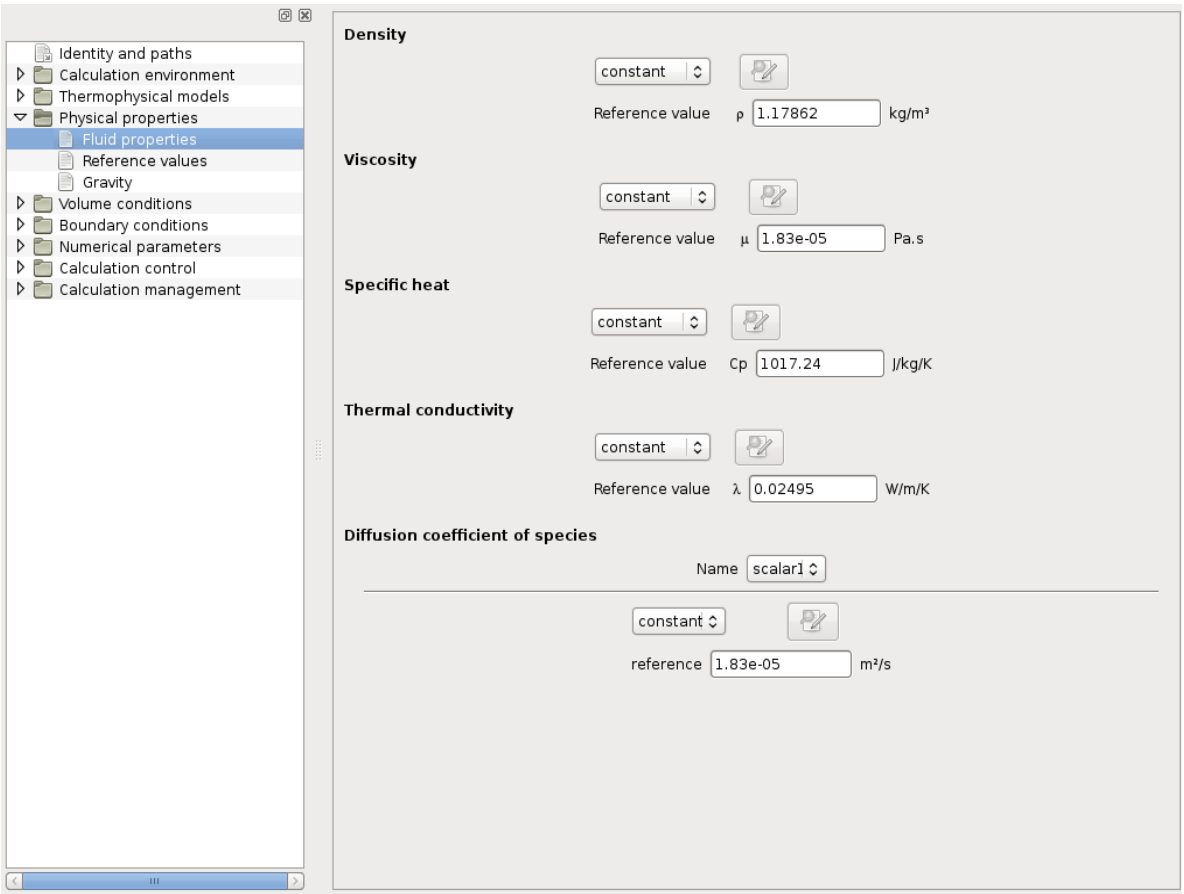

Figure 16: Fluid properties

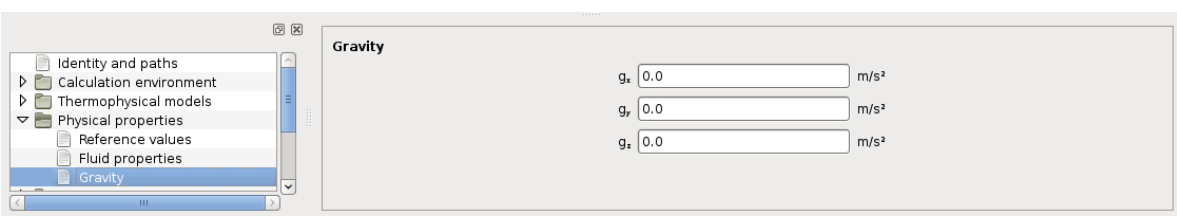

Figure 17: Setting of the gravity

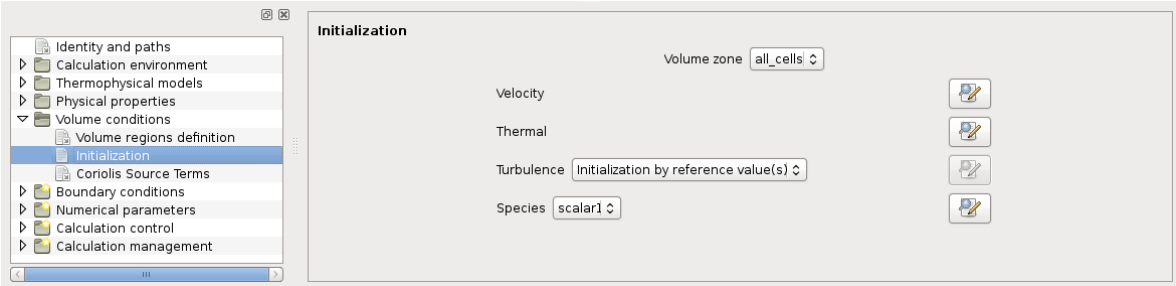

Figure 18: Initialisation of variables

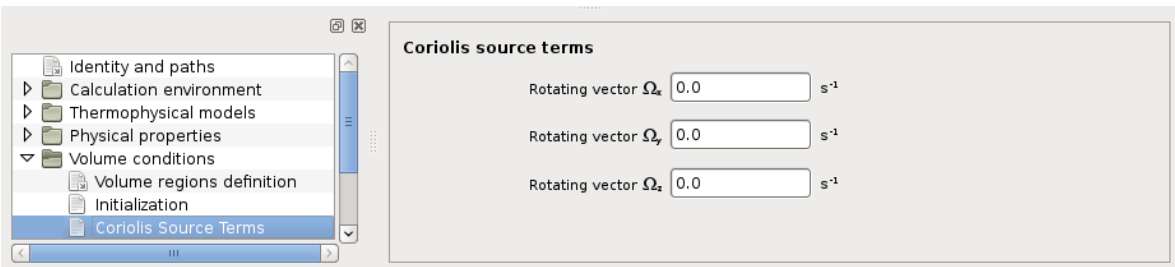

Figure 19: Setting of the Coriolis source term

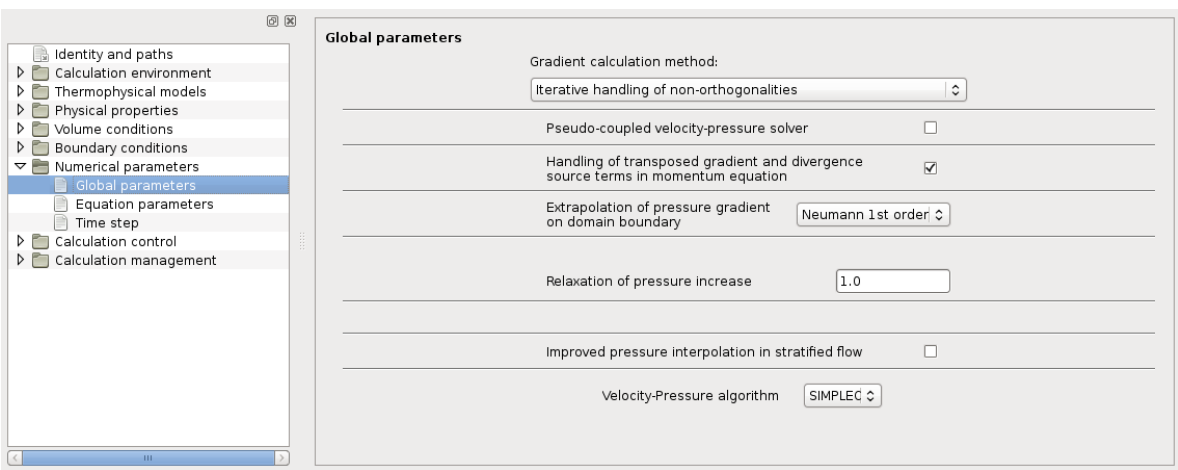

Figure 20: Global resolution parameters

can also be seen in the help sections of the interface.

• When using the interface, only the additional parameters (which can not be defined in the interface) should appear in `cs_user_parameters.f90`. The user needs then only to activate examples which are useful for his case (replacing `if (.false.)` with `if (.true.)`, or removing such tests).

6.2 Selection of mesh inputs: `cs_user_mesh_input`

Subroutine called only during the calculation initialisation.

This C function may be used to select which mesh input files are read, and apply optional coordinate

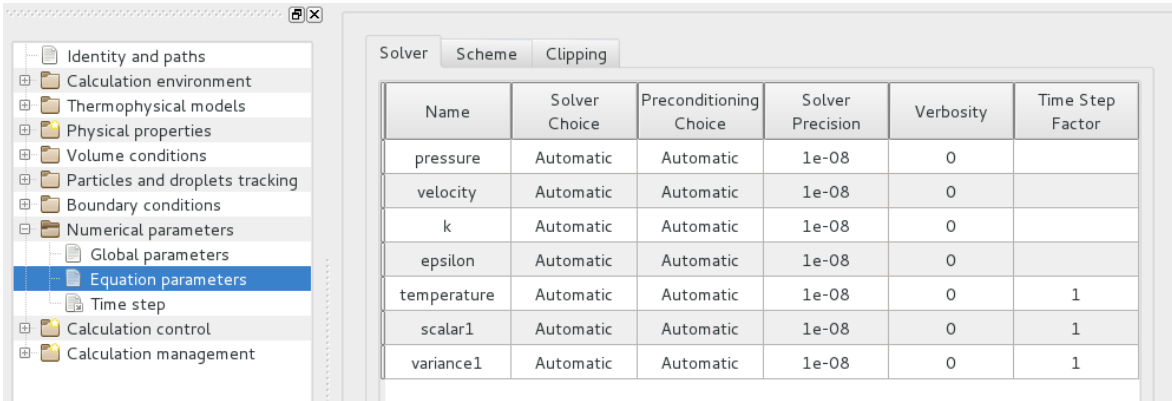

Figure 21: Numerical parameters for the main variables resolution

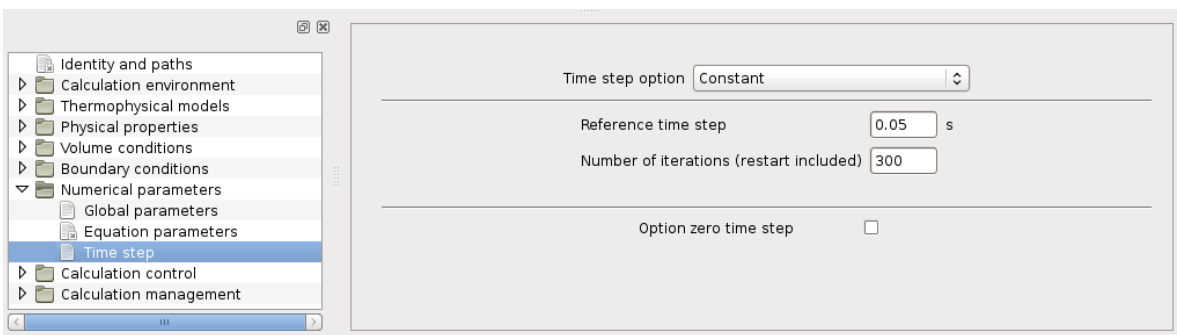

Figure 22: Time step settings

transformations or group renumberings to them. By default, the input read is a file or directory named `mesh_input`, but if this function is used, any file can be selected, and the same file can be read multiple times (applying a different coordinate transformation each time). All inputs read through this function are automatically concatenated, and may be later joined using the mesh joining options.

Geometric transformations are defined using a homogeneous coordinates transformation matrix. Such a matrix has 3 lines and 4 columns, with the first 3 columns describing a rotation/scaling factor, and the last column describing a translation. A 4th line is implicit, containing zeroes off-diagonal, and 1 on the diagonal. The advantages of this representation is that any rotation/translation/scaling combination may be expressed by matrix multiplication, while simple rotations or translations may still be defined easily.

6.3 Non-default variables initialisation

The non-default variables initialisation is performed in the subroutine `cs_user_initialization` (called only during the calculation initialisation).
At the calculation beginning, the variables are initialised automatically by the code. Velocities and scalars are set to 0 (or `scamax` or `scamin` if 0 is outside the acceptable scalar variation range), and the turbulent variables are estimated from `uref` and `almax`.
For k (of variable index `ik`) in the $k-\varepsilon$, $R_{ij}-\varepsilon$, v2f or $k-\omega$ models:

$$k = 1.5\,(0.02\ \texttt{uref})^2$$

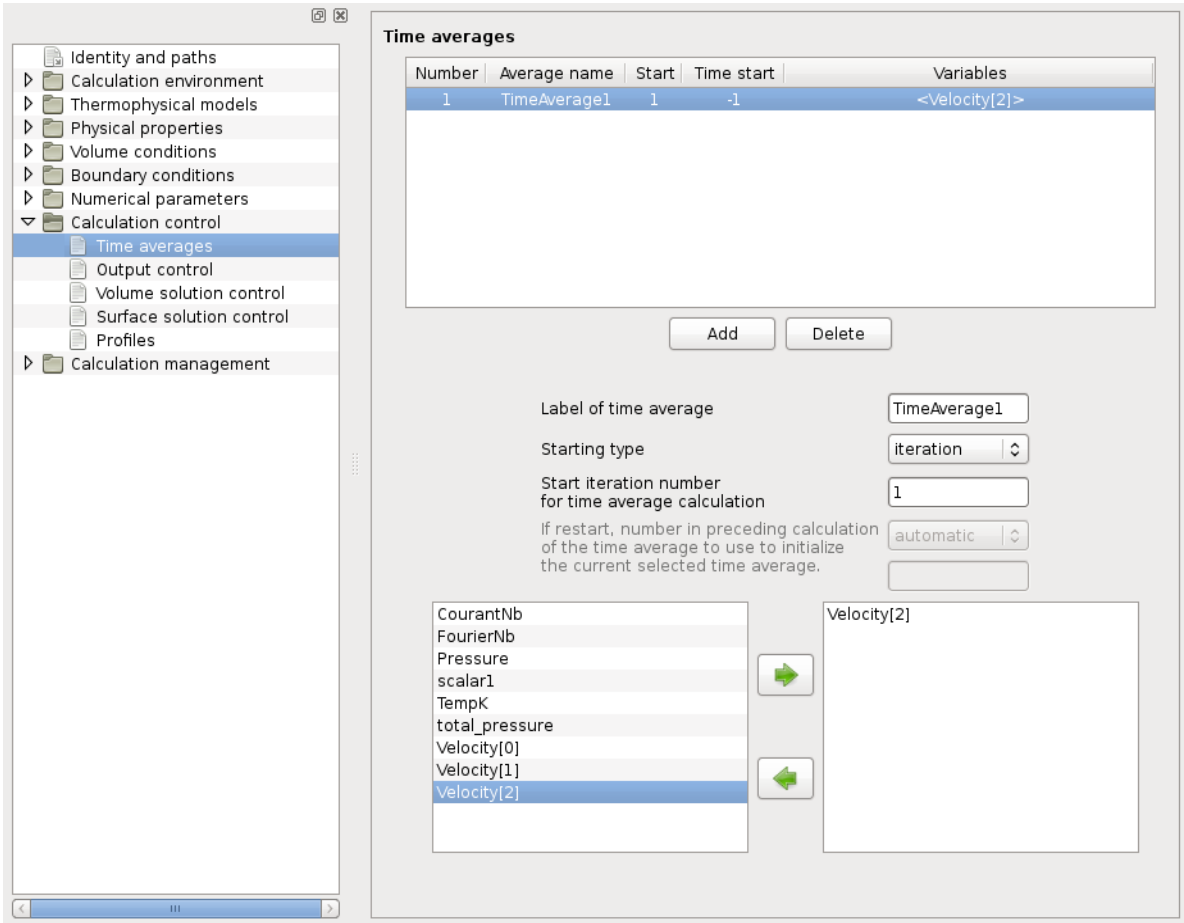

Figure 23: Management of time averaged variables

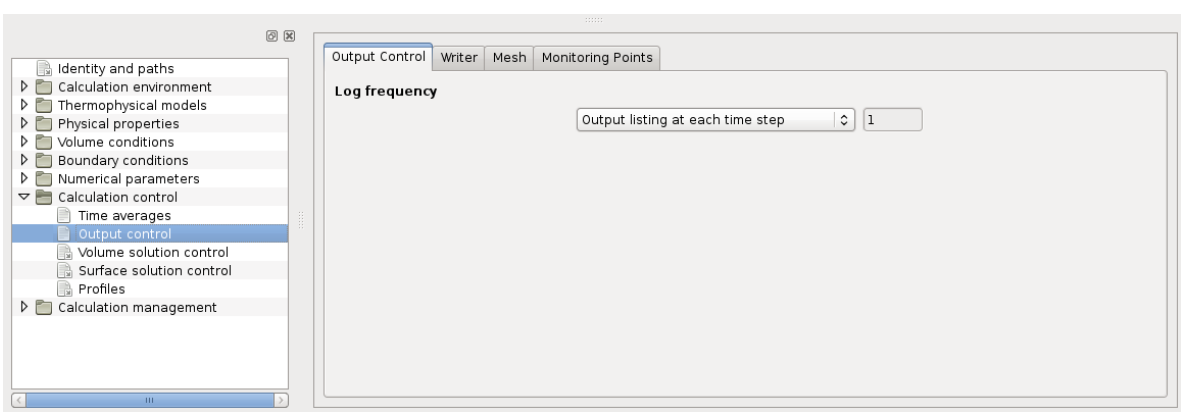

Figure 24: Parameters of chronological logging options

and in $R_{ij} - \varepsilon$:

$$R_{ij} = \frac{2}{3} k \delta_{ij}$$

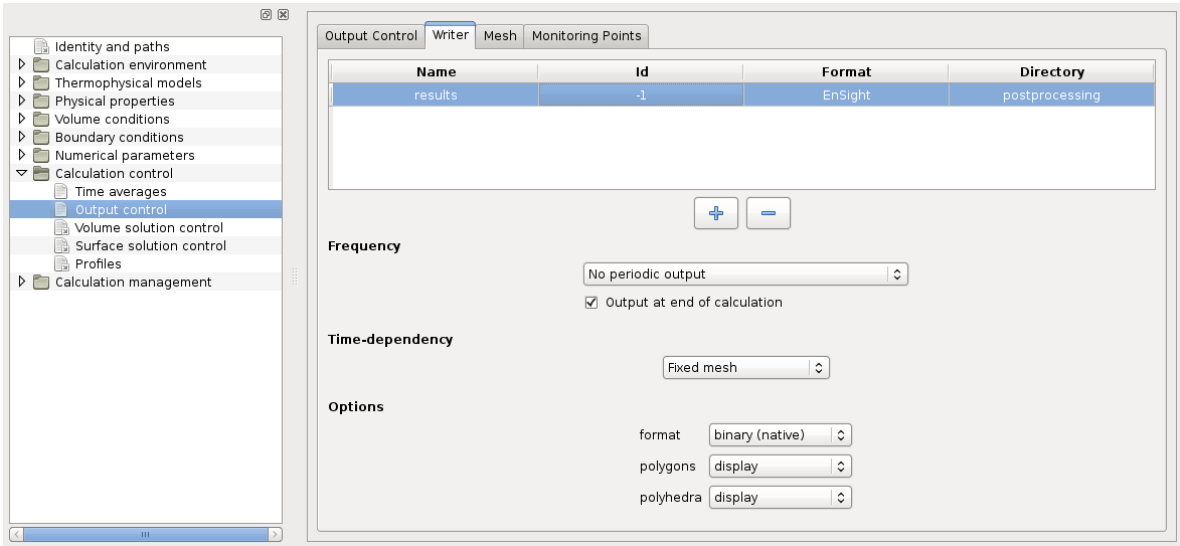

Figure 25: Management of postprocessing writers

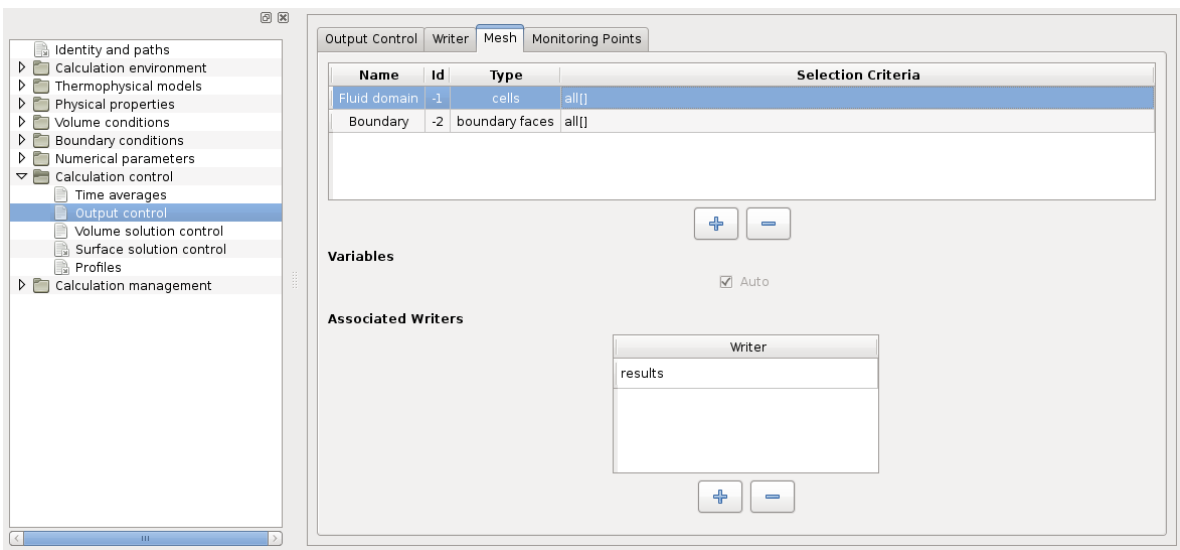

Figure 26: Management of postprocessing meshes

For ε (of variable index `iep`) in the $k-\varepsilon$, $R_{ij}-\varepsilon$ or v2f models:

$$\varepsilon = k^{1.5}\frac{C_\mu}{\texttt{almax}}$$

For ω (of variable index `iomg`) in the $k-\omega$ model:

$$\omega = k^{0.5}\frac{1}{\texttt{almax}}$$

For φ and $\overline{f}$ (of variable indices `iphi` and `ifb`) in the v2f models:

$$\left\{\begin{array}{ll}\varphi = & \frac{2}{3}\\ \overline{f} = & 0\end{array}\right.$$

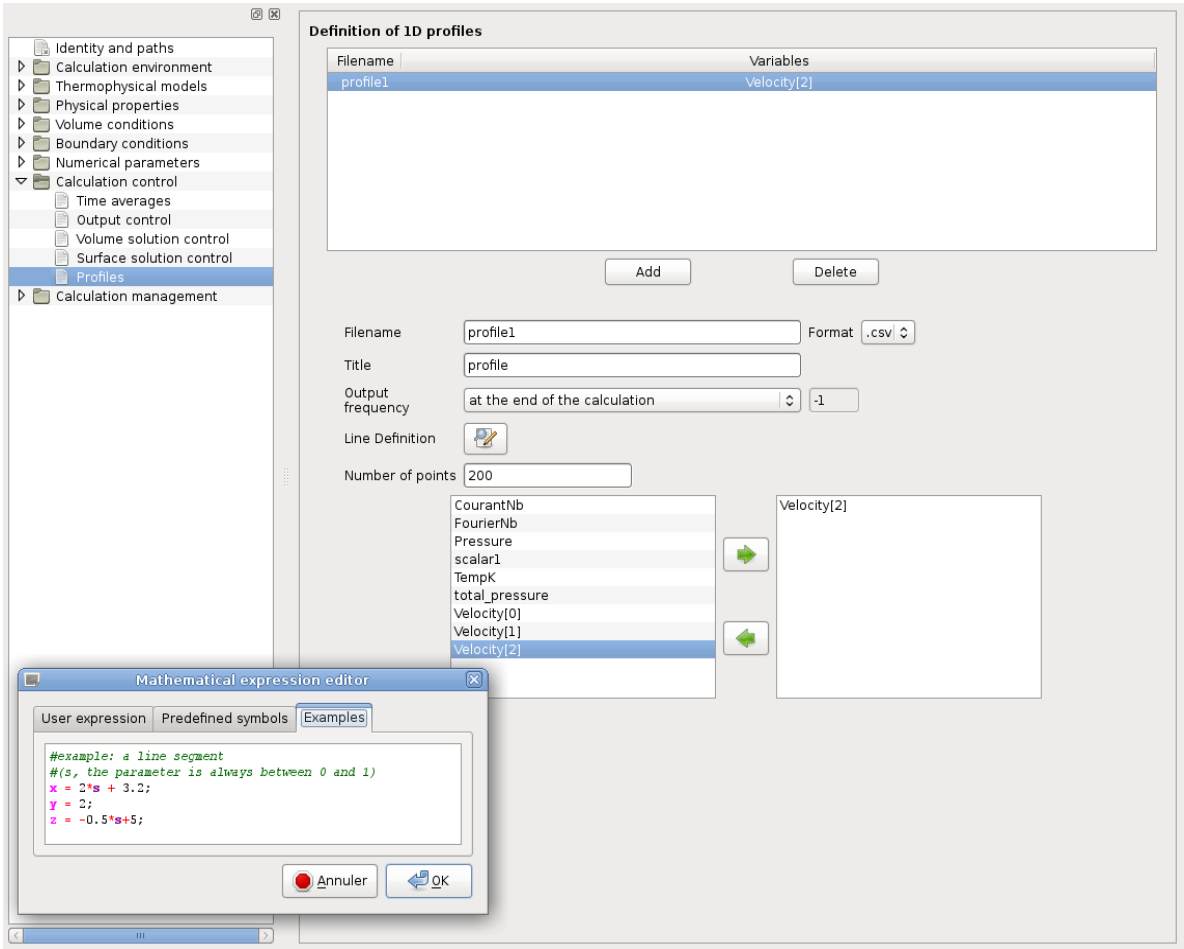

Figure 27: Management of 1D profiles of the solution

For α (of variable index `ial`) in the EBRSM and BL-v2/k models:

$$\alpha = 1$$

For $\tilde{\nu}_t$ in the Spalart-Allmaras model:

$$\tilde{\nu}_t = 0.02\sqrt{\frac{3}{2}}(\texttt{uref})(\texttt{almax})$$

The subroutine `cs_user_initialization` allows if necessary to initialise certain variables to values closer to their estimated final values, in order to obtain a faster convergence.

This subroutine allows also the user to make a non-standard initialisation of physical parameters (density, viscosity, ...), to impose a local value of the time step, or to modify some parameters (time step, variable specific heat, ...) in the case of a calculation restart.

<u>NOTE: VALUE OF THE TIME STEP</u>

- For calculations with constant and uniform time step (`idtvar`=0), the value of the time step is `dtref`, given in the parametric file of the interface or in `cs_user_parameters.f90`.

- For calculations with a non-constant time step (`idtvar`=1 or 2), which is not a calculation restart, the value of `dtref` given in the parametric file of the interface or in `cs_user_parameters.f90` is used to initialise the time step.
- For calculations with a non-constant time step (`idtvar`=1 or 2) which is a restart of a calculation whose time step type was different (for instance, restart using a variable time step of a calculation run using a constant time step), the value of `dtref`, given in the parametric file of the interface or in `cs_user_parameters.f90`, is used to initialise the time step.
- For calculations with non-constant time step (`idtvar`=1 or 2) which is a restart of a calculation whose time step type was the same (for instance, restart with `idtvar`=1 of a calculation run with `idtvar`=1), the time step is read from the restart file and the value of `dtref` given in the parametric file of the interface, or in `cs_user_parameters.f90`, is not used.

It follows, that for a calculation with a non-constant time step (`idtvar`=1 or 2) which is a restart of a calculation in which `idtvar` had the same value, `dtref` does not allow to modify the time step. The user subroutine `cs_user_initialization` allows modifying the array `dt` which contains the value of the time step read from the restart file (array whose size is `ncelet`, defined at the cell centres whatever the chosen time step type is).

6.4 Manage boundary conditions

The boundary conditions can be specified in the Graphical User Interface (GUI) under the heading "Boundary conditions" or in the user subroutine `cs_user_boundary_conditions` called every time step. With the GUI, each region and the type of boundary condition associated to it are defined in Figure 28. Then, the parameters of the boundary condition are specified in Figure 29. The colors of the boundary faces may be read directly from a "listing" file created by the Preprocessor. This file can be generated directly by the interface under the heading "Definition of boundary regions → Add from Preprocessor listing → import groups and references from Preprocessor listing", see Figure 28.

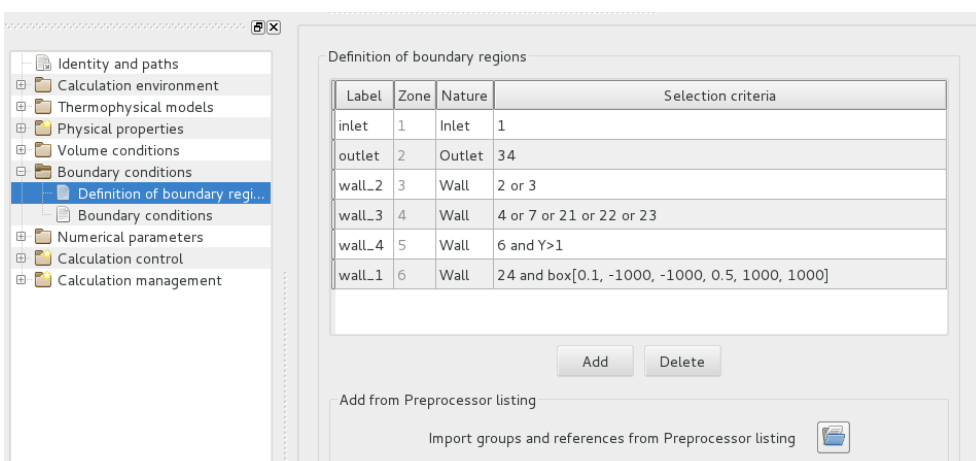

Figure 28: Definition of the boundary conditions

`cs_user_boundary_conditions` is the second compulsory subroutine for every calculation launched without interface (except in the case of specific physics where the corresponding boundary condition user subroutine must be used).

When using the interface, only complex boundary conditions (input profiles, conditions varying in time, ...) need to be defined with `cs_user_boundary_conditions`. In the case of a calculation launched without the interface, all the boundary conditions must appear in `cs_user_boundary_conditions`.

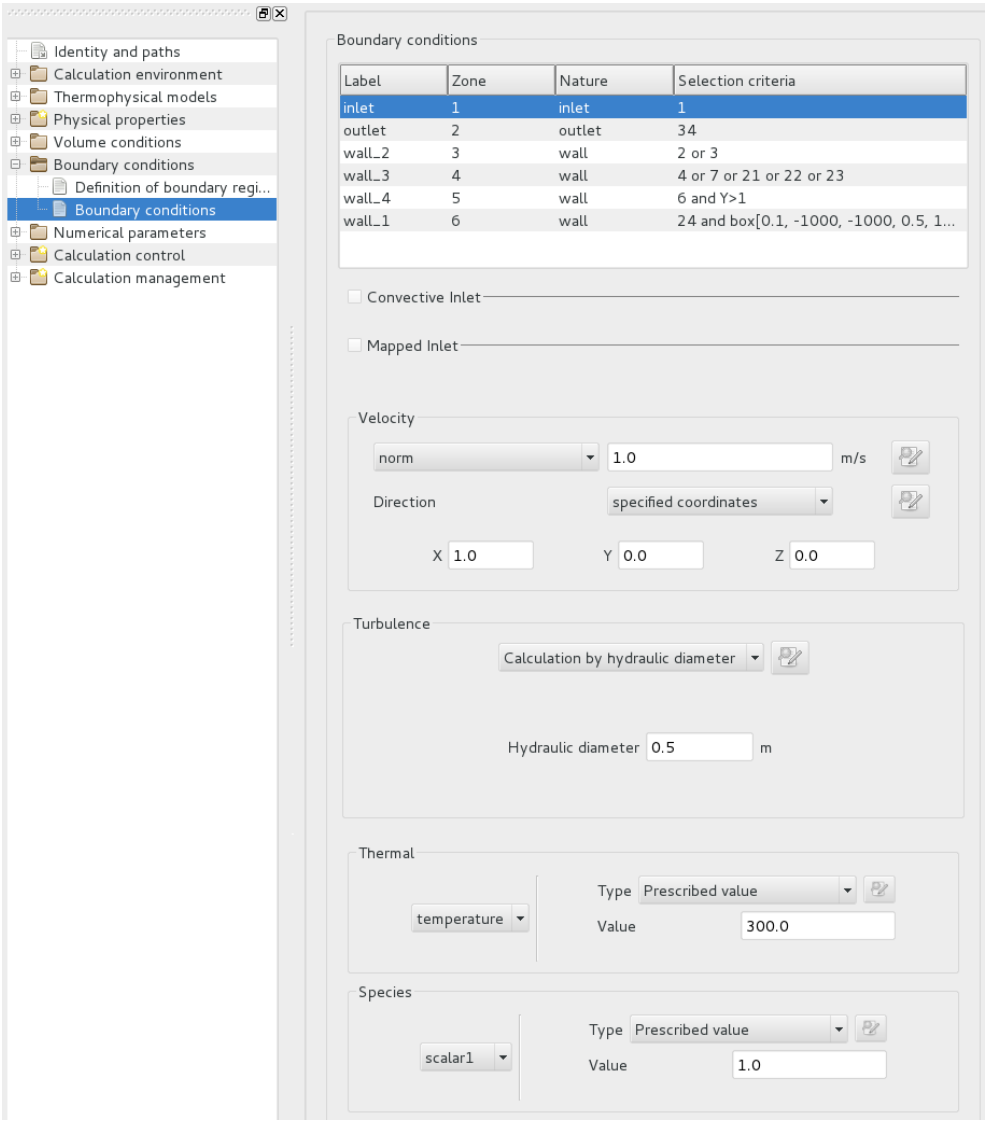

Figure 29: Parameters of the boundary conditions

`cs_user_boundary_conditions` is essentially constituted of loops on boundary face subsets. Several sequences of `call getfbr ('criterion', nlelt, lstelt)` (cf. §3.9.4) allow selecting the boundary faces with respect to their group(s), their color(s) or geometric criteria. If needed, geometric and physical variables are also available to the user. These allow him to select the boundary faces using other criteria.

For more details about the treatment of boundary conditions, the user may refer to the theoretical and computer documentation [11] of the subroutine `condli` (for wall conditions, see `clptur`) (to access this document on a workstation, use `code_saturne info --guide theory`).

From the user point of view, the boundary conditions are fully defined by three arrays[20]: `itypfb(nfabor)`, `icodcl(nfabor,nvar)` and `rcodcl(nfabor,nvar,3)`.

- `itypfb(ifac)` defines the type of the face `ifac` (input, wall, ...).
- `icodcl(ifac,ivar)` defines the type of boundary condition for the variable `ivar` on the face

[20]Except with Lagrangian boundary condition

`ifac` (Dirichlet, flux ...).

- `rcodcl(ifac,ivar,.)` gives the numerical values associated with the type of boundary condition (value of the Dirichlet condition, of the flux ...).

In the case of standard boundary conditions (see §6.4.1), it is sufficient to complete `itypfb(ifac)` and parts of the array `rcodcl`; the array `icodcl` and most of `rcodcl` are filled automatically. For non-standard boundary conditions (see §6.4.2), the arrays `icodcl` and `rcodcl` must be fully completed.

6.4.1 Coding of standard boundary conditions

The standard keywords used by the indicator `itypfb` are: `ientre`, `iparoi`, `iparug`, `isymet`, `isolib`, `ifrent`, `ifresf`, `i_convective_inlet` and `iindef`.

- If `itypfb=ientre`: inlet face.

  → Zero-flux condition for pressure and Dirichlet condition for all other variables. The value of the Dirichlet condition must be given in `rcodcl(ifac,ivar,1)` for every value of `ivar`, except for `ivar=ipr`. The other values of `rcodcl` and `icodcl` are filled automatically.

- If `itypfb=iparoi`: smooth solid wall face, impermeable and with friction.

  → the eventual sliding wall velocity of the face is found in `rcodcl(ifac,ivar,1)` (`ivar` being `iu`, `iv` or `iw`). The initial values of `rcodcl(ifac,ivar,1)` are zero for the three velocity components (and therefore are to be specified only if the velocity is not equal to zero).
  WARNING: the wall sliding velocity must belong to the boundary face plane. For safety, the code only uses the projection of this velocity on the face. As a consequence, if the velocity specified by the user does not belong to the face plane, the wall sliding velocity really taken into account will be different.

  → For scalars, two kinds of boundary conditions can be defined:

  ⇝ Imposed value at the wall. The user must write
  `icodcl(ifac,ivar)`=5
  `rcodcl(ifac,ivar,1)`=imposed value

  ⇝ Imposed flux at the wall. The user must write
  `icodcl(ifac,ivar)`=3
  `rcodcl(ifac,ivar,3)`=imposed flux value (depending on the variable, the user may refer to the case `icodcl`=3 of § 6.4.2 for the flux definition).

  ⇝ If the user does not fill these arrays, the default condition is zero flux.

- If `itypfb=iparug`: rough solid wall face, impermeable and with friction.

  → the eventual moving velocity of the wall tangent to the face is given by `rcodcl(ifac,ivar,1)` (`ivar` being `iu`, `iv` or `iw`). The initial value of `rcodcl(ifac,ivar,1)` is zero for the three velocity components (and therefore must be specified only in the case of the existence of a slipping velocity).
  WARNING: the wall moving velocity must be in the boundary face plane. By security, the code uses only the projection of this velocity on the face. As a consequence, if the velocity specified by the user is not in the face plane, the wall moving velocity really taken into account will be different.

  → The dynamic roughness must be specified in `rcodcl(ifac,iu,3)`. The values of `rcodcl(ifac,iv,3)` stores the thermal and scalar roughness. The values of `rcodcl(ifac,iw,3)` is not used.

  → For scalars, two kinds of boundary conditions can be defined:

⇝ Imposed value at the wall. The user must write
`icodcl(ifac,ivar)`=6
`rcodcl(ifac,ivar,1)`=imposed value

⇝ Imposed flux at the wall. The user must write
`icodcl(ifac,ivar)`=3
`rcodcl(ifac,ivar,3)`= imposed flux value (definition of the flux condition according to the variable, the user can refer to the case `icodcl`=3 of the paragraph 6.4.2).

⇝ If the user does not complete these arrays, the default condition is zero flux.

- If `itypfb=isymet`: symmetry face (or wall without friction).
  → Nothing to be writen in `icodcl` and `rcodcl`.
- If `itypfb=isolib`: free outlet face (or more precisely free inlet/outlet with forced pressure)
  → The pressure is always treated with a Dirichlet condition, calculated with the constraint $\frac{\partial}{\partial n}\left(\frac{\partial P}{\partial \tau}\right) = 0$. The pressure is set to P_0 at the first `isolib` face met. The pressure calibration is always done on a single face, even if there are several outlets.
  → If the mass flow is coming in, the velocity is set to zero and a Dirichlet condition for the scalars and the turbulent quantities is used (or zero-flux condition if no Dirichlet value has been specified).
  → If the mass flow is going out, zero-flux condition are set for the velocity, the scalars and the turbulent quantities.
  → Nothing is written in `icodcl` or `rcodcl` for the pressure or the velocity. An optional Dirichlet condition can be specified for the scalars and turbulent quantities.
- If `itypfb=ifrent`: free outlet, free inlet (based on Bernoulli relationship) face.
  → if outlet, the equivalent to standard outlet. In case of ingoing flux, the Benoulli relationship which links pressure and velocity is used (see the thory guide for more information). An additional head loss modelling what is going on outward of the domain can be added by the user.
- If `itypfb=ifresf`: free-surface boundary condition.
- If `itypfb=i_convective_inlet`: inlet with zero diffusive flux for all transported variables (species and velocity). This allows to exactly impose the ingoing flux.
- If `itypfb=iindef`: undefined type face (non-standard case).
  → Coding is done in a non-standard way by filling both arrays `rcodcl` and `icodcl` (see § 6.4.2).

NOTES

• Whatever is the value of the indicator `itypfb(ifac)`, if the array `icodcl(ifac,ivar)` is modified by the user (*i.e.* filled with a non-zero value), the code will not use the default conditions for the variable `ivar` at the face `ifac`. It will take into account only the values of `icodcl` and `rcodcl` provided by the user (these arrays must then be fully completed, like in the non-standard case).
For instance, for a normal symmetry face where scalar 1 is associated with a Dirichlet condition equal to 23.8 (with an infinite exchange coefficient):

```
itypfb(ifac)=isymet
icodcl(ifac,isca(1))=1
rcodcl(ifac,isca(1),1)=23.8
```

(`rcodcl(ifac,isca(1),2)=rinfin` is the default value, therefore it is not necessary to specify a value)
The boundary conditions for the other variables are automatically defined.

• The user can define new types of boundary faces. He only must choose a value N and to fully specify the boundary conditions (see §6.4.2). He must specify `itypfb(ifac)`$=N$ where N range is 1 to `ntypmx` (maximum number of boundary face types), and of course different from the values `ientre`, `iparoi`, `iparug`, `isymet`, `isolib` and `iindef` (the values of these variables are given in the `paramx` module). This allows to easily isolate some boundary faces, in order for instance to calculate balances.

6.4.2 Coding of non-standard boundary conditions

Ifa face does not correspond to a standard type, the user must completely fill the arrays `itypfb`, `icodcl` and `rcodcl`. `itypfb(ifac)` is then equal to `iindef` or another value defined by the user (see note at the end of § 6.4.1). The arrays `icodcl` and `rcodcl` must be filled as follows:

- If `icodcl(ifac,ivar)`=1: Dirichlet condition at the face `ifac` for the variable `ivar`.
  - → `rcodcl(ifac,ivar,1)` is the value of the variable `ivar` at the face `ifac`.
  - → `rcodcl(ifac,ivar,2)` is the value of the exchange coefficient between the outside and the fluid for the variable `ivar`. An infinite value (`rcodcl(ifac,ivar,2)=rinfin`) indicates an ideal transfer between the outside and the fluid (default case).
  - → `rcodcl(ifac,ivar,3)` is not used.
  - → `rcodcl(ifac,ivar,1)` has the units of the variable `ivar`, *i.e.*:
    - ⇝ m/s for the velocity
    - ⇝ m^2/s^2 for the Reynolds stress
    - ⇝ m^2/s^3 for the dissipation
    - ⇝ Pa for the pressure
    - ⇝ °C for the temperature
    - ⇝ $J.kg^{-1}$ for the enthalpy
    - ⇝ $°C^2$ for temperature fluctuations
    - ⇝ $J^2.kg^{-2}$ for enthalpy fluctuations
  - → `rcodcl(ifac,ivar,2)` has the following units (defined in such way that when multiplying the exchange coefficient by the variable, the given flux has the same units as the flux defined below when `icodcl=3`):
    - ⇝ $kg.m^{-2}.s^{-1}$ for the velocity
    - ⇝ $kg.m^{-2}.s^{-1}$ for the Reynolds stress
    - ⇝ $s.m^{-1}$ for the pressure
    - ⇝ $W.m^{-2}.°C^{-1}$ for the temperature
    - ⇝ $kg.m^{-2}.s^{-1}$ for the enthalpy
- If `icodcl(ifac,ivar)=2`: radiative outlet at the face `ifac` for the variable `ivar`. It reads $\dfrac{\partial Y}{\partial t} + C\dfrac{\partial Y}{\partial n} = 0$, where C is a to be defined celerity of radiation.
  - → `rcodcl(ifac,ivar,3)` is not used.
  - → `rcodcl(ifac,ivar,1)` is the flux value of `ivar` at the cell center I', projection of the center of the adjacent cell on the straight line perpendicular to the boundary face and crossing its center, at the previous time step. It corresponds to:
  - → `rcodcl(ifac,ivar,2)` is CFL number based on the parameter C, the distance to the boundary $I'F$ and the time step: $CFL = \dfrac{Cdt}{I'F}$,
- If `icodcl(ifac,ivar)=3`: flux condition at the face `ifac` for the variable `ivar`.
  - → `rcodcl(ifac,ivar,1)` and `rcodcl(ifac,ivar,2)` are not used.

→ `rcodcl(ifac,ivar,3)` is the flux value of `ivar` at the wall. This flux is negative if it is a source for the fluid. It corresponds to:

⇝ $-(\lambda_T + C_p \frac{\mu_t}{\sigma_T})\underline{\nabla} T \cdot \underline{n}$ for a temperature (in W/m^2)

$-(\frac{\lambda_T}{C_p} + \frac{\mu_t}{\sigma_h})\underline{\nabla} h \cdot \underline{n}$ for an enthalpy (in W/m^2).

$-(\lambda_\varphi + \frac{\mu_t}{\sigma_\varphi})\underline{\nabla}\varphi \cdot \underline{n}$ in the case of another scalar φ (in $kg.m^{-2}.s^{-1}.[\varphi]$, where $[\varphi]$ are the units of φ).

⇝ $-\Delta t\ \underline{\nabla} P \cdot \underline{n}$ for the pressure (in $kg.m^{-2}.s^{-1}$).

⇝ $-(\mu + \mu_t)\underline{\nabla} U_i \cdot \underline{n}$ for a velocity component (in $kg.m^{-1}.s^{-2}$).

⇝ $-\mu\underline{\nabla} R_{ij} \cdot \underline{n}$ for a R_{ij} tensor component (in W/m^2).

- If `icodcl(ifac,ivar)`=4: symmetry condition, for the symmetry faces or wall faces without friction. This condition can only be used for velocity components ($\underline{U} \cdot \underline{n} = 0$) and the R_{ij} tensor components (for other variables, a zero-flux condition type is usually used).

- If `icodcl(ifac,ivar)`=5: friction condition, for wall faces with friction. This condition can not be applied to the pressure.

  ⇝ For the velocity and (if necessary) the turbulent variables, the values at the wall are calculated from theoretical profiles. In the case of a sliding wall, the three components of the sliding velocity are given by (`rcodcl(ifac,iu,1)`, `rcodcl(ifac,iv,1)`, and `rcodcl(ifac,iw,1)`).
  WARNING: the wall sliding velocity must belong to the boundary face plane. For safety, the code uses only the projection of this velocity on the face. Therefore, if the velocity vector specified by the user does not belong to the face plane, the wall sliding velocity really taken into account will be different.

  ⇝ For other scalars, the condition `icodcl`=5 is similar to `icodcl=1`, but with a wall exchange coefficient calculated from a theoretical law. Therefore, the values of `rcodcl(ifac,ivar,1)` and `rcodcl(ifac,ivar,2)` must be specified: see [11].

- If `icodcl(ifac,ivar)`=6: friction condition, for the rough-wall faces with friction. This condition can not be used with the pressure.

  ⇝ For the velocity and (if necessary) the turbulent variables, the values at the wall are calculated from theoretical profiles. In the case of a sliding wall, the three components of the sliding velocity are given by (`rcodcl(ifac,iu,1)`, `rcodcl(ifac,iv,1)`, and `rcodcl(ifac,iw,1)`).
  WARNING: the wall sliding velocity must belong to the boundary face plane. For safety, the code uses only the projection of this velocity on the face. Therefore, if the velocity vector specified by the user does not belong to the face plane, the wall sliding velocity really taken into account will be different.
  The dynamic roughness height is given by `rcodcl(ifac,iu,3)` only.

  ⇝ For the other scalars, the condition `icodcl`=6 is similar to `icodcl`=1, but with a wall exchange coefficient calculated from a theoretical law. The values of `rcodcl(ifac,ivar,1)` and `rcodcl(ifac,ivar,2)` must therefore be specified: see [11]. The thermal roughness height is then given by `rcodcl(ifac,ivar,3)`.

- If `icodcl(ifac,ivar)`=9: free outlet condition for the velocity. This condition is only applicable to velocity components.
  If the mass flow at the face is negative, this condition is equivalent to a zero-flux condition.
  If the mass flow at the face is positive, the velocity at the face is set to zero (but not the mass flow).
  `rcodcl` is not used.

- If `icodcl(ifac,ivar)`=14: generalized symmetry boundary condition for vectors (Marangoni effect for the velocity for instance). This condition is only applicable to vectors and set a Dirichlet boundary condition on the normal component and a Neumann condition on the tangential components.
  If the three components are `ivar1, ivar2, ivar3`, the required values are:

  → `rcodcl(ifac,ivar1,1)`: Dirichlet value in the x direction.

  → `rcodcl(ifac,ivar2,1)`: Dirichlet value in the y direction.

  → `rcodcl(ifac,ivar3,1)`: Dirichlet value in the z direction.

  → `rcodcl(ifac,ivar1,3)`: flux value for the x direction.

  → `rcodcl(ifac,ivar2,3)`: flux value for the y direction.

  → `rcodcl(ifac,ivar3,3)`: flux value for the z direction.

  Therefore, the code automatically computes the boundary condition to impose to the normal and to the tangential components.

NOTE
• A standard `isolib` outlet face amounts to a Dirichlet condition (`icodcl`=1) for the pressure, a free outlet condition (`icodcl`=9) for the velocity and a Dirichlet condition (`icodcl`=1) if the user has specified a Dirichlet value or a zero-flux condition (`icodcl`=3) for the other variables.

6.4.3 Checking of the boundary conditions

The code checks the main compatibilities between the boundary conditions. In particular, the following rules must be respected:
• On each face, the boundary conditions of the three velocity components must belong to the same type. The same is true for the components of the R_{ij} tensor.
• If the boundary conditions for the velocity belong to the "sliding" type (`icodcl`=4), the conditions for R_{ij} must belong to the "symmetry" type (`icodcl`=4), and vice versa.
• If the boundary conditions for the velocity belong to the "friction" type (`icodcl`=5 or 6), the boundary conditions for the turbulent variables must belong to the "friction" type, too.
• If the boundary condition of a scalar belongs to the "friction" type, the boundary condition of the velocity must belong to the "friction" type, too.

In case of mistakes, if the post-processing output is activated (which is the default setting), a special error output, similar to the mesh format, is produced in order to help correcting boundary condition definitions.

6.4.4 Sorting of the boundary faces

In the code, it may be necessary to have access to all the boundary faces of a given type. To ease this kind of search, an array made of sorted faces is automatically filled (and updated at each time step): `itrifb(nfabor)`.
`ifac=itrifb(i)` is the number of the ith face of type 1.
`ifac=itrifb(i+n)` is the number of the ith face of type 2, if there are n faces of type 1.
... etc.

Two auxiliary arrays of size `ntypmx` are also defined.
`idebty(ityp)` is the index corresponding to the first face of type `ityp` in the array `itrifb`.
`ifinty(ityp)` is the index corresponding to the last face of type `ityp` in the array `itrifb`.

Therefore, a value `ifac0` found between `idebty(ityp)` and `ifinty(ityp)` is associated to each face `ifac` of type `ityp=itypfb(ifac)`, so that `ifac=itrifb(ifac0)`.

If there is no face of type `ityp`, the code set
`ifinty(ityp)=idebty(ityp)-1`,
which enables to bypass, for all the missing `ityp`, the loops such as
`do ii=idebty(ityp),ifinty(ityp)`.

The values of all these indicators are displayed at the beginning of the code execution listing.

6.4.5 Boundary conditions with LES

6.4.5.1 Vortex method

The subroutine `usvort` allows generating the unsteady inlet boundary conditions for the LES by the vortex method. The method is based on the generation of vortices in the 2D inlet plane with help from the pre-defined functions. The fluctuation normal to the inlet plane is generated by a Langevin equation. It is in the subroutine `usvort` where the parameters of this method are given.

Subroutine called at each time step

To allow the application of the vortex method, an indicator must be informed of the method in the user subroutine `cs_user_parameters.f90` (ivrtex=1)

The subroutine `usvort` contains 3 separate parts:

- The 1st part defines the number of inlets concerned with the vortex method (`nnentt`) and the number of vortex for each inlet (`nvort`), where `ient` represents the number of inlets.
- The 2nd part (`iappel=1`) defines the boundary faces at which the vortex method is applicable. The `irepvo` array is informed by `ient` which defines the number of inlets concerned with the vortex (essentially, the vortex method can be applied with many independent inlets).
- The 3rd section defines the main parameters of the method at each inlet. With the complexity of any given geometry, 4 cases are distinguished (the first 3 use the data file `ficvor` and in the final case only 1 initial velocity and energy are imposed.):
  * `icas`=1, For the outlet of a rectangular pipe; 1 boundary condition is defined for each side of the rectangle taking into account their interaction with the vortex.
  * `icas`=2, For the outlet of a circular pipe; the entry face is considered as a wall (as far as interaction with the vortex is concerned)
  * `icas`=3, For inlets of any geometry; no boundary conditions are defined at the inlet face (i.e no specific treatment on the interaction between the vortex and the boundary)
  * `icas`=4, similar to `icas`=3 except the data file is not used (`ficvor`); the outflow parameters are estimated by the code from the global data (initial velocity, level of turbulence and dissipation), information which is supplied by the user.

  When the geometry allows, cases 1 and 2 are used. Case 4 is only used if it is not possible to use the other three.

  In the first 3 cases, the 2 base vectors in the plane of each inlet must be defined (vectors `dir1` and `dir2`). The 3rd vector is automatically calculated by the code, defined as a product of `dir1` and `dir2`. `dir1` and `dir2` must be chosen imperatively to give (`cen`, `dir1`, `dir2`) an orthogonal reference of the inlet plane and so `dir3` is oriented in the entry domain. If `icas`=2, the `cen` position must be the center of gravity of the rectangle or disc.

  The reference points (`cen`, `dir1`, `dir2`, `dir3`) define the values of the variable in the `ficvor` file. In the case where `icas`=4, the vectors `dir1` and `dir2` are generated by the code.

  If `icas`=1, the boundary conditions at the rectangle's edges must be defined. They are defined in the array `iclvor`. `iclvor(ii,ient)` represents the standard boundary conditions at the edge II ($1 \leqslant II \leqslant 4$) of the inlet `ient`. The code for the boundary conditions is as follows:

* `iclvor`=1 for a wall
* `iclvor`=2 for symmetry
* `iclvor`=3 for periodicity of translation (the face corresponding to periodicity will automatically be taken as 3)

The 4 edges are numbered relative to the directions `dir1` and `dir2` as shown in Figure 30:

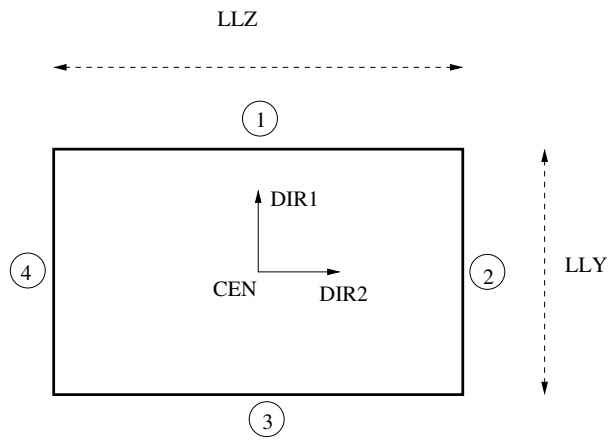

Figure 30: Numbering of the edges of a rectangular inlet(`icas`=1) treated by the vortex method

If `icas`=1, the user must define `llx` and `lly` which give the lengths of the rectangular pipe in the directions `dir1` and `dir2`.
If `icas`=2, `lld` represents the diameter of the circular pipe. If `icas`=4, `udebit`, `kdebit` and `edebit` are defined for each inlet, these give respectively, initial speed, turbulent energy level and the dissipation level. These can be used to obtain their magnitude using the correlations in the user routine `cs_user_boundary_conditions` for fully developed flow in a pipe.

The independent parameters are defined as follows:

* `itmpl` represents the indicator of the advancement in time of the vortex. If `itmpli`=1, the vortex will be regenerated after a fixed time of `tmplim` second (defined as `itmpli`=1). If `itmpli`=2, following the data indicated in `ficvor` file, the vortex will have a variable life span equal to $5C_\mu \frac{k^{\frac{3}{2}}}{\varepsilon U}$, where $C_\mu = 0.09$ and k, ε and U represent respectively, turbulent energy, turbulent dissipation and the convective velocity in the direction normal to the inlet plane.
* `xsgmvo` represents the support functions used in the vortex method. These are representative of the eddy sizes entered in the vortex method. `isgmvo` is used to define their size: if `isgmvo`=1, `xsgmvo` will be constant across the inlet face and is defined in `usvort`, if `isgmvo`=2, `xsgmvo` will be variable and equal to the mixing length of the standard $k-\varepsilon$ model ($C_\mu{}^{\frac{3}{4}} \frac{k^{\frac{3}{2}}}{\varepsilon}$), if `isgmvo`=3, `xsgmvo` will be equal to the maximum of L_t et L_K where L_t and L_K are the $\frac{\partial U}{\partial y}\frac{\partial U}{\partial y}$ Taylor and Kolmogrov coefficients ($L_T = (5\nu\frac{k}{\varepsilon})^{\frac{1}{2}}$, $L_K = 200(\frac{\nu^3}{\varepsilon})^{\frac{1}{4}}$).
* `idepvo` gives the vortex displacement method in the 2D inlet plane (the vortex method is a Lagrangian method in which the eddy centres are replaced by a set velocity). If `idepvo`=1, the velocity displacement referred to by `ud` which is the vortex following a random sampling (a sample number r, is taken for each vortex, at each time step and for each direction and the center of the vortex is replaced by the 2 principal directions, $r\mathrm{ud}\Delta t$ where Δt is the time step of the calculation). If `idepvo`=2, the vortex will be convected by itself (with the speed given by the time step before the vortex method)

A data file, `ficvor`, must be defined in the cases of `icas`=1,2,3, for each inlet. The data file must contain the following data in order (x, y, U, $\frac{\partial U}{\partial y}$, k, ε). The number of lines of the file is given by the integer `ndat`. x and y are the co-ordinates in the inlet plane defined by the vectors `dir1` and `dir2`. U, k and ε are respectively, the average speed normal to the inlet, the turbulent energy and the turbulent dissipation. $\frac{\partial U}{\partial y}$ is the derivative in the direction normal to the inlet boundary in the cases, `icas`=1, `icas`=2. Where `icas`=3 and `icas`=4 this variable is not applied (it is given the value 0) so the Langevin equations, used to generate fluctuations normal to the inlet plane, is de-activated (the fluctuations normal to the inlet is 0 on both these cases). Note that the application of many different test of the Langevin equation doesn't have a notable influence on the results and that, by contrast it simply increases the computing time per iteration and so it decreases the random sampling which slows down the pressure solver. The interpolation used in the vortex method is defined by the function `phidat`. An example is given at the end of the subroutine `usvort` where the user can define the interpolation required. In the `phidat` function, `xx` and `yy` are the co-ordinates by which the value of `phidat` is calculated. `xdat` and `ydat` are the co-ordinates in the `ficvor` file. `vardat` is the value of the `phidat` function with the co-ordinates `xdat` and `ydat` (given in the `ficvor` file). Note that using an indicator `iii` accelerates the calculations (the user need not modify or delete). The user must also define the parameter `isuivo` which indicates if the vortex was started at 0 or if the file must be re-read (`ficmvo`).

WARNING

- Be sure that the `ficvor` file and the interpolation in the user function `phidat` are compatible (in particular that all the entry region is covered by `ficvor`)
- If the user wants to use a 1D profile in the `dir2` direction, set x =0 in the `ficvor` file and define the interpolation in `phidat`.

6.4.5.2 Synthetic Eddy Method

The user file `cs_user_les_inflow.f90` allows to generate the unsteady boundary conditions for the LES by the Synthetic Eddy Method. The basic principle of this method is illustrated in figure 31: the turbulent fluctuations at the inlet are generated by a set of synthetic eddies advected across the inlet boundaries. The eddies evolve in a virtual "box" surrounding the inlet boudaries and each of them contributes to the normalized velocity fluctuations, depending on its relative position with the inlet faces and on a form function characterizing the shape of the eddies. By this way, the Synthetic Eddy Method provides a coherent flow with a target mean velocity and target Reynolds stresses at LES inlet.

WARNING: As for laminar or RANS inlets, the type of boundary for LES inlets is `ientre`. It has to be specified in the GUI or in the `cs_user_boundary_conditions` surboutine. On the contrary, if Dirichlet values are given for these faces in the GUI or in the `cs_user_boundary_conditions` subroutine (`rcodcl(ifac,ivar,1)` array), they are erased by those provided by the Synthetic Eddy Method.

In the current version of *Code_Saturne*, the Synthetic Eddy Method is not available through the GUI but only through the `cs_user_les_inflow.f90` user file. The user file contains 3 subroutines:

- `cs_user_les_inflow_init` (mandatory): global definition of synthetic turbulence inlets
- `cs_user_les_inflow_define` (mandatory): specific definition of each synthetic turbulence inlet
- `cs_user_les_inflow_advanced` (not mandatory): advanced definition of each synthetic turbulence inlet

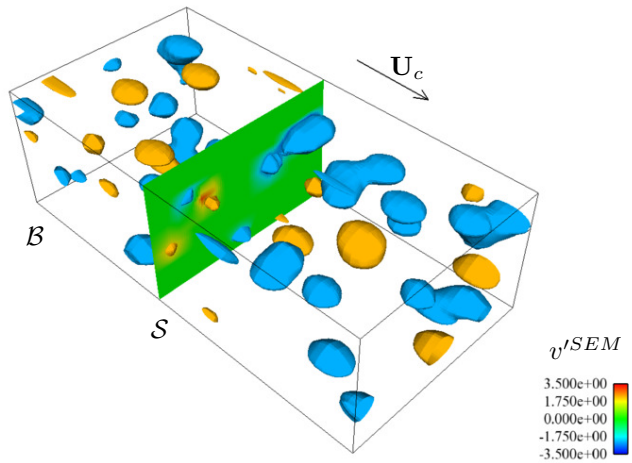

Figure 31: Illustration of the principle of the Synthetic Eddy Method, with $\mathcal{S}$ the inlet boundary, $\mathcal{B}$ the virtual box and $\mathbf{U}_c$ the advection velocity of the eddies

`cs_user_les_inflow_init`: this subroutine defines some global parameters shared by all LES inlets. These parameters are:

- `nent`: number of LES inlet boundaries
- `isuisy`: in case of a restart calculation, it indicates if the synthetic turbulence is re-initialize (0) or read from the previous calculation (1). In that case, the checkpoint folder must contain the `les_inflow` restart file. This file is generated during a computation with synthetic turbulence, at the same physical times as the main and auxiliary restart files.

`cs_user_les_inflow_define`: this subroutine defines the specific parameters of each LES inlet. These parameters are:

- `typent`: type of LES inflow method. The Synthetic Eddy Method corresponds to `typent=3`. For the sake of comparision, other methods can be selected through this user file (see remark 2).
- `nelent`: number of synthetic eddies in the "box". This parameter might be adjusted, depending on the case (in particular the size of the inlet plane and the level of turbulence). As a general rule, the greater is the better since an insufficient number can lead to an intermittent signal while some numerical tests have shown that this parameter does not have a great influence beyond a threshold value. Given the inlet of size h^2 of a shear flow at a given Reynolds number $Re = u_\tau h/\nu$, an appropriate number of eddies can be evaluated by $(Re/50)^3$ (Re and 50 approximates respectively the size, in wall unit, of the largest and the smallest synthetic eddy. Note the latter can depend on the grid size, see remark 1).
- `iverbo`: level of verbosity in the listing. `iverbo=1` provides mainly informations about the size of the eddies and the size of the "box" surrounding the inlet boundary.
- `nfbent` and `lfbent`: number and list of boundary faces composing the LES inlet boundary.
- `vitent`: reference mean velocity at inlet. This parameter imposes the target mean velocity at inlet. A finer (non homogeneous) definition of the mean velocity can be done in the `cs_user_les_inflow_advanced` subroutine (see below).
- `enrent`: reference turbulence kinetic energy k at inlet. This parameter imposes the target Reynolds stresses R_{ij} at inlet, computed by $R_{ij} = \frac{2}{3}k\delta_{ij}$ (isotropy). A finer (non isotropic and/or non homogeneous) definition of the Reynolds stresses can be done in the `cs_user_les_inflow_advanced` subroutine (see below).

- `dspent`: reference dissipation rate ε at inlet. This parameter is used to compute the size of the synthetic eddies (see remark 1). A finer (non homogeneous) definition of the dissipation rate can be done in the `cs_user_les_inflow_advanced` subroutine (see below).

`cs_user_les_inflow_advanced`: this optional subroutine enables to give an accurate (non homogeneous) specification of inflow statistics: mean velocity (`uvwent` array), Reynolds stresses (`rijent` array) and dissipation rate (`epsent` array). In that case, this accurate specification replaces the one given in `cs_user_les_inflow_define` subroutine (`vitent`, `enrent` and `dspent` variables).

REMARK 1: The specification of the dissipation rate ε at inlet is used to compute the size σ_i of the synthetic eddies in the i cartesian direction. One has:

$$\sigma_i = \max\left\{C\frac{\left(\frac{3}{2}R_{ii}\right)^{3/2}}{\varepsilon}, \Delta\right\}, \qquad C = 0.5.$$

Δ is a reference size of the grid, in order to assume that all synthetic eddies are discretized. In the implementation of *Code_Saturne*, it is computed at each inlet boundary face F as:

$$\Delta = 2 \max_{i\leq 3, V\in\mathcal{V}}\left\{\left|x_i^V - x_i^C\right|\right\}$$

with $\mathcal{V}$ the subset of the vertices of the boundary face F and C the cell adjacent to F.

REMARK 2: For the sake of comparison, others LES inflow methods are available through the `cs_user_les_inflow.f90` user file, in addition to the Synthetic Eddy Method:

- The Batten method corresponds to `typent=2` in `cs_user_les_inflow_define` subroutine. With this method, the inflow velocity signal is the superposition of several Fourier modes. The number of modes is indicated through the `nelent` keyword. As for Synthetic Eddy Method, the mean velocity, the turbulent kinetic energy and the dissipation rate have to be specified at inlet: either giving their reference values (`vitent`, `enrent` and `dspent`) in the `cs_user_les_inflow_define` subroutine, either providing an accurate local description in the `cs_user_les_inflow_advanced` subroutine.

- `typent=1`: turbulent fluctuations are given by a Gaussian noise. The mean velocity and Reynolds stresses have to be specified (in `cs_user_les_inflow_define` or in `cs_user_les_inflow_advanced`). The other parameters of the user subroutines are useless. The turbulent fluctuations provided by this method are much less realistic than those provided by the Synthetic Eddy Method or the Batten method. Especially for low Reynolds number flows, this could lead to the rapid dissipation of this fluctuations and the laminarization of the flow.

- `typent=0`: No fluctuation. This method does not require any parameter. It should be reserved to regions where the flow is laminar.

6.5 Manage the variable physical properties

6.5.1 Basic variable physical properties

When the fluid properties are not constant, the user is offered the choice to define the variation laws in the Graphical User Interface (GUI) or in the subroutine `cs_user_physical_properties` which is called at each time step. In the GUI, in the item "Fluid properties" under the heading "Physical properties", the variation laws are defined for the fluid density, viscosity, specific heat, thermal conductivity and scalar diffusivity through the use of a formula editor, see Figure 32 and Figure 33.

If necessary, all the variation laws related to the fluid physical properties are written in the subroutine `cs_user_physical_properties`.

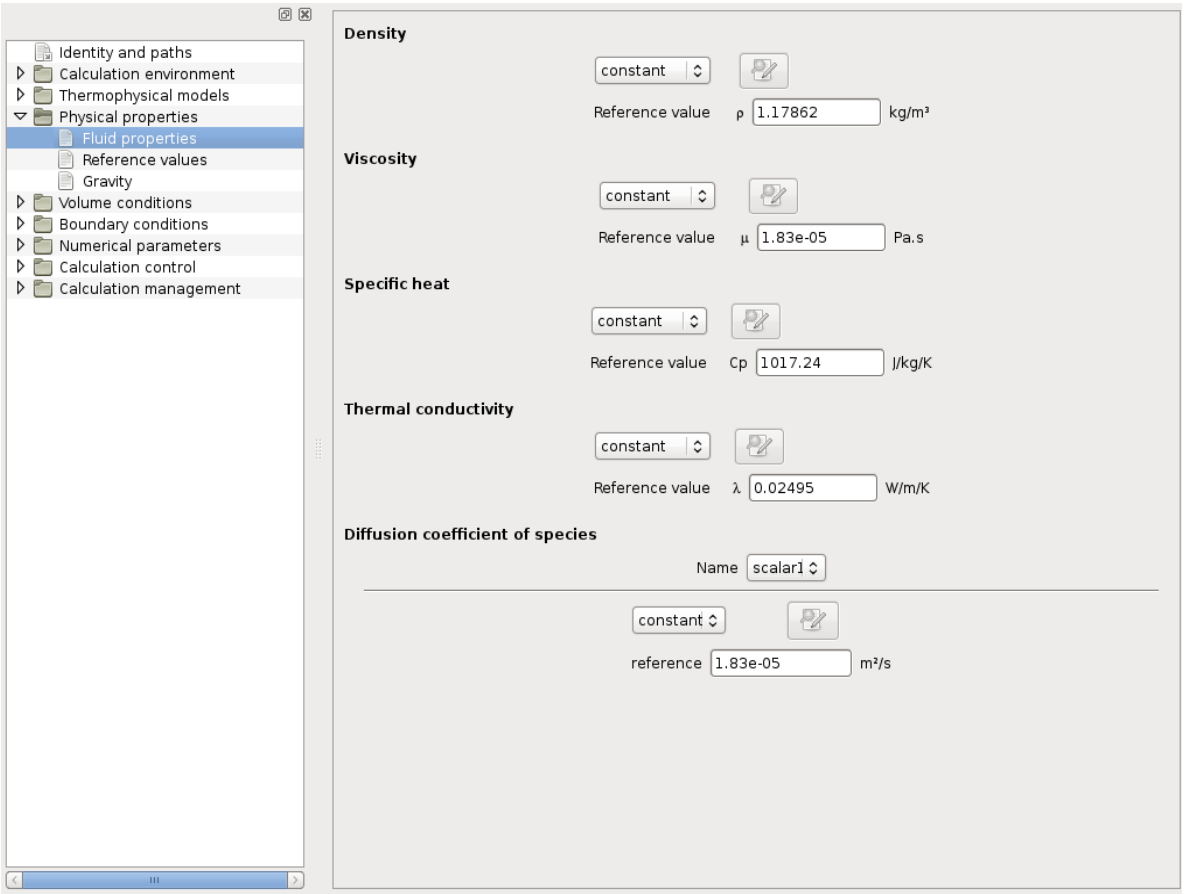

Figure 32: Physical properties - Fluid properties

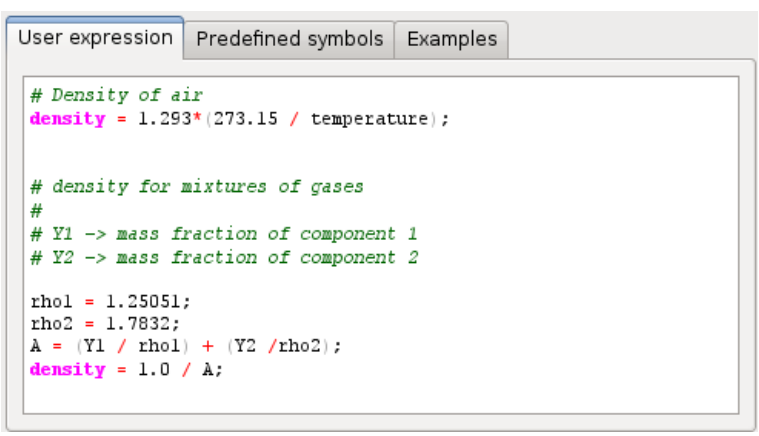

Figure 33: Definition of a user law for the density

The validity of the variation laws must be checked, particularly when non-linear laws are defined (for instance, a third-degree polynomial law may produce negative density values).

WARNING

- If the user wishes to impose a variable density or variable viscosity in `usphyv`, it must be flagged

either in the interface or in `cs_user_parameters.f90`(`irovar`=1, `ivivar`=1).

- In order to impose a physical property (ρ, μ, λ, C_p)[21], a reference value should be provided in the interface or in `cs_user_parameters.f90` (in particular for ρ, the pressure will be function of $\rho_0 g z$)

- By default, the C_p coefficient and the diffusivity for the scalars `iscal` (λ_T for the temperature) are considered as constant in time and uniform in space, with the values `cp0` and `visls0(iscal)` specified in the interface or in `cs_user_parameters.f90`.
  To assign a variable value to C_p, the user **must** specify it in the interface (with a user law) or assign the value 1 to `icp` in `cs_user_parameters.f90`, and fill for each cell `iel` the array `cpro_cp` which can be retrieved by calling `field_get_val_s(icp, cpro_cp)` in `cs_user_physical_properties`.
  NB: completing the array `cpro_cp` while `icp=0` induces array overwriting problems and produces wrong results.

- In the same way, to have variable diffusivities for the scalars `iscal`, the user **must** specify it in the interface (with a user law) or calling `field_set_key_int(ivarfl(isca(iscal)), kivisl, 0)` in `cs_user_parameters.f90` (in `usipsu`), and complete for each cell `iel` the values array of the field id `ifcvsl` returned by calling `field_get_key_id(ivarfl(isca(iscal)), kivisl, ifcvsl)` in `cs_user_physical_properties`.

  Note: The scalar diffusivity id must not be defined for user scalars representing the average of the square of the fluctuations of another scalar, because the diffusivity of a user scalar `jj` representing the average of the square of the fluctuations of a user scalar `kk` comes directly from the diffusivity of this last scalar. In particular, the diffusivity of the scalar `jj` is variable if the diffusivity of `kk` is variable.

6.5.2 Modification of the turbulent viscosity

The subroutine `usvist` is used to modify the calculation of the turbulent viscosity, *i.e.* μ_t in $kg.m^{-1}.s^{-1}$ (this piece of information, at the mesh cell centres, is conveyed by the variable `cpro_visct` which can be retrieved by calling `field_get_val_s(ivisct, cpro_cp)`). The subroutine is called at the beginning of every time step, after the calculation of the physical parameters of the flow and of the "conventional" value of μ_t corresponding to the chosen turbulence model (indicator `iturb`).
WARNING: The calculation of the turbulent viscosity being a particularly sensible stage, a wrong use of `usvist` *may seriously distort the results.*

6.5.3 Modification of the variable C of the dynamic LES model

Subroutine called every time step in the case of LES with the dynamic model.

The subroutine `ussmag` is used to modify the calculation of the variable C of the LES sub-grid scale dynamic model.

It worth to recalling that the LES approach introduces the notion of filtering between large eddies and small motions. The solved variables are said to be filtered in an "implicit" way. Sub-grid scale models ("dynamic" models) introduce in addition an explicit filtering.

The notations used for the definition of the variable C used in the dynamic models of *Code_Saturne* are specified below. These notations are the ones assumed in the document [3], to which the user may refer to for more details.

The value of a filtered by the explicit filter (of width $\widetilde{\overline{\Delta}}$) is called $\widetilde{a}$ and the value of a filtered by the

[21] Except for some specific physics

implicit filter (of width $\overline{\Delta}$) is called $\overline{a}$. We define:

$$\begin{array}{ll} \overline{S}_{ij} = \frac{1}{2}(\frac{\partial \overline{u}_i}{\partial x_j} + \frac{\partial \overline{u}_j}{\partial x_i}) & ||\overline{S}|| = \sqrt{2\overline{S}_{ij}\overline{S}_{ij}} \\ \alpha_{ij} = -2\widetilde{\overline{\Delta}}^2 ||\widetilde{\overline{S}}||\widetilde{\overline{S}}_{ij} & \beta_{ij} = -2\overline{\Delta}^2||\overline{S}||\overline{S}_{ij} \\ L_{ij} = \widetilde{\overline{u}_i\overline{u}_j} - \widetilde{\overline{u}}_i\widetilde{\overline{u}}_j & M_{ij} = \alpha_{ij} - \widetilde{\beta}_{ij} \end{array} \tag{4}$$

In the framework of LES, the total viscosity (molecular + sub-grid) in $kg.m^{-1}.s^{-1}$ may be written in *Code_Saturne*:

$$\begin{array}{lcll} \mu_{\text{total}} & = & \mu + \mu_{\text{sub-grid}} & \text{if } \mu_{\text{sub-grid}} > 0 \\ & = & \mu & \text{otherwise} \\ \text{with } \mu_{\text{sub-grid}} & = & \rho C\overline{\Delta}^2||\overline{S}|| & \end{array} \tag{5}$$

$\overline{\Delta}$ is the width of the implicit filter, defined at the cell Ω_i by
$\overline{\Delta} = XLESFL * (ALES * |\Omega_i|)^{BLES}$.

In the case of the Smagorinsky model (`iturb=40`), C is a constant which is worth C_s^2. C_s^2 is the so-called Smagorinsky constant and is stored in the variable *csmago*.

In the case of the dynamic model (`iturb=41`), C is variable in time and in space. It is determined by $C = \dfrac{M_{ij}Lij}{M_{kl}M_{kl}}$.

In practice, in order to increase the stability, the code does not use the value of C obtained in each cell, but an average with the values obtained in the neighbouring cells (this average uses the extended neighbourhood and corresponds to the explicit filter). By default, the value calculated by the code is

$$C = \frac{\widetilde{M_{ij}Lij}}{\widetilde{M_{kl}M_{kl}}}$$

The subroutine `ussmag` allows to modify this value. It is for example possible to calculate the local average after having calculated the ratio

$$C = \left[\frac{\widetilde{M_{ij}Lij}}{M_{kl}M_{kl}}\right]$$

WARNING: The subroutine `ussmag` *can be activated only when the dynamic model is used.*

6.6 User source terms

Assume, for example, that the user source terms modify the equation of a variable φ in the following way:

$$\rho\frac{\partial\varphi}{\partial t} + \ldots = \ldots + S_{impl} \times \varphi + S_{expl}$$

The example is valid for a velocity component, for a turbulent variable (k, ε, R_{ij}, ω, φ or $\overline{f}$) and for a scalar (or for the average of the square of the fluctuations of a scalar), because the syntax of all the subroutines `ustsnv`, `cs_user_turbulence_source_terms` and `ustssc` in the `cs_user_source_terms` file is similar.

In the finite volume formulation, the solved system is then modified as follows:

$$\left(\frac{\rho_i\Omega_i}{\Delta t_i} - \Omega_i S_{impl,i}\right)\left(\varphi_i^{(n+1)} - \varphi_i^{(n)}\right) + \ldots = \ldots + \Omega_i S_{impl,i}\varphi_i^{(n)} + \Omega_i S_{expl,i}$$

The user needs therefore to provide the following values:
$\texttt{crvimp}_i = \Omega_i S_{impl,i}$
$\texttt{crvexp}_i = \Omega_i S_{expl,i}$

In practice, it is essential for the term $\left(\frac{\rho_i\Omega_i}{\Delta t_i} - \Omega_i S_{impl,i}\right)$ to be positive. To ensure this property, the equation really taken into account by the code is the following:

$$\left(\frac{\rho_i\Omega_i}{\Delta t_i} - \mathrm{Min}(\Omega_i S_{impl,i};0)\right)\left(\varphi_i^{(n+1)} - \varphi_i^{(n)}\right) + \ldots = \ldots + \Omega_i S_{impl,i}\varphi_i^{(n)} + \Omega_i S_{expl,i}$$

To make the "implicitation" effective, the source term decomposition between the implicit and explicit parts will be done by the user who must ensure that $\mathtt{crvimp}_i = \Omega_i S_{impl,i}$ is always negative (otherwise the solved equation remains right, but there will not be "implicitation").

WARNING: When the second-order in time is used along with the extrapolation of the source terms[22]*, it is no longer possible to test the sign of $S_{impl,i}$, because of coherence reasons (for more details, the user may refer to the theoretical and computer documentation [11] of the subroutine* `preduv`*). The user must therefore make sure it is always positive (or take the risk to affect the calculation stability).*

Particular case of a linearised source term

In some cases, the added source term is not linear, but the user may want to linearise it using a first-order Taylor development, in order to make it partially implicit.
Consider an equation of the type:

$$\rho\frac{\partial\varphi}{\partial t} = F(\varphi)$$

To make it implicit using the following method:

$$\begin{aligned}\frac{\rho_i\Omega_i}{\Delta t}\left(\varphi_i^{(n+1)} - \varphi_i^{(n)}\right) &= \Omega_i\left[F(\varphi_i^{(n)}) + \left(\varphi_i^{(n+1)} - \varphi_i^{(n)}\right)\frac{dF}{d\varphi}(\varphi_i^{(n)})\right] \\ &= \Omega_i\frac{dF}{d\varphi}(\varphi_i^{(n)}) \times \varphi_i^{(n+1)} + \Omega_i\left[F(\varphi_i^{(n)}) - \frac{dF}{d\varphi}(\varphi_i^{(n)}) \times \varphi_i^{(n)}\right]\end{aligned}$$

The user must therefore specify:
$\mathtt{crvimp}_i = \Omega_i\frac{dF}{d\varphi}(\varphi_i^{(n)})$
$\mathtt{crvexp}_i = \Omega_i\left[F(\varphi_i^{(n)}) - \frac{dF}{d\varphi}(\varphi_i^{(n)}) \times \varphi_i^{(n)}\right]$

Example:
If the equation is $\rho\frac{\partial\varphi}{\partial t} = -K\varphi^2$, the user must set:
$\mathtt{crvimp}_i = -2K\Omega_i\varphi_i^{(n)}$
$\mathtt{crvexp}_i = K\Omega_i[\varphi_i^{(n)}]^2$

6.6.1 In Navier-Stokes

The source term in Navier-Stokes can be filled in thanks to the GUI or the `cs_user_source_terms` user file. Without the GUI, the subroutine `ustsnv` is used to add user source terms to the Navier-Stokes equations (at each time step).

`ustsnv` is called only once per time step; for each cell `iel`, the vector `crvexp(.,iel)` (explicit part) and the matrix `crvimp(.,.,iel)` (implicit part) must be filled in for the whole velocity vector.

6.6.2 For k and ε

Subroutine called every time step, for the $k-\varepsilon$ and the v2f models.

[22] indicator `isno2t` for the velocity, `isto2t` for the turbulence and `isso2t` for the scalars

The subroutine `cs_user_turbulence_source_terms` is used to add source terms to the transport equations related to the turbulent kinetics energy k and to the turbulent dissipation ε. This subroutine is called every time step (the treatment of the two variables k and ε is made simultaneously). The user is expected to provide the arrays `crkimp` and `crkexp` for k, and `creimp` and `creexp` for ε. These arrays are similar to the arrays `crvimp` and `crvexp` given for the velocity in the user subroutine `ustsnv`. The way of making implicit the resulting source terms is the same as the one presented in `ustsnv`. For φ and $\overline{f}$ in the v2f model, see `cs_user_turbulence_source_terms`, §6.6.4.

6.6.3 For R_{ij} and ε

Subroutine called every time step, for the $R_{ij} - \varepsilon$ models.

The subroutine `cs_user_turbulence_source_terms` is used to add source terms to the transport equations related to the Reynolds stress variables R_{ij} and to the turbulent dissipation ε. This subroutine is called 7 times every time step (once for each Reynolds stress component and once for the dissipation). The user must provide the arrays `crvimp` and `crvexp` for the field variable of index `f_id` (referring successively to `ir11`, `ir22`, `ir33`, `ir12`, `ir13`, `ir23` and `iep`). These arrays are similar to the arrays `crvimp` and `crvexp` given for the velocity in the user subroutine `ustsnv`. The method for impliciting the resulting source terms is the same as that presented in `ustsnv`.

6.6.4 For φ and $\overline{f}$

Subroutine called every time step, for the v2f models.

The subroutine `cs_user_turbulence_source_terms` is used to add source terms to the transport equations related to the variables φ and $\overline{f}$ of the v2f φ-model. This subroutine is called twice every time step (once for φ and once for $\overline{f}$). The user is expected to provide the arrays `crvimp` and `crvexp` for `ivar` referring successively to `iphi` and `ifb`. Concerning φ, these arrays are similar to the arrays `crvimp` and `crvexp` given for the velocity in the user subroutine `ustsnv`. Concerning $\overline{f}$, the equation is slightly different:

$$L^2 div(\underline{\nabla}(\overline{f})) = \overline{f} + \ldots + S_{impl} \times \overline{f} + S_{expl}$$

In the finite volume formulation, the solved system is written as:

$$\int_{\partial\Omega_i} \underline{\nabla}(\overline{f})^{(n+1)} dS = \frac{1}{L_i^2}\left(\Omega_i \overline{f}_i^{(n+1)} + \ldots + \Omega_i S_{impl,i} \overline{f}_i^{(n+1)} + \Omega_i S_{expl,i}\right)$$

The user must then specify:
$\texttt{crvimp}_i = \Omega_i S_{impl,i}$
$\texttt{crvexp}_i = \Omega_i S_{expl,i}$

The way of making implicit the resulting source terms is the same as the one presented in `ustsnv`.

6.6.5 For k and ω

Subroutine called every time step, for the $k - \omega$ SST model.

The subroutine `cs_user_turbulence_source_terms` is used to add source terms to the transport equations related to the turbulent kinetics energy k and to the specific dissipation rate ω. This subroutine is called every time step (the treatment of the two variables k and ω is made simultaneously). The user is expected to provide the arrays `crkimp` and `crkexp` for the variable k, and the arrays `crwimp` and `crwexp` for the variable ω. These arrays are similar to the arrays `crvimp` and `crvexp` given for the velocity in the user subroutine `ustsnv`. The way of making implicit the resulting source terms is the same as the one presented in `ustsnv`.

6.6.6 For $\tilde{\nu}_t$

Subroutine called every time step, or the Spalart-Allmaras model.

The subroutine `cs_user_turbulence_source_terms` is used to add source terms to the transport equations related to the turbulent viscosity ν_t for the Spalart-Allmaras model. This subroutine is called every time step. The user is expected to provide the arrays `crkimp` and `crkexp` for the variable $\tilde{\nu}_t$. These arrays are similar to the arrays `crvimp` and `crvexp` given for the velocity in the user subroutine `ustsnv`. The way of making implicit the resulting source terms is the same as the one presented in `ustsnv`.

6.6.7 For user scalars

Subroutine called every time step.

The source terms in the transport equations related to the user scalars (passive or not, average of the square of the fluctuations of a scalar, ...) can be filled in thanks to the GUI or the `cs_user_source_terms` user file. Without the GUI, the subroutine `ustssc` is used to add source terms to the transport equations related to the user scalars. In the same way as `ustsnv`, this subroutine is called every time step, once for each user scalar. The user must provide the arrays `crvimp` and `crvexp` related to each scalar. `cvimp` and `crvexp` must be set to 0 for the scalars on which it is not wished for the user source term to be applied (the arrays are initially set to 0 at each inlet in the subroutine).

6.7 Pressure drops (head losses) and porosity

6.7.1 Head losses

Pressure drops can be defined in the Graphical User Interface (GUI) or in the subroutine `cs_user_head_losses.f90` (called three times every time step). In the GUI, under the heading "Volume conditions", the item "Volume regions definition" allows to define areas where pressure drops occur, see an example in fig 34. The item "Head losses" allows to specify the head loss coefficients, see Figure 35. The tensor representing the pressure drops is supposed to be symmetric and positive.

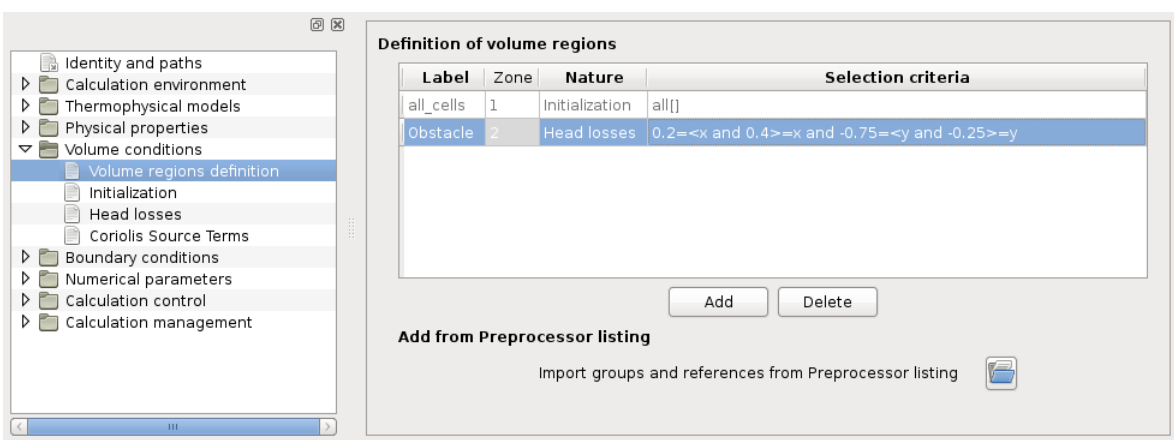

Figure 34: Creation of head losses region

If necessary, the pressure drops are written in the subroutine `cs_user_head_losses.f90`.

- During the first call, all the cells are checked to know the number of cells in which a pressure drop is present. This number is called `ncepdp` in `cs_user_head_losses.f90` (and corresponds to `ncepdc`). It is used to lay out the arrays related to the pressure drops. If there is no pressure

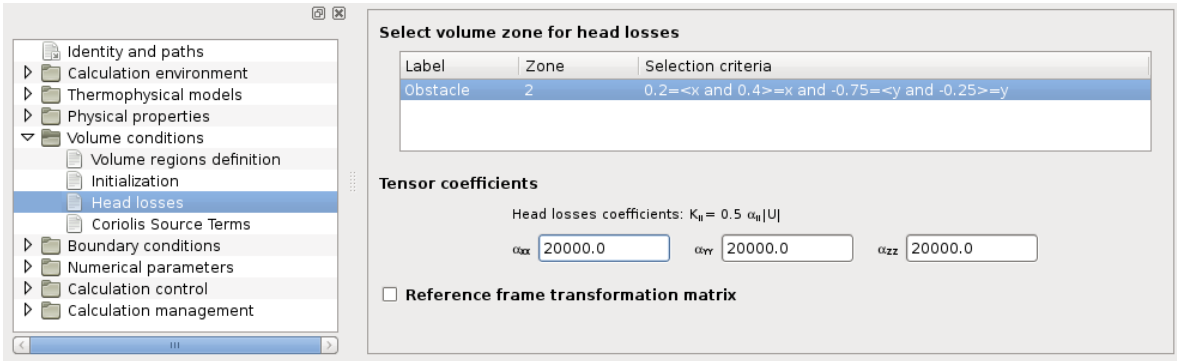

Figure 35: Head losses coefficients

drop, `ncepdp` must be equal to zero (it is the default value, and the rest of the subroutine is then useless).

- During the second call, all the cells are checked again to complete the array `icepdp` whose size is `ncepdp`. `icepdc(ielpdc)` is the number of the `ielpdc`th cell containing pressure drops.
- During the third call, all the cells containing pressure drops are checked in order to complete the array containing the components of the tensor of pressure drops `ckupdc(ncepdp,6)`. This array is so that the equation related to the velocity may be written:

$$\rho \frac{\partial}{\partial t} \underline{u} = \ldots - \rho \underline{\underline{K}}_{pdc} \cdot \underline{u}$$

  The tensor components are given in the following order (in the general reference frame): `k11`, `k22`, `k33`, `k12`, `k23`, `k13` with `k12`, `k23` and `k13` being zero if the tensor is diagonal.

The three calls are made at every time step, so that variable pressure drop zones or values may be treated.

6.7.2 Porosity

The management of the porosity is not yet available in the GUI. To define the porosity, the user must fill in the subroutine `usporo` and set the keyword `iporos` to `1` in the `cs_user_parameters` file. This subroutine is called every time step.
When using the subroutine `usporo`, the user is expected to fill in the array `porosi` for each cell in order to give the porosity.

6.8 Management of the mass sources

The subroutine `cs_user_mass_source_terms` is used to add a density source term in some cells of the domain (called at each time step). The mass conservation equation is then modified as follows:

$$\frac{\partial \rho}{\partial t} + div(\rho \underline{u}) = \Gamma$$

Γ is the mass source term expressed in $kg.m^{-3}.s^{-1}$.

The presence of a mass source term modifies the evolution equation of the other variables, too. Let φ be any solved variable apart from the pressure (velocity component, turbulent energy, dissipation, scalar, ...). Its evolution equation becomes:

$$\rho \frac{\partial \varphi}{\partial t} + \ldots = \ldots + \Gamma(\varphi_i - \varphi)$$

φ_i is the value of φ associated with the mass entering or leaving the domain. After discretisation, the equation may be written:

$$\rho \frac{\varphi^{(n+1)} - \varphi^{(n)}}{\Delta t} + \ldots = \ldots + \Gamma(\varphi_i - \varphi^{(n+1)})$$

For each variable φ, there are two possibilities:

- We can consider that the mass is added (or removed) with the ambient value of φ. In this case $\varphi_i = \varphi^{(n+1)}$ and the equation of φ is not modified.
- Or we can consider that the mass is added with an imposed value φ_i (this solution is physically correct only when the mass is effectively added, $\Gamma > 0$).

This subroutine is called three times every time step.

- During the first call, all the cells are checked to know the number of cells containing a mass source term. This number is called `ncesmp` in `cs_user_mass_source_terms` (and corresponds to `ncetsm`). It is used to lay out the arrays related to the mass sources. If there is no mass source, `ncesmp` must be equal to zero (it is the default value, and the rest of the subroutine is then useless).
- During the second call, all the cells are checked again to complete the array `icetsm` whose dimension is `ncesmp`. `icetsm(ieltsm)` is the number of the `ieltsm`th cell containing a mass source.
- During the third call, all the cells containing mass sources are checked in order to complete the arrays `itypsm(ncesmp,nvar)` and `smacel(ncesmp,nvar)`:
  - `itypsm(ieltsm,ivar)` is the flow type associated with the variable `ivar` in the `ielstm`th cell containing a mass source.
    `itypsm`=0: $\varphi_i = \varphi^{(n+1)}$ condition
    `itypsm`=1: imposed φ_i condition
    `itypsm` is not used for `ivar=ipr`
  - `smacel(ieltsm,ipr)` is the value of the mass source term Γ, in $kg.m^{-3}.s^{-1}$.
  - `smacel(ieltsm,ivar)`, for `ivar` different from `ipr`, is the value of φ_i for the variable `ivar` in the `ielstm`th cell containing a mass source.

  NOTES
  • If `itypsm(ieltsm,ivar)=0`, `smacel(ieltsm,ivar)` is not used.
  • If Γ=`smacel(ieltsm,ipr)`<0, mass is removed from the system, and *Code_Saturne* considers automatically a $\varphi_i = \varphi^{(n+1)}$ condition, whatever the values given to `itypsm(ieltsm,ivar)` and `smacel(ieltsm,ivar)` (the extraction of a variable is done at ambient value).

The three calls are made every time step, so that variable mass source zones or values may be treated.

For the variance, do not take into account the scalar φ_i in the environment where $\varphi \neq \varphi_i$ generates a variance source.

6.9 User law editor of the GUI

A formula interpreter is embedded in *Code_Saturne*, which can be used through the GUI. In order to call the formula editor of the GUI, click on the button:

The formula editor is a window with three tabs:

- User expression

  This tab is the formula editor. At the opening of the window only the required symbols are displayed. The syntax colorization shows to the user symbols which are required symbols, functions, or user variables. Each expression must be closed by a semicolon (";"). The required symbols must be present in the final user law. A syntax checker is used when the user clicks on the OK button.

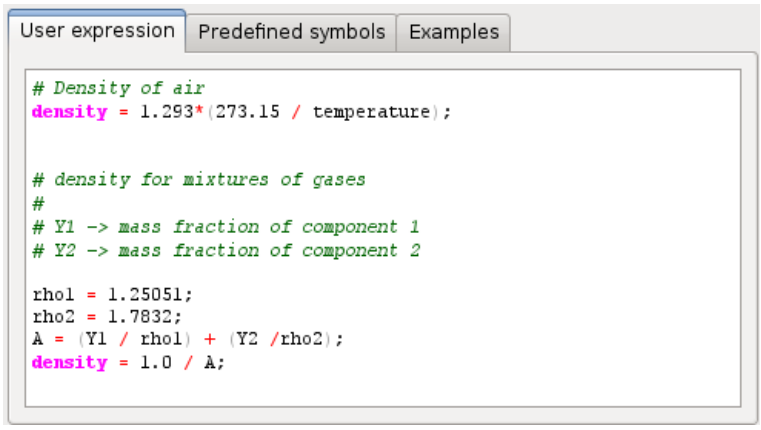

Figure 36: Example of the user law editor

- Predefined symbols

  There are three types of symbols

  Useful functions:

  `cos`: cosine

  `sin`: sine

  `tan`: tangent

  `exp`: exponential

  `sqrt`: square root

  `log`: Napierian logarithm

  `acos`: arc cosine

  `asin`: arc sine

  `atan(x)`: arc tangent (arc tangent of x in radians; the return value is in the range [-pi/2, pi/2])

  `atan2(y,x)`: arc tangent (arc tangent of y/x in radians; the return value is in the range [-pi, pi])

  `cosh`: hyperbolic cosine

  `sinh`: hyperbolic sine

  `tanh`: hyperbolic tangent

  `abs`: absolute value

  `mod`: modulo

  `int`: floor

`min`: minimum

`max`: maximum

Useful constants:

pi = 3.14159265358979323846

e = 2.718281828459045235

Operators and statements:

+ − * / ∧

! < > <= >= == ! = && ||

`while if else print`

- Examples

  This tab displays examples of formula, which could be copy and paste.

6.10 Modification of the variables at the end of a time step

The subroutine `cs_user_extra_operations` is called at the end of every time step. It is used to print of modify any variable at the end of every time step.

Several examples are given in the directory `EXAMPLES`:

- Calculation of a thermal balance at the boundaries and in the domain (including the mass source terms)
- Modification of the temperature in a given area starting from a given time
- Extraction of a 1D profile (which is also possible with the GUI, see Figure 27)
- Printing of a moment
- Usage of utility subroutines in the case of a parallel calculation (calculation of a sum on the processors, of a maximum, ...)

WARNING: As all the variables (solved variables, physical properties, geometric parameters) can be modified in this subroutine, a wrong use may distort totally the calculation.

The thermal balance example is particularly interesting.

- It can be easily adapted to another scalar (only three simple modifications to do, as indicated in the subroutine).
- It shows how to make a sum on all the sub-domains in the framework of a parallel calculation (see the calls to the subroutines `par*`).
- It shows the precautions to take before doing some operations in the framework of periodic or parallel calculations (in particular when we want to calculate the gradient of a variable or to have access to values at the neighbouring cells of a face).
- Finally it must not be forgotten that the resolution with temperature (and not enthalpy) as a solved variable is questionable when the specific heat is not constant.

7 Advanced modelling setup

7.1 Use of a specific physics

Specific physics such as dispersed phase, atmospheric flows, gas combustion, pulverised fuel combustion, electrical model and compressible model can be added by the user from the interface, or by using the subroutine `usppmo` of the `cs_user_parameters` file (called only during the calculation initialisation). With the interface, when a specific physics is activated in Figure 37, additional items or headings may appear (see for instance Sections 7.6.4 and 7.2.0.1).

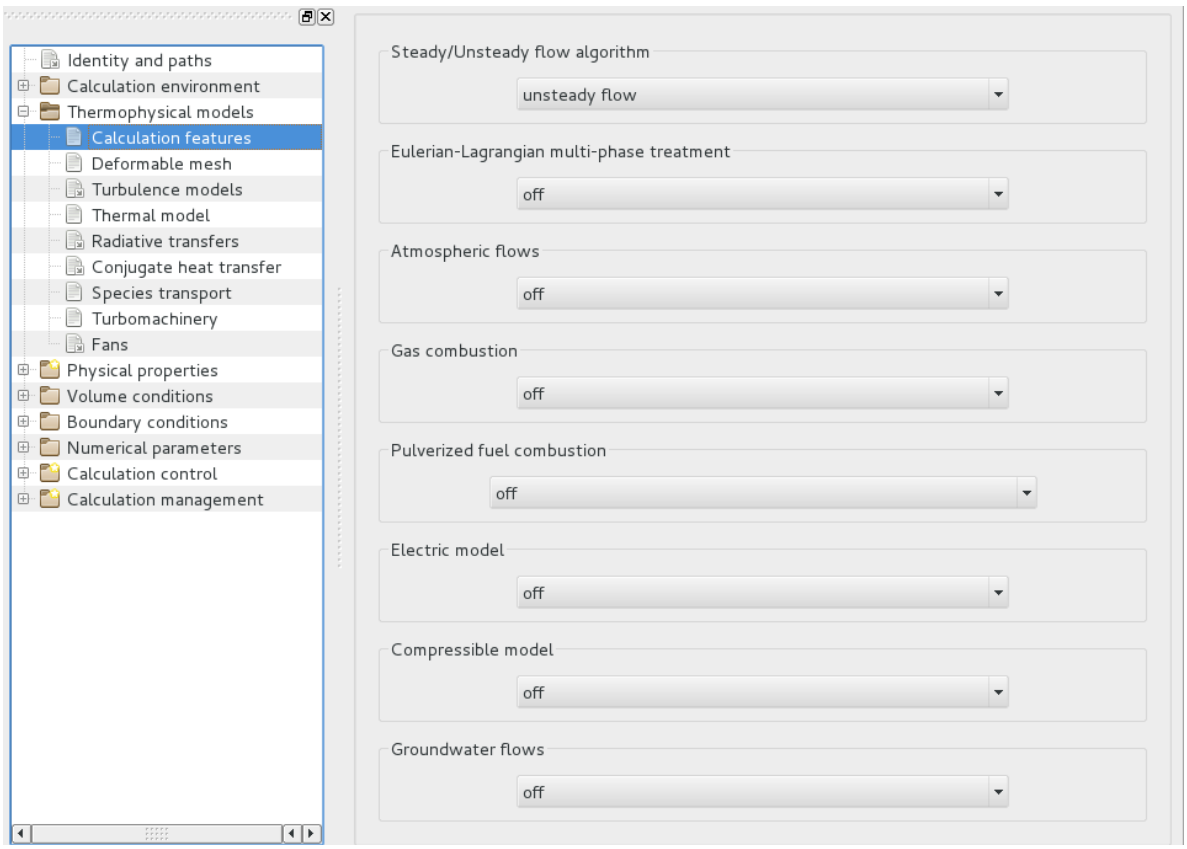

Figure 37: Specific physics models selection

When the interface is not used, `usppmo` is one of the three subroutines which must be obligatory completed by the user in order to use a specific physics module (only heavy fuel combustion is not available with the GUI). At the moment, *Code_Saturne* allows to use two "pulverised coal" modules (with Lagrangian coupling or not) and one "pulverised heavy fuel" module, two "gas combustion" modules, two "electrical" modules, a "compressible" module and an "atmospheric" module. To activate one of these modules, the user must complete one (and only one) of the indicators `ippmod(i.....)` in the subroutine `usppmo`. By default, all the indicators `ippmod(i.....)` are initialised at -1, which means that no specific physics is activated.

- Diffusion flame in the framework of "3 points" rapid complete chemistry: indicator `ippmod(icod3p)`

  → `ippmod(icod3p)` = 0 adiabatic conditions

  → `ippmod(icod3p)` = 1 permeatic conditions (enthalpy transport)

  → `ippmod(icod3p)` =-1 module not activated

- Eddy Break Up pre-mixed flame: indicator `ippmod(icoebu)`
  - → `ippmod(icoebu)` = 0 adiabatic conditions at constant richness
  - → `ippmod(icoebu)` = 1 permeatic conditions at constant richness
  - → `ippmod(icoebu)` = 2 adiabatic conditions at variable richness
  - → `ippmod(icoebu)` = 3 permeatic conditions at variable richness
  - → `ippmod(icoebu)` =-1 module not activated
- Libby-Williams pre-mixed flame: indicator `ippmod(icolwc)`
  - → `ippmod(icolwc)`=0 two peak model with adiabiatic conditions.
  - → `ippmod(icolwc)`=1 two peak model with permeatic conditions.
  - → `ippmod(icolwc)`=2 three peak model with adiabiatic conditions.
  - → `ippmod(icolwc)`=3 three peak model with permeatic conditions.
  - → `ippmod(icolwc)`=4 four peak model with adiabiatic conditions.
  - → `ippmod(icolwc)`=5 four peak model with permeatic conditions.
  - → `ippmod(icolwc)`=-1 module not activated.
- Multi-coals and multi-classes pulverised coal combustion: indicator `ippmod(iccoal)` The number of different coals must be less than or equal to `ncharm` = 3. The number of particle size classes `nclpch(icha)` for the coal `icha`, must be less than or equal to `ncpcmx` = 10.
  - → `ippmod(iccoal)` = 0 imbalance between the temperature of the continuous and the solid phases
  - → `ippmod(iccoal)` = 1 otherwise
  - → `ippmod(iccoal)` =-1 module not activated
- Multi-classes pulverised heavy fuel combustion: indicator `ippmod(icfuel)`
  - → `ippmod(icfuel)` = 0 module activated
  - → `ippmod(icfuel)` =-1 module not activated
- Lagrangian modelling of multi-coals and multi-classes pulverised coal combustion: indicator `ippmod(icpl3c)` The number of different coals must be less than or equal to `ncharm` = 3. The number of particle size classes `nclpch(icha)` for the coal `icha`, must be less than or equal to `ncpcmx` = 10.
  - → `ippmod(icpl3c)` = 1 coupling with the Lagrangian module, with transport of H_2
  - → `ippmod(icpl3c)` =-1 module not activated
- Electric arcs module (Joule effect and Laplace forces): indicator `ippmod(ielarc)`
  - → `ippmod(ielarc)` = 1 determination of the magnetic field by means of the Ampere's theorem (not available)
  - → `ippmod(ielarc)` = 2 determination of the magnetic field by means of the vector potential
  - → `ippmod(ielarc)` =-1 module not activated
- Joule effect module (Laplace forces not taken into account): indicator `ippmod(ieljou)`
  - → `ippmod(ieljou)` = 1 use of a real potential
  - → `ippmod(ieljou)` = 2 use of a complex potential
  - → `ippmod(ieljou)` = 3 use of real potential and specific boundary conditions for transformers.
  - → `ippmod(ieljou)` = 4 use of complex potential and specific boundary conditions for transformers.

→ `ippmod(ieljou)` =-1 module not activated

- Compressible module: indicator `ippmod(icompf)`

  → `ippmod(icompf)` = 0 module activated

  → `ippmod(icompf)` =-1 module not activated

- Atmospheric flow module: indicator `ippmod(iatmos)`

  → `ippmod(iatmos)` =-1 module not activated

  → `ippmod(iatmos)` = 0 standard modelling

  → `ippmod(iatmos)` = 1 dry atmosphere

  → `ippmod(iatmos)` = 2 humid atmosphere

WARNING: Only one specific physics module can be activated at the same time.

In the framework of the gas combustion modelling, the user may impose his own enthalpy-temperature tabulation (conversion law). He needs then to give the value zero to the indicator `indjon` (the default value being 1). For more details, the user may refer to the following note (thermochemical files).

NOTE: THE THERMO-CHEMICAL FILES

The user must not forget to place in the directory DATA the thermochemical file `dp_FCP.xml`, `dp_C3P`, `dp_C3PSJ` or `dp_ELE` (depending on the specific physics module he activated) Some example files are placed in the directory `DATA/REFERENCE` at the creation of the study case. Their content is described below.

- Example of file for the gas combustion:

  → if the enthalpy-temperature conversion data base JANAF is used: `dp_C3P` (see array 1).

  → if the user provides his own enthalpy-temperature tabulation (there must be three chemical species and only one reaction): `dp_C3PSJ` (see array 2). This file replaces `dp_C3P`.

- Example of file for the pulverised coal combustion: `dp_FCP.xml` (see the example in the directory `DATA/REFERENCE`, this file can be filled in thanks to the GUI).

- Example of file for the heavy fuel combustion: `DP_FUE_new` (see the example in `DATA/REFERENCE`).

- Example of file for the electric arcs: `dp_ELE` (see array 3).

Lines	Examples of values	Variables	Observations
1	5	`ngaze`	Number of current species
2	10	`npo`	Number of points for the enthalpy-temperature table
3	300.	`tmin`	Lower temperature limit for the table
4	3000.	`tmax`	Upper temperature limi t for the tabulation
5			Empty line
6	CH4 O2 CO2 H2O N2	`nomcoe(ngaze)`	List of the current species
7	.35 .35 .35 .35 .35	`kabse(ngaze)`	Absorption coefficient of the current species
8	4	`nato`	Number of elemental species
9 10 11 12	.012 1 0 1 0 0 .001 4 0 0 2 0 .016 0 2 2 1 0 .014 0 0 0 0 2	`wmolat(nato)`, `atgaze(ngaze,nato)`	Molar mass of the elemental species (first column) Composition of the current species as a function of the elemental species (`ngaze` following columns)
13	3	`ngazg`	Number of global species Here, `ngazg` = 3 (Fuel, Oxidiser and Products)
14 15 16	1. 0. 0. 0. 0. 0. 1. 0. 0. 3.76 0. 0. 1. 2. 7.52	`compog(ngaze,ngazg)`	Composition of the global species as a function of the current species of line 6 In the order: Fuel (line 15), Oxidiser (line 16) and Product (line 17)
17	1	`nrgaz`	Number of global reactions Here `nrgaz` = 1 (always equal to 1 in this version)
18	1 2 -1 -9.52 10.52	`igfuel(nrgaz)`, `igoxy(nrgaz)`, `stoeg(ngazg,nrgaz)`	Numbers of the global species concerned by the stoichiometric ratio (first 2 integers) Stoichiometry in global species reaction. Negative for the reactants (here "Fuel" and "Oxidiser") and positive for the products (here "Products")

Table 1: Example of file for the gas combustion when JANAF is used: `dp_C3P`

Lines	Examples of values	Variables	Observations
1	6	`npo`	Number of tabulation points
2 3 4 5 6 7	50. -0.32E+07 -0.22E+06 -0.13E+08 250. -0.68E+06 -0.44E+05 -0.13E+08 450. 0.21E+07 0.14E+06 -0.13E+08 650. 0.50E+07 0.33E+06 -0.12E+08 850. 0.80E+07 0.54E+06 -0.12E+08 1050. 0.11E+08 0.76E+06 -0.11E+08	`th(npo)`, `ehgazg(1,npo)`, `ehgazg(2,npo)`, `ehgazg(3,npo)`	Temperature(first column), mass enthalpies of fuel, oxidiser and products (columns 2,3 and 4) from line 2 to line `npo`+1
8	.00219 .1387 .159	`wmolg(1)`, `wmolg(2)`, `wmolg(3)`	Molar masses of fuel, oxidiser and products
9	.11111	`fs(1)`	Mixing rate at the stoichiometry (relating to Fuel and Oxidiser)
10	0.4 0.5 0.87	`ckabsg(1)`, `ckabsg(2)`, `ckabsg(3)`	Absorption coefficients of the fuel, oxidiser and products
11	1. 2.	`xco2`, `xh2o`	Molar coefficients of CO_2 and H_2O in the products (using Modak radiation)

Table 2: Example of file for the gas combustion when the user provides his own enthalpy-temperature table (there must be three species and only one reaction): `dp_C3PSJ` (this file replaces `dp_C3P`)

Lines	Examples of values	Variables	Observations
1	# Free format ASCII file ...		Free comment
2	# Comment lines ...		Free comment
3	# ...		Free comment
4	# Argon propoerties ...		Free comment
5	# ...		Free comment
6	# No of NGAZG and No ...		Free comment
7	# NGAZG NPO ...		Free comment
8	1 238	`ngazg` `npo`	Number of species Number of given temperature points for the tabulated physical properties (`npo` $\leqslant$ `npot` set in `ppthch`) So there will be `ngazg` blocks of `npo` lines each
9	# ...		Free comment
14	0	`ixkabe`	Radiation options for `xkabe`
15	# ...		Free comment
16	# Propreties ...		Free comment
17	# T H ...		Free comment
18	# Temperature Enthalpy ...		Free comment
19	# ...		Free comment
20	# K J/kg ...		Free comment
21	# ...		Free comment

22	300. 14000. ...	
		
		`h`
		`roel`
		`cpel`
		`sigel`
		`visel`
		`xlabel`
		`xkabel`
			In line tabulation of the physical properties
			as a function of the temperature in Kelvin
			for each of the `ngazg` species
			Enthalpy in J/kg
			Density in kg/m3
			Specific heat in J/(kg K)
			Electric conductivity in Ohm/m
			Dynamic viscosity in kg/(m s)
			Thermal conductivity in W/(m K)
			Absorption coefficient (radiation)

Table 3: Example of file for the electric arcs module: `dp_ELE`

7.2 Pulverised coal and gas combustion module (needs update)

7.2.0.1 Initialisation of the variables

For coal combustion, it is possible to initialise the specific variables in the Graphical User Interface (GUI) or in the subroutine `cs_user_initialization`. In the GUI, when a coal combustion physics is selected in the item "Calculation features" under the heading "Thermophysical models", an additional item appears: "Pulverized coal combustion". In this item the user can define coal types, their composition, the oxidant and reactions parameters, see Figure 38 to Figure 42.

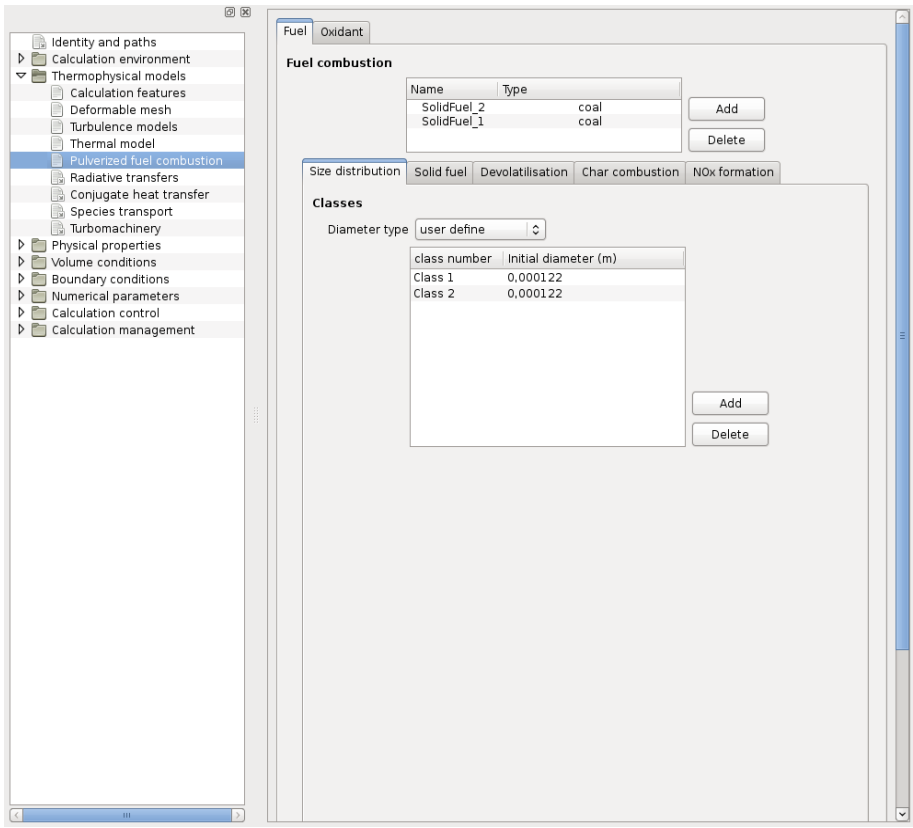

Figure 38: Thermophysical models - Pulverized coal combustion, coal classes

If the user deals with gas combustion or if he (or she) does not want to use the GUI for coal combustion, the subroutine `cs_user_initialization` must be used (only during the calculation initialisation). In this section, "specific physics" will refer to gas combustion or to pulverised coal combustion.

These subroutines allow the user to initialise some variables specific to the specific physics activated *via* `usppmo`. As usual, the user may have access to several geometric variables to discriminate between different initialisation zones if needed.

It should be recalled again that the user can access the array of values of the variables as described in the the `doxygen` documentation dedicated to the fields management. In the following description, only variables indices `ivar` are given, but field indices can be retrieved easily by using `ivarfl(ivar)`.

WARNING: in the case of a specific physics modelling, all the variables will be initialised here, even the potential user scalars: `cs_user_initialization` *is no longer used.*

- in the case of the EBU pre-mixed flame module, the user can initialise in every cell `iel`: the mixing rate `isca(ifm)` in variable richness, the fresh gas mass fraction

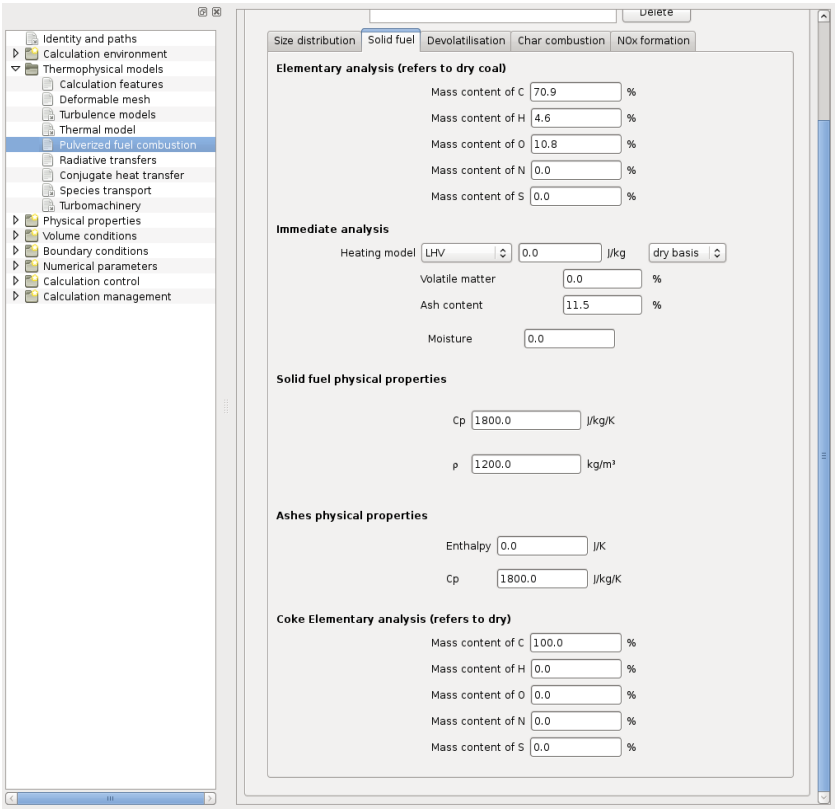

Figure 39: Pulverized coal combustion, coke

`isca(iygfm)` and the mixture enthalpy `isca(iscalt)` in permeatic conditions

- in the case of the rapid complete chemistry diffusion flame module, the user can initialise in every cell `iel`: the mixing rate `isca(ifm)`, its variance `isca(ifp2m)` and the mixture mass enthalpy `isca(iscalt)` in permeatic conditions
- in the case of the pulverised coal combustion module, the user can initialise in every cell `iel`:
  - → the transport variables related to the solid phase

    `isca(ixch(icla))` the reactive coal mass fraction related to the class `icla` (`icla` from 1 to `nclacp` which is the total number of classes, *i.e.* for all the coal type)

    `isca(ixck(icla))` the coke mass fraction related to the class `icla`

    `isca(inp(icla))` the number of particles related to class `icla` per kg of air-coal mixture

    `isca(ih2(icla))` the mass enthalpy related to the class `icla` in permeatic conditions
  - → `isca(iscalt)` the mixture enthalpy
  - → the transport variables related to the gas phase

    `isca(if1m(icha))` the mean value of the tracer 1 representing the light volatile matters released by the coal `icha`

    `isca(if2m(icha))` the mean value of the tracer 2 representing the heavy volatile matters released by the coal `icha`

    `isca(if3m)` the mean value of the tracer 3 representing the carbon released as CO during coke burnout

    `isca(if4p2m)` the variance associated with the tracer 4 representing the air (the mean value of this tracer is not transported, it can be deduced directly from the three others)

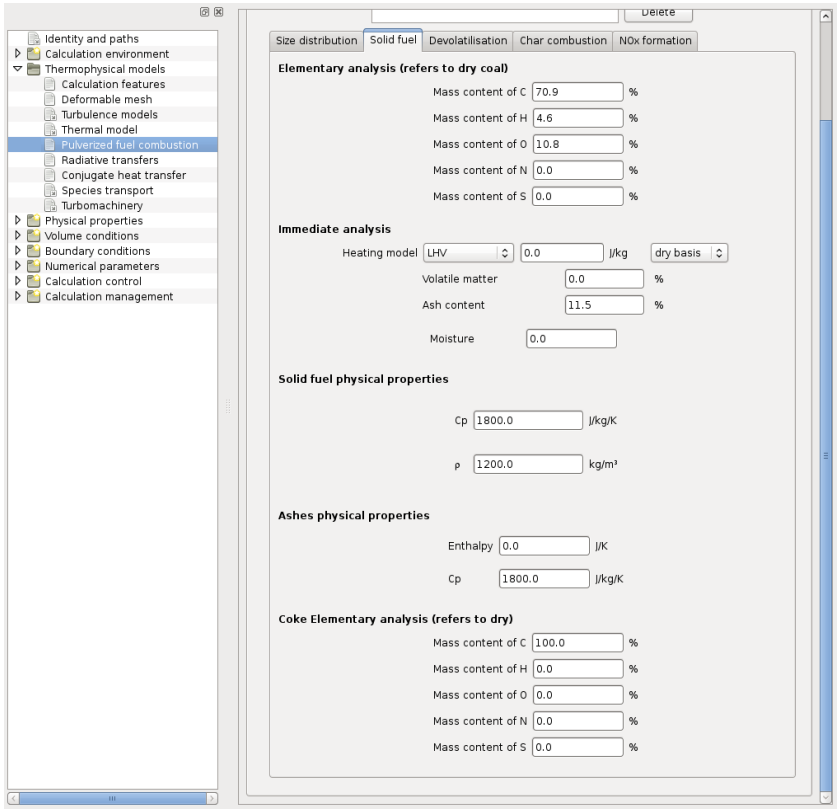

Figure 40: Pulverized coal combustion, coal composition

`isca(ifp3m)` the variance associated with the tracer 3

7.2.1 Boundary conditions

In this section, "specific physics" refers to gas combustion or to pulverised coal combustion.
For coal combustion, it is possible to manage the boundary conditions in the Graphical User Interface (GUI). When the coal combustion physics is selected in the heading "Thermophysical models", specific boundary conditions are activated for inlets, see Figure 43. The user fills for each type of coal previously defined (see § 7.2.0.1) the initial temperature and initial composition of the inlet flow, as well as the mass flow rate.

For gas combustion or if the GUI is not used for coal combustion, the use of `cs_user_boundary_conditions` (called at every time step) is as mandatory as `cs_user_parameters.f90` and `usppmo` to run a calculation involving specific physics. The way of using them is the same as using in the framework of standard calculations, that is, run several loops on the boundary faces lists (cf. §3.9.4) marked out by their colors, groups, or geometrical criterion, where the type of face, the type of boundary condition for each variable and eventually the value of each variable are defined.

WARNING: In the case of a specific physics modelling, all the boundary conditions for every variable must be defined here, even for the eventual user scalars: `cs_user_boundary_conditions` *is not used at all.*

In the case of a specific physics modelling, a zone number `izone` [23] (for instance the color `icoul`) is

[23] `izone` must be less than the maximum number of boundary zone allowable by the code, `nozppm`. This is fixed at 2000 in `pppvar`;not to be modified

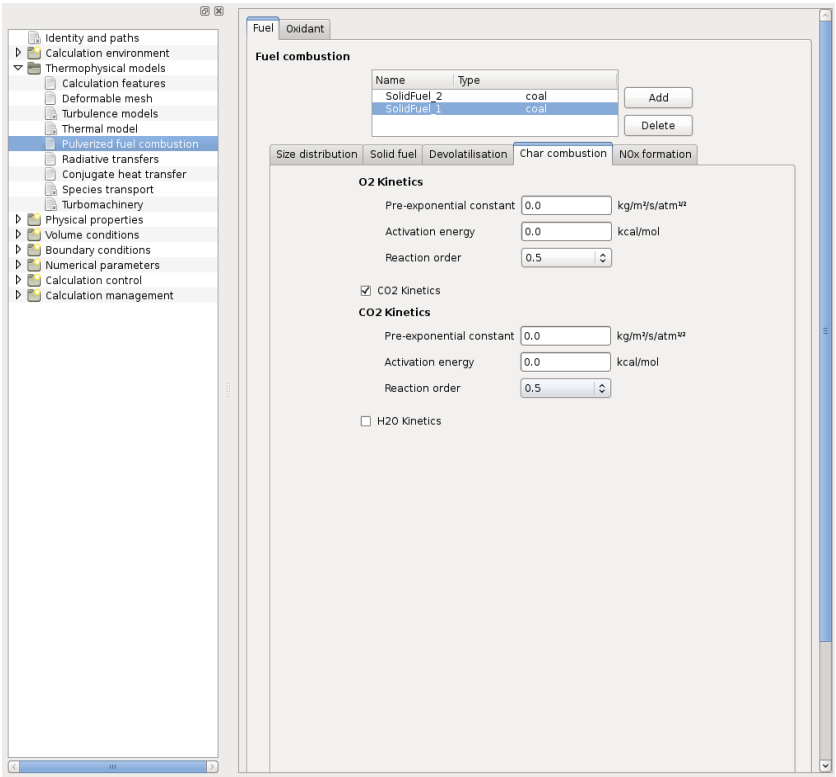

Figure 41: Pulverized coal combustion, reaction parameters

associated with every boundary face, in order to gather together all the boundary faces of the same type. In comparison to `cs_user_boundary_conditions`, the main change from the user point of view concerns the faces whose boundary conditions belong to the type `itypfb=ientre`:

- for the EBU pre-mixed flame module:
  - → the user can choose between the "burned gas inlet" type (marked out by the burned gas indicator `ientgb(izone)`=1) and the "fresh gas inlet" type (marked out by the fresh gas indicator `ientgf(izone)`=1)
  - → for each inlet type (fresh or burned gas), a mass flow or a velocity must be imposed:
    - to impose the mass flow,
      - the user gives to the indicator `iqimp(izone)` the value 1,
      - the mass flow value is set in `qimp(izone)` (positive value, in kgs^{-1})
      - finally he imposes the velocity vector direction by giving the components of a direction vector in `rcodcl(ifac,iu)`, `rcodcl(ifac,iv)` and `rcodcl(ifac,iw)`

      WARNING:
      - *the variable `qimp(izone)` refers to the mass flow across the whole zone `izone` and not across a boundary face (specifically for the axi-symmetric calculations, the inlet surface of the mesh must be broken up)*
      - *the variable `qimp(izone)` deals with the inflow across the area `izoz` and only across this zone; it is recommended to pay attention to the boundary conditions.*
      - *the velocity direction vector is neither necessarily normed, nor necessarily incoming.*
    - to impose a velocity, the user must give to the indicator `iqimp(izone)` the value 0 and set the three velocity components (in $m.s^{-1}$) in `rcodcl(ifac,iu)`, `rcodcl(ifac,iv)` and `rcodcl(ifac,iw)`

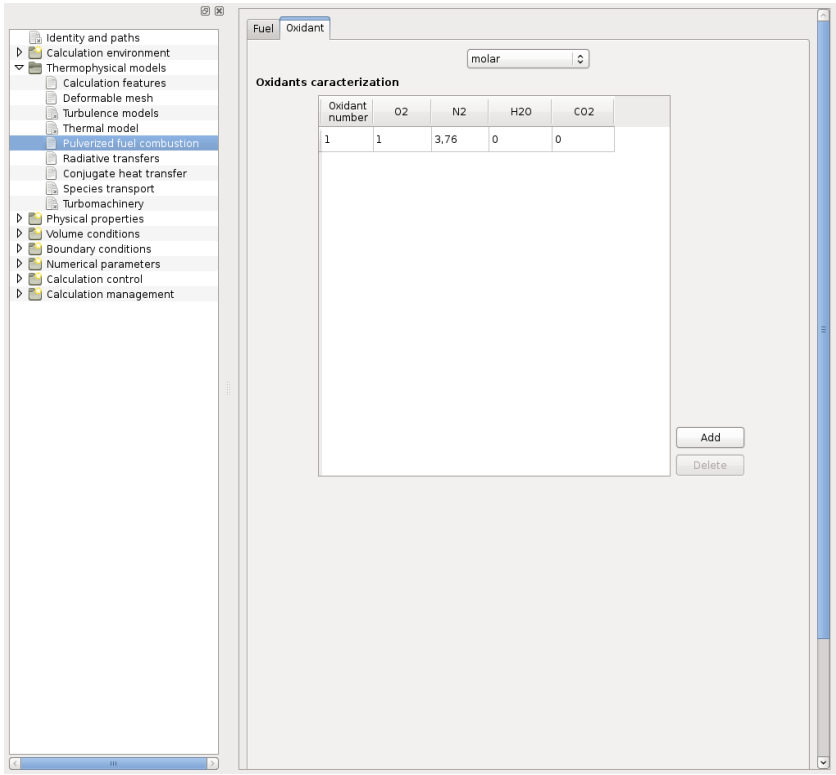

Figure 42: Pulverized coal combustion, oxydant

→ finally he specifies for each gas inlet type the mixing rate `fment(izone)` and the temperature `tkent(izone)` in Kelvin

- for the "3 points" diffusion flame module:
  - → the user can choose between the "oxidiser inlet" type marked out by `ientox(izone)`=1 and the "fuel inlet" type marked out by `ientfu(izone)`=1
  - → concerning the input mass flow or the input velocity, the method is the same as for the EBU pre-mixed flame module
  - → finally, the user sets the temperatures `tinoxy` for each oxidiser inlet and `tinfue`, for each fuel inlet

    Note: In the standard version, only the cases with only one oxidising inlet type and one fuel inlet type can be treated. In particular, there must be only one input temperature for the oxidiser (`tinoxy`) and one input temperature for the fuel (`tinfuel`).

- for the pulverised coal module:
  - → the inlet faces can belong to the "primary air and pulverised coal inlet" type, marked out by `ientcp(izone)`=1, or to the "secondary or tertiary air inlet" type, marked out by `ientat(izone)`=1
  - → in a way which is similar to the process described in the framework of the EBU module, the user chooses for every inlet face to impose the mass flow or not (`iqimp(izone)`=1 or 0). If the mass flow is imposed, the user must set the air mass flow value `qimpat(izone)`, its direction in `rcodcl(ifac,iu)`, `rcodcl(ifac,iv)` and `rcodcl(ifac,iw)` and if
  - → incoming air temperature `timpat(izone)` in Kelvin. If the velocity is imposed, he must set `rcodcl(ifac,iu)`, `rcodcl(ifac,iv)` and `rcodcl(ifac,iw)`.

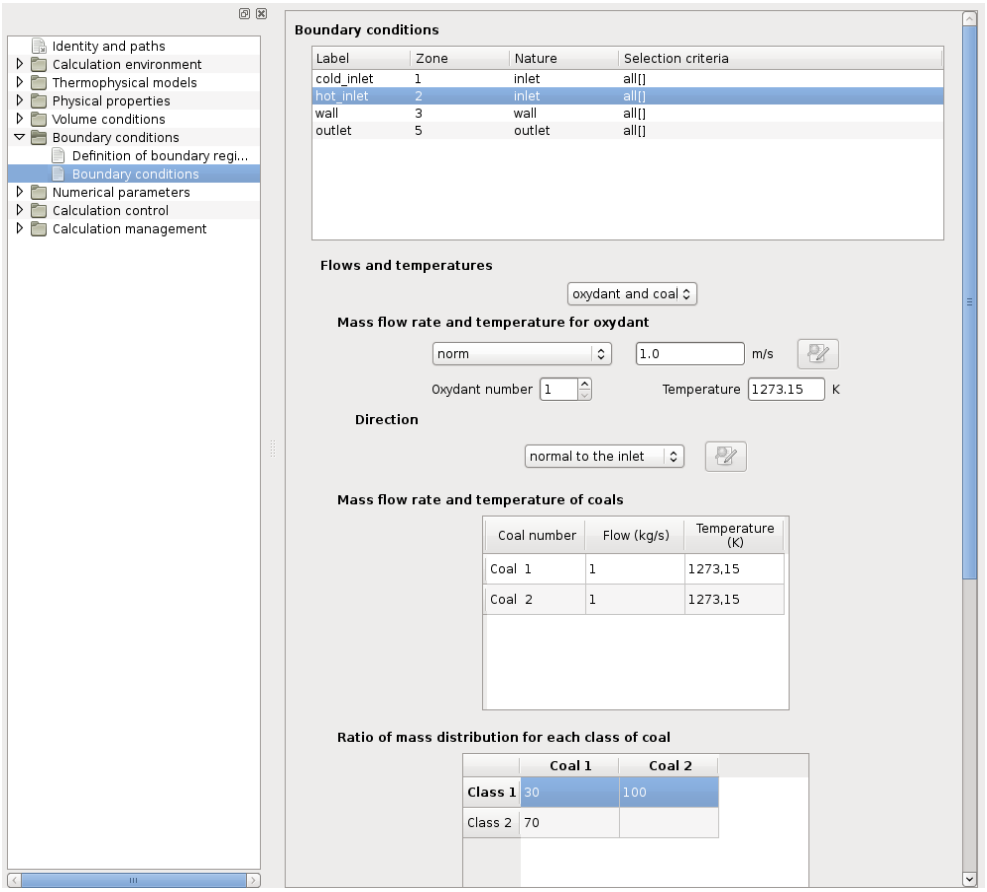

Figure 43: Boundary conditions for the combustion of coal

→ if the inlet belongs to the "primary air and pluverised coal" `type (ientcp(izone) = 1)` the user must also define for each coal type `icha`: the mass flow `qimpcp(izone,icha)`, the granulometric distribution `distch(izone,icha,iclapc)` related to each class `iclacp`, and the injection temperature `timpcp(izone,icha)`

7.2.2 Initialisation of the options of the variables

In the case of coal combustion, time averages, chronological records and listings follow-ups can be set in the Graphical User Interface (GUI) or in the subroutines `cs_user_combustion`. In the GUI, under the heading "Calculation control", additional variables appear in the list in the items "Time averages" and "Profiles", as well as in the item Volume solution control", see Figure 44 and Figure 45.

In this section, "specific physics" refers to gas combustion or pulverised coal combustion.

For gas combustion or if the GUI is not used for coal combustion, the 3 subroutines `cs_user_combustion` can be used to complete `cs_user_parameters.f90` for the considered specific physics. These subroutines are called at the calculation start. They allow to:

- activate, for the variables which are specific to the activated specific physics module, chronological records at the probes defined in `cs_user_parameters.f90`.
  Concerning the main variables (velocity, pressure, etc ...) the user must still complete `cs_user_parameters.f90` if he wants to get chronological records, printings in the listing or chronological outputs. The variables which can be activated by the user for each specific physics are listed below. The solved

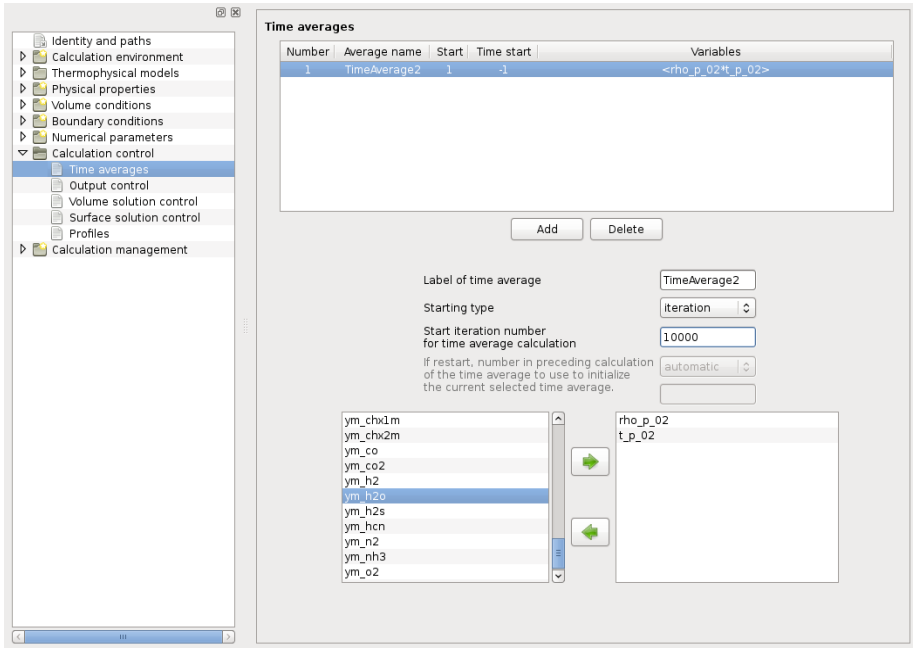

Figure 44: Calculation control - Time averages

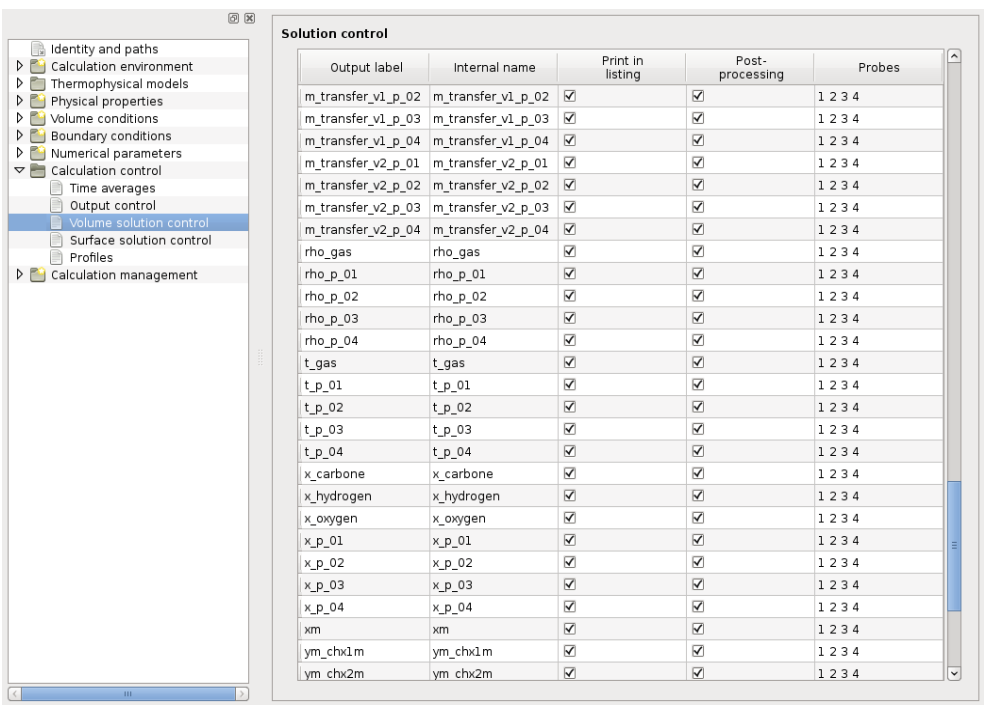

Figure 45: Calculation control - Volume solution control

variables (of variable indices `ivar`) and the properties of indices `iprop` (defined at the cell `iel` by `cpro_prop(iel)` which is obtained by calling `field_get_val_s(iprop, cpro_prop)`) are listed below:

→ EBU pre-mixed flame modelling:

- Solved variables

  `ivar` = `isca(iygfm)` fresh gas mass fraction

  `ivar` = `isca(ifm)` mixing rate

  `ivar` = `isca(ihm)` enthalpy, if transported

- Properties `cpro_prop(iel)`

  `iprop` = `itemp` temperature

  `iprop` = `iym(1)` fuel mass fraction

  `iprop` = `iym(2)` oxidiser mass fraction

  `iprop` = `iym(3)` product mass fraction

  `iprop` = `ickabs` absorption coefficient, when the radiation modelling is activated

  `iprop` = `it3m` and `it4m` "T^3" and "T^4" terms, when the radiation modelling is activated

→ rapid complete chemistry diffusion flame modelling:

everything is identical to the "EBU" case, except the fresh gas mass fraction which is replaced by the variance of the mixing rate `ivar=isca(ifp2m)`

→ pulverised coal modelling with 3 combustibles:

variables shared by the two phases:

- Solved variables

  `ivar` = `isca(ihm)`: gas-coal mixture enthalpy

  `ivar` = `isca(immel)`: molar mass of the gas mixture

variables specific to the dispersed phase:

- Solved variables

  `ivar` = `isca(ixck(icla))`: coke mass fraction related to the class `icla`

  `ivar` = `isca(ixch(icla))`: reactive coal mass fraction related to the class `icla`

  `ivar` = `isca(inp(icla))`: number of particles of the class `icla` per kg of air-coal mixture

  `ivar` = `isca(ih2(icla))`: mass enthalpy of the coal of class `icla`, if we are in permeatic conditions

- Properties `cpro_prop(iel)`

  `iprop` = `immel`: molar mass of the gas mixture

  `iprop` = `itemp2(icla)`: temperature of the particles of the class `icla`

  `iprop` = `irom2(icla)`: density of the particles of the class `icla`

  `iprop` = `idiam2(icla)`: diameter of the particles of the class `icla`

  `iprop` = `igmdch(icla)`: disappearance rate of the reactive coal of the class `icla`

  `iprop` = `igmdv1(icla)`: mass transfer caused by the release of light volatiles from the class `icla`

  `iprop` = `igmdv2(icla)`: mass transfer caused by the release of heavy volatiles from the class `icla`

  `iprop` = `igmhet(icla)`: coke disappearance rate during the coke burnout of the class `icla`

  `iprop` = `ix2(icla)`: solid mass fraction of the class `icla`

variables specific to the continuous phase:

- Solved variables

  `ivar` = `isca(if1m(icha))`: mean value of the tracer 1 representing the light volatiles released by the coal `icha`

  `ivar` = `isca(if2m(icha))`: mean value of the tracer 2 representing the heavy volatiles released by the coal `icha`

  `ivar` = `isca(if3m)`: mean value of the tracer 3 representing the carbon released as CO during coke burnout

`ivar` = `isca(if4pm)`: variance of the tracer 4 representing the air
`ivar` = `isca(if3p2m)`: variance of the tracer 3

- Properties `cpro_prop(iel)`
  `iprop` = `itemp1`: temperature of the gas mixture
  `iprop` = `iym1(1)`: mass fraction of CH_{X1m} (light volatiles) in the gas mixture
  `iprop` = `iym1(2)`: mass fraction of CH_{X2m} (heavy volatiles) in the gas mixture
  `iprop` = `iym1(3)`: mass fraction of CO in the gas mixture
  `iprop` = `iym1(4)`: mass fraction of O_2 in the gas mixture
  `iprop` = `iym1(5)`: mass fraction of CO_2 in the gas mixture
  `iprop` = `iym1(6)`: mass fraction of H_2O in the gas mixture
  `iprop` = `iym1(7)`: mass fraction of N_2 in the gas mixture

- set the relaxation coefficient of the density `srrom`, with
  $\rho^{n+1} = \texttt{srrom} * \rho^n + (1 - \texttt{srrom})\rho^{n+1}$
  (the default value is `srrom` = 0.8. At the beginning of a calculation, a sub-relaxation of 0.95 may reduce the numerical "shocks").

- set the dynamic viscosity `diftl0`. By default `diftl0`= 4.25 $kgm^{-1}s^{-1}$ (the dynamic diffusivity being the ratio between the thermal conductivity λ and the mixture specific heat C_p in the equation of enthalpy).

- set the value of the constant `cebu` of the Eddy Break Up model (only in `cs_user_combustion`. By default `cebu`=2.5)

7.3 Heavy fuel oil combustion module

7.3.1 Initialisation of transported variables

To initialise or modify (in case of a continuation) values of transported variables and of the time step, the standard subroutine `cs_user_initialization` is used.

Physical properties are stored using the `cs_field` API (cell center). For instance, to obtain `rom(iel)`, the mean density (in $kg.m^{-3}$), one must declare a `ncelet` array `cpro_rom` and then call `call field_get_val_s(icrom, cpro_rom)`.
Physical properties (`rom, viscl, cp, ...`) are computed in `ppphyv` and are not to be modified here.

The `cs_user_initialization-fuel.f90` example illustrates how the user may initialise quantities related to gaseous species and droplets compositions in addition to the chosen turbulent model.

7.3.2 Boundary conditions

Boundary conditions are defined as usual on a per-face basis in `cs_user_boundary_conditions`. They may be assigned in two ways:

. for "standard" boundary conditions (inlet, free outlet, wall, symmetry): a code is defined in the array `itypfb` (of dimensions equal to the number of boundary faces). This code will then be used by a non-user subroutine to assign the conditions.

. for "non-standard" conditions: see details given in `cs_user_boundary_conditions-fuel.f90` example.

7.4 Radiative thermal transfers in semi-transparent gray media

7.4.1 Initialisation of the radiation main parameters

The main radiation parameters can be initialise in the Graphical User Interface (GUI) or in the user subroutine `cs_user_radiative_transfer_param`. In the GUI, under the heading "Thermophysical models", when one of the two thermal radiative transfers models is selected, see Figure 46, additional items appear. The user is asked to choose the number of directions for angular discretisation, to define the absorption coefficient and select if the radiative calculation are restarted or not, see Figure 47 and Figure 49. When "Advanced options" is selected for both models Figure 48 or Figure 50 appear, the user must fill the resolution frequency and verbosity levels. In addition, the activation of the radiative transfer leads to the creation of an item "Surface solution control" under the heading "Calculation control", see Figure 51, where radiative transfer variables can be selected to appear in the output listing.

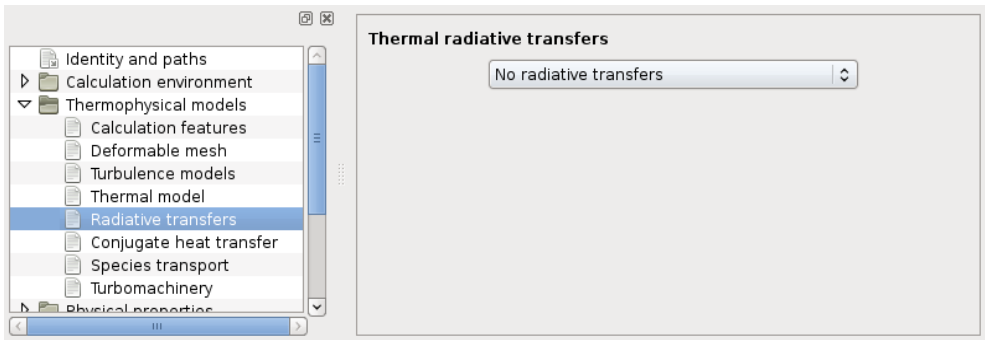

Figure 46: Radiative transfers models

If the GUI is not used, `cs_user_radiative_transfer_param` is one of the two subroutine which must be completed by the user for all calculations including radiative thermal transfers. It is called only during the calculation initialisation. It is composed of three headings. The first one is dedicated to the activation of the radiation module, only in the case of classic physics.
WARNING: when a calculation is ran using a specific physics module, this first heading must not be completed. The radiation module is then activated or not, according to the parameter file related to the considered specific physics.

In the second heading the basic parameters of the radiation module are indicated.
Finally, the third heading deals with the selection of the post-processing graphic outputs. The variables to treat are splitted into two categories: the volumetric variables and those related to the boundary faces.

For more details about the different parameters, the user may refer to the keyword list (§ 8).

7.4.2 Radiative transfers boundary conditions

These informations can be filled by the user through the Graphical User Interface (GUI) or by using the subroutine `cs_user_radiative_transfer_bcs.f90` (called every time step). If the interface is used, when one of the "Radiative transfers" options is selected in Figure 46, it activates specific boundary conditions each time a "Wall" is defined, see Figure 52. The user can then choose between 3 cases. The parameters the user must specify are displayed for one of them in Figure 53.

When the GUI is not used, `cs_user_radiative_transfer_bcs.f90` is the second subroutine necessary for every calculation which includes radiative thermal transfers. It is used to give all the necessary

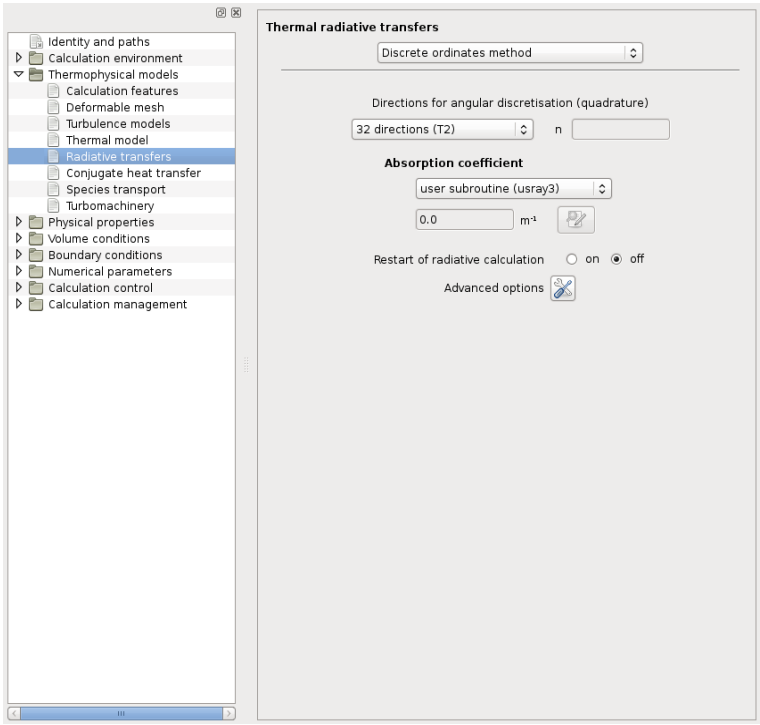

Figure 47: Radiative transfers - parameters of the DO method

parameters concerning, in the one case, the wall temperature calculation, and in the other, the coupling between the thermal scalar (temperature or enthalpy), and the radiation module at the calculation domain boundaries. It must be noted that the boundary conditions concerning the thermal scalar which may have been defined in the subroutine `cs_user_boundary_conditions` will be modified by the radiation module according to the data given in `cs_user_radiative_transfer_bcs.f90` (cf. §3.9.4).
A zone number must be given to each boundary face [24] and, specifically for the walls, a boundary condition type and an initialisation temperature (in Kelvin). The initialisation temperature is only used to make the solving implicit at the first time step. The zone number allows assigning an arbitrary integer to a set of boundary faces having the same radiation boundary condition type. This gathering is used by the calculation, and in the listing to print some physical values (mean temperature, net radiative flux ...). An independent graphic output in *EnSight* format is associated with each zone and allows the display on the boundary faces of the variables selected in the third heading of the subroutine `cs_user_radiative_transfer_param`.
A boundary condition type stored in the array ISOTHP is associated with each boundary face. There are five different types:

- `itpimp`: wall face with imposed temperature,
- `ipgrno`: for a grey or black wall face, calculation of the temperature by means of a flux balance,
- `iprefl`: for a reflecting wall face, calculation of the temperature by means of a flux balance. This is fixed at 2000 in `radiat` and cannot be modified.
- `ifgrno`: grey or black wall face to which a conduction flux is imposed,
- `ifrefl`: reflecting wall face to which a conduction flux is imposed, which is equivalent to impose this flux directly to the fluid.

[24] This must be less than the maximum allowable by the code, `nozrdm`. This is fixed at 2000 in `radiat` and cannot be modified.

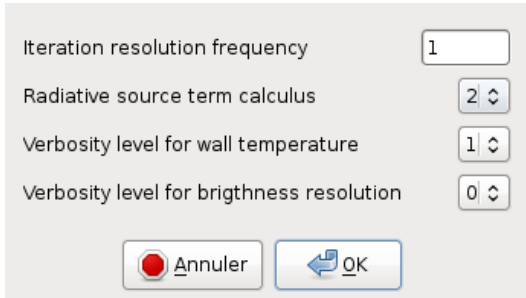

Figure 48: Radiative transfers - advanced parameters of the DO method

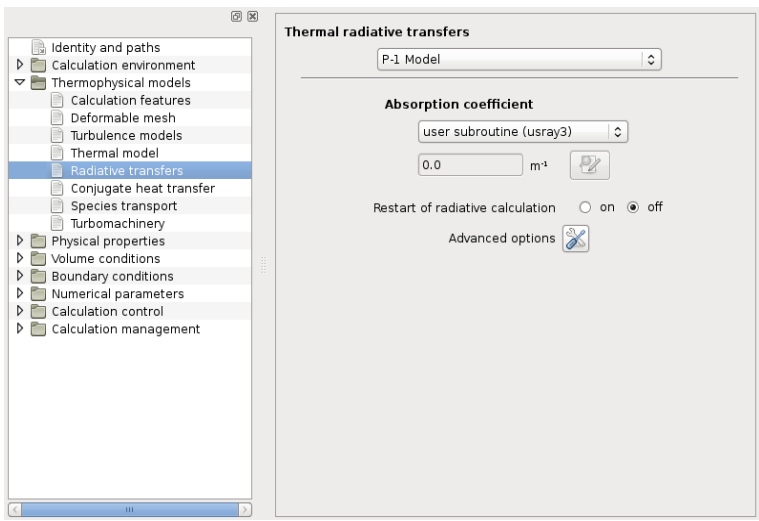

Figure 49: Radiative transfers - parameters of the P-1 model

Depending on the selected boundary condition type at every wall face, the code needs to be given some additional information:

- `itpimp`: the array `tintp` must be completed with the imposed temperature value and the array `epsp` must be completed with the emissivity value (strictly positive).
- `ipgrno`: must be given: an initialisation temperature in the array `tintp`, the wall emissivity (strictly positive, in `epsp`), thickness (in `epap`), thermal conductivity (in `xlamp`) and an external temperature (in `textp`) in order to calculate a conduction flux across the wall.
- `iprefl`: must be given: an initialisation temperature (in `tintp`), the wall thickness (in `epap`) and thermal conductivity (in `xlamp`) and an external temperature (in `textp`).
- `ifgrno`: must be given: an initialisation temperature (in `tintp`), the wall emissivity (in `epsp`) and the conduction flux (in W/m^2 whatever the thermal scalar, enthalpy or temperature) in the array `rcodcl`. The value of `rcodcl` is positive when the conduction flux is directed from the inside of the fluid domain to the outside (for instance, when the fluid heats the walls). If the conduction flux is null, the wall is adiabatic.
- `ifrefl`: must be given: an initialisation temperature (in `tintp`) and the conduction flux (in W/m^2 whatever the thermal scalar) in the array `rcodcl`. The value of `rcodcl` is positive when the conduction flux is directed from the inside of the fluid domain to the outside (for instance,

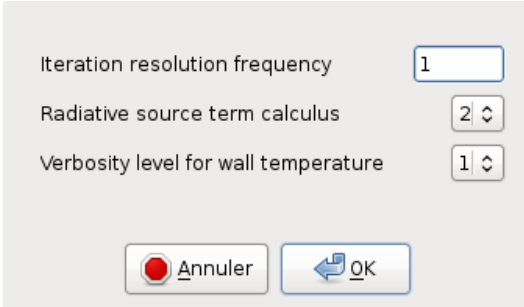

Figure 50: Radiative transfers - advanced parameters of the P-1 model

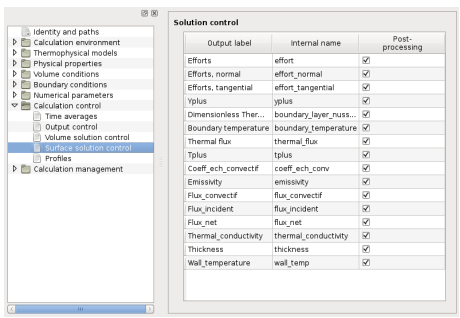

Figure 51: Calculation control - Radiative transfers post-processing output

when the fluid heats the walls). If the conduction flux is null, the wall is adiabatic. The flux received by `rcodcl` is directly imposed as boundary condition for the fluid.

WARNING: it is mandatory to set a zone number to every boundary face, even those which are not wall faces. These zones will be used during the printing in the listing. It is recommended to gather together the boundary faces of the same type, in order to ease the reading of the listing.

7.4.3 Absorption coefficient of the medium, boundary conditions for the luminance and calculation of the net radiative flux

When the absorption coefficient is not constant, the subroutine `cs_user_rad_transfer_absorption` is called instead at each time step. It is composed of three parts. In the first one, the user must provide the absorption coefficient of the medium in the array CK, for each cell of the fluid mesh. By default, the absorption coefficient of the medium is 0, which corresponds to a transparent medium.

WARNING: when a specific physics is activated, it is forbidden to give a value to the absorption coefficient in this subroutine. In this case, the coefficient is either calculated automatically, or provided by the user via *a thermo-chemical parameter file (dp_C3P or dp_C3PSJ for gas combustion, and dp_FCP for pulverised coal combustion).*

The two following parts of this subroutine concern a more advanced use of the radiation module. It is about imposing boundary conditions to the equation of radiative transfer and net radiative flux calculation, in coherence with the luminance at the boundary faces, when the user wants to give it a particular value. In most cases, the given examples do not need to be modified.

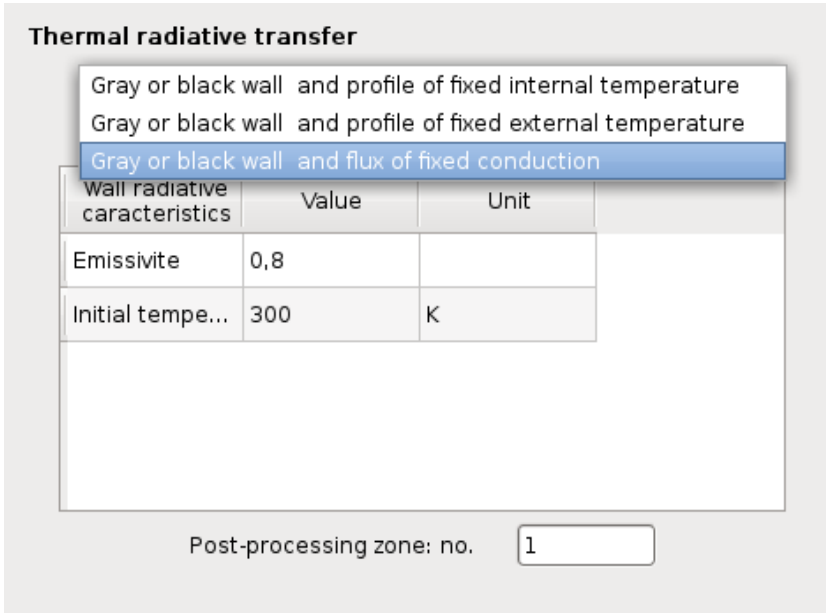

Figure 52: Boundary conditions - choice of wall thermal radiative transfers

7.5 Conjugate heat transfer

7.5.1 Thermal module in a 1D wall

subroutine called at every time step

This subroutine takes into account the wall-affected thermal inertia. Some boundary faces are treated as a solid wall with a given thickness, on which the code resolves a one-dimensional equation for the heat conduction. The coupling between the 1D module and the fluid works in a similar way to the coupling with the SYRTHES. By construction, the user is not able to account for the heat transfer between different parts of the wall. A physical analysis of each problem, case by case is required in order to evaluate the relevance of its usage by way of a report of the simple conditions (temperature, zero-flux) or a coupling with SYRTHES.

The use of this code requires that the thermal scalar is defined as (`iscalt`> 0).

WARNING: The 1D thermal module is developed assuming the thermal scalar as a temperature. If the thermal scalar is an enthalpy, the code calls the subroutine `usthht` *for each transfer of data between the fluid and the wall in order to convert the enthalpy to temperature and vice-versa. This function has not been tested and is firmly discouraged. If the thermal variable is the total (compressible) energy, the thermal module will not work.*

7.5.2 Fluid-Thermal coupling with SYRTHES

When the user wishes to couple *Code_Saturne* with SYRTHES to include heat transfers, he can do so with using with the Graphical User Interface (GUI) or the `cs_syrthes_coupling` user function. To set such a coupling in the Graphic User Interfacee (GUI), a thermal scalar must be selected first in the item "Thermal scalar" under the heading "Thermophysical models". Then the item "Conjugate heat transfer" will appear, see Figure54. The zones where the coupling occurs must be defined and a projection axis can be specified in case of 2D coupling.

If the function `cs_user_syrthes_coupling` is used, the user must specify the arguments passed to the

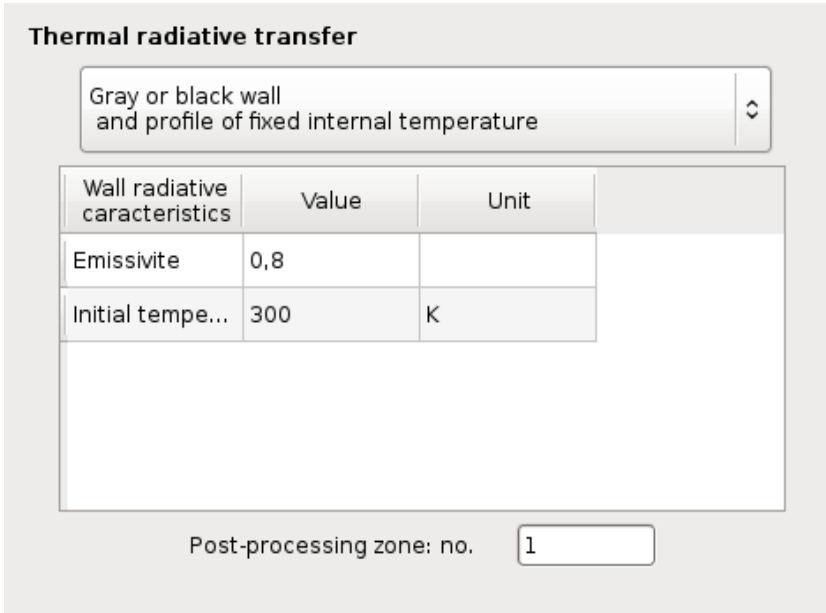

Figure 53: Boundary conditions - example of wall thermal radiative transfer

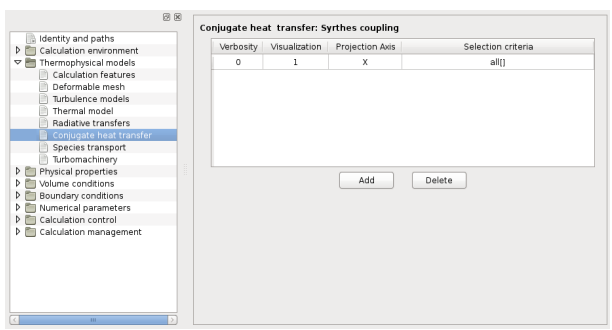

Figure 54: Thermophysical models - coupling with SYRTHES

'`cs_syr_coupling_define`' function. These arguments are:

- `syrthes_name` is the matching SYRTHES application name (useful only when more than one SYRTHES and one *Code_Saturne* domain are present),
- `boundary_criteria` is the surface selection criteria,
- `volume_criteria` is the volume selection criteria,
- `projection_axis`: ' ' if the user wishes to use a 3D standard coupling, or specify 'x', 'y', or 'z' as the projection axis if a 2D coupling with SYRTHES is used,
- `verbosity` is the verbosity level.
- `visualization` is the visualization level.

Examples are provided in `cs_user_coupling.c`.

The user may also define global coupling options relative to the handling of time-stepping, by adapting the example `cs_user_coupling` in the `cs_user_coupling.c` file. In the case of multiple couplings, these options are global to all SYRTHES and *Code_Saturne* couplings.

7.6 Particle-tracking (Lagrangian) Module

7.6.1 General information

- The particle-tracking (or Lagrangian) module enables the simulation of poly-dispersed particulate flows, by calculating the trajectories of individual particles, mainly characterized by their diameter and density (if no heat nor mass transfer between particle and fluid are activated).
- The standard use of the particle-tracking module follows the **Moments/PDF approach**: the instantaneous properties of the underlying flow needed to calculate the particle motion are reconstructed from the averaged values (obtained by Reynolds-Averaged Navier-Stokes simulation) by using stochastic processes. The statistics of interest are then obtained through Monte-Carlo simulation.
- As a consequence, is is important to emphasize that the most important (and physically meaningful) results of a particle-tracking calculation following the Moments/PDF approach are **statistics**. Volume and surface statistics, steady or unsteady, can be calculated. Individual particle trajectories (as 1D, *EnSight*-readable cases) and displacements (as *EnSight*-readable animations) can also be provided, but only for illustrative purposes.

7.6.2 Activating the particle-tracking module

The activation of the particle-tracking module is performed either:

- in the Graphical User Interface (GUI): `Calculation features` → `Thermophysical models` → `Eulerian-Lagrangian multi-phase treatment` → `particles and droplets tracking`
- or in the user function `cs_user_lagr_model`.

7.6.3 Basic guidelines for standard simulations

Except for cases in which the flow conditions depend on time, it is generally recommended to perform a first Lagrangian calculation whose aim is to reach a steady-state (i.e. to reach a time starting from which the relevant statistics do not depend on time anymore). In a second step, a calculation restart is done to calculate the statistics. When the single-phase flow is steady and the particle volume fraction is low enough to neglect the particles influence on the continuous phase behaviour, it is recommended to perform a Lagrangian calculation on a frozen field.

It is then possible to calculate steady-state volumetric statistics and to give a statistical weight higher than 1 to the particles, in order to reduce the number of simulated ("numerical") particles to treat while keeping the right concentrations. Otherwise, when the continuous phase flow is steady, but the two-coupling coupling must be taken into consideration, it is still possible to activate steady statistics. When the continuous phase flow is unsteady, it is no longer possible to use steady statistics. To have correct statistics at every moment in the whole calculation domain, it is imperative to have an established particle seeding and it is recommended (when it is possible) not to impose statistical weights different from the unity.

Finally, when the so-called complete model is used for turbulent dispersion modelling, the user must make sure that the volumetric statistics are directly used for the calculation of the locally undisturbed fluid flow field.

When the thermal evolution of the particles is activated, the associated particulate scalars are always the inclusion temperature and the locally undisturbed fluid flow temperature expressed in degrees Celsius, whatever the thermal scalar associated with the continuous phase is (*i.e.* temperature or

enthalpy). If the thermal scalar associated with the continuous phase is the temperature in Kelvin, the unit is converted automatically into Celsius. If the thermal scalar associated with the continuous phase is the enthalpy, the enthalpy-temperature conversion subroutine `usthht` must be completed for `mode`=1, and must express temperatures in degrees Celsius. In all cases, the thermal backward coupling of the dispersed phase on the continuous phase is adapted to the thermal scalar transported by the fluid.

7.6.4 Prescribing the main modelling parameters (GUI and/or `cs_user_lagr_model`)

Use of the GUI

In the GUI, the selection of the Lagrangian module activates the heading `Particle and droplets tracking` in the tree menu. The initialization is performed in the three items included in this heading:

- `Global settings`. The user defines in this item the kind of Euler/Lagrange multi-phase treatment, the main parameters, the specific physics associated with the particles and advanced numerical options, see Figure 55 to Figure56.
- `Statistics`. The user can select the volume and boundary statistics to be post-processed.
- `Output`. The user defines the output frequency and post-processing options for particles and select the variables that will appear in the listing.

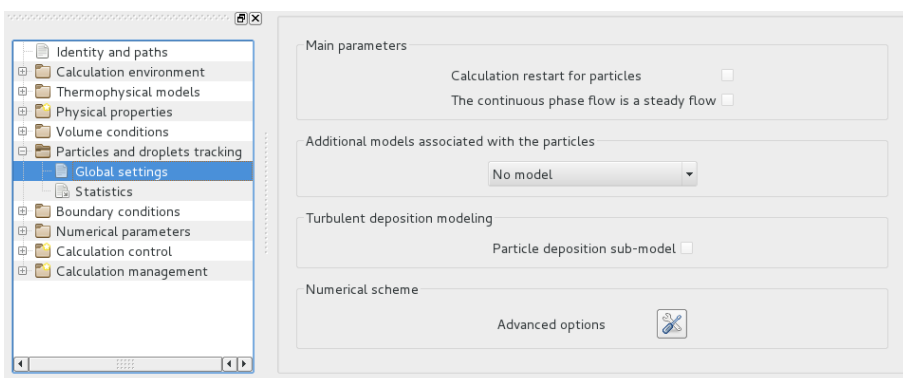

Figure 55: Lagrangian module - View of the `Global Settings` page

Use of the subroutine `cs_user_lagr_model`

When the GUI is not used, `cs_user_lagr_model` must be completed. This function gathers in different headings all the keywords which are necessary to configure the Lagrangian module. The different headings refer to:

- the global configuration parameters
- the specific physical models describing the particle behaviour
- the backward coupling (influence of the dispersed phase on the continuous phase)
- the numerical parameters
- the volumetric statistics

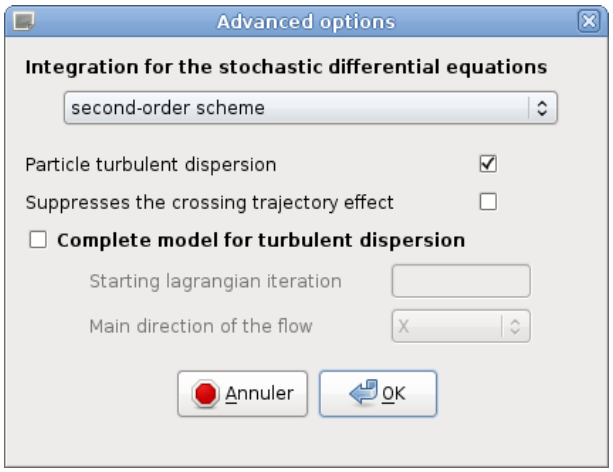

Figure 56: Lagrangian module - Global Settings, advanced numerical options

- the boundary statistics

For more details about the different parameters, the user may refer to the keyword list (§ **??**).

7.6.5 Prescribing particle boundary conditions (GUI and/or `uslag2`)

In the framework of the multiphase Lagrangian modelling, the management of the boundary conditions concerns the particle behaviour when there is an interaction between its trajectory and a boundary face. These boundary conditions may be imposed independently of those concerning the Eulerian fluid phase (but they are of course generally consistent). The boundary condition zones are actually redefined by the Lagrangian module (cf. §3.9.4), and a type of particle behaviour is associated with each one. The boundary conditions related to particles can be defined in the Graphical User Interface (GUI) or in the subroutine `uslag2`. More advanced user-defined boundary conditions can be prescribed in the user-subroutine `uslain`.

Use of the GUI

In the GUI, selecting the Lagrangian module in the activates the item `Particle boundary conditions` under the heading `Boundary conditions` in the tree menu. Different options are available depending on the type of standard boundary conditions selected (wall, inlet/outlet, etc...), see Figure 57.

7.6.6 Advanced particle-tracking set-up

In this section, some information is provided for a more advanced numerical set-up of a particle-tracking simulation.

User-defined stochastic differential equations

An intervention in the `cs_user_lagr_sded` function is required if supplementary user variables are added to the particle state vector. This function is called at each Lagrangian sub-step.

The integration of the stochastic differential equations associated with supplementary particulate variables is done in this function.
When the integration scheme of the stochastic differential equations is a first-order (`nordre` = 1), this

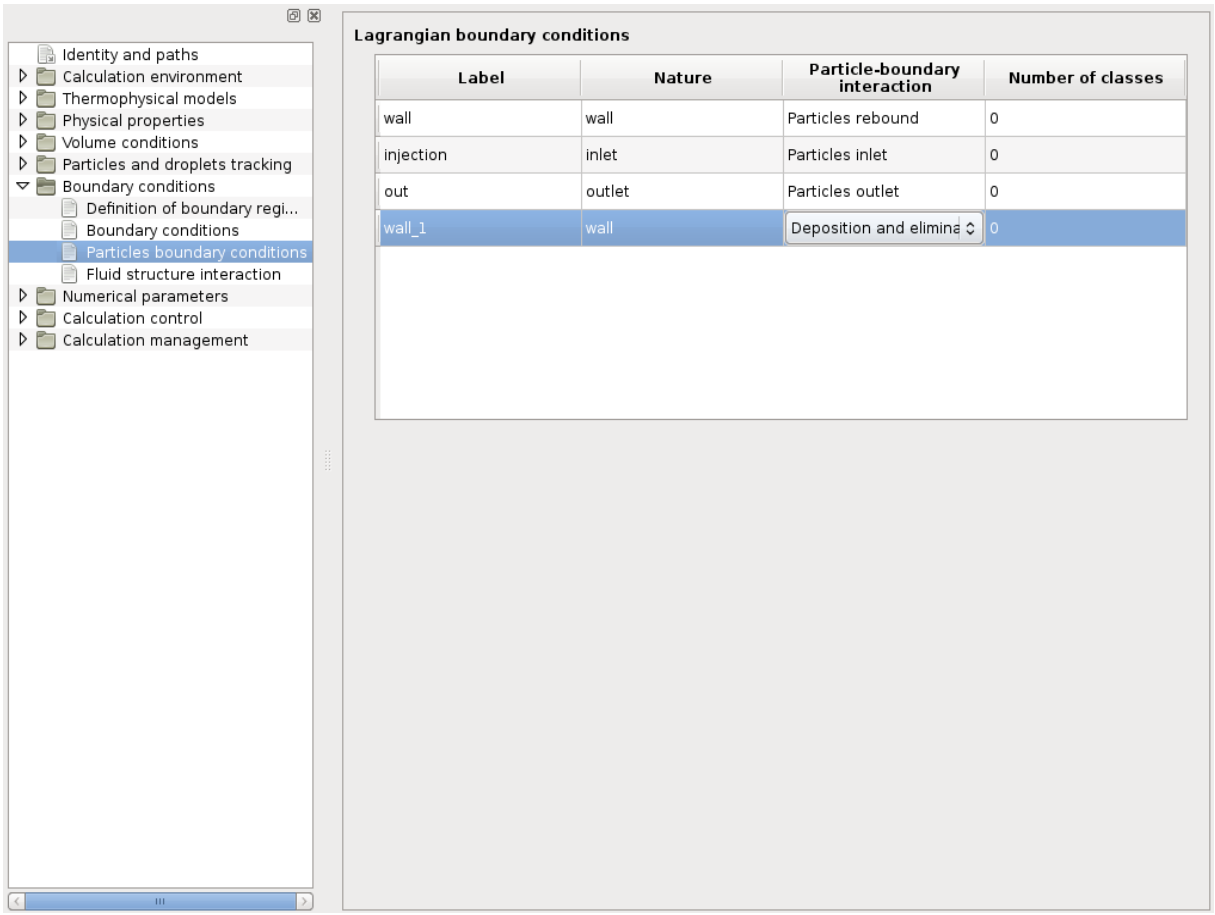

Figure 57: Lagrangian module - boundary conditions

subroutine is called once every Lagrangian iteration, if it is a second-order (`nordre` = 2), it is called twice.

The solved stochastic differential equations must be written in the form:

$$\frac{d\Phi_p}{dt} = -\frac{\Phi_p - \Pi}{\tau_\phi}$$

where Φ_p is the Ith supplementary user variable, τ_ϕ is a quantity homogeneous to a characteristic time, and Π is a coefficient which may be expressed as a function of the other particulate variables.
In order to do the integration of this equation, the following parameters must be provided:

- τ_ϕ, equation characteristic time, in the array `auxl1` for every particle,
- Π , equation coefficient, in the array `auxl2`. If the integration scheme is a first-order, then Π is expressed as a function of the particulate variables at the previous iteration, stored in the array `eptpa`. If the chosen scheme is a second-order, then Π is expressed at the first call of the subroutine (prediction step `nor` = 1) as a function of the variables at the previous iteration (stored in `eptpa`), then at the second call (correction step `nor` = 2) as a function of the predicted variables stored in the array `eptp`.

If necessary, the thermal characteristic time τ_c, whose calculation can be modified by the user in the function `cs_user_lagr_rt_t`, is stored for each particle in the first part of the array `tempct`.

User-defined particle relaxation time

The particle relaxation time may be modified in the `cs_user_lagr_rt` function according to the chosen formulation of the drag coefficient. The particle relaxation time, modified or not by the user, is available in the array `taup`.

User-defined particle thermal characteristic time

The particle thermal characteristic time may be modified in the subroutine `uslatc` according to the chosen correlation for the calculation of the Nusselt number. This subroutine is called at each Lagrangian sub-step. The thermal characteristic time, modified or not by the user, is available in the zone `tempct(nbpmax,1)` of the array `tempct`.

7.7 Compressible module

When the compressible module[25] is activated, it is recommended to:

- use the option "time step variable in time and uniform in space" (`idtvar`=1) with a maximum Courant number of 0.4 (`coumax`=0.4): these choices must be written in `cs_user_parameters.f90` or specified with the GUI.
- keep the convective numerical schemes proposed by default (*i.e.*: upwind scheme).

With the compressible algorithm, the specific total energy is a new solved variable `isca(ienerg)`). The temperature variable deduced from the specific total energy variable is `isca(itempk)` for the compressible module.
Initialisation of the options of the variables, boundary conditions, initialisation of the variables and management of variable physical properties can be done with the GUI. We describe below the subroutines the user has to fill in without the GUI.

7.7.1 Initialisation of the options of the variables

Subroutines called at each time step.

When the GUI is not being used, the subroutines `uscfx1` and `uscfx2` in `cs_user_parameters.f90` must be completed by the user.

`uscfx1` allows to specify:

- `ieos`: equation of state (only perfect gas with a constant adiabatic coefficient, `ieos=1` is available, but the user can complete the subroutine `cfther`, which is not a user subroutine, to add new equations of state).
- `call field_set_key_int(ivarfl(isca(itempk)), kivisl, ...)`: molecular thermal conductivity, constant (`-1`) or variable (`0`).
- `iviscv`: volumetric molecular viscosity, constant (`0`) or variable (`1`).

`uscfx2` allows to specify:

- `ivivar`: molecular viscosity, constant (`0`) or variable (`1`).

[25] For more details concerning the compressible version, the user may refer to the theory guide [11] and the document "Implantation d'un algorithme compressible dans *Code_Saturne*", Rapport EDF 2003, HI-83/03/016/A, P. Mathon, F. Archambeau et J.-M. Hérard.

- `visls0(itempk)`: reference molecular thermal conductivity.
- `viscv0`: reference volumetric molecular viscosity.
- `xmasmr`: molar mass of the perfect gas (`ieos=1`).
- `icfgrp`: specify if the hydrostatic equilibrium must be accounted for in the boundary conditions.

7.7.2 Management of the boundary conditions

Subroutine called at each time step.

When running the compressible module without a GUI, the `cs_user_boundary_conditions` subroutine can be used to define specific boundary conditions (see the `cs_user_boundary_conditions-compressible` file in the directory `EXAMPLES` for examples of boundary conditions with the compressible module).

With the compressible module, the following types of boundary condition are avaliable:

- Inlet/outlet for which velocity and two thermodynamics variables are known.
- Subsonic inlet with imposed total pressure and total energy.
- Subsonic outlet with imposed static pressure.
- Supersonic outlet.
- Wall (adiabatic or not).
- Symmetry.

It is advised to only use these predefined boundary conditions type for the compressible module.

7.7.3 Initialisation of the variables

Subroutine called only at the initialisation of the calculation

When the GUI is not used, the subroutine `cs_user_initialization` is used initialize the velocity, turbulence and passive scalars (see the `cs_user_initialization-compressible` file in the directory `EXAMPLES` for examples of initialisations with the compressible module). Concerning pressure, density, temperature and specific total energy, only 2 variables out of these 4 are independent. The user may then initialise the desired variable pair (apart from temperature-energy) and the two other variables will be calculated automatically by giving the right value to the variable `ithvar` used for the call to the subroutine `cfther`.

7.7.4 Management of variable physical properties

Subroutine called at each time step.

Without the GUI, all of the laws governing the physical properties of the fluid (molecular viscosity, molecular volumetric viscosity, molecular thermal conductivity and molecular diffusivity of the user-defined scalars) can be specified in the subroutine `usphyv` of the `cs_user_physical_properties` file, which is then called at each time step. This subroutine replaces and is similar to `usphyv`.

The user should check that the defined laws are valid for the whole variation range of the variables. Moreover, as only the perfect gas with a constant adiabatic coefficient equation of state is available, it is not advised to give a law for the isobaric specific heat without modifying the equation of state in the subroutine `cfther` which is not a user subroutine.

7.8 Management of the electric arcs module

7.8.1 Activating the electric arcs module

The electric arcs module is activated either:

- in the Graphical User Interface (GUI): `Calculation features` → `Electrical models`
- or in the user subroutine `usppmo`, by setting the `ielarc` or `ieljou` parameter to a non-null value.

7.8.2 Initialisation of the variables

Subroutine called only at initialisation of the calculation

The subroutine `cs_user_initialization` allows the user to initialise some of the specific physics variables prompted via `usppmo`. It is called only during the initialisation of the calculation. As usual,the user has access to many geometric variables so that the zones can be treated separately if needed.

The values of potential and its constituents are initialised if required.

It should be noted that the enthalpy is relevant.

- For the electric arcs module, the enthalpy value is taken from the temperature of reference `t0` (given in `cs_user_parameters.f90`) from the temperature-enthalpy tables supplied in the data file `dp_ELE`. The user must not intervene here.
- For the Joule effect module, the value of enthalpy must be specified by the user . An example is given of how to obtain the enthalpy from the temperature of reference `t0`(given in `cs_user_parameters.f90`), the temperature-enthalpy law must be supplied. A code is suggested in the `usthht` subroutine (provided for the determination of physical properties).

7.8.3 Variable physical properties

All the laws of the variation of physical data of the fluid are written (when necessary) in the subroutine `cs_user_physical_properties`. It is called at each time step.

WARNING: For the electric module, it is here that all the physical variables are defined (including the relative cells and the eventual user scalars): `cs_user_physical_properties` *is not used.*

The user should ensure that the defined variation laws are valid for the whole range of variables. Particular care should be taken with non-linear laws (for example, a 3rd degree polynomial law giving negative values of density)

WARNING: In the electric module, all of the physical properties are considered as variables and are therefore stored using the `cs_field` *API.* `cp0`*,* `viscls0` *and* `viscl0` *are not used*

For the Joule effect, the user is required to supply the physical properties in the subroutine. Examples are given which are to be adapted by the user. If the temperature is to be determined to calculate the physical properties, the solved variable, enthalpy must be deduced. The preferred temperature-enthalpy law can be selected in the subroutine `usthht` (an example of the interpolation is given from the law table. This subroutine can be re-used for the initialisation of the variables(`cs_user_initialization`)) For the electric arcs module, the physical properties are interpolated from the data file `dp_ELE` supplied by the user. Modifications are generally not necessary.

7.8.4 Boundary conditions

For the electric module,each boundary face in `cs_user_boundary_conditions` should be associated with a `izone` number [26](the color `icoul` for example) in order to group together all the boundary faces of the same type. In the `cs_user_boundary_conditions` report, the main change from the users point of view concerns the specification of the boundary conditions of the potential, which isn't implied by default. The Dirichlet and Neumann conditions must be imposed explicitly using `icodcl` and `rcodcl` (as would be done for the classical scalar).

Furthermore, if one wishes to slow down the power dissipation (Joule effect module) or the current (electric arcs module) from the imposed values (`puismp` and `couimp` respectively), they can be changed by the potential scalar as shown below:

- For the electric arcs, the imposed potential difference can be a fixed variable: for example, the cathode can be fixed at 0 and the potential at the anode contains the variable `dpot`. This variable is initialised in in `cs_user_parameters.c` by an estimated potential difference. If `ielcor`=1 (see `cs_user_parameters.c`), `dpot` is updated automatically during the calculation to obtain the required current.
- For the Joule effect module, `dpot` is again used with the same signification as in the electric arcs module. If `dpot` is not wanted in the setting of the boundary conditions, the variable `coejou` can be used. `coejou` is the coefficient by which the potential difference is multiplied to obtain the desired power dissipation. By default this begins at 1 and is updated automatically. If `ielcor`=1 (see `cs_user_parameters.c`), multiply the imposed potentials in `cs_user_boundary_conditions` by `coejou` at each time step to achieve the desired power dissipation.

WARNING: In the case of alternating current, attention should be paid to the values of potential imposed at the limits: the variable named "real potential" represents an affective value if the current is in single phase, and a "real part" if not.

- For the Joule studies, a complex potential is sometimes needed (`ippmod(ieljou)`=2): this is the case in particular where the current has three phases. To have access to the phase of the potential, and not just to its amplitude, the two variables must be deleted: in *Code_Saturne*, there are two arrays specified for this role, the real part and the imaginary part of the potential. For use in the code, these variables are named "real potential" and "imaginary potential". For an alternative sinusoidal potential Pp, the maximum value is noted as $Pp_{\max}$, the phase is noted as ϕ, the real potential and the imaginary potential are respectively $Pp_{\max}\, cos\phi$ and $Pp_{\max}\, sin\phi$.
- For the Joule studies in which one does not have access to the phases, the real potential (imaginary part =0) will suffice (`ippmod(ieljou)=1`): this is obviously the case with continuous current, but also with single phase alternative current. In *Code_Saturne* there is only 1 variable for the potential, called "real potential". Pay attention to the fact that in alternate current, the "real potential" represents a effective value of potential , $\frac{1}{\sqrt{2}}\, Pp_{\max}$ (in continuous current there is no such ambiguity).

<u>Additions for transformers</u>

The following additional boundary conditions must be defined for tansformers:

- the intensity at each electrode
- the voltage on each terminal of transformers. To achieve it, the intensity, the rvoltage at each termin, the Rvoltage, and the total intensity of the transformer are calculated.

[26] `izone` must be less than the maximum value allowed by the code, `nozzppm`. This is fixed at 2000 in `ppvar` and cannot be modified.

Finally, a test is performed to check if the offset is zero or if a boundary face is in contact with the ground.

7.8.5 Initialisation of the variable options

The subroutine `cs_user_parameters` (in `cs_user_parameters.c`) is called at each time step. It allows:

- to give the coefficient of relaxation of the density `srrom`:
  $\rho^{n+1} = \texttt{srrom} * \rho^n + (1 - \texttt{srrom})\rho^n$
  (for the electric arcs, the sub-relaxation is taken into account during the 2nd time step;)
- to indicate if the data will be fixed in the power dissipation or in the current, done in `ielcor`.
- target either the current fixed as `couimp` (electric arcs module) or the power dissipation `puism` (Joule module effect).
- to fix the initial value of potential difference `dpot`, the for the calculations with a single fixed parameter as `couimp` or `puism`.
- to define type of scaling model for electric arcs `modrec`. If scaling by a resetting plane is choosen then `idreca` defines the current density component and `crit_reca` the plane used for resetting of electromagnetic variables.

7.8.6 Post-processing output

The algebraic variables related to the electric module are provided by default:

- gradient of real potential in Vm^{-1} ($\underline{\nabla} Pot_R = -\underline{E}$)
- density of real current in Am^{-2} ($\underline{j} = \sigma \underline{E}$)

specifically for the Joule module effect with `ippmod(ieljou)`=2 :

- gradient of imaginary potential in Vm^{-1}
- density of real current in Am^{-2}

specifically for the electric arcs module with `ippmod(ielarc)`=2 :

- magnetic field in T ($\underline{\mathrm{B}} = \underline{\mathrm{rot}}\,\underline{\mathrm{A}}$)

The post-processing output will be created automatically (on all output volume meshes for which the automatic output of main variables is active).

7.9 *Code_Saturne*-*Code_Saturne* coupling

Subroutine called once during the calculation initialisation.

The user function `cs_user_saturne_coupling` (in `cs_user_coupling.c` is used to couple *Code_Saturne* with itself. It is used for turbo-machine applications for instance, the first *Code_Saturne* managing the fluid around the rotor and the other the fluid around the stator. In the case of a coupling between two *Code_Saturne* instances, first argument `saturne_name` of the function '`cs_sat_coupling_define`' is ignored. In case of multiple couplings, a coupling will be matched with available *Code_Saturne* instances based on that argument, which should match the directory name for the given coupled domain..
The arguments of '`cs_sat_coupling_define`' are:

- `saturne_name`: the matching *Code_Saturne* application name,
- `volume_sup_criteria`: the cell selection criteria for support,
- `boundary_sup_criteria`: the boundary face selection criteria for support (not functional),
- `volume_cpl_criteria`: the cell selection criteria for coupled cells,
- `boundary_cpl_criteria`: the boundary face selection criteria for coupled faces,
- `verbosity`: the verbosity level.

7.10 Fluid-Structure external coupling

Subroutine called only once

The subroutine `usaste` belongs to the module dedicated to external Fluid-Structure coupling with *Code_Aster*. Here one defines the boundary faces coupled with *Code_Aster* and the fluid forces components which are given to structural calculation. When using external coupling with *Code_Aster*, structure numbers necessarily need to be negative; the references of coupled faces being i.e. -1, -2, *etc.* The subroutine performs the following operations:

- '`getfbr`' is called to get a list of elements matching a geometrical criterion or reference number then a structure number (negative value) is associated to these elements.
- the value passed to `asddlf`, for user-chosen component, for every negative structure number, defines the movement imposed to the external structure.

7.11 ALE module

7.11.1 Initialisation of the options

This initialisation can be performed in the Graphical User Interface (GUI) or in the subroutines `usipph` and `usstr1`. Firstly, when the "Mobile mesh" is selected in GUI under the "Thermophysical models" heading, additional options are displayed. The user must choose the type of mesh viscosity and describe its spatial distribution, see Figure 58.

The following paragraphs are relevant if the GUI is not used.

SUBROUTINE `USIPPH`
Subroutine called at the beginning. This subroutine completes `cs_user_parameters.f90`.

`usipph` allows setting options for the ALE module, and in particular to activate the ALE module (`iale=1`).

SUBROUTINE `USSTR1`

This subroutine reads in `cs_user_fluid_structure_interaction.f90`. It allows to specify the following pieces of information for the structure module:

- the index of the structure, (`idfstr(ifac)` where `ifac` is the index of the face). Then the total number of structures `nbstru` is automatically computed by the code. Be careful, the value must belong to 1, ..., `nbstru`.
- the initial value of displacement, velocity and acceleration (`xstr0`, `xstreq` and `vstr0`).

Below is a list of the different variables that might be modified:

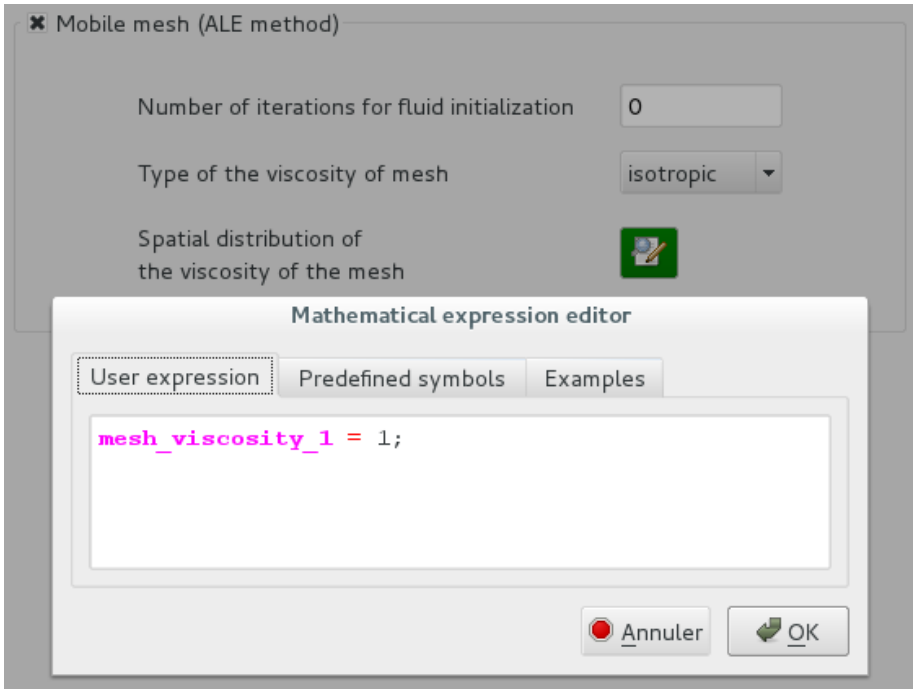

Figure 58: Thermophysical models - mobile mesh (ALE method)

- `idfstr(ifac)`
  the index of the structure, (`idfstr(ifac)` where `ifac` is the index of the face), 0 if the face is not coupled to any structure.
- `xstr0(i,k)`
  initial position of a structure, where `i` is the dimension of space and `k` the index of the structure
- `xstreq(i,k)`
  equilibrum position of a structure, where `i` is the dimension of space and `k` the index of the structure
- `vstr0(i,k)`
  initial velicity of a structure, where `i` is the dimension of space and `k` the index of the structure

7.11.2 Mesh velocity boundary conditions

These boundary conditions can be managed through the Graphical User Interface (GUI) or using the subroutine `usalcl` (called at each time step). With the GUI, when the item "Mobile mesh" is activated the item "Fluid structure interaction" appears under the heading "Boundary conditions". Two types of fluid-structure coupling are offered. The first one is internal, using a simplified structure model and the second is external with *Code_Aster*, see Figure 59 and Figure 60.

SUBROUTINE `usalcl`
When the GUI is not used, the use of `usalcl` is mandatory to run a calculation using the ale module just as it is in `cs_user_parameters.f90`. It is used the same way as `cs_user_boundary_conditions` in the framework of standard calculations, that is to say a loop on the boundary faces marked out by their colour (or more generally by a property of their family), where the type of mesh velocity boundary condition is definied for each variable.

The main numerical variables are described below.

`ialtyb(nfabor)` [ia]: In the ale module, the user defines the mesh velocity from the colour of the boundary faces, or more generally from their properties (colours, groups, ...), from the boundary conditions defined in `cs_user_boundary_conditions`, or even from their coordinates. To do so, the array `ialtyb(nfabor)` gives for each face `ifac` the mesh velocity boundary condition types marked out by the key words `ivimpo`, `igliss`, `ibfixe` or `ifresf`..

- If `ialtyb(ifac) = ivimpo`: imposed velocity.

  → In the cases where all the nodes of a face have a imposed displacement, it is not necessary to fill the tables with mesh velocity boundary conditions for this face, these will be erased. In the other case, the value of the Dirichlet must be given in `rcodcl(ifac,ivar,1)` for every value of `ivar` (`iuma`, `ivma` and `iwma`). The other boxes of `rcodcl` and `icodcl` are completed automatically.

  The tangential mesh velocity is taken like a tape speed under the boundary conditions of wall for the fluid, except if wall fluid velocity was specified by the user in the interface or `cs_user_boundary_conditions` (in which case it is this speed which is considered).

- if `ialtyb(ifac) = ibfixe`: fixed wall

  → the velocity is null.

- if `ialtyb(ifac) = igliss`: sliding wall

  → symmetry boundary condition on the mesh velocity vector, which means a homogeneous Neumann on the tangential mesh velocity and a zero Dirichlet on the normal mesh velocity.

- if `ialtyb(ifac) = ifresf`: free-surface

  → an imposed mesh velocity such that the fluid mass flux is equal to the mesh displacement in order to mimic the free-surface automatically. Note that the boundary condition on the fluid velocity must be set separately (homogeneous Neumann conditionfor instance).

7.11.3 Modification of the mesh viscosity

The user subroutine `usvima` is used along the ALE (Arbitrary Lagrangian Eulerian Method) module, and allows modifying the mesh viscosity. It is called before the time loop, and before reading restart files (so the mesh is always in its initial position at this stage). The user can modify mesh viscosity values to prevent cells and nodes from huge displacements in awkward areas, such as boundary layer for example. If `iortvm` = 0, the mesh viscosity modelling is considered as isotropic and therefore the `ivisma` field is scalar. If `iortvm` = 1, mesh viscosity modelling is orthotropic therefore that field is a vector field.

Note that for more complex settings, the mesh viscosity could be modified in `cs_user_initialization` or `cs_user_extra_operations`. The matching field's name is `mesh_viscosity`.

7.11.4 Fluid - Structure internal coupling

In the subroutine `cs_user_fluid_structure_interaction` the user provides the parameters of two other subroutines. `usstr1` is called at the beginning of the calculation. It is used to define and initialise the internal structures where fluid-Structure coupling occurs. For each boundary face `ifac`, `idfstr(ifac)` is the index of the structure the face belongs to (if `idfstr(ifac)` = 0, the face `ifac` doesn't belong to any structure). When using internal coupling, structure index necessarily must be strictly positive and smaller than the number of structures. The number of "internal" structures is automatically defined with the maximum value of the `idfstr` table, meaning that internal structure numbers must be defined sequentially with positive values, beginning with integer value '1'.

For each internal structure the user can define:

- an initial velocity `vstr0`
- an initial displacement `xstr0` (*i.e.* `xstr0` is the value of the displacement `xstr` compared to the initial mesh at time t = 0)
- a displacement compared to equilibrium `xstreq` (i.e. `xstreq` is the initial displacement of the internal structure compared to its position at equilibrium; at each time step t and for a displacement xstr(t), the associated internal structure will undergo a force $-k*(t+XSTREQ)$ due to the spring).

xstr0 and vstr0 are initialised with the value 0. When starting a calculation using ALE, or re-starting a calculation with ALE, based on a first calculation without ALE, an initial iteration 0 is automatically performed in order to take initial arrays xstr0, vstr0 and xstreq into account. In any other case, add the following expression 'italin=1' in subroutine usipsu, so that the code can deal with the arrays xstr0, vstr0 and xstreq.

When `ihistr` is set to 1, the code writes in the output the history of the displacement, of the structural velocity, of the structural acceleration and of the fluid force. The value of structural history output step is the same as the one for standard variables nthist.

The second subroutine, usstr2, is called at each iteration. One defines in this subroutine structural parameters (considered as potentially time dependent): *i.e.*, mass m xmstru, friction coefficients c xcstru, and stiffness k xkstru. forstr array gives fluid stresses acting on each internal structure. Moreover it is also possible to take external forces (gravity for example) into account.

. the xstr array indicates the displacement of the structure compared to its position in the initial mesh,

. the xstr0 array gives the displacement of the structures in the initial mesh compared to structural equilibrium,

. the vstr array stands for structural velocity.

xstr, xstr0 and vstr are DATA tables that can be used to define the Mass, Friction and Stiffness arays. These are not to be modified.

The 3D structural equation that is solved is the following one:

$$\underline{\underline{m}}.\partial_{tt}\underline{x} + \underline{\underline{c}}.\partial_t\underline{x} + \underline{\underline{k}}.\left(\underline{x}+\underline{x_0}\right) = \underline{f}, \tag{6}$$

where x stands for the structural displacement compared to initial mesh position xstr, x_0 represents the displacement of the structure in initial mesh compared to equilibrium. Note that $\underline{\underline{m}}$,$\underline{\underline{c}}$, and $\underline{\underline{k}}$ are 3x3 matrices. Equation (6) is solved using a Newmark HHT algorithm. Note that the time step used to solve this equation, dtstr, can be different from the one of fluid calculations. The user is free to define dtstr array. At the beginning of the calculation dtstr is initialised to the value of dtcel (fluid time step).

7.12 Management of the structure property

The use of `usstr2` is mandatory to run a calculation using the ALE module with a structure module. It is called at each time step.

For each structure, the system that will be solved is:

$$M.x^{''} + C.x^{''} + K.(x - x_0 = 0 \tag{7}$$

where

- M is the mass structure (`xmstru`).
- C is the damping coefficient of the structure (`xcstru`).
- K is the spring constant or force constant of the structure (`xkstru`).
- x_0 is the initial position.

Below is a list of the different variables that might be modified:

- `xmstru(i,j,k)`
  mass matrix of the structure, where `i,j` is the array of mass structure and `k` the index of the structure.
- `xcstru(i,j,k)`
  damping matrix coefficient of the structure, where `i,j` is the array of damping coefficient and `k` the index of the structure.
- `xkstru(i,j,k)`
  spring matrix constant of the structure, where `i,j` is the array of spring constant and `k` the index of the structure.
- `forstr(i,k)`
  force vector of the structure, where `i` is the force vector and `k` the index of the structure.

7.13 Management of the atmospheric module

This section describes how to set a calculation using the atmospheric module of *Code_Saturne*. Each paragraph describes a step of the data setting process.

7.13.1 Directory structure

The flowchart (Figure 61) recalls the directory structure of a study generated by *Code_Saturne* (see also 3.1.3). When using the atmospheric module, the structure is identical but a file called `meteo` may be added to the data settings in order to provide vertical profiles of the main variables. This file should be put in the `DATA` directory. For more details about the `meteo` file, see § 7.13.5).

7.13.2 The atmospheric mesh features

An atmospheric mesh has the following specific features:

- The boundary located at the top of the domain should be a plane. So, horizontal wind speed at a given altitude can be prescribed at the top face as an inlet boundary.
- Cells may have very different sizes, from very small (near ground or buildings) to very large (near the top of domain or far from zone of interest).
- Vertical resolution: from tiny cells (e.g. $\Delta z = 1$ m) near the ground to a few hundreds of meters at the top.
- Horizontal resolution: from a few meters to hundreds of meters.
- The length ratio between two adjacent cells (in each direction) should preferably be between 0.7 and 1.3.
- The z axis represents the vertical axis.

A topography map can be used to generate a mesh. In this case, the preprocessor mode is particularly useful to check the quality of the mesh (run type Mesh quality criteria).

7.13.3 Atmospheric flow model and steady/unsteady algorithm

The Graphical User Interface (GUI) may be used to enable the atmospheric flow module and set up the following calculation parameters in the `Thermophysical models-Calculation features` page (see Figure 62):

7.13.3.1 The atmospheric flow model

The user can choose one of the following atmospheric flow models:

- **Constant density**: To simulate neutral atmosphere.
- **Dry atmosphere**: To simulate dry, thermally-stratified atmospheric flows (enables `Potential temperature` as thermal model).
- **Humid atmosphere**: To simulate thermally stratified atmospheric flows (air-water mixture) with phase changes (enables `Liquid potential temperature` as thermal model). The model is described in Bouzereau [15].

7.13.3.2 The time algorithm

- Steady flow algorithm: is the one usually set. It sets a time step variable in space and time. It has to be selected if constant boundary conditions are used.
- Unsteady flow algorithm has to be selected for time varying boundary conditions (the time step can then be variable in time or constant).

Then global numerical parameters have to be set (see Figure 63). Table Table 7.13.4 can help to choose the right parameters depending on the type of atmospheric flow.

7.13.3.3 Warnings

The following points have to be considered when setting the parameters described above:

- The potential temperature thermal model and the liquid potential temperature one (see the paragraph "Atmospheric main variables" for the definition) requires that the vertical component of the gravity is set to $g_z = -9.81 m.s^{-2}$ ($g_x = g_y = 0 m.s^{-2}$), otherwise pressure and density won't be correctly computed.
- As well, the use of scalar with drift for atmospheric dispersion requires the gravity to be set to $g_z = -9.81$ ($g_x = g_y = 0 m.s^{-2}$), even if the density is constant.

7.13.4 Physical properties

The specific heat value has to be set to the atmospheric value $C_p = 1005 J/kg/K$.

7.13.5 Boundary and initial conditions

The `meteo` file can be used to define initial conditions for the different fields and to set up the inlet boundary conditions. For the velocity field, *Code_Saturne* can automatically detect if the boundary is an inlet boundary or an outflow boundary, according to the wind speed components given in the `meteo` file with respect to the boundary face orientation. This is often used for the lateral boundaries of the atmospheric domain, especially if the profile is evolving in time. In the case of inlet flow, the data given in the `meteo` file will be used as the input data (Dirichlet boundary condition) for velocity,

Parameters	**Constant density**	**Dry atmosphere**	**Humid atmosphere**	**Explanation**
pressure boundary condition	Neumann first order	Extrapolation	Extrapolation	In case of **Extrapolation**, the pressure gradient is assumed (and set) constant, whereas in case of **Neumann first order**, the pressure gradient is assumed (and set) to zero.
Improved pressure interpolation in stratified flows	no	yes	yes	If yes, exact balance between the hydrostatic part of the pressure gradient and the gravity term ρg is numerically ensured.
Gravity (gravity is assumed aligned with the z-axis)	$g_z = 0$ or $g_z = -9.81 m.s^{-2}$ (the latter is useful for scalar with drift)	$g_z = -9.81 m.s^{-2}$	$g_z = -9.81 m.s^{-2}$	
Thermal variable	no	potential temperature	liquid potential temperature	
Others variables	no	no	total water content, droplets number	

Table 4: List of parameters

temperature, humidity and turbulent variables. In the case of outflow, a Neumann boundary condition is automatically imposed (except for the pressure). The unit of temperature in the `meteo` file is the degree Celsius whereas the unit in the GUI is the kelvin.

To be taken into account, the `meteo` file has to be selected in the GUI (`Atmospheric flows` page, see Figure 64) and the check box on the side ticked. This file gives the profiles of prognostic atmospheric variables containing one or a list of time stamps. The file has to be put in the `DATA` directory. An example of file `meteo` is given in the directory `DATA/REFERENCE/`. The file format has to be strictly respected. The horizontal coordinates are not used at the present time (except when boundary conditions are based on several meteorological vertical profiles) and the vertical profiles are defined with the altitude above sea level. The highest altitude of the profile should be above the top of the simulation domain and the lowest altitude of the profile should be below or equal to the lowest level of the simulation domain. The line at the end of the `meteo` file should not be empty.

If the boundary conditions are variable in time, the vertical profiles for the different time stamps have to be written sequentially in the `meteo` file.

You can also set the profiles of atmospheric variables directly in the GUI. The following boundary conditions can be selected in the GUI:

- Inlet/Outlet is automatically calculated for lateral boundaries (e.g. North, West...) of the computational domain (see Figure 65).
- Inlet for the top of the domain (see Figure 66).
- Rough wall for building walls (see Figure 67) or for the ground (see Figure 68). The user has

to enter the roughness length. In case of variable roughness length, the user has to provide the land use data and the association between the roughness length values and land use categories.

Remark: If a meteorological file is given, it is used by default to initialize the variables. If a meteorological file is not given, the user can use the standard *Code_Saturne* initial and boundary conditions set up but has to be aware that even small inconsistencies can create very large buoyancy forces and spurious circulations.

7.13.5.1 Boundary conditions based on several meteorological vertical profiles

In some cases, especially when outputs of a mesoscale model are used, you need to build input boundary conditions from several meteorological vertical wind profiles. Cressman interpolation is then used to create the boundary conditions. The following files need to be put in the `DATA` directory:

- All `meteo` files giving the different vertical profiles of prognostic variables (wind, temperature, turbulent kinetic energy and dissipation).
- A file called `imbrication_files_list.txt` which is a list of the `meteo` files used.
- A separate `meteo` file which is used for the initial conditions and to impose inlet boundary conditions for the variables for which Cressman interpolation is not used (for example: temperature, turbulent kinetic energy). This file must follow the rules indicated previously.

The following files should be put in the SRC directory:

- The user source file `cs_user_parameters.f90`. In this file, set the `cressman_` flag of each variable, for which the Cressman interpolation should be enabled, to `.true.`.

7.13.6 User subroutines

The user subroutines are used when the graphical user interface is not sufficient to set up the calculation. We give some examples of user file for atmospheric application:

- `cs_user_source_terms.f90`: to add a source term in the prognostic equations for forest canopy modelling, wind turbine wake modelling... See the associated `doxygen` documentation for examples of use of `cs_user_source_terms.f90`.
- `cs_user_parameters.f90`: to activate the Cressman interpolation. For example, it is used to impose inhomogeneous boundary conditions. See the associated `doxygen` documentation for examples of use of `cs_user_parameters.f90`.
- `cs_user_extra_operations-extract.f90`: to generate vertical profiles for post processing. See the associated `doxygen` documentation for examples of use of `cs_user_extra_operations.f90`.
- `cs_user_boundary_conditions-atmospheric.f90`: show how to set up the boundary conditions and to put a heterogeneous roughness length... See the associated `doxygen` documentation for examples of use of `cs_user_boundary_conditions.f90`.

Remark: If the computation is set without the GUI, other user subroutines such as the following have to be used:

- `cs_user_initialization-atmospheric.f90`: allows to initialize or modify (in case of a restarted calculation) the calculation variables and the values of the time step. See the associated `doxygen` documentation for examples of use of `cs_user_initialization.f90`.

- `cs_user_boundary_conditions-atmospheric.f90`: allows to define all the boundary conditions. For each type of boundary condition, faces should be grouped as physical zones characterized by an arbitrary number `izone` chosen by the user. If a boundary condition is retrieved from a meteorological profile, the variable `iprofm(izone)` of the zone has to be set to 1. The vertical profiles of atmospheric variables can be described in this file.

Examples are available in the directory `SRC/EXAMPLE`.

7.13.7 Physical models

7.13.7.1 Atmospheric dispersion of pollutants

To simulate the atmospheric dispersion of pollutant, one first need to define the source(s) term(s). That is to say the location i.e. the list of cells or boundary faces, the total air flow, the emitted mass fraction of pollutant, the emission temperature and the speed with the associated turbulent parameters. The mass fraction of pollutant is simulated through a user added scalar that could be a 'scalar with drift' if wanted (aerosols for example).

The simulations can be done using 3 different methods:

1. Using a mass source term, that is added in the Navier-Stokes equations using the `cs_user_mass_source_terms.f90` user subroutine.
2. Prescribing a boundary condition code "total imposed mass flux" for some boundary faces using the `cs_user_boundary_conditions.f90` user subroutine.
3. Using a scalar source term. In this case, the air inflow is not taken into account. The user has to add an explicit part to the equations for the scalar through the `cs_user_source_terms.f90` file. This is done by selecting the cells and adding the source term `crvexp (cells)` which equals to the air flux multiplied by the mass fraction, while the implicit part `crvimp` is set to zero.

The first method is recommended, but one must take care that each source influences the dispersion of the others, which is physically realistic. So if the impact of several sources has to be analyzed independently it has first to be verified that these influences are negligible or as many simulations as there are sources have to be run.

With the second method, the same problem of sources interactions appears, and moreover standard Dirichlet conditions should not be used (use `itypfb=i_convective_inlet` and `icodcl=13` instead) as the exact emission rate cannot be prescribed because the diffusive part (usually negligible) cannot be quantified. Additionally, it requires that the boundary faces of the emission are explicitly represented in the mesh.

Finally the third method does not take into account the jet effect of the emission and so must be used only if it is sure that the emission does not modify the flow.

Whatever solution is chosen, the mass conservation should be verified by using for example the `cs_user_extra_operations-scalar_balance_by_zone.f90` file.

7.13.7.2 Soil/atmosphere interaction model

This model is based on the force restore model (Deardorff [17]). It takes into account heat and humidity exchanges between the ground and the atmosphere at daily scale and the time evolution of ground surface temperature and humidity. Surface temperature is calculated with a prognostic equation whereas a 2-layers model is used to compute surface humidity.

The parameter `iatsoil` in the file `atini0.f90` needs to be equal to one to activate the model. Then, the source file `solvar.f90` is used.

Three variables need to be initialized in the file `atini0.f90`: deep soil temperature, surface temperature and humidity.

The user needs to give the values of the model constants in the file `solcat.f90`: roughness length, albedo, emissivity...

In case of a 3D simulation domain, land use data has to be provided for the domain. Values of model constants for the land use categories have also to be provided.

7.13.7.3 Radiative model (1D)

The 1D-radiative model calculates the radiative exchange between different atmospheric layers and the surface radiative fluxes.

The radiative exchange is computed separately for two wave lengths intervals

- Calculation in the infrared spectral domain (file `rayir.f90`)
- Calculation in the spectral range of solar radiation (file `rayso.f90`)

This 1D-radiative model is needed if the soil/atmosphere interaction model is activated.

This model is activated if the parameter `iatra1` is equal to one in the file `cs_users_parameters.f90`.

7.13.8 Atmospheric main variables

For more details on the topic of atmospheric boundary layers, see Stull [16].

- Definition of the potential temperature:

$$\theta = T\left(\frac{P}{P_r}\right)^{-\frac{R_d}{C_p}}$$

- Definition of liquid potential temperature:

$$\theta_l = \theta\left(1 - \frac{L}{C_p T} q_l\right)$$

- Definition of virtual temperature:

$$T_v = (1 + 0.61q)\,T$$

- Gas law:

$$P = \rho \frac{R}{M_d}\,(1 + 0,61q)\,T$$

  with $R = R_d M_d$.

- Hydrostatic state:

$$\frac{\partial P}{\partial z} = -\rho g$$

7.13.9 Recommendations

This part is a list of recommendations for atmospheric numerical simulations.

- Enough probes at different vertical levels in the domain should be used to check the convergence of the calculation.

Constant name	Symbol	Values	Unit
Gravity acceleration at sea level	g	9.81	$m.s^{-2}$
Effective Molecular Mass for dry air	M_d	28.97	$kg.kmol^{-1}$
Standard reference pressure	P_r	10^5	Pa
Universal gas constant	R	8.3143	$J.K^{-1}.mol$
Gas constant for dry air	R_d	287	$J.kg^{-1}.K^{-1}$

Table 5: Constant name

Variable name	Symbol
Specific heat capacity of dry air	C_p
Atmospheric pressure	P
Specific humidity	q
Specific content for liquid water	q_l
Temperature	T
Virtual temperature	T_v
Potential temperature	θ
Liquid potential temperature	θ_l
Latent heat of vaporization	L
Density	ρ
Altitude	z

Table 6: Variable name

- An inflow boundary condition at the top level of the domain should be set (symmetry and automatic inlet/outlet are not appropriate).
- A Courant number too small or too big has to be avoided (see *Code_Saturne* Best Practice Guidelines). That is the reason why the option `variable time step in space and in time` is recommended for steady simulations when there are large differences of cell size inside the domain (which is generally the case for atmospheric simulations). With this option, it can be necessary to change the reference time step and the time step maximal increase (by default, the time step increase rate is 10%).

In some cases, results can be improved with the following modifications:

- In some case, the turbulent eddy viscosity can drop to unrealistically low values (especially with $k-\varepsilon$ model in stable atmospheric condition). In those cases, it is suggested to put an artificial molecular viscosity around $0.1m^2.s^{-1}$.
- If the main direction of wind is parallel to the boundary of your computing domain, try to set symmetry boundary conditions for the lateral boundaries to avoid inflow and outflow on the same boundary zone (side of your domain). Another possibility is to use a cylindrical mesh.
- To avoid inflow and outflow on the same boundary zone (side of your domain), avoid the case of vertical profile in the input data `meteo` file with changes of the sign of velocity of wind (V_x or/and V_y).

7.14 Cavitation module

The cavitation module is based on an homogeneous mixture model. The physical properties (density and dynamic viscosity) of the mixture depends on a resolved void fraction and constant reference properties of the liquid phase and the gas phase.

For a description of the user management of the cavitation module, please refer to the dedicated `doxygen` documentation.

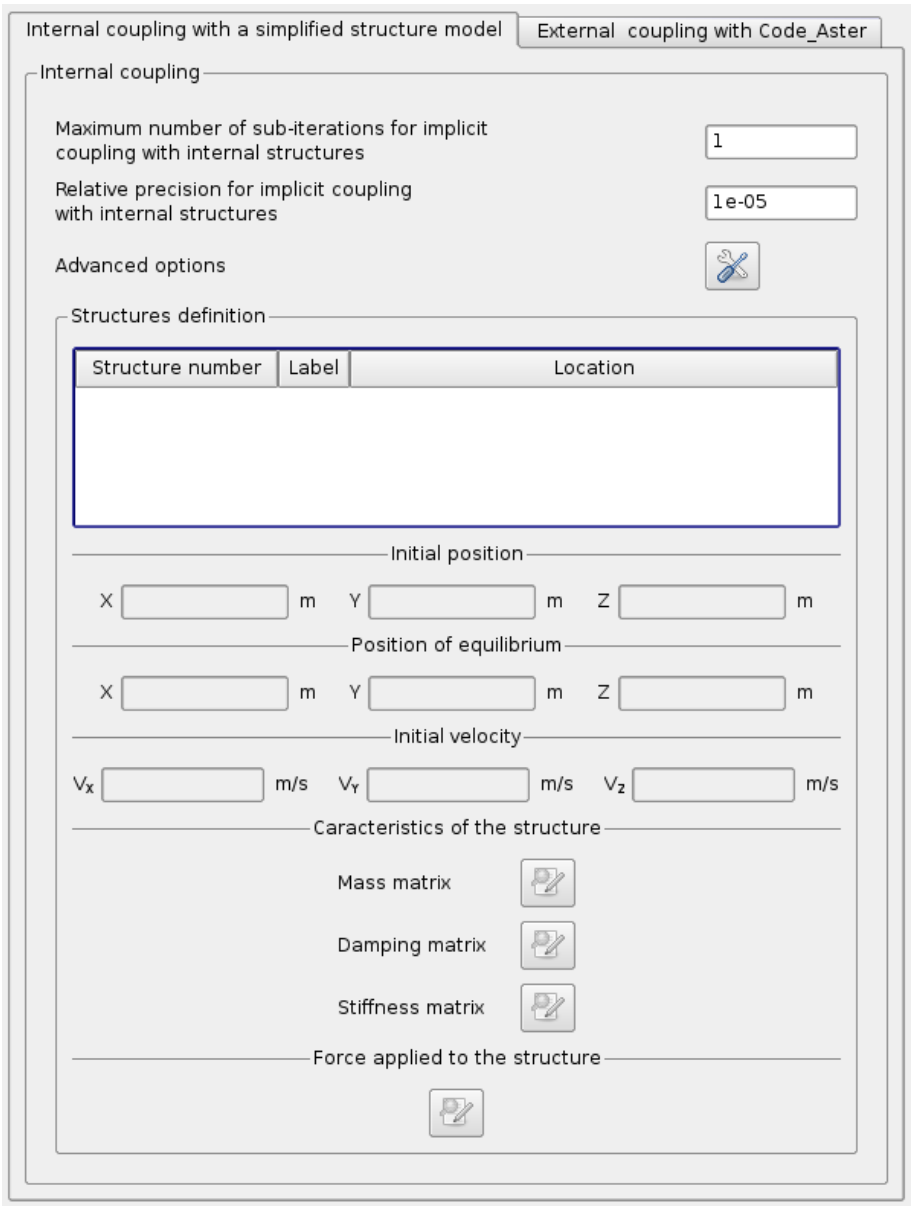

Figure 59: Boundary conditions - internal coupling

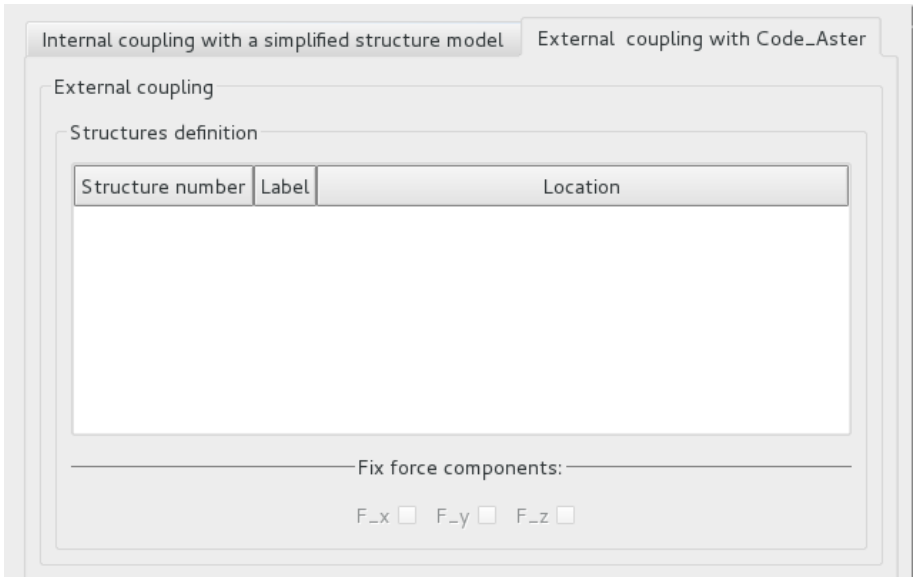

Figure 60: Boundary conditions - external coupling

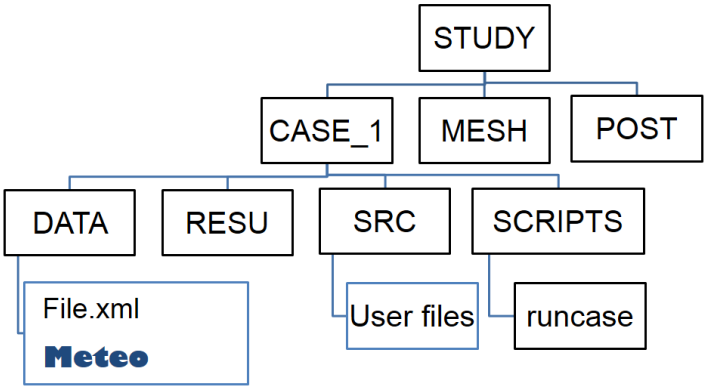

Figure 61: Organization of a study (specific files of atmospheric version in bold type)

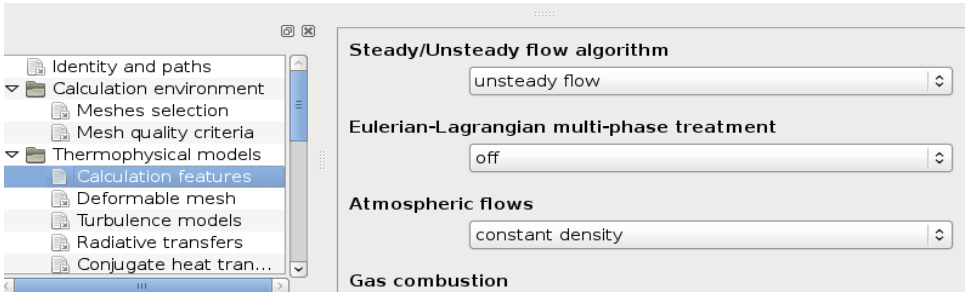

Figure 62: Selection of atmospheric model and steady/unsteady flow algorithm

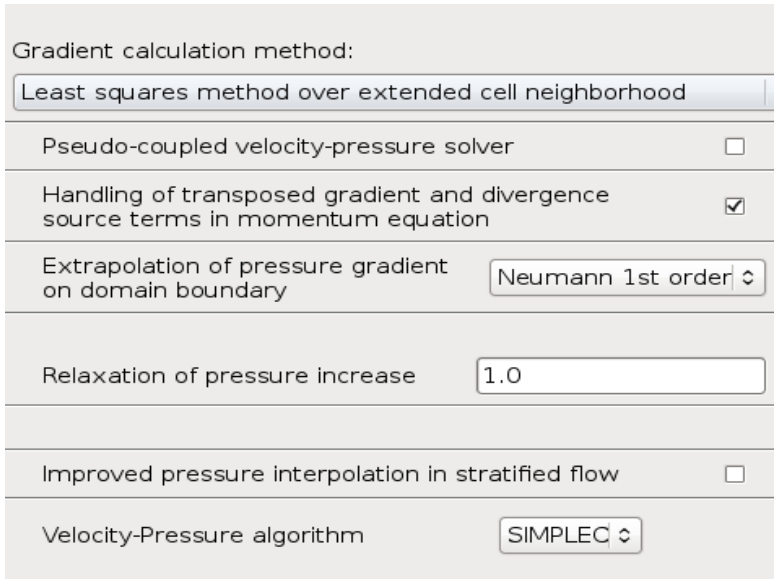

Figure 63: Example of global parameters

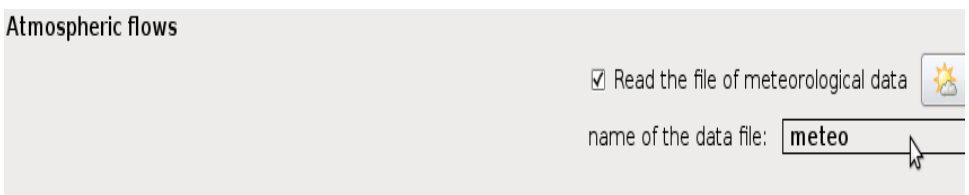

Figure 64: Selection of the `meteo` file

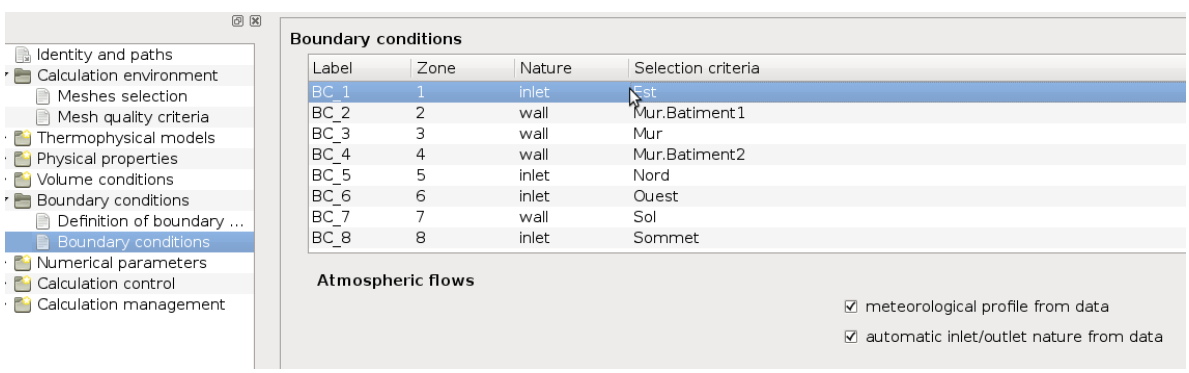

Figure 65: Selection of automatic inlet/ outlet for boundary conditions

Figure 66: Selection of the boundary condition for the top of the domain

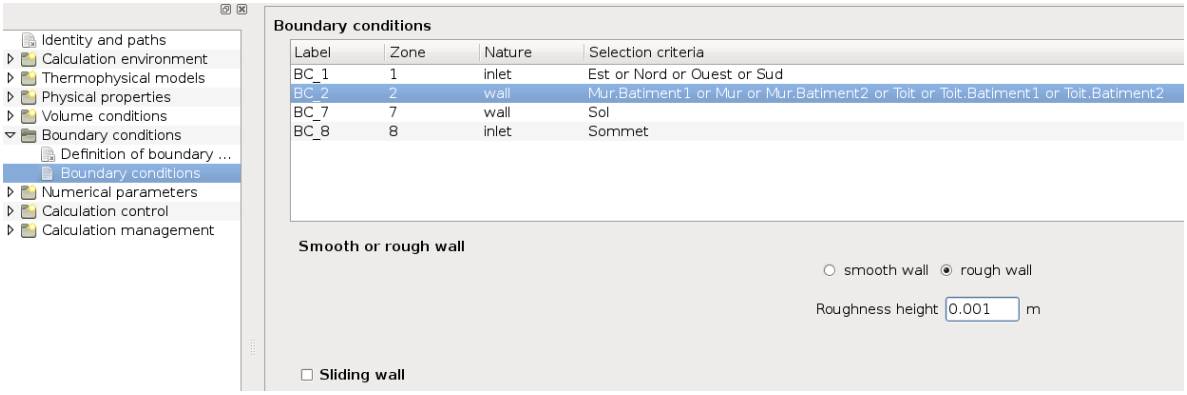

Figure 67: Selection of the boundary condition for building walls

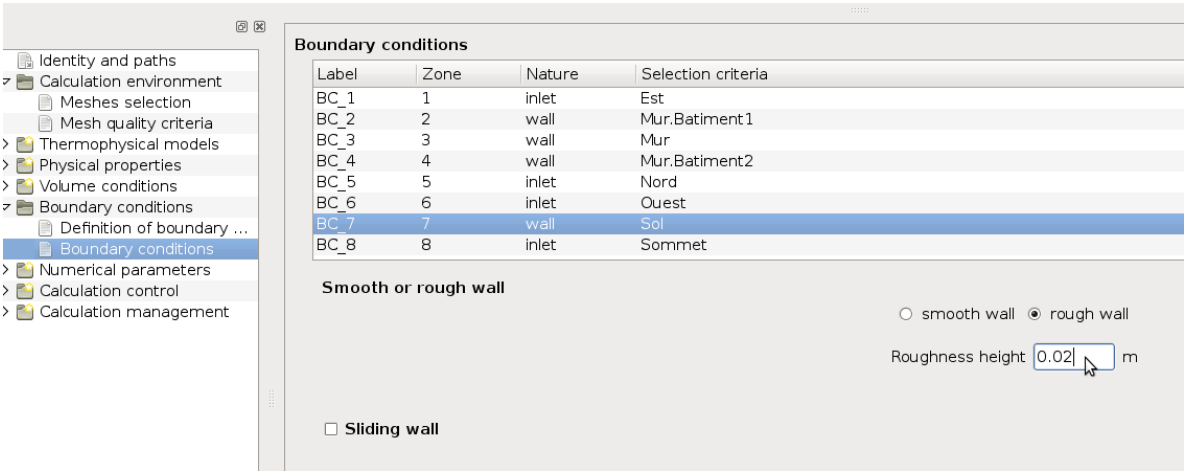

Figure 68: Selection of the boundary condition for the ground

8 Keyword list

The keywords are classified under relevant headings. For each keyword of *Code_Saturne* Kernel, the following informations are given:

Variable name	Type	Allowed values	[Default]	O/C	Level
	Description				
	Potential dependences				

- **Variable name**: Name of the variable containing the keyword.
- **Type**: a (Array), i (Integer), r (Real number), c (Character string).
- **Allowed values**: list or range of allowed values.
- **Default**: value defined by the code before any user modification (every keyword has one). In some cases, a non-allowed value is given (generally -999 or -10^{12}), forcing the user to specify a value. If he does not do it, the code may:
  - automatically use a recommended value (for example, automatic choice of the variables for which chronological records will be generated).
  - stop, if the keyword is essential.
- **O/C**: Optional/Compulsory
  - O: optional keyword, whose default value may be enough.
  - C: keyword which must imperatively be specified.
- **Level**: L1, L2 or L3
  - L1 (level 1): the users will have to modify it in the framework of standard applications. The L1 keywords are written in bold.
  - L2 (level 2): the users may have to modify it in the framework of advanced applications. The L2 keywords are all optional.
  - L3 (level 3): the developers may have to modify it; it keeps its default value in any other case. The L3 keywords are all optional.
- **Description**: keyword description, with its potential dependences.

The L1 keywords can be modified through the Graphical Use Interface or in the `cs_user_parameters.f90` file. L2 and L3 keywords can only be modified through the `cs_user_parameters.f90` file, even if they do not appear in the version proposed as example it the `SRC/REFERENCE/base` directory.
It is however recommended not to modify the keywords which do not belong to the L1 level.

The alphabetical keyword list is displayed in the index, in the end of this report.

NOTES
• The notation "d" refers to a double precision real. For instance, 1.8d-2 means 0.018.
• The notation "`grand`" (which can be used in the code) corresponds to 10^{12}.

8.1 Input-output

NOTES

- Two different files can have neither the same unit number nor the same name.

8.1.1 "Calculation" files

General

Vortex method for LES

For calculation files related to the vortex method for LES, please refer to the dedicated Doxygen documentation.

Thermochemistry

For the calculation file related to the thermochemistry, please refer to the dedicated Doxygen documentation.

8.1.2 Post-processing for *EnSight* or other tools

Notes

- The format depends on the user choices, and most options are defined using the GUI or `cs_user_postprocess.c`.
- The post-processing files can be of the following formats: *Ensight Gold*, *MED* or *CGNS*. The use of the two latter formats depends on the installation of the corresponding external libraries.
- For each quantity (problem unknown, preselected numerical variable or preselected physical parameter), the user specifies if a post-processing output is wanted. The output frequency can be set.

See the dedicated Doxygen documentation about `keyvis`.

8.1.3 Chronological records of the variables on specific points

Standard use through Interface or `cs_user_parameters.f90`

For each quantity (problem unknown, preselected numerical variable or preselected physical parameter), the user indicates whether chronological records should be generated, the output period and the position of the probes. The code generates chronological records at the cell centers located closest to the geometric points defined by the user by means of their coordinates. For each quantity, the number of probes and their index-numbers must be specified (it is not mandatory to generate all the variables at all the probes).

Please refer to the dedicated Doxygen documentation.

8.1.4 Time averages

See the dedicated Doxygen documentation.

8.1.5 Others

For user calculation file, see the following Doxygen documentation. For other printing options, please refer to the Doxygen documentation dealing with input/output options.

8.2 Numerical options

8.2.1 Calculation management

The following Doxygen documentation provides information about the various calculation management options available in *Code_Saturne* such as `ntmabs`, `ntcabs`, etc.

8.2.2 Scalar unknowns

Several keywords refering to the scalar unknowns are detailed in the following Doxygen documentation. The Doxygen page of the Stokes model structure also contains some keywords such as `icpsyr`, `iclvfl` or `itbrrb`. For other keywords, please refer to the following `Doxygen` pages refering to `nscaus` and `iscacp`.

8.2.3 Definition of the equations

For informations about `istat`, `iconv`, `idiff` or `idifft`, please refer to the following Doxygen documentation.

Moreover, one can find details about the `idircl` keyword here and about the `ivisse` keyword there.

8.2.4 Definition of the time advancement

idilat	i	1, 2, 3, 4	[1]	O	L1

Algorithm to take into account the density variation in time
= 1: steady dilatable flow algorithm (default)
= 2: unsteady dilatable flow algorithm
= 3: low-Mach number algorithm
= 4: non conservative algorithm for fire simulation
always useful

cdtvar	ra	strictly positive real number	[1]	O	L1

multiplicative factor applied to the time step for each scalar
Hence, the time step used when solving the evolution equation for the variable is the time step used for the dynamic equations (velocity/pressure) multiplied by `cdtvar`. The size of the array `cdtvar` is `nvar`. For instance, the multiplicative coefficient applied to the scalar 2 is `cdtvar(isca(2))`. Yet, the value of `cdtvar` for the velocity components and the pressure is not used. Also, although it is possible to change the value of `cdtvar` for the turbulent variables, it is highly not recommended
useful if and only if `nscal` $\geqslant 1$

`varrdt`	r	strictly positive real number	[0.1]	O	L3

maximum allowed relative increase in the calculated time step value between two successive time steps (to ensure stability, any decrease in the time step is immediate and without limit)
useful if `idtvar` $\neq 0$

For details about time stepping options, please refer to the dedicated Doxygen documentation.

Non-constant time step

The calculation of the time step uses a reference time step `dtref` (at the calculation beginning). Later, every time step, the time step value is calculated by taking into account the different existing limits,

in the following order:

- `coumax`, `foumax`: the more restrictive limit between both is used (in the compressible module, the acoustic limitation is added),
- `varrdt`: progressive increase and immediate decrease in the time step,
- `iptlro`: limitation by the thermal time step,
- `dtmax` and `dtmin`: clipping of the time step to the maximum, then to the minimum limit.

8.2.5 Turbulence

The $k-\varepsilon$ (standard and linearized production) and $R_{ij}-\varepsilon$ (LRR and SSG) turbulence models implemented in *Code_Saturne* are "High-Reynolds" models. It is therefore necessary to make sure that the thickness of the first cell neighboring the wall is larger than the thickness of the viscous sub-layer (at the wall, $y^+ > 2.5$ is required as a minimum, and preferably between 30 and 100)[27]. If the mesh does not respect this condition, the results may be biased (particularly if thermal processes are involved). Using scalable wall-functions (cf. keyword `iwallf`) may help avoiding this problem.
The v2-f model is a "Low-Reynolds" model, it is therefore necessary to make sure that the thickness of the first cell neighboring the wall is smaller than the thickness of the viscous sub-layer ($y^+ < 1$).
The $k-\omega$ SST model provides correct results whatever the thickness of the first cell. Yet, it requires the knowledge of the distance to the wall in every cell of the calculation domain. The user may refer to the keyword `icdpar` for more details about the potential limitations.
The $k-\varepsilon$ model with linear production allows to correct the known flaw of the standard $k-\varepsilon$ model which overestimates the turbulence level in case of strong velocity gradients (stopping point).
With LES, the wall functions are usually not greatly adapted. It is generally more advisable (if possible) to refine the mesh towards the wall so that the first cell is in the viscous sub-layer, where the boundary conditions are simple natural no-slip conditions.
Concerning the LES model, the user may refer to the subroutine `ussmag` for complements about the dynamic model. Its usage and the interpretation of its results require particular attention. In addition, the user must pay further attention when using the dynamic model with the least squares method based on a partial extended neighbourhood (`imrgra`=3). Indeed, the results may be degraded if the user does not implement his own way of averaging the dynamic constant in `ussmag` (*i.e.* if the user keeps the local average based on the extended neighbourhood).

For further details, please refer to the following `Doxygen` documentation dealing with turbulence options and turbulence constants.

8.2.6 Time scheme

By default, the standard time scheme is a first-order. A second-order scheme is activated automatically with LES modelling. On the other hand, when "specific physics" (gas combustion, pulverised coal, compressible module) are activated, the second-order scheme is not allowed.

In the current version, the second-order time scheme is not compatible with the estimators (`iescal`), the velocity-pressure coupling (`ipucou`), the modelling of hydrostatic pressure (`icalhy` and `iphydr`) and the time- or space-variable time step (`idtvar`).

Also, in the case of a rotation periodicity, a proper second-order is not ensured for the velocity, but calculations remain possible.

It is recommended to keep the default values of the variables listed below. Hence, in standard cases, the user does not need to specify these options.

Please refer to the dedicated `Doxygen` documentation for detailed informations about the time stepping

[27] While creating the mesh, $y^+ = \frac{yu*}{\nu}$ is generally unknown. It can be roughly estimated as $\frac{yU}{10\nu}$, where U is the characteristic velocity, ν is the kinematic viscosity of the fluid and y is the mid-height of the first cell near the wall.

parameters.

8.2.7 Gradient reconstruction

The gradient reconstruction keywords such as `imrgra`, `nswrgr`, `epsrgr`, `imligr`, `climgr` or `extrag` are members of the `cs_var_cal_opt_t` structure for which informations can be found in the following `Doxygen` documentation.

Details on the `anomax` keyword can be found here as well.

8.2.8 Solution of the linear systems

See the dedicated `Doxygen` documentation for most settings related to linear solver options.

More informations on these settings can also be found here.

8.2.9 Convective scheme

For informations on the keywords related to the convective scheme (i.e. `blencv`, `ischcv`, `isstpc`) please refer to the following `Doxygen` documentation.

8.2.10 Pressure-continuity step

Several options related to the pressure-continuity step are available and can be modified by the user. These options can be found in the following `Doxygen` documentation. For details about the porosity keyword `iporos`, please refer to the dedicated `Doxygen` documentation.

8.2.11 Error estimators for Navier-Stokes

There are currently `nestmx`=4 types of local estimators provided at every time step, with two possible definitions for each[28]. These scalars indicate the areas (cells) in which some error types may be important. They are stored using the `cs_field` API (see `field_get_val_s(iestim(iestim), c_estim)`). For each estimator, the code writes the minimum and maximum values in the listing and generates post-processing outputs along with the other variables.

The additional memory cost is about one real number per cell and per estimator. The additional calculation cost is variable. For instance, on a simple test case, the total estimator `iestot` generates an additional cost of 15 to 20 % on the CPU time[29]; the cost of the three others may be neglected. If the user wants to avoid the calculation of the estimators during the computation, it is possible to run a calculation without estimators first, and then activate them on a restart of one or two time steps.

It is recommended to use the estimators only for visual and qualitative analysis. Also, their use is compatible neither with a second-order time scheme nor with a calculation with a frozen velocity field.

iest = iespre: prediction (default name: EsPre). After the velocity prediction step (yielding $\widetilde{\underline{u}}$), the estimator $\eta_{i,k}^{pred}(\widetilde{\underline{u}})$, local variable calculated at every cell Ω_i, is created from $\underline{\mathcal{R}}^{pred}(\widetilde{\underline{u}})$, which represents the residual of the equation solved during this step:

$$\begin{aligned}\underline{\mathcal{R}}^{pred}(\widetilde{\underline{u}}) &= \rho^n \frac{\widetilde{\underline{u}} - \underline{u}^n}{\Delta t} + \underline{\underline{\nabla}}(\widetilde{\underline{u}}) \cdot (\rho \underline{u})^n - \underline{\mathrm{div}}\left((\mu+\mu_t)^n \underline{\underline{\nabla}}(\widetilde{\underline{u}})\right) + \underline{\nabla}(P^n) \\ &- \text{rest of the right-hand side}(\underline{u}^n, P^n, \text{other variables}^n)\end{aligned}$$

By definition:

$$\eta_{i,k}^{pred}(\widetilde{\underline{u}}) = |\Omega_i|^{(k-2)/2} \, ||\underline{\mathcal{R}}^{pred}(\widetilde{\underline{u}})||_{\mathbb{L}^2(\Omega_i)}$$

[28] Choice made by the user

[29] Indeed, all the first-order in space differential terms have to be recalculated at the time t^{n+1}

- The first family, $k = 1$, suppresses the volume $|\Omega_i|$ which intrinsically appears with the norm $\mathbb{L}^2(\Omega_i)$.
- The second family, $k = 2$, exactly represents the norm $\mathbb{L}^2(\Omega_i)$. The size of the cell therefore appears in its calculation and induces a weighting effect.

$\eta_{i,k}^{pred}(\widetilde{\underline{u}})$ is ideally equal to zero when the reconstruction methods are perfect and the associated system is solved exactly.

iest = iesder: drift (default name: EsDer). The estimator $\eta_{i,k}^{der}(\underline{u}^{n+1})$ is based on the following quantity (intrinsic to the code):

$$\begin{aligned} \eta_{i,k}^{der}(\underline{u}^{n+1}) &= |\Omega_i|^{(k-2)/2} ||\text{div (corrected mass flow after the pressure step)} - \Gamma||_{L^2(\Omega_i)} \\ &= |\Omega_i|^{(1-k)/2} |div(\text{corrected mass flow after the pressure step}) - \Gamma| \end{aligned} \tag{8}$$

Ideally, it is equal to zero when the Poisson equation related to the pressure is solved exactly.

iest = iescor: correction (default name: EsCor). The estimator $\eta_{i,k}^{corr}(\underline{u}^{n+1})$ comes directly from the mass flow calculated with the updated velocity field:

$$\eta_{i,k}^{corr}(\underline{u}^{n+1}) = |\Omega_i|^{\delta_{2,k}} \; |div(\rho^n \underline{u}^{n+1}) - \Gamma|$$

The velocities $\underline{u}^{n+1}$ are taken at the cell centers, the divergence is calculated after projection on the faces.
$\delta_{2,k}$ represents the Kronecker symbol.

- The first family, $k = 1$, is the absolute raw value of the divergence of the mass flow minus the mass source term.
- The second family, $k = 2$, represents a physical property and allows to evaluate the difference in $kg.s^{-1}$.

Ideally, it is equal to zero when the Poisson equation is solved exactly and the projection from the mass flux at the faces to the velocity at the cell centers is made in a set of functions with null divergence.

iest = iestot: total (default name: EsTot). The estimator $\eta_{i,k}^{tot}(\underline{u}^{n+1})$, local variable calculated at every cell Ω_i, is based on the quantity $\underline{\mathcal{R}}^{tot}(\underline{u}^{n+1})$, which represents the residual of the equation using the updated values of $\underline{u}$ and P:

$$\begin{aligned} \underline{\mathcal{R}}^{tot}(\underline{u}^{n+1}) = \rho^n \frac{\underline{u}^{n+1} - \underline{u}^n}{\Delta t} + \underline{\underline{\nabla}}\left(\underline{u}^{n+1}\right) \cdot (\rho \underline{u})^{n+1} - \underline{\text{div}}\left((\mu + \mu_t)^n \underline{\underline{\nabla}}(\underline{u}^{n+1})\right) + \underline{\nabla}(P^{n+1}) \\ - \text{rest of the right-hand side}(\underline{u}^{n+1}, P^{n+1}, \text{other variables}^n) \end{aligned}$$

By definition:

$$\eta_{i,k}^{tot}(\underline{u}^{n+1}) = |\Omega_i|^{(k-2)/2} \; ||\underline{\mathcal{R}}^{tot}(\underline{u}^{n+1})||_{\mathbb{L}^2(\Omega_i)}$$

The mass flux in the convective term is recalculated from $\underline{u}^{n+1}$ expressed at the cell centres (and not taken from the updated mass flow at the faces).

As for the prediction estimator:

- The first family, $k = 1$, suppresses the volume $|\Omega_i|$ which intrinsicly appears with the norm $\mathbb{L}^2(\Omega_i)$.
- The second family, $k = 2$, exactly represents the norm $\mathbb{L}^2(\Omega_i)$. The size of the cell therefore appears in its calculation and induces a weighting effect.

The estimators are evaluated depending on the values of `iescal`.

8.2.12 Calculation of the distance to the wall

The options related to the calculation of the distance to the wall are described in the following Doxygen documentation. Some options are used only in the case of the calculation of the non-dimensional distance to the wall y^+ (LES model with van Driest damping). Most of the keywords are simple copies of the keywords for the numerical options of the general equations, with a potentially specific value in the case of the calculation of the distance to the wall.

8.2.13 Others

Informations concerning the remaining keywords can be reached through the following Doxygen pages:

- iccvfg and ipucou
- nterup and epsup
- imvisf
- irclu, nswrsm and epsrsm
- isuit1

8.3 Numerical, physical and modelling parameters

8.3.1 Numeric parameters

These parameters correspond to numeric reference values in the code. They can be used but shall not be modified (they are defined as parameter).

For a list of these physical parameters, please refer to the following Doxygen documentation.

8.3.2 Physical parameters

These parameters correspond to physical reference values in the code. They can be used but shall not be modified (they are defined as parameter).

For a list of these physical parameters, please refer to the following Doxygen documentation.

8.3.3 Physical variables

Most physical variables are listed in the following Doxygen documentation.

Other physical variables such as diftl0, srrom, visls0, sigmas or rvarfl are described in the following Doxygen pages :

- diftl0,
- srrom,
- visls0, sigmas, rvarfl.

8.3.4 Modelling parameters

Please refer to the following Doxygen documentation for more informations about modelling parameters such as xlomlg, almax or uref.

8.4 ALE

For further details about the ALE calculation options, please refer to the dedicated `Doxygen` pages here and there. The following `Doxygen` documentation might be useful as well.

8.5 Thermal radiative transfers: global settings

Most of radiative module keywords may be modified in the user subroutines `cs_user_radiative_*` (or, for some of them, through the thermochemical data files).

For a detailed list of these keywords, please refer to the following `Doxygen` documentation.

8.6 Electric module (Joule effect and electric arcs): specificities

The electric module is composed of a Joule effect module (`ippmod(ieljou)`) and an electric arcs module (`ippmod(ielarc)`).

The Joule effect module is designed to take into account the Joule effect (for instance in glass furnaces) with real or complex potential in the enthalpy equation. The Laplace forces are not taken into account in the impulse momentum equation. Specific boundary conditions can be applied to account for the coupled effect of transformers (offset) in glass furnaces.

The electric arcs module is designed to take into account the Joule effect (only with real potential) in the enthalpy equation. The Laplace forces are taken into account in the impulse momentum equation.

The different keywords used in the electric module are detailed in the following `Doxygen` documentation.

8.7 Compressible module: specificities

The keywords used in the global settings are quite few. They are found in the subroutines `uscfx1` and `uscfx2`, in the `cs_user_parameters.f90` file (see the description of these user subroutines, §7.7.1).

Detailed informations can be found here for the keywords `igrdpp`, `viscv0` and `icfgrp`.

For `iviscv`, `ieos` and `xmasmr`, please refer to the dedicated `Doxygen` documentation.

9 Bibliography

[1] F. Archambeau, N. Méchitoua, M. Sakiz,
Code_Saturne: a Finite Volume Code for the Computation of Turbulent Incompressible Flows,
Industrial Applications, International Journal on Finite Volumes, Vol. 1, 2004.

[2] F. Archambeau, *et al.*,
Note de validation de Code_Saturne version 1.1.0,
EDF Report HI-83/04/003/A, 2004 (in French).

[3] S. Benhamadouche,
Modélisation de sous-maille pour la LES - Validation avec la Turbulence Homogène Isotrope (THI) dans une version de développement de Code_Saturne,
EDF Report HI-83/01/033/A, 2001 (in French).

[4] M. Boucker, F. Archambeau, N. Méchitoua,
Quelques éléments concernant la structure informatique du Solveur Commun - Version 1.0_init0,
Compte-rendu express EDF I81-00-8, 2000 (in French).

[5] M. Boucker, J.D. Mattéi,
Proposition de modification des conditions aux limites de paroi turbulente pour le Solveur Commun dans le cadre du modèle $k - \varepsilon$ standard,
EDF Report HI-81/00/019/A, 2000 (in French).

[6] A. Douce, N. Méchitoua,
Mise en œuvre dans Code_Saturne des physiques particulières. Tome3 : Transfert thermique radiatif en milieu gris semi-transparent,
EDF Report HI-81/02/019/A, 2002 (in French).

[7] A. Douce,
Physiques particulières dans Code_Saturne 1.1, Tome 5 : modélisation stochastique lagrangienne d'écoulements turbulents diphasiques polydispersés,
EDF Report, HI-81/04/03/A, 2005 (in French).

[8] A. Escaich, P. Plion, *Mise en œuvre dans Code_Saturne des modélisations physiques particulières. Tome 1 : Combustion en phase gaz*,
EDF Report, HI-81/02/03/A, 2002 (in French).

[9] A. Escaich, *Mise en œuvre dans Code_Saturne des modélisations physiques particulières. Tome 2 : Combustion du charbon pulvérisé*,
EDF Report, HI-81/02/09/A, 2002 (in French).

[10] N. Méchitoua, F. Archambeau,
Prototype de solveur volumes finis co-localisé sur maillage non-structuré pour les équations de Navier-Stokes 3D incompressibles et dilatables avec turbulence et scalaire passif,
EDF Report HE-41/98/010/B, 1998 (in French).

[11] *Code_Saturne* documentation,
Code_Saturne 5.3.3 Theory and Programmer's guide,
on line with the release of *Code_Saturne* 5.3.3 (`code_saturne info --guide theory`).

[12] M. Sakiz, Validation team,
Validation de Code_Saturne version 1.2 : note de synthèse,
EDF Report H-I83-2006-00818-FR, 2006 (in French).

[13] M. Tagorti., S. Dal-Secco, A. Douce, N. Méchitoua,
Physiques particulières dans Code_Saturne, tome 4 : le modèle P-1 pour la modélisation des transferts thermiques radiatifs en milieu gris semi-transparent,
EDF Report HI-81/03/017/A, 2003 (in French).

[14] *Code_Saturne* DOCUMENTATION,
Code_Saturne tutorial, in line on main *Code_Saturne* website (http://www.paraview.org).

[15] E. BOUZEREAU,
Représentation des nuages chauds dans le modèle météorologique "Mercure": Application aux panaches d'aéroréfrigérants et aux précipitations orographiques,
EDF,Universite Pierre et Marie Curie-Paris VI, PHD, 2004 (in French).

[16] R. STULL,
An introduction to boundary layer meteorology ,
Springer, 1988.

[17] J.W. DEARDORFF,
Efficient prediction of ground surface temperature and moisture with inclusion of a layer of vegetation,
Journal of Geophysical Research,83:1889-1903 , 1978.

Index of the main variables and keywords

– Symbols –

isvhb 34
isvtb 34
coejou 40
dpot 40
icdpar 39
icodcl 64
iscapp 33
itypfb 64
rcodcl 64

– A –

ales 77
atgaze 88

– B –

bles 77

– C –

cdtvar 132
cebu 98
ckabsg 89
ckupdc 81
compog 88
couimp 112
csmago 77

– D –

diftl0 98
distch 95
divukw 35
dt 35

– E –

ehgazg 89
eppt1d 35

– F –

fment 94
fs(1) 89

– H –

hbord 34

– I –

i_convective_inlet 65
ialtyb 116
ibfixe 116
iccoal 86
icdpar 133
icepdp 81
icetsm 82
icfuel 86
ickabs 97
iclvor 70
icod3p 85
icoebu 86
icolwc 86
icompf 87
icpl3c 86
idebty 69
idiam2 97
idilat 132
iecaux 39
ielarc 86, 137
ieljou 86, 137
ientat 94
ientcp 94
ientfu 94
ientgb 93
ientgf 93
ientox 94
ientre 65, 93
iescor 135
iesder 135
iespre 134
iestot 135
if1m 91, 97
if2m 91, 97
if3m 91, 97
if3p2m 98
if4p2m 91
if4pm 98
ifinty 69
ifm 91, 97
ifmcel 35
ifp2m 91, 97
ifp3m 92
ifpt1d 34
ifrent 65
ifresf 65, 116
igfuel 88
igliss 116
igmdch 97
igmdv1 97
igmdv2 97
igmhet 97
igoxy 88
ih2 ..... 91, 97
ihm 97
iindef 65
ileaux 39
immel 97
indjon 87
inp 91, 97
iparoi 65

iparug ... 65
ippmod ... 85
iqimp ... 93
irepvo ... 70
irom2 ... 97
iscalt ... 33
isolib ... 65
isymet ... 65
it3m ... 97
it4m ... 97
itemp ... 97
itemp1 ... 98
itemp2 ... 97
itrifb ... 69
itypsm ... 82
iu ... 93
iv ... 93
ivimpo ... 116
ivrtex ... 39
iw ... 93
ix2 ... 97
ixch ... 91, 97
ixck ... 91, 97
ixkabe ... 89
iygfm ... 91, 97
iym(1) ... 97
iym(2) ... 97
iym(3) ... 97
iym1(1) ... 98
iym1(2) ... 98
iym1(3) ... 98
iym1(4) ... 98
iym1(5) ... 98
iym1(6) ... 98
iym1(7) ... 98
izone ... 93

– K –

kabse ... 88

– N –

nato ... 88
ncepdc ... 80
ncepdp ... 80
ncesmp ... 82
ncetsm ... 82
ncharm ... 86
nclpch ... 86
ncpcmx ... 86
nestmx ... 134
nfpt1d ... 34
ngaze ... 88
ngazg ... 88, 89
nnent ... 70
nomcoe ... 88
npo ... 88, 89
nppt1d ... 35
nrgaz ... 88
ntypmx ... 67
nvort ... 70

– P –

puismp ... 112

– Q –

qimp ... 93
qimpat ... 94
qimpcp ... 95

– R –

rcodcl ... 93

– S –

s2kw ... 35
smacel ... 82
srrom ... 98
stoeg ... 88

– T –

tbord ... 34
th ... 89
timpat ... 94
timpcp ... 95
tinfue ... 94
tinoxy ... 94
tkent ... 94
tmax ... 88
tmin ... 88

– V –

varrdt ... 132

– W –

wmolat ... 88
wmolg ... 89

– X –

xco2 ... 89
xh2o ... 89
xkabe ... 89
xkabel ... 89
xlesfl ... 77