EDF R&D

Fluid Dynamics, Power Generation and Environment Department
Single Phase Thermal-Hydraulics Group

6, quai Watier
F-78401 Chatou Cedex

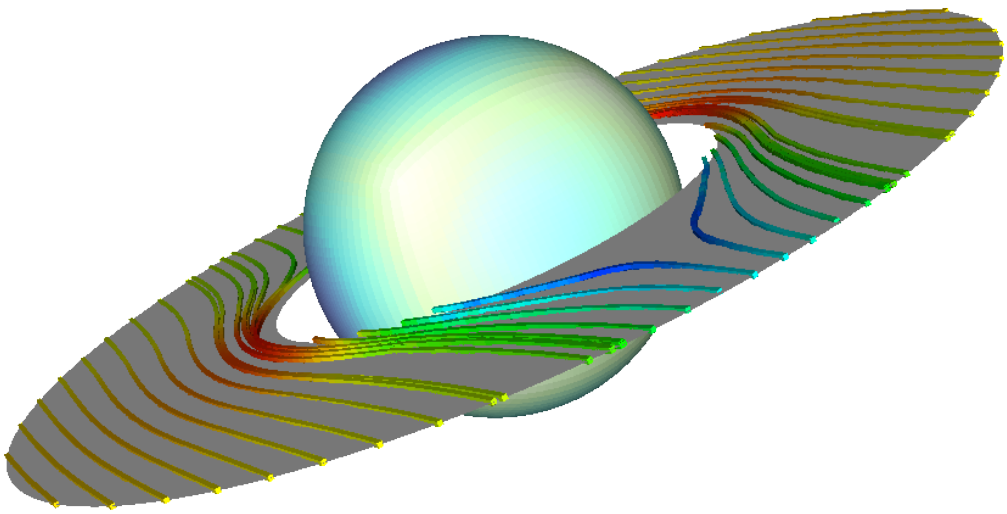
Tel: 33 1 30 87 75 40
Fax: 33 1 30 87 79 16

OCTOBER 2010

Code_Saturne documentation

Code_Saturne version 2.0.0-rc1 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. The code also includes an engineering module, Matisse, for the simulation of nuclear waste surface storage.
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 2.0.0-rc1. 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 2.0.0-rc1. 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.

TABLE OF CONTENTS

1 Introduction . . . 9

2 Practical information about *Code_Saturne* . . . 10

2.1 System Environment for *Code_Saturne* . . . 10

2.1.1 Preliminary settings . . . 10

2.1.2 Standard directory hierarchy . . . 10

2.1.3 Code_Saturne Kernel library files . . . 13

2.2 Setting up and running of a calculation . . . 13

2.2.1 Step by step calculation . . . 13

2.2.2 Temporary execution directory . . . 15

2.2.3 Execution modes . . . 16

2.2.4 Interactive modification of the target time step . . . 16

2.3 Case preparer . . . 16

2.4 Supported mesh and post-processing output formats . . . 17

2.4.1 Formats supported for input . . . 18

2.4.2 Formats supported for input or output . . . 21

2.4.3 Mesh meta-files . . . 24

2.4.4 Meshing tools and associated formats . . . 24

2.4.5 Meshing remarks . . . 25

2.5 Preprocessor command line options . . . 25

2.6 Kernel command line options . . . 26

2.7 Parameters in the launch script . . . 27

2.8 Graphical User Interface . . . 30

2.9 Face and cell mesh-defined properties and selection . . . 31

3 Preprocessing . . . 33

3.1 Preprocessor options and sub-options . . . 34

3.1.1 Option files . . . 34

3.1.2 Mesh selection . . . 34

3.1.3 Post-processing output . . . 35

3.1.4 Faces selection . . . 35

3.1.5 Joining of non-conforming meshes . . . 36

3.1.6 Periodicity . . . 37

3.1.7 Element orientation correction . . . 38

3.2 Environment variables . . . 38

3.2.1 System environment variables . . . 39

3.3 Optional functionality . . . 39

3.4 General remarks . . . 39
3.5 Files passed to the Kernel . . . 39
4 Partitioning for parallel runs . . . 40
4.1 Options . . . 40
4.1.1 Ignore periodicity . . . 40
4.1.2 Partitioner choice . . . 40
4.1.3 Simulation mode . . . 40
4.1.4 Environment variables . . . 40
5 Main variables . . . 41
5.1 Array sizes . . . 41
5.2 Geometric variables . . . 43
5.3 Physical variables . . . 44
5.4 Variables related to the numerical methods . . . 49
5.5 User arrays . . . 52
5.6 Developer arrays . . . 52
5.7 Parallelism and periodicity . . . 53
5.8 Geometry and particule arrays related to Lagrangian modeling . . . 56
5.9 Variables saved to allow calculation restarts . . . 59
6 User subroutines . . . 61
6.1 Preliminary comments . . . 61
6.2 Using selection criteria in user subroutines . . . 61
6.3 Initialisation of the main key words: `usini1` . . . 63
6.4 Management of boundary conditions: `usclim` . . . 63
6.4.1 Coding of standard boundary conditions . . . 64
6.4.2 Coding of non-standard boundary conditions . . . 66
6.4.3 Checking of the boundary conditions . . . 68
6.4.4 Sorting of the boundary faces . . . 68
6.5 Management of the boundary conditions with LES: `usvort` . . . 68
6.6 Management of the variable physical properties: `usphyv` . . . 70
6.7 Non-default variables initialisation: `usiniv` . . . 71
6.8 Non-standard management of the chronological record files: `ushist` . . . 72
6.9 User source terms in Navier-Stokes: `ustsns` . . . 73
6.10 User source terms for k and ε: `ustske` . . . 74
6.11 User source terms for R_{ij} and ε: `ustsri` . . . 74
6.12 User source terms for φ and $\overline{f}$: `ustsv2` . . . 75
6.13 User source terms for k and ω: `ustskw` . . . 75
6.14 User source terms for the user scalars: `ustssc` . . . 75

6.15 Management of the pressure drops: uskpdc . . . 75
6.16 Management of the mass sources: ustsma . . . 76
6.17 Thermal module in a 1D wall . . . 77
6.18 Initialization of the options of the variables related to the ALE module: usalin and usstr1 . . . 78
6.19 Management of the boundary conditions of velocity mesh related to the ALE module: usalcl . . . 79
6.20 Management of the structure property: usstr2 . . . 80
6.21 Modification of the turbulent viscosity: usvist . . . 80
6.22 Modification of the variable C of the dynamic LES model: ussmag . . . 81
6.23 Temperature-enthalpy and enthalpy-temperature conversions: usthht . . . 81
6.24 Modification of the mesh geometry: usmodg . . . 82
6.25 Management of the post-processing intermediate outputs: usnpst . . . 82
6.26 Definition of post-processing and mesh zones: usdpst . . . 83
6.27 Modification of the mesh zones to post-process: usmpst . . . 85
6.28 Definition of the variables to post-process: usvpst . . . 85
6.29 Modification of the variables at the end of a time step: usproj . . . 86
6.30 Radiative thermal transfers in semi-transparent gray media . . . 87
6.30.1 Initialisation of the radiation main key words: usray1 . . . 87
6.30.2 Management of the radiation boundary conditions: usray2 . . . 87
6.30.3 Absorption coefficient of the medium, boundary conditions for the luminance and cualtion of the net radiative flux: usray3 . . . 88
6.30.4 Encapsulation of the temperature-enthalpy conversion: usray4 . . . 89
6.31 Utilisation of a specific physics: usppmo . . . 89
6.32 Management of the boundary conditions related to pulverised coal and gas combustion: usebuc, usd3pc, uslwcc, uscpcl and uscplc . . . 95
6.33 Initialisation of the variables related to pulverised coal and gas combustion: usebui, usd3pi, uslwci and uscpiv . . . 96
6.34 Initialisation of the options of the variables related to pulverised coal and gas combustion: usebu1, usd3p1, uslwc1, uscpi1 and uscpl1 . . . 97
6.35 Management of Boundary Conditions of the electric arc: uselcl . . . 99
6.36 Initialisation of the variables in the electric module . . . 100
6.37 Initialisation of the variable options in the electric module . . . 100
6.38 Management of variable physical properties in the electric module . . . 101
6.39 Management of the *EnSight* output in the electric module: uselen . . . 101
6.40 Compressible module . . . 102
6.40.1 Initialisation of the options of the variables related to the compressible module: uscfx1 and uscfx2 . . . 102
6.40.2 Management of the boundary conditions related to the compressible module: uscfcl . 103

6.40.3 Ininitialisation of the variables related to the compressible module: `uscfxi` . . . 103
6.40.4 Compressible module thermodynamics: `uscfth` . . . 103
6.40.5 Management of the variable physical properties in the compressible module: `uscfpv` . . . 103
6.41 Lagrangian modeling of multiphasic flows with dipersed inclusions . . . 104
6.41.1 Initialisation of the main key words in the Lagrangian modeling: `uslag1` . . . 104
6.41.2 Management of the boundary conditions related to the particles: `uslag2` *and* `uslain` . . . 105
6.41.3 Treatment of the particle/boundary interaction: `uslabo` . . . 108
6.41.4 Option for particle cloning/merging: `uslaru` . . . 109
6.41.5 Manipulation of particulate variables at the end of an iteration and user volumetric statistics: `uslast` *and* `uslaen` . . . 110
6.41.6 User stochastic differential equations: `uslaed` . . . 110
6.41.7 Particle relaxation time: `uslatp` . . . 111
6.41.8 Particle thermal characteristic time: `uslatc` . . . 111
7 Key word list . . . 112
7.1 Input-output . . . 112
7.1.1 "Calculation" files . . . 113
7.1.2 Post-processing for EnSight *or other tools* . . . 116
7.1.3 Chronological records of the variables on specific points . . . 117
7.1.4 Time averages . . . 119
7.1.5 Others . . . 121
7.2 Numerical options . . . 122
7.2.1 Calculation management . . . 122
7.2.2 Scalar unknowns . . . 123
7.2.3 Definition of the equations . . . 125
7.2.4 Definition of the time advancement . . . 126
7.2.5 Turbulence . . . 128
7.2.6 Time scheme . . . 132
7.2.7 Gradient reconstruction . . . 137
7.2.8 Solution of the linear systems . . . 139
7.2.9 Convective scheme . . . 140
7.2.10 Pressure-continuity step . . . 141
7.2.11 Error estimators for Navier-Stokes . . . 142
7.2.12 Calculation of the distance to the wall . . . 143
7.2.13 Others . . . 146
7.3 Numerical, physical and modeling parameters . . . 147
7.3.1 Numeric Parameters . . . 147
7.3.2 Physical parameters . . . 147

7.3.3 Physical variables . . . 148
7.3.4 Modeling parameters . . . 152
7.4 ALE . . . 156
7.5 Thermal radiative transfers: global settings . . . 157
7.6 Electric module (Joule effect and electric arc): specificities . . . 160
7.7 Compressible module: specificities . . . 161
7.8 Lagrangian multiphase flows . . . 162
7.8.1 Global settings . . . 162
7.8.2 Specific physics models associated with the particles . . . 164
7.8.3 Options for two-way coupling . . . 165
7.8.4 Numerical modeling . . . 165
7.8.5 Volume statistics . . . 166
7.8.6 Display of trajectories and particle movements . . . 168
7.8.7 Display of the particle/boundary interactions and the statistics at the boundaries . . . 169
8 Bibliography . . . 173
9 Appendix 1 : automatic validation procedure . . . 175
9.1 Introduction . . . 175
9.2 Practical informations on the procedure . . . 175
9.3 Directories architecture . . . 175
9.4 Validation base . . . 175
9.4.1 Elementary tests : gradient calculations . . . 176
9.4.2 Laplacien calculation . . . 176
9.5 Architecture description . . . 176
9.5.1 Python files in the modules directory . . . 176
9.5.2 XML file description . . . 177
9.5.3 To add a new study . . . 178
9.5.4 Report files . . . 178
Index of the main variables and keywords . . . 179

1 Introduction

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. The code also includes an engineering module, Matisse, for the simulation of nuclear waste surface storage.

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 three main elements and an optional GUI, as shown on figure 1:

- the Kernel module is the numerical solver
- the Preprocessor module is in charge of mesh import, mesh joining (arbitrary interfaces), and definition of periodicity boundary conditions (translation and/or rotation)
- the Partitioner is in charge of optimizing domain decomposition for parallel computing (optional, but highly recommended for parallel performance)

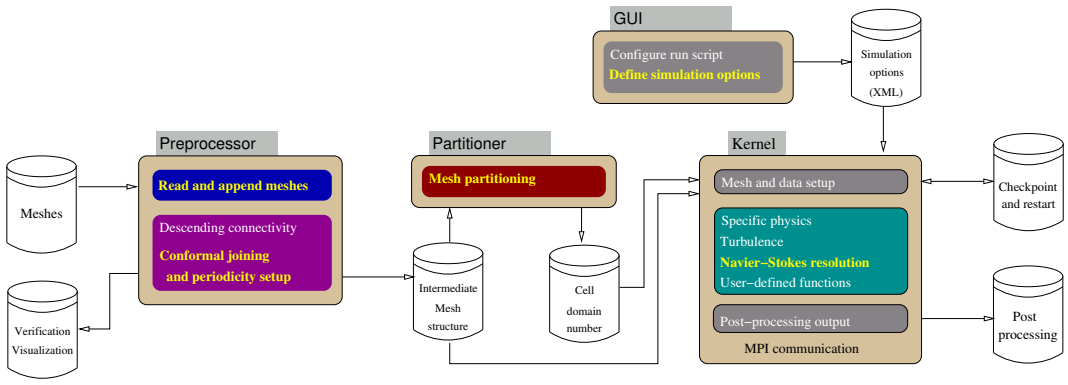


Figure 1: *Code_Saturne* elements

Code_Saturne also relies on two libraries (by the same team, under LGPL licence), which can also be used independently:

- BFT (Base Functions and Types) for the management of memory and input/output as well as specific utilities (estimation of time and memory usage for instance)
- FVM (Finite Volume Mesh) for the post-processing output and the management of code coupling

[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 present document is a practical user's guide for *Code_Saturne* version 2.0.0-rc1. It is the result of the joint effort of all the members in the development team.

The aim of this document is to give practical information to the users of *Code_Saturne*. It is therefore strictly oriented towards the usage of the code. For more details about the algorithms and their numerical implementation, please refer to the reports [10] and [3], and to the theoretical documentation [11], which is newer and more detailed (the latest updated version of this document is available on-line with the version of *Code_Saturne* and accessible through the command `cs_info --guide theory`).

The present document first presents all the necessary elements to run a calculation with *Code_Saturne* version 2.0.0-rc1. 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.

2 Practical information about *Code_Saturne*

2.1 System Environment for *Code_Saturne*

2.1.1 Preliminary settings

In order to use *Code_Saturne*, every user must add the following line (in their `.profile`, or equivalent, depending on the environment):

```
export PATH={prefix}/bin:$PATH
```

or define the following alias (in their `.bashrc`, or equivalent, or `.alias` file, depending on the environment):

```
alias cs='{prefix}/bin/cs'
```

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

2.1.2 Standard directory hierarchy

The standard architecture for the simulation studies is:

A study directory containing:

- A directory `MESH` containing the mesh(es) necessary for the study
- A directory `POST` for the potential post-processing routines (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, thermochemical data, ...)
- A directory `SCRIPTS` for the launch script

[2] At EDF R&D, `/home/saturne/Code_Saturne/2.0.0-rc1` is used

- A directory `RESU` for the results
  To improve the calculation traceability, the files and directories sent to `RESU` after a calculation are given a suffix identifying the calculation start date and time by an eight-digit number (two digits for each month, day, hour and minute; the result of a calculation started at 14h03 on December 31st will therefore be indexed 12311403)

In the standard cases, `RESU` contains a directory `CHR.ENSIGHT.mmddhhmm` with the post-processing files in *EnSight* format, a directory `RESTART.mmddhhmm` for the calculation restart files, a directory `HIST.mmddhhmm` for the files of chronological record of the results at specific locations (probes), `listpre.mmddhhmm` and `listing.mmddhhmm` files reporting the Preprocessor and the Kernel execution. For an easier follow-up of the modifications in former calculations, the user-subroutines used in a calculation are stored in a directory `SRC.mmddhhmm` in the directory `RESU`. The *Xml* Interface data file, thermo-chemical data files and launch script are also copied into the directory `RESU` with the appropriate suffix (whatever its name, the launch script appears in the directory `RESU` as `runcase.mmddhhmm`). `compil.log.mmddhhmm` and `summary.mmddhhmm` are respectively reports of the compilation phase and general information on the calculation (which kind of machine, which user, which version of the code, ...). Eventually, when the user subroutines produce specific result files (extraction of 1D profiles for instance), a directory `RES_USERS.mmddhhmm` is created in the directory `RESU` for these files[3].

During calculations coupled with SYRTHES (option specified in the launch script of *Code_Saturne* or *via* the Interface) the same organisation is used for the files related to *Code_Saturne*. For the files related to SYRTHES, the location of the upstream files is specified in the `syrthes.env` file. Yet, the launch script is built presuming that the following organisation is applied:

- a directory `SRC_SYR` for the potential SYRTHES user subroutines
- a directory `DATA_SYR` containing the configuration file `syrthes.env` (location of files specific to SYRTHES). The file defining the SYRTHES calculation options (`syrthes.data`) and the potential restart files can also be placed in this directory.

The SYRTHES result files (geometry file, chronological result files, calculation restart files and the historic file) are placed in a sub-directory `RESU_SYR.mmddhhmm` of the `RESU` directory, where `mmddhhmm` corresponds to the calculation identification suffix.

The SYRTHES execution report file is placed in the `RESU` directory (same as for the *Code_Saturne* review) under the name `listsyr.mmddhhmm` and the compilation report file is under the name `compil_syrthes.log.mmddhhmm`. For an easier follow-up of the modifications in former calculations, the potential SYRTHES user-subroutines used in a calculation are stored in a directory `SRC_SYR.mmddhhmm` in the directory `RESU`.

[3] in order for the script to copy them properly, their names have to be given in the variable `USER_OUTPUT_FILES` of the launch script, see §2.7

Below are typical contents of a case directory CASE1 in a study STUDY (*Code_Saturne* calculation coupled with SYRTHES):

`STUDY/CASE1/DATA:`	***Code_Saturne* data**
`SaturneGUI`	Graphical User Interface launch script
`study.xml`	Graphical User Interface parameter file
`THCH`	example of thermochemical files (used with the specific physics modules for gas combustion, pulverised coal or electric arcs)
`STUDY/CASE1/DATA_SYR:`	**SYRTHES data**
`syrthes.data`	SYRTHES data file
`syrthes.env`	SYRTHES configuration file
`STUDY/CASE1/SRC:`	***Code_Saturne* user subroutines**
`REFERENCE`	examples of a user subroutines
`usclim.f90`	user subroutines used for the present the calculation
`usini1.f90`	
`STUDY/CASE1/RESU:`	**results**
`CHR.ENSIGHT.08211921`	directory containing the *Code_Saturne* post-processing results in the *EnSight* format for the calculation 08211921 (contains both volume and boundary results; the contents of the directory are user modifiable)
`SRC.08211921`	*Code_Saturne* user subroutines used for the calculation 08211921
`SRC_SYR.08211921`	SYRTHES user subroutines used in the calculation 08211921
`HIST.08211921`	directory containing the chronological records for *Code_Saturne*
`RES_USERS.08211921`	optional directory containing the user results files
`RESTART.08211921`	directory containing the *Code_Saturne* restart files
`compile.log.08211921`	compilation report
`study.xml.08211921`	Graphical User Interface parameter file used for the calculation 08211921
`runcase.08211921`	launch script used for the calculation 08211921 (whatever the name given to the file in the `SCRIPT` directory, the file will be referred as "`runcase.*`" in the `RESU` directory)
`listpre.08211921`	execution report for the Preprocessor module of *Code_Saturne*
`listing.08211921`	execution report for the Kernel module of *Code_Saturne*
`listsyr.08211921`	execution report for SYRTHES
`summary.08211921`	general information (machine, user, version, ...)
`RESU_SYR.08211921:`	**SYRTHES results (file names given in the `syrthes.env` file)**
`geoms`	SYRTHES solid geometry file
`histos1`	SYRTHES chronological records at specified probes
`resus1`	SYRTHES calculation restart file (1 time step)
`resusc1`	SYRTHES chronological solid post-processing file (may be transformed into the *EnSight* format with the *syrthes2ensight* utility)
`STUDY/CASE1/SCRIPTS:`	**launch script**
`runcase`	launch script (compliant with all architectures on which *Code_Saturne* has been ported)

2.1.3 *Code_Saturne* Kernel 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 `cs 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 §2.3). The `cs info` command gives direct access to the most needed information (especially the user and programmer's guides and the tutorial) without the user having to look for them in the *Code_Saturne* directories.

The subdirectories `{prefix}/lib` and `{prefix}/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 `users`, under subdirectories corresponding to each module: `base` (general routines), `cfbl` (compressible flows), `cogz` (gas combustion), `cplv` (pulverised coal combustion), `ctwr` (cooling towers modelling), `elec` (electric module), `fuel` (heavy fuel oil combustion module), `lagr` (Lagrangian module, `mati` (Matisse module), `pprt` (general specific physics routines) and `rayt` (radiative heat transfer). The case preparer command `cs create` copies all these files in the user directory `SRC/REFERENCE` during the case preparation.

The "include" files are available in the directory `include`, under similar subdirectories corresponding to each module: `base`, `cfbl`, `cogz`, `cplv`, `elec`, `fuel`, `lagr`, `mati`, `pprt` and `rayt`.

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

2.2 Setting up and running of a calculation

2.2.1 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 (`cs info --version`). If it does not correspond to the desired version, update the `.profile` file to set the environment variables correctly. Log out of the session and log in again to take the modifications into account properly (cf. §2.1.1).
- Prepare the different directories using the `cs create` command (see §2.3) and, when needed, add the directories `DATA_SYR` and `SRC_SYR` which are required to accomodate the SYRTHES files.
- Place the mesh(es) in the directory `MESH`. Make sure they are in a format compliant with *Code_Saturne* (see §2.4.5). There can be several meshes in case of mesh joining or coupling with SYRTHES[4].
- Go to the directory `DATA` and launch the Graphical User Interface using the command `./SaturneGUI` (see §**??**).
- Place the necessary user subroutines in the directory `SRC` (see §6). When not using the Interface, some subroutines are compulsory.

  For the standard physics:

  compulsory without Graphical User Interface:
  - `usini1` to specify the calculation parameters
  - `usclim` to manage the boundary conditions

  very useful:

[4] SYRTHES uses meshes composed of 10-node tetrahedra (vertices and centers of edges)

- `usphyv` to manage the variable physical properties (fluid density, viscosity ...)
- `usiniv` to manage the non-standard initialisations

For the specific physics "gas combustion":

(not accessible through the Graphical User Interface in version 2.0.0-rc1)

compulsory:

- `usini1` to specify the calculation parameters
- `usppmo` to select a specific physics module and combustion model
- `usebuc`, `usd3pc` or `uslwcc` (depending on the selected combustion model) to manage the boundary conditions of *all variables* (*i.e.* not only the ones related to the combustion model)

very useful:

- `usebu1`, `usd3p1` or `uslwc1` (depending on the selected combustion model) to specify the calculation options for the variables corresponding to combustion model
- `usebui`, `usd3pi` or `uslwci` (depending on the selected combustion model) to manage the initialisation of the variables corresponding to the combustion model

For the specific physics "coal combustion":

compulsory without Graphical User Interface:

- `usini1` to specify the calculation parameters
- `usppmo` to select the specific physics module
- `uscpcl` or `uscplc` (depending on the specific physics module) to manage the boundary conditions of *all variables* (*i.e.* not only the ones related to the specific physics module)

very useful:

- `uscpi1` to specify the calculation options for the variables corresponding to the specific physics module
- `uscpiv` to manage the initialisation of the variables corresponding to the specific physics module

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

(not accessible through the Graphical User Interface in version 2.0.0-rc1)

compulsory:

- `usini1` to specify the calculation parameters
- `usppmo` to select the specific physics module
- `uselcl` to manage the boundary conditions of *all variables* (*i.e.* not only the ones related to the electric module)
- `useliv` to initialise the enthalpy in case of Joule effect
- `uselph` to define the physical properties in case of Joule effect

very useful:

- `useli1` to manage the options related to the variables corresponding to the electric module
- `useliv` to manage the initialisation of the variables corresponding to the electric module

For the specific physics "heavy fuel oil combustion module":

(not accessible through the Graphical User Interface in version 2.0.0-rc1)

compulsory:

- `usini1` to specify the calculation parameters
- `usppmo` to select the specific physics module
- `usfucl` to manage the boundary conditions of *all variables* (*i.e.* not only the ones related to the specific physics module)

very useful:

- `usfui1` to specify the calculation options for the variables corresponding to the specific physics module
- `usfuiv` to manage the initialisation of the variables corresponding to the specific physics module

For the Lagrangian module (dispersed phase):
(the continuous phase is managed in the same way as for a case of standard physics)
(the Lagrangian module is not accessible through the Graphical User Interface in version 2.0.0-rc1)

compulsory:

- `uslag1` to manage the calculation conditions
- `uslag2` to manage the boundary conditions for the dispersed phase

very useful:

- `uslabo` to manage potential specific treatments at the boundaries (rebound conditions, specific statistics, ...)

For the compressible module:
(not accessible through the Graphical User Interface in version 2.0.0-rc1)

compulsory:

- `uscfx1` and `uscfx2` to manage the calculation parameters
- `uscfcl` to manage the boundary conditions
- `uscfth` to define the thermodynamics.

very useful:

- `uscfxi` to manage non-standard initialisations of the variables

The comprehensive list of the user subroutines and their instructions for use are given in §6.

- 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 §2.7)
- Run the calculation and analyse the results
- Purge the temporary files (in the directory `RUN` defined in the launch script, see §2.7)

2.2.2 Temporary execution directory

During a calculation, *Code_Saturne* uses a temporary directory for the compilation and the execution, the result files being only copied at the end in the directory `RESU`. This temporary directory is defined in the variable `RUN` of the launch script. This variable is set top a default value in the non-user section of the launch script, depending on the architecture:
`RUN=$HOME/tmp_Saturne/$STUDY/$CASE.mmddhhmm` for stand-alone workstations or for the Chatou cluster
`RUN=$SCRATCHDIR/tmp_Saturne/$STUDY/$CASE.mmddhhmm` for Platine at the CCRT
where `$STUDY` and `$CASE` are the names of the study and the case. The usual suffix with the date and time is added so that successive calculations will not get mixed-up.

This default value might not always be the optimal choice. Indeed, on a stand-alone machine, it might be useful to take advantage of large sized local disks on a machine when the `$HOME` account is on an NFS disk.

For this matter, the variable `CS_TMP_PREFIX` of the launch script (see §2.7) allows the user to change this directory. If the variable is empty, the default `RUN` directory will be used. If it is not empty, the launch script will set the `RUN` directory to `$CS_TMP_PREFIX/tmp_Saturne/$STUDY/$CASE.mmddhhmm`.

WARNING: in most cases, the temporary directories are not deleted after a calculation. They will accumulate and may hinder the correct running of the machine.
It is therefore essential to remove them regularly.

2.2.3 Execution modes

As explained before, *Code_Saturne* is composed of two main modules, the Preprocessor and the Kernel, and an optional Partitioner. The Preprocessor reads the meshes and performs the necessary joinings. The Partitioner optimizes domain decomposition for parallel runs. The resulting data is transfered to the Kernel through specific files, named `preprocessor_output` and `domain_number_*`, where `*` is the number of processors used. 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 and Partitioner do not work in parallel and may require a large amount of memory. Hence it is sometimes useful to run them separately, on a machine or in batch queues with extended memory, and to run the proper parallel calculation on another machine or in another batch queue. The launch scripts therefore allows specifically choosing which modules to run, using the variables `EXEC_PREPROCESS`, `EXEC_PARTITION`, and `EXEC_KERNEL` (see §2.7).

If `EXEC_PREPROCESS=no`, the Partitioner and Kernel will copy or link the file defined by the `PREPROCESSOR_OUTPUT_IN` variable to `preprocessor_output`.

If `EXEC_PARTITION=no`, the Kernel will search for a corresponding `domain_number_*` file in the directory defined by `PARTITION_OUTPUT_IN`.

If `EXEC_KERNEL=no`, the Preprocessor and Partitioner output will be saved respectively to a `$RESU/preprocessor_output.$SUFFIX` file and to a `$RESU/PARTITION_OUTPUT.$SUFFIX` directory. In this case, the number of processors for which partitioning is required is given by the `PARTITION_LIST` variable of the launch script (multiple partitionings may be done in one run).

2.2.4 Interactive modification of the target time step

During a calculation, it is possible to change the limit time step number (`ntmabs`) specified through the Interface or in `usini1`. To do so, a file named `ficstp` must be placed in the temporary execution directory (see §2.2.2). This file must contain a blank first line and the second line indicating the value of the new limit number of time steps.
If this new limit has already been passed in the calculation, *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.

2.3 Case preparer

The case preparer command `cs 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 `cs create` is as follows:

`cs 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.

`cs create --case DEBIT3 --case DEBIT4`
executed in the directory `STUDY` adds the case directories `DEBIT3` and `DEBIT4`.

An option `--nogui` is available for the use of *Code_Saturne* without Graphic Interface (see §**??**). This option must be either the first or the last argument and appear only once.

In the directory `DATA`, the `cs create` command places a subdirectory `THCH` containing examples of thermochemical data files used for pulverised coal combustion, gas combustion or electric arc. The file to be used for the calculation must be copied directly in the `DATA` directory and its name must be referenced in the launch script in the variable THERMOCHEMISTRY_DATA. All other files in the `DATA` or in the `THCH` will be ignored.
The `cs create` command also places in the directory `DATA` the launch script for the Graphical User Interface: `SaturneGUI`.

In the directory `SRC`, the `cs create` command creates a subdirectory `REFERENCE` containing all the user subroutines, classified by module type: `base`, `cfbl`, `cogz`, `cplv`, `elec`, `fuel`, `lagr`, `pprt` and `rayt`. Only the user subroutines placed directly under the directory `SRC` will be considered. The others will be ignored.

In the directory `SCRIPTS`, the `cs create` command copies and pre-fills an example of the launch script: `runcase`. The study, case and user name are filled automatically in the launch script, as are the paths leading to the different directories. Other parameters must be specified in the script (see §2.7), especially the mesh file(s) to use, but everything can be specified through the Graphical Interface.

2.4 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
- NUMECA IGG/Hexa
- MED
- CGNS
- EnSight 6
- EnSight Gold
- GAMBIT neutral
- Gmsh
- pro-STAR/STAR4
- STAR-CCM+

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, *".hex"* for IGG/Hexa, *".med"* for MED, *".msh"* for Gmsh, *".neu"* for GAMBIT neutral, *".ngeom"* for pro-STAR/STAR4, *".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.

2.4.1 Formats supported for input

2.4.1.1 NOPO/SIMAIL (INRIA/SIMULOG)

This format output by SIMAIL is still heavily used at EDF. We do 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,[5], 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 colors)
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

2.4.1.2 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 CS, 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.

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

[5] "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

Default extension:	`.unv`
File type:	text
Surface elements:	triangles, quadrangles
Volume elements:	tetrahedra, prisms, hexahedra
Zone selection:	colors (systematic) and named groups
Compatibility:	I-deas (*Master Series* 5 to 9, *NX Series* 10 to 12) at least
Documentation:	Online I-deas NX Series documentation

2.4.1.3 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

2.4.1.4 pro-STAR

This polyhedral format from CD-Adapco seems to be usable both with the **STAR-CD** and **STAR-CCM+** tools, and the **pro-STAR** tool should be able to generate it. The test meshes we have were generated by the **Comet-Design** tool, which has since been replaced by other CD-Adapco tools, especially **STAR-CD** V4 and **STAR-CCM+**. The available test cases are thus not extensive in terms of functionality (especially when considering definition of descriptions), so support for this format is lightly tested.

Currently, geometric entity numbers are converted to "color" numbers. This tends to lead to a large number of colors.

Default extension:	`.ngeom`
File type:	binary file using portable XDR encoding.
Surface elements:	polygons
Volume elements:	polyhedra
Zone selection:	face and cell sets (interpreted as numbered colors)
Compatibility:	all files of this type ? (tested on purely polyhedral meshes)
Documentation:	documentation accompanying and source code provided by CD-adapco in the context of a collaboration with UMIST (now University of Manchester) and EDF R&D/MFEE

2.4.1.5 STAR-CCM+

This polyhedral format is the current CD-Adapco 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 HDF-5). 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 "color" numbers, 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 interpret such sets as named "groups" rather than numbered "colors".

The CCMIO library is distributed freely by CD-Adapco upon demand.

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 colors, with names appearing in log)
Compatibility:	all files of this type ? (tested on purely polyhedral meshes)
Documentation:	documentation and source code provided by CD-adapco

2.4.1.6 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 explicitely 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 interpret their number as a color.

Also note that files produced by Harpoon may contain badly oriented prisms, so the Preprocessor orientation correction option (`--reorient`)may need to be used. Meshes built by this tool also contain hanging nodes, with non-conforming elements sharing some vertices. The `--join --semi-conf` preprocessor option must thus be used. This option is not set 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 the integers 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 color numbers
Compatibility:	All files of this type
Documentation:	online documentation, also available at: http://www.ensight.com/downloads/cat_view-5.html

2.4.1.7 Gmsh

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

Note that some meshes produced byGmsh 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 it' 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.

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

So as to obtain distinct colors 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 colors associated with the mesh, while using only elementary entities could lead to a high number of colors.

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 color numbers
Compatibility:	all files of this type
Documentation:	included documentation, also available at: http://www.geuz.org/gmsh

2.4.1.8 IGG/Hexa (NUMECA)

This format is quite peculiar in the sense that it defines a hierarchical mesh built exclusively of hexahedra, of quadrangular faces, and of edges, in which edges may be split in 2, faces in 2 or 4, and cells in 2, 4, or 8. Two neighboring elements are thus not always conforming. The Preprocessor only keeps the finest mesh level, and uses the hierarchical information so as to automatically build the appropriate conformal joining. CAD face information is transformed into color numbers.

Remark: The reader is based on the IGG/Hexa format as it was defined at the end of 2001. Since that time, the mesher seems to have been renamed $HEXPRESS^{TM}$, and we do not have any recent documentation or test file. For this reason, support for this format is not directly included in the main Preprocessor executable, but requires conversion using the `igghexa_to_med` tool.

Default extension:	`.hex`
File type:	portable binary by default (4-byte integers, 8-byte IEEE double-precision reals, big-endian), or text
Surface elements:	quadrangles
Volume elements:	hexahedra
Zone selection:	no volume selection, CAD surface numbers interpreted as colors
Compatibility:	unknown (at least files from late 2001 are readable)
Documentation:	elements provided by NUMECA (http://www.numeca.be) in 2001.

2.4.2 Formats supported for input or output

2.4.2.1 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, and has only recently added support for polyhedral elements in EnSight Gold files (interpreted as convex point sets, and not true polyhedra, but at least usable). 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.

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, so 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 colors. Without vertex IDs, only one part is read, and no colors 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 color number if vertex IDs are given
Compatibility:	files readable by EnSight 7.4 to 9.0, as well as tools based on the VTK library, especially ParaView (`http://www.paraview.org`)
Documentation:	online documentation, also available at: `http://www.ensight.com/downloads/cat_view-5.html`

2.4.2.2 MED 2.3

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 (the *Code_Saturne* team being represented). 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. Since MED 2.2 in 2003, this format 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 color/attribute and 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 licence, and is even packaged in some Linux distributions (at least Debian and Ubuntu). Versions 2.3.5 and older require HDF5 1.6, version 2.3.6 may compile with either HDF5 1.6 or HDF5 1.8 (if the latter has HDF5 1.6 compatibility enabled).

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)
Compatibility:	all versions of MED 2.2 or 2.3 (only unstructured nodal connectivity is supported)
Documentation:	online documentation. Download link at `http://files.opencascade.com/Salome/Salome5.1.2/med-fichier_2.3.5.tar.gz`

2.4.2.3 CGNS 2.5

Promoted especially by NASA, Boeing, and ICEM CFD (as well as ONERA in France), 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. The opposite of MED, the first version of this format was limited to multibloc 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 polygonale faces although the format defines them), and an interpolation tool.

We should be able to read almost any mesh written in this format, though meshes with overset interfaces may not be usable for a calculation. 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[6]), 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 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.[7]

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.[8]

Though many tools support CGNS, that support is often quite dissapointing, at least for unstructured meshes. Thus, some editors seem to use different means to mark zones to associate with boundary conditions than the ones recommended in the CGNS documentation, and some behaviors are worse: for example, under EnSight Gold 8, whenever a mesh contains multiple element types, variables are assigned to the wrong cells. Regarding support of polygons (*ngons* in the CGNS standard), even the verification tools published alongside the CGNS library are unable to handle them, and report errors in valid files containing such elements. VisIt 1.11.1 reports an error when a mesh contains such faces, while EnSight Gold 8 ignores them. CGNS 3 should improve this situation, allowing for polyhedra, but it is still in beta stage as of August 2009.

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

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

[8] 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.

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
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
Compatibility:	CGNS 2.0 to 2.5 on input, CGNS 2.5 on output
Documentation:	See CGNS site: `http://www.cgns.org`

2.4.3 Mesh meta-files

The Preprocessor allows use of text files (preferably with a `.mesh` extension) describing a set of meshes and their transformations, in place of (or combined with) "true" mesh files. These meta-files are described here:

Empty lines are ignored, and the # character may be used to define comments (the part of a line following this character is ignored).

One may request the reading of as many meshes as one needs, using for each mesh a section such as:

`read_mesh:` *filename*

or:

`read_mesh:` *filename* `<sub-options>`

If this section type appears more than once, the corresponding meshes are automatically appended. If needed, a mesh meta-file may itself declare another meta-file as a mesh file. Possible sub-options associated with a file may be separated by commas, semicolumns, or spaces, and are of the following form:

`format=`	format name (identical to command-line options)
`num=`	mesh number (useful when a file contains multiple meshes)
`grp_cel=`	`<section` or `zone>`, useful only for files in CGNS format
`grp_fac=`	`<section` or `zone>`, useful only for files in CGNS format

It is also possible to define a geometric transformation to apply to a mesh, using a homogeneous coordinates transformation matrix (3 lines, 4 columns, with the 3 first columns describing a rotation/scaling factor, and the last column describing a translation. The corresponding section is as follows (values may be spread over several lines):

`transformation_matrix:` t_{11} t_{12} t_{13} t_{14} t_{21} t_{22} t_{23} t_{24} t_{31} t_{32} t_{33} t_{34}

Note that the order in which multiple meshes are declared defines the order in which they are read and appended, but the geometric transformation is only applied at the end (this is a description file, not a command file). If multiple transformations are needed, a hierarchy of mesh meta-files may be used.

2.4.4 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 (export to I-deas universal is also possible, but has some limitations).

The use of files of the "Common Solver" type[9] is still possible but is deprecated. Such files are read directly from the Kernel, without the Preprocessor. The variable SOLCOM must be set to 1 in the launch scripts. Many potentialities of *Code_Saturne* are not usable with this file format (mesh joining with hanging nodes, periodicity, parallel computing, ...). For all the other formats, the Preprocessor must be used (SOLCOM=0).

2.4.5 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 neighboring the wall needs to be greater than the thickness of the viscous sublayer (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`.

2.5 Preprocessor command line options

The main options are:

- `--help`: gives a summary of the different command line options
- `-m mesh1 mesh2`: used to specify the names of the different meshes used. The launch script automatically calls the Preprocessor with the option `-m $MESH`, where `MESH` is the variable where the user has specified the different meshes to be used.
- `--join`: triggers the mesh joining functions. If nothing more is specified, every area of contact between two meshes will be joined together. The joining can be limited to certain selected faces. For instance, to join only the faces of colors 6 and 7, the full option will be `--join --color 6 7`. These options are to be specified in the `COMMAND_JOIN` variable in the launch script, to be automatically passed to the command line.
- `--perio`: triggers periodic boundary conditions. Two types of periodic boundaries are possible: translation or rotation (see §5.7 for additional details). For the translation, the basic option line is `--perio --trans tx ty tz`
  where `tx`, `ty` and `tz` are the coordinates of the translation vector. For the rotation, there are two possibilities. The transformation can be defined with a rotation angle (in degrees, between -180 and 180), a direction and an invariant point
  `--perio --rota --angle a --dir dx dy dz --invpt px py pz`
  (with obvious notations), or by giving the rotation matrix and an invariant point
  `--perio --rota --matrix m11 m12 m13 m21 m22 m23 m31 m32 m33 --invpt px py pz`
  A rotation and a translation can be combined, by giving both `--rota` and `--trans` specifications. The translation will always be applied first, whatever the order in which the rotation and the translation have been given.
  The orientation of the transformations is not important since both the transformation and its inverse will be used to connect faces. Yet, when combining a translation and a rotation, the orientations given for both have to be consistent.
  It is possible (and usually recommended) to restrict the search for periodic connections between faces to a certain group of faces, by adding selection arguments like `--color`. It is also possible to specify up to 3 independent periodicities, simply by repeating the `--perio` option. Below is given a example of the option line for a triple periodicity (the \ indicates the continuation of the

[9] File type specifically developed for the early prototype versions of *Code_Saturne* (`tlc`) extension

command line):
```
--perio --trans -10.2 0 0 --color 2 \
--perio --rota --angle 90 --dir 0 0 1 --invpt 0 0 0 --color 3 4 \
--perio --trans 0 1 0 --rota --matrix 1 0 0 0 0 -1 0 1 0 --invpt 0 0 -0.2
```
This option is to be specified in the `COMMAND_PERIO` variable in the launch script, to be automatically passed to the command line.

- `--reorient`: try to re-orient badly-oriented cells if it is necessary to compensate for mesh-generation software whose output does not conform to the format specifications.

2.6 Kernel 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 script. This way, the user only has to fill these variables and doesn't need to search deep in the script for the Kernel command line. Yet, below is given the complete list of options, with the variables in which they should be specified in the script.

- `--solcom`: this option indicates that the Kernel will read the mesh directly, not using the Preprocessor output files. This is only possible with "Common Solver" type of mesh files (see §2.4.5 for restrictions).
  This option is triggered by the `SOLCOM` variable in the launch script. If `SOLCOM` is set to 1, the `-solcom` option is automatically added to the Kernel command line. The variable `ifoenv` in the Fortran code will be set to 0 if the `--solcom` option has been used, otherwise it will be set to 1.

- `--mpi`: specifies that the calculation is running with MPI communications. The number of processors used will be determined automatically by the Kernel. If necessary, the launch script automatically passes the `--mpi` option to the Kernel command line (see 2.7).

- `-q` or `--quality`: triggers the verification mode. The code runs without any Interface parameter file nor any user subroutine. The mesh is read and elementary tests are performed:

  - the quality criteria of the mesh are calculated (non-orthogonality angles, internal faces offset, ...) and corresponding EnSight post-processing parts are created.

  - test calculation of the gradient of $sin(x + 2y + 3z)$. The calculated value is compared to the exact value, and an EnSight part for the corresponding error is created. The gradient is calculated with each option IMRGRA from 0 to 4.

  The command `-q` is to be placed in the `ARG_CS_VERIF` variable in the launch script to be added automatically to the Kernel command line.

- `--cwf`: triggers the cutting of boundary and internal faces which have a warp angle larger than a certain limit[10]. The concerned faces are divided into triangles. This option is to handle with care, since the division of the faces increases the non-orthogonalities of the mesh, but it is sometimes required (for the Lagrangian modeling, for instance, where non-plane faces lead to noticeable particle loss). By default, the faces are divided if their warp angle is larger than 0.01 degrees. This default value can be changed by adding an optional angle value to the command. For instance, to devide faces with a warp angle larger than 0.02 degrees, the full option will be `-cwf 0.02`.
  This option is to be specified in the `COMMAND_CWF` variable in the launch script, to be automatically passed to the command line.

[10] the warp angle is an indicator of the non-coplanarity of the different vertices of the face

- `--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 a 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 `ARG_CS_VERIF` variable in the launch script to be added automatically to the Kernel 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 `ARG_CS_OUTPUT` variable of the launch 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_n0002` to `listing_n`N. This option can be specified in the `ARG_CS_OUTPUT` variable of the launch script.

- `-p xxx` or `--param xxx`: specifies the name of the GUI parameter file to use for the calculation. The value of `xxx` is to be placed in the `PARAM` variable in the launch script (the file will be looked for in the directory `DATA`). The option `-param $PARAM` is automatically added to the Kernel command line.

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

2.7 Parameters in the launch script

The case preparer command `cs create` places an example of launch script, `runcase`, in the `SCRIPTS` directory. This script is quite general and known to work on every architecture *Code_Saturne* has been tested on. The beginning if the script contains the definition of certain parameters (environment variables) necessary to set the calculation. The second part of the script contains the commands for the preparation and execution of the calculation. No user intervention should be necessary in this second part.
The Graphical User Interface allows to fill in the major parameters of the script without having to edit the file.

Below is a list of the different variables and parameters that might be modified for a calculation, in their order of apparition in the script:

- LSF headers: definition of the headers for an LSF batch system, as can be found on the machines of the CCRT (Platine). The data expected are the number of processors reserved (`#BSUB -n`), the CPU time limit (`#BSUB -W`), the name of the standard output file (`#BSUB -o`), the name of the standard error file (`#BSUB -e`) and the name of the job (`#BSUB -J`).

- PBS headers: definition of the headers for a PBS batch system, as can be found on the machines of the Chatou cluster. The data expected are the number of nodes reserved (`nodes`), the number of processors per node (`ppn`), the total CPU time (`walltime`), the memory reserved (`mem`), and the name of the job (`#PBS -N`).

- Manchester headers: definition of the headers for the batch system specific to the cluster of the University of Manchester

- `SOLCOM`: a value of 1 will pass the `-solcom` option to the Kernel (see 2.6)

- `STUDY`: name of the study directory (automatically set by `cs create`, see §2.1.2)
- `CASE`: name of the case directory (automatically set by `cs create`, see §2.1.2)
- `PARAM`: name of the Interface parameter file, if necessary (see 2.6)
- `MESH`: name(s) of the mesh(es) used for the calculation (see 2.5 and 2.4.5). The files will be looked for in the directory `MESHDIR` (see below).
- `COMMAND_JOIN`: Preprocessor command line option for mesh joining (see 2.5)
- `COMMAND_CWF`: Kernel command line option for the division of faces with too large a warp angle (see 2.6)
- `COMMAND_PERIO`: Preprocessor command line option for the definition of periodic boundary conditions (see 2.5)
- `THERMOCHEMISTRY_DATA`: name of the thermochemical data file, if necessary (the file is looked for in the directory `DATA`, see §6.31)
- `NUMBER_OF_PROCESSORS`: number of processors (potentially virtual) to be used for the calculation.
  If the variable is left empty, the launch script will fill it automatically: on a batch system, `NUMBER_OF_PROCESSORS` will be equal to the number of processors reserved; in case of an interactive calculation, it will be set to 1.
  When using a batch system, `NUMBER_OF_PROCESSORS` should ideally be equal to the number of processors reserved, and can never be larger (one executable per processor). With an interactive calculation (like a Linux PC), `NUMBER_OF_PROCESSORS` can be larger than the total number of processors available, although it is not recommended (loss of efficiency because several executables share the same processor).
  In case of coupling with SYRTHES, one processor is reserved for SYRTHES, and the Kernel of *Code_Saturne* will therefore automatically be set to run on `NUMBER_OF_PROCESSORS-1` processors.
- `PROCESSOR_LIST`: list of nodes on which the calculation is to be run. On batch systems, this list is set automatically by the batch manager. For calculations on a stand-alone machine, the list is not used. Hence, except for very specific test (mainly for developing purposes), it is recommended to leave this variable empty.
- `USER_INPUT_FILES`: list of the user data files to be copied in the temporary execution directory before the calculation (input profiles for instance). The files will be looked for in the directory `DATA`. The thermochemical data files, Interface parameter file and calculation restart files are specified in other variables and do not need to appear here. When using the vortex method for LES entry conditions, the corresponding data files have to be specified in `USER_INPUT_FILES` (see §6.5)
- `USER_OUTPUT_FILES`: list of user result files to be copied in the directory `RESU` at the end of the calculation. A directory `RES_USERS.mmddhhmm` will be created in the directory `RESU` and all the files will be stored in it. The files automatically created by the code (listings, post-processing, automatic chronological records[11], restart files) do not need to be specified in `USER_OUTPUT_FILES`.
- `CS_TMP_PREFIX`: alternate temporary directory for the calculation (see §2.2.2)
- `OPTIMIZATION`: optimisation level for compilation (LO, DBG, EF or PROF; see §2.1.3). This optimisation level will be applied to all the modules of *Code_Saturne* (BASE, CFBL, COGZ, CPLV, ELEC, FUEL, LAGR, MATI, RAYT). Leaving the variable empty stands for "standard" optimisation.

[11] when using `ushist` for user-defined chronological records, the files created need to be specified in `USER_OUTPUT_FILES`

- `CS_LIB_ADD`: additional commands for the link stage of the compilation. This can be especially useful if the user subroutines call routines provided by external libraries. To link with an external library "foo", the variable would be for instance
  `CS_LIB_ADD=''-L/opt/foo/lib -lfoo''`
- `VALGRIND`: command to be placed before the *Code_Saturne* executable name on the execution command line (*i.e.* the launch script will execute the command `$VALGRIND cs_solver ...`). It is especially designed to use the valgrind debugging and profiling tool. The usual value to use valgrind is `VALGRIND=''valgrind --tool=memcheck''`
- `ARG_CS_VERIF`: verification mode to be used for *Code_Saturne* (see 2.6). An empty variable implies standard calculation mode (IVERIF=0).
- `ARG_CS_OUTPUT`: options for the redirection of the standard output (see 2.6)
- `ECHOCOMM`: level for the `--echo-comm` option of the Kernel command line (see 2.6)
- `ADAPTATION`: commands to trigger the automatic mesh adaptation with the software Homard. This option is still under development and restricted to developpers use.
- `CASEDIR`: root directory of the calculation. This variable is automatically set by `cs create` and should not be changed.
- `DATA`: DATA directory of the case (see 2.1.2). This variable is automatically set by `cs create` and should not be changed.
- `RESU`: RESU directory of the case (see 2.1.2). This variable is automatically set by `cs create` and should not be changed.
- `SRC`: SRC directory of the case (see 2.1.2). This variable is automatically set by `cs create` and should not be changed.
- `SCRIPTS`: SCRIPTS directory of the case (see 2.1.2). This variable is automatically set by `cs create` and should not be changed.
- `RESTART_IN`: directory containing the files for calculation restart.
- `PREPROCESSOR_OUTPUT_IN`: `preprocessor_ouput` file for a calculation in "calculation" mode (see 2.2.3)
- `MESHDIR`: directory containing the mesh files (see 2.1.2). This variable is automatically set by `cs create` and should generally not be changed.
- `DATA_SYR`: directory for the SYRTHES data. This directory has to be created by the user. It is advised to keep the location proposed in the launch script, which complies with the standard architecture of *Code_Saturne* (see 2.1.2).
- `SYRTHES_ENV`: name of the environment file for SYRTHES (usually `syrthes.env`, as proposed in the launch script).
- `SRC_SYR`: directory for the SYRTHES user subroutines. This directory has to be created by the user. It is advised to keep the location proposed in the launch script, which complies with the standard architecture of *Code_Saturne* (see 2.1.2).
- `COUPLING_MODE`: coupling mode between *Code_Saturne* and SYRTHES 3.4, when such coupling is activated (see `COMMAND_SYRTHES`). Two options are available:
  `MPI`: for a coupling based on MPI messages (requires a MPI library)
  `sockets`: for a coupling based on sockets
- `EXEC_PREPROCESSOR`: execution mode for *Code_Saturne* preprocessor (see 2.2.3)
- `EXEC_PARTITION`: execution mode for *Code_Saturne* partitionner (see 2.2.3)
- `EXEC_KERNEL`: execution mode for *Code_Saturne* kernel (see 2.2.3)

2.8 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* syntax which is then interpreted by the code.

In version 2.0.0-rc1, the Graphical Interface manages calculation parameters, standard initialisation values and boundary conditions for standard physics, pulverised coal combustion and radiative transfers. The other specific physics are not yet managed by the Graphical Interface. In these particular cases, user subroutines have to be completed.

The Interface is optional. Every data that can be specified through the Interface can also still 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 `usini1.f90`, or to define additional scalars in `usini1` 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 `usclim`, and their nature (entry, outlet, wall[12], 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 `usclim`. The wall and output areas do not need to appear in `usclim`. 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 followings:

- Analysis environment: definition of the calculation directories (STUDY, CASE), mesh file(s), periodicity, coupling with SYRTHES, stand-alone execution of the Preprocessor module (used by the Interface to get the colors of the boundary faces).

- Thermophysical models: physical model, ALE mobile mesh features, turbulence model, thermal model, initialisation of the variables.

- Physical properties: reference pressure, fluid characteristics, gravity.

- Additional scalars: definition, physical characteristics and initialisation of the scalars (apart from the temperature, which is treated separately in the Interface).

- Boundary conditions: definition of the boundary conditions for each variable. 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
"Analysis environment → Solution Domain → Stand-alone running".

- Analysis 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.

- Numerical parameters: advanced parameters for the numerical solution of the equations

[12] smooth and rough walls are considered of 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, ...), interactive launch of the calculation and user arrays definition.

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 to the variable `PARAM` in the launch script (see §2.7). When the launch script is edited from the Interface (Calculation management → Prepare batch analysis), the PARAM section is filled automatically as are the other parameters specified through the Interface.

NOTE: OPTION `--nogui` OF THE `cs create` COMMAND
When a calculation is using the Interface but, for some reason, some extra parameters need to be specified in the subroutine `usini1`, the latter must be placed in the directory `SRC`. But, while doing this, all the parameters appearing in `usini1` will also be taken into account. In order to prevent the user from having to respecify in `usini1` all that he has already specified through the Interface, `cs create` automatically comments out the examples in `usini1` (`Cex` at the beginning of each line) while copying it in the directory `REFERENCE`. Therefore, the user only needs to uncomment the specific parts of `usini1` he wants to modify, and the rest of the examples will be ignored.
On the contrary, if the Interface will not be used, then all the parameters in `usini1` have to be specified. In that case, using the `--nogui` option of `cs create` will prevent it from commenting `usini1` out, thus saving the user the tedious task of uncommenting all the lines (and the risk of skipping some of them).

2.9 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 Comet Design (pro-STAR/STAR4) and NUMECA Hex file formats have a CAD section id that is interpreted as a color number. In the latter case, this notion only applies to faces, so volume elements are given color.
  - The MED format allow integer "attributes", though many tools working with this format ignore 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" to 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 whitespace or having names similar to reserved operators may be protected using "escape characters".[13] More complex examples of strings whith 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 capitals/lowercase 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

[13] 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.

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 centers. Normals are of course usable only for face selections, not cell selections.

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`]`
	`plane[`a`, `b`, `c`, `d`, epsilon = `*epsilon*`]`
	`plane[`a`, `b`, `c`, `d`, inside]`
	`plane[`a`, `b`, `c`, `d`, outside]`
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.

3 Preprocessing

The Preprocessor module of *Code_Saturne* reads the mesh file(s) (under any supported format) and translates the necessary information into a Kernel input file. Mesh joining and domain partitioning for parallel calculations are done at this stage. In case of periodic boundary conditions, the Preprocessor module also identifies the boundary faces that are related through periodicity and generates the corresponding connectivity.

The executable of the Preprocessor module is `cs_preprocess`, and the 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.

For the main options, the launch script `runcase` contains corresponding variables, that are used to define options for the Preprocessor. This way, the user only has to define these variables and does not detailed knowledge of the Preprocessor command line.

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 `cs check_mesh` command, which takes a subset of the same command-line arguments.

The Preprocessor is controlled using command-line arguments, some of these accepting sub-arguments. It is possible to complete the command-line with a file, using the same syntax as the command-line, but also allowing multiple lines and comments. A few environment variables allow an expert user to modify some behaviors or to obtain a trace of memory management.

3.1 Preprocessor options and sub-options

Main choices are done using command-line options. Sub-option arguments must directly follow the one activating the corresponding option, and the definition of sub-options ends with the first argument which does not match the same sub-option. Some options may be applied multiple times. For example, `-j` being the option which activates face joining, `--fraction` the sub-option modifying the default tolerance, and `--color` a face selection sub-option:

```
cs_preprocess -m fluid.msh -j --fraction 0.2 --color 2 --color 3 -j
```

means that we apply a joining to boundary faces of color 2 or 3 of mesh `fluid.msh` with a tolerance of 0.2, then that we apply a second joining to all remaining boundary faces (with default parameters), while:

```
cs_preprocess -m fluid.msh -j --fraction 0.2 -color 2 -j --color 3
```

means that we apply a joining to boundary faces of color 2 of mesh `fluid.msh` with a tolerance of 0.2, then that we apply a second joining to boundary faces of color 3 (with default values for other parameters).

3.1.1 Option files

It is possible to complete the command-line with a file, using the same syntax as the command-line, but allowing the addition of arbitrary line jumps and comments. On a given line, anything that follows a`#` character until the end of the line is ignored in such a file. The use of such a file is activated by the argument `-i`. If for example a file `pp_cmd` contains the following lines:

```
--ensight # activate EnSight output
--med     # activate MED output
```

then command: `cs_preprocess -i pp_cmd -m mesh.unv`
is equivalent to: `cs_preprocess -m mesh.unv --ensight --med`

The use of option files may circumvent command-line length limits (now rarely an issue), and allows easier manipulation and better readability of complex option combinations.

3.1.2 Mesh selection

Any use of the preprocessor requires one or several meshes (except for `cs_preprocess` and `cs_preprocess -h` which respectively print the version number and list of options). They are selected using the `--mesh` or `-m` option, followed by th list of meshes to read. The file format is usually automatically determined based on its extension (c.f. 2.4.1 page 18) but this option also allows a `--format` sub-option, to force the format choice of selected files.

For formats allowing multiple meshes in a single file, the `--num` sub-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` sub-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.

We may define as many mesh selections as we wish. Meshes are read and automatically compounded (but not joined) in the order of their appearance. For example:

`cs_preprocess -m file.1 file.2 --format ideas -m branch.cgns --grp-cel section`

reads mesh files `file.1` and `file.2` as I-deas files, and appends the first mesh from the `branch.cgns` file, on which additional groups of cells corresponding to the CGNS section names to which cells belong.

3.1.3 Post-processing output

By default, the Preprocessor does not generate any post-processor output. By adding `--ensight`, `--med`, or `--cgns` to the command-line arguments, the output of the mesh to the indicated format is provoked. It is possible to generate output to different formats simultaneously. Note that we will obtain additional surface meshes corresponding to the calculation domain boundary, to faces joined or modified by joinings or periodicity, etc. Note also that periodic faces are not considered to be part of the domain boundary. [14]

We consider 4 types of output: the volume mesh (output before possible joinings), the boundary mesh, informational meshes (such as faces stemming from or modified by a joining of periodicity, or selected interior faces), and meshes outlining to zones with errors. By default, all types of meshes are output. If one or several of the three filtering sub-options is used , only the meshes of the requested types will be output, as well as meshes corresponding to zones with errors (non-filterable). The filtering sub-options are: `--volume`, `--boundary`, and `--info`.

One may also optionally avoid outputting polygons or polyhedra by adding the `--discard-poly` sub-option. This allow working around bugs or limited polyhedron support in visualization tools or limitations of the CGNS 2 format. In this case, the post-processing output will be incomplete, but at least readable.

In the case of *EnSight Gold* output, we may use the `--simple` sub-option to avoid splitting the output in several *parts* corresponding to different attributes (colors and groups) borne by this mesh.

Still in the case of the *EnSight Gold* output, we may force the output in text mode using the `--text` sub-option, or force binary output to "big-endian" variant using the `--big-endian` sub-option. By default, the output is native binary.

Finally, in the case of output in the *MED* format, we may force the conversion of attributes (colors) to groups using the `--color-to-group` sub-option.

3.1.4 Faces selection

Selection of faces occurs at several levels, especially for conforming joining.

It is also possible to select a set of interior faces bearing attributes (colors or groups) for verification purposes using the option `--int-face`. The Preprocessor prints the number of interior faces matching a selection criterion, and if in addition post-processing output is activated, a surface mesh corresponding to the selected faces will be generated.

Preprocessor selection criteria are more limited than those used by th kernel, and are generated using the following sub-options:

`--color` c_1 c_2 ... c_n	to select faces of the indicated color numbers
`--group` g_1 g_2 ... g_n	to select faces of the groups named
`--invsel`	to use the complement of the indicated colors and groups

The same selection sub-options are used for conforming joining, periodicity, and interior faces verification. In the case of joining and periodicity, the selection is automatically restricted to boundary faces, while in the case of interior face verification, it is restricted to interior faces bearing attributes.

For example, to select all interior faces of mesh `example.des` which bear an attribute (color or group), but are not of color 2 and do not belong to group "inlet", with output of an *Ensight Gold* part and

[14] Periodicity is interpreted as a "geometric" condition rather than a classical boundary condition.

with no kernel output (using `-sc`), we use the following command line (the `\` character marks line continuation):

```
cs_preprocess -m example.des -sc --ensight  \
              --int-face --group entree --color 2 --invsel
```

3.1.5 Joining of non-conforming meshes

Conforming joining of possibly non-conforming meshes is one of the Preprocessor's main functions. To activate it, use the `--join` or `-j` command-line options, possibly followed by face selection criteria and sub-options controlling associated tolerance parameters.

For a simple mesh, it is rarely useful to specify 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 sub-arguments (see §2.4.1). This has the additional advantage of reducing the number of faces to test for in the intersection/overlap search, and thus reduced to time required by the joining algorithm.

One may also modify tolerance criteria using 2 sub-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 2;

`--plane` c assigns the minimum cosine between normals for two faces to be considered coplanar ($0,8$ by default, which corresponds to almost 37°); this parameter is used in the second stage of the algorithm, to reconstruct conforming faces, as shown on the right of figure 2.

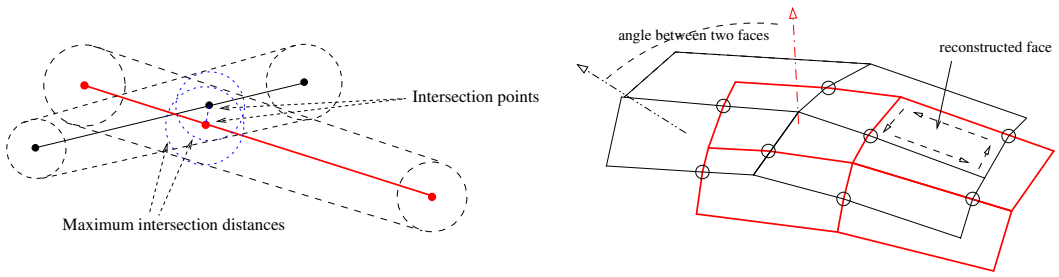


Figure 2: Maximum intersection tolerance and faces normal angle

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` sub-option, its use has only been necessary on a few meshes up to now, and always in the sense of reducing the tolerance (i.e. bringing the minimum cosine closer to 1) so that face reconstruction does not try to generate faces from initial faces on different surfaces.

As an example, to join faces with color 2 of the mesh `mesh.des` with an intersection distance multiplier of 0.15, then joining all remaining joinable boundary faces with a default parameter of 0.1, we may use the following command:

```
cs_preprocess -m mesh.des -j --fraction 0.15 --color 2 -j
```

In cases where any two faces to join always have at least one common vertex (which is the case with meshes containing hanging nodes generated by tools such as Harpoon), we may use the `--semi-conf`

sub-option, for a much faster edge intersection search than with the usual algorithm.

3.1.6 Periodicity

Handling of periodicity is based on an extension of conforming joining, as shown on figure 3. It is thus not necessary that periodic faces be conforming (though it usually leads to better mesh quality). All sub-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).

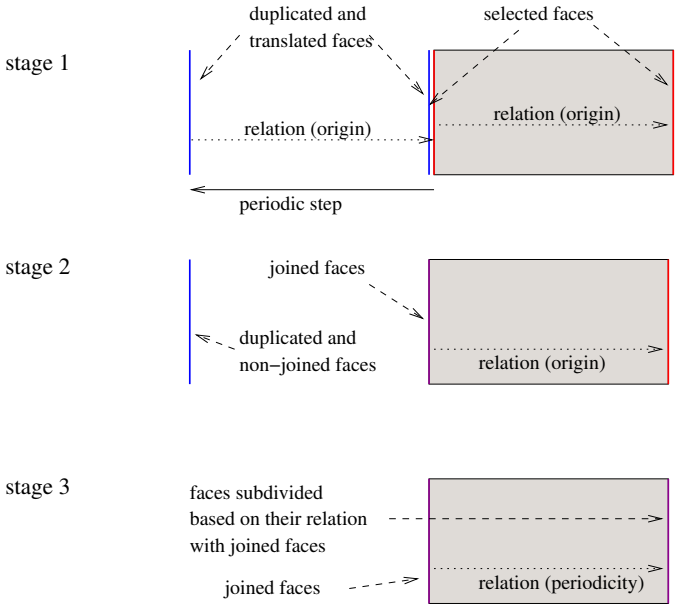


Figure 3: Matching of periodic faces

The sub-option for a translation-type periodicity is `--trans` followed by the three translation vector components (tx, ty, tz). The sub-option for a rotation-type periodicity is `--rota`, which must be completed either by:

`--angle` α `--dir` dx dy dz

where α is the rotation angle in degrees and (dx, dy, dz) defines the rotation axis (using the trigonometric orientation), or by:

`--matrix` $m11$ $m12$ $m13$ $m21$ $m22$ $m23$ $m31$ $m32$ $m33$

where $\begin{pmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ m_{31} & m_{32} & m_{33} \end{pmatrix}$ defines the rotation matrix.

In either case, we may define an invariant point different from $(0, 0, 0)$ by adding the rotation sub-option `--invpt` px py pz. A translation and a rotation may be combined: in this case, the composite transformation is always $R \circ T$, independently of the order in which the translation and rotation are defined.

The orientation of the transformations has no importance (but in the case of a composite transformation $R \circ T$, R and T must be consistent). 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. One 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.

We finish with the example of a periodicity of rotation of 90 degrees around an axis of direction $(0, 0, 1)$ passing through $(1, 0, 0)$, for faces of group "perio_1" and with a joining tolerance of 20% (the

\ character marks line continuation):

```
--perio --rota --angle 90 --dir 0 0 1 --invpt 1 0 0 \
       --fraction 0.2 --group perio_1
```

3.1.7 Element orientation correction

We may active the possible element orientation correction 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 `--ensight`, visualizable meshes of corrected elements as well as remaining badly oriented elements are generated.

3.2 Environment variables

Setting a few environment variables specific to the Preprocessor allows modifying its default behavior. In general, if a given behavior 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:

`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 Kernel 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 behavior 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_PREPROCESS_JOIN_MAX_SUB_FACES`

Defines the number of new faces originating from an initial face (100 by default) above which we consider that the joining reconstruction has probably entered an infinite loop and has thus failed. This criterion is the last to intervene in the detection of errors and only comes into play in pathological cases probably related to excessive deformation of faces to join (and thus a too large tolerance factor). It should not be necessary to increase it for a mesh really intended for a calculation, as 100 faces correspond to a refinement ratio of 10 in each direction for joined cells, which is excessive from a numerical standpoint.

3.2.1 System environment variables

Some system environment variables may also modify the behavior 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, and thus experiment with another version 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}`.

3.3 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.
- Read compressed files using Zlib. With this option, it is possible to diretly 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 or MED files), and has memory and CPU time overhead, but may be practical. Without this library, files must be uncompressed before use.

3.4 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 centers 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. [15]

At this stage, volume elements are sorted by type, and the fluid domain post-processing output is generated if required. Conforming joining, which may transform cells into general polyhedra only occurs after this. This allows bypassing a limitation of most post-processing tools, which do not handle polyhedral elements. *It is possible to require the simultaneous output with multiple formats.*

In general, colors or 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 descending connectivity, all faces with the same topology are merged: the initial presence of two layers of identical surface elements bearing different colors would thus lead to a calculation mesh with faces bearing two colors.

3.5 Files passed to the Kernel

Data passed to the Kernel by the Preprocessor is transmitted using a binary file, using "big endian" data representation, named `preprocessor_output`.

When using the Preprocessor for mesh verification, data for the Kernel is usually not needed. In this case, the `-sc` option may be used to simulate output, without creation of a `preprocessor_output` file.

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

4 Partitioning for parallel runs

Graph partitioning (using one of the optional METIS or SCOTCH libraries) is done using a secondary executable,
`cs_partition`, which reads the file produced by the Preprocessor and builds one or several "cell → domain" distribution files, named `domain_number_`p for a partitioning on p sub-domains.

This separation leads to extra work for the Kernel, which must redistribute data read in the `preprocessor_output` file based on the associated partitioning, but avoids requiring re-running the Preprocessor whenever running on a different number of files. This useful mainly for large files (20 to 100 million cells, for which running the Preprocessor may require several hours and a machine with a very large memory capacity. Simple domain partitioning requires much less time, and in general slightly less memory than the Preprocessor.

Without partitioning (for example if neither METIS nor SCOTCH is available, or the partitioner has not been run for the required number of sub-domains), the Kernel will use a built-in partitioning using a space-filling curve (Z-curve) technique. This usually leads to partitionings of lower quality than with graph partitioning, but parallel performance remains reasonable.

4.1 Options

To list the partitioner's options, use the following command: `cs_partition -h`

We provide the list of required partitionings an optionally additional options. For example, to simulate a partitioning for calculations on 64 and 128 processes with no output, we may use the following command:

```
cs_partition 64 128 --no-write
```

4.1.1 Ignore 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 using the `--no-perio` option.

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

4.1.2 Partitioner choice

If the Partitioner has been configured with both METIS and SCOTCH libaries, using the `--metis` or `--scotch` option allows choosing between either library. By default, *metis* is used if both choices are available.

4.1.3 Simulation mode

Using the `--no-write` option, we can tell the partitioner not to output a `domain_number_`p file. Partitioning is thus computed, but not saved.

4.1.4 Environment variables

`CS_PARTITION_MEM_LOG`

Allows defining a file name in which memory allocations, reallocations, and frees will be logged.

5 Main variables

This section presents a non-exhaustive list of the main variables which 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. Developpers can also refer to [3] and [11].

These variables are listed in the alphabetical index at the end of this document.

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

5.1 Array sizes

`ndim`: Space dimension (ndim=3).

`ncel`: Number of real cells in the mesh.

`ncelet`: Number of cells in the mesh, including the ghost cells of the "halos" (see note 1).

`nfac`: Number of internal faces (see note 2).

`nfabor`: Number of boundary faces (see note 2).

`ncelbr`: Number of cells with at least one boundary face (see note 2).

`lndfac`: Size of the array `nodfac` of internal faces - nodes connectivity (see note 3).

`lndfbr`: Size of the array `nodfbr` of boundary faces - nodes connectivity (see note 3).

`nnod`: Number of vertices in the mesh.

`nfml`: Number of referenced families of entities (boundary faces, elements, ...).

`nprfml`: Number of properties per referenced entity family.

`nphas`: Effective number of phases. `nphas` must be inferior or equal to `nphsmx`. In the current version, `nphas` is forced to 1 and should not be changed..

`nphsmx`: Maximum number of phases (default value: 1)[16].

`nvar`: Number of solved variables (must be lower than `nvrmax`).

`nscamx`: Maximum number of scalars solutions of an advection equation, apart from the variables of the turbulence model (k, ε, R_{ij}, ω, φ, $\overline{f}$). That is to say the temperature and other scalars (passive or not, user-defined or not).

`nscal`: Effective number of scalars solutions of an advection equation, apart from the variables of the turbulence model (k, ε, R_{ij}, ω, φ, $\overline{f}$). That is to say the temperature and other scalars (passive or not, user-defined or not). These scalars can be divided into two distinct groups: `nscaus` user-defined scalars and `nscapp` scalars related to a "specific physics". `nscal=nscaus+nscapp`, and `nscal` must be inferior or equal to `nscamx`.

`nscapp`: Effective number of scalars related to a "specific physics". These scalars are solutions of an advection equation and distinct from the scalars of the turbulence model (k, ε, R_{ij}, ω, φ, $\overline{f}$). They are automatically defined by the choice of the selected specific physics model (gas combustion with Eddy Break-Up model, pulverised coal combustion, ...). For example: mass fractions, enthalpy, ...

`nscaus`: Effective number of user-defined scalars. These scalars are solutions of an advection equation and distinct from the scalars of the turbulence model (k, ε, R_{ij}, ω, φ, $\overline{f}$) and from the `nscapp` scalars related to the "specific physics". For example: passive tracers, temperature (when no specific physics model is selected), ...

`nestmx`: Maximum number of error estimators for Navier-Stokes.

`longia`: Size of the macro array of integer `ia`.

`longra`: Size of the macro array of real `ra`.

`npromx`: Maximum number of physical properties. They will be stored in the arrays `propce`, `propfa` or `propfb`.

`nproce`: Number of properties defined at the cells. They will be stored in the array `propce`.

`nprofa`: Number of properties defined at the internal faces. They will be stored in the array `propfa`.

`nprofb`: Number of properties defined at the boundary faces. They will be stored in the array `propfb`.

`nvisls`: Number of scalars with variable diffusivity.

`nushmx`: Maximum number of user chronological files (in the case where `ushist` is used).

`nbmomt`: Effective number of calculated time-averages. NBMOMT must be inferior or equal to `nbmomx`.

`nbmomx`: Maximum number of calculated time-averages (default value: 50).

`ndgmox`: Maximum degree of the time-averages (default value: 5).

`nclacp`: Number of coal classes for the pulverised coal combustion module. It is the total number of classes, *i.e.* the sum of the number of classes for every represented coal. `nclacp` must be inferior or equal to `nclcpm`.

`nclcpm`: Maximum number of coal classes for the pulverised coal combustion module.

Note 1: ghost cells - "halos"

A cell (real cell) is an elementary mesh element of the spatial discretisation of the calculation domain. The mesh is made of NCEL cells.
When using periodicity and parallelism, extra "ghost" cells (called "halo" cells) are defined for temporary storage of some information (on a given processor). The total number of real and ghost cells is `ncelet`.

Indeed, when periodicity is enabled, the cells with periodic faces do not have any real neighboring cell across these particular faces. Their neighboring cell is elsewhere in the calculation domain (its position is determined by the periodicity). In order to temporarily store the information coming from this "distant" neighboring cell, a ghost cell ("halo") is created.

The same kind of problem exists in the case of a calculation on parallel machines: due to the decomposition of the calculation domain, some cells no longer have access to all their neighboring cells, some of them being treated by another processor. The creation of ghost cells allows to temporarily store the information coming from real neighboring cells treated by other processors.
The variables are generally arrays of size `ncelet` (number of real and fictitious cells). The calculations (loops) are made on `ncel` cells (only the real cells, the fictitious cells are only used to store information).

Note 2: internal faces

An internal face is an inferface shared by two cells (real or ghost ones) of the mesh. A boundary face is a face which has only one real neighboring cell. In the case of periodic calculations, a periodic face is an internal face. In the case of parallel running calculations, the faces situated at the boundary of a partition may be internal faces or boundary faces (of the whole mesh);

NOTE 3: FACES-NODES CONNECTIVITY

The faces - nodes connectivity is stored by means of four integer arrays: `ipnfac` and `nodfac` for the internal faces, `ipnfbr` and `nodfbr` for the boundary faces. `nodfac` (size `lndfac`) contains the list of all the nodes of all the internal faces; first the nodes of the first face, then the nodes of the second face, and so on. `ipnfac` (size: `nfac+1`) gives the position `ipnfac(ifac)` in `nodfac` of the first node of each internal face `ifac`. Therefore, the reference numbers of all the nodes of the internal face `ifac` are: `nodfac(ipnfac(ifac))`, `nodfac(ipnfac(ifac)+1)`, ..., `nodfac(ipnfac(ifac+1)-1)`. In order for this last formula to be valid even for `ifac=nfac`, `ipnfac` is of size `nfac+1` and `ipnfac(nfac+1)` is equal to `lndfac+1`.
The composition of the arrays `nodfbr` and `ipnfbr` is similar.

NOTE 4: COMMONS

The user will not modify the existing "commons". This would require the recompilation of the complete version, operation which is not allowed in standard use.

5.2 Geometric variables

The main geometric variables are available in most of the subroutines and directly accessible through the following arrays.

`cdgfac(ndim,nfac)` [ra]: Coordinates of the centers of the internal faces.

`cdgfbo(ndim,nfabor)` [ra]: Coordinates of the centers of the boundary face.

`ifacel(2,nfac)` [ia]: Index-numbers of the two (only) neighboring cells for each internal face.

`ifabor(nfabor)` [ia]: Index-number of the (unique) neighboring cell for each boundary face.

`ipnfac(nfac+1)` [ia]: Position of the first node of the each internal face in the array `nodfac` (see note 3 in paragraph 5.1)..

`ipnfbr(nfabor+1)` [ia]: Position of the first node of the each boundary face in the array `nodfbr` (see note 3 in paragraph 5.1)..

`nodfac(lndfac)` [ia]: Index-numbers of the nodes of each internal face (see note 3 in paragraph 5.1)..

`nodfbr(lndfbr)` [ia]: Index-numbers of the nodes of each boundary face (see note 3 in paragraph 5.1)..

`surfac(ndim,nfac)` [ra]: Surface vector of the internal faces. Its norm is the surface of the face and it is oriented from `ifacel(1,.)` to `ifacel(2,.)`..

`surfbo(ndim,nfabor)` [ra]: Surface vector of the boundary faces. Its norm is the surface of the face and it is oriented outwards.

`volume(ncelet)` [ra]: Volume of each cell.

`xyzcen(ndim,ncelet)` [ra]: Coordinates of the cell centers.

`xyznod(ndim,nnod)` [ra]: Coordinates of the mesh vertices.

In addition, other geometric variables are accessible in sections of the unidimensional macro-arrays IA (for integers) and RA (for real numbers) which are passed as arguments in every subroutine (apart from a few ones of very low level). The index-number of the first element of these sections is stored in a "common" (in the file `pointe.h`), passed to most of the routines. Hence, the surface of an internal face `ifac` is stored in `ra(isrfan+ifac-1)`. Or, the coordinate of vector $\underline{OF}$ (see below for definition) in the IIth direction for face `ifac` is stored in `ra(idofij+(ifac-1)*ndim+ii-1)`[17].

The main variables of this type are the following:

[17] in Fortran, a multidimensional array `a(3,2)` is in fact a unidimensional array containing the elements `a(1,1)`, `a(2,1)`, `a(3,1)`, `a(1,2)`, `a(2,2)` and `a(3,2)` in this order.

`idijpf` [i]: In `ra`, pointer to `dijpf(ndim,nfac)`, real array giving, for every internal face, the three components of the vector $\underline{I'J'}$, where I' and J' are respectively the orthogonal projections of the neighboring cell centers I and J on a straight line orthogonal to the face and passing through its center..

`idiipb` [i]: In `ra`, pointer to `diipb(ndim,nfabor)`, real array giving, for every boundary face, the three components of the vector $\underline{II'}$. I' is the orthogonal projection of I, center of the neighboring cell, on the straight line perpendicular to the face and passign through its center.

`idist` [i]: In `ra`, pointer to `dist(nfac)`, real array giving, for every internal face, the scalar product between the vectors $\underline{IJ}$ and $\underline{n}$. I and J are respectively the centers of the first and the second neighboring cell. The vector $\underline{n}$ is the unit vector normal to the face and oriented from the first to the second cell.

`idistb` [i]: In `ra`, pointer to `distbr(nfabor)`, real array giving, for every boundary face, the scalar product between the vectors $\underline{IF}$ and $\underline{n}$. I is the center of the neighboring cell. F is the face center. The vector $\underline{n}$ is the unit vector normal to the face and oriented to the exterior of the domain.

`idofij` [i]: In `ra`, pointer to `dofij(ndim,nfac)`, real array giving, for every internal face, the components of the vector $\underline{OF}$. O is the intersection point between the face and the straight line joining the centers of the two neighboring cells. F is the face center.

`iicelb` [i]: In `ia`, pointer to `icelbr(ncelbr)`, integer array giving the list of cells having at least one boundary face.

`ipond` [i]: In `ra`, pointer to `pond(nfac)`, real array giving $\frac{\underline{FJ}.\underline{n}}{\underline{IJ}.\underline{n}}$, for every internal face. With regard to the mesh quality, its ideal value is 0.5.

`isrfan` [i]: In `ra`, pointer to `surfan(nfac)`, real array giving the norm of the surface vector of the internal faces.

`isrfbn` [i]: In `ra`, pointer to `surfbn(nfabor)`, real array giving the norm of the surface of the boundary faces.

5.3 Physical variables

The main physical variables are available in the majority of the subroutines and brought together according to their type in the multidimensional arrays listed below. In some paricular subroutines, some variables may be given a more explicit name, in order to ease the comprehension.

`propce(ncelet,nproce)` [ra]: Properties defined at the cell centers. For instance: density, viscosity, ...

`propfa(nfac,nprofa)` [ra]: Properties defined at the internal faces. For instance: mass flow across internal faces.

`propfb(nfabor,nprofb)` [ra]: Properties defined at the boundary faces. For instance: mass flow across boundary faces, density at boundary faces, ...

`rtp(ncelet,nvar)` [ra]: Array storing the values of the solved variables at the current time step.

`rtpa(ncelet,nvar)` [ra]: Array storing the values of the solved variables at the previous time step.

About `rtp` and `rtpa`

The indexes allowing to mark out the different variables (from 1 to `nvar`) are integers available in a "common" file called `numvar.h`. Some solved variables (pressure, velocity, turbulence) depend on the considered phase, and the index which refers to it is then a array of size `nphsmx`, the maximum number of phases.

For example, `ipr(iphas)` refers to the variable "pressure" of the phase `iphas` (with 1⩽iphas⩽nphas): the pressure of the phase `iphas` in the cell `iel` at the current time step is therefore `rtp(iel,ipr(iphas))`.

The list of integers referring to solved variables is given below. These variable index-numbers are not only used for the `rtp` and `rtpa` arrays, but also for some arrays of variable associated options (for instance, `blencv(ik(iphas))` is the percentage of second-order convective scheme for the turbulent energy of the phase `iphas` when a corresponding turbulent model is used).

- `ipr(iphas)`: pressure [18].
- `iu(iphas)`: velocity along the X axis.
- `iv(iphas)`: velocity along the Y axis.
- `iw(iphas)`: velocity along the Z axis.
- `ik(iphas)`: turbulent energy, in $k-\varepsilon$, $k-\omega$ modeling or v2f (φ-model) modeling.
- `ir11(iphas)`: Reynolds stress R11, in $R_{ij}-\varepsilon$ or SSG modeling.
- `ir22(iphas)`: Reynolds stress R22, in $R_{ij}-\varepsilon$ or SSG modeling.
- `ir33(iphas)`: Reynolds stress R33, in $R_{ij}-\varepsilon$ modeling.
- `ir12(iphas)`: Reynolds stress R12, in $R_{ij}-\varepsilon$ modeling.
- `ir13(iphas)`: Reynolds stress R13, in $R_{ij}-\varepsilon$ modeling.
- `ir23(iphas)`: Reynolds stress R23, in $R_{ij}-\varepsilon$ modeling.
- `iep(iphas)`: turbulent dissipation in $k-\varepsilon$, $R_{ij}-\varepsilon$ or v2f (φ-model) modeling.
- `iomg(iphas)`: Specific dissipation rate ω, in $k-\omega$ SST modeling.
- `iphi(iphas)`: variable $\varphi=\overline{v^2}/k$ in v2f (φ-model).
- `ifb(iphas)`: variable $\overline{f}$ in v2f (φ-model).
- `isca(j)`: scalar j(1⩽j⩽nscal).

Concerning the solved scalar variables (apart from the variables pressure, k, ε, R_{ij}, ω, φ, $\overline{f}$), the following are highly 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, $\overline{f}$): 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: `nscaus` "user" scalars and `nscapp` "specific physics" scalars, with `nscal=nscaus+nscapp`. `nscal` must be inferior or equal to `nscamx`.
- The phase related to the scalar `j` is `iphsca(j)`.
- The `j`$^{\text{th}}$ 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)`, its value in the cell `iel` at the current time step is given by `rtp(iel,isca(j))`.

[18] `ipr(iphas)` corresponds to a reduced pressure, from which the standard hydrostatic pressure has be deduced. The total pressure is stored in the PROPCE array

- The `j`$^{\text{th}}$ 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))`, its value in the cell `iel` at the current time step is given by `rtp(iel,isca(iscapp(j)))`.

- The temperature (or the enthalpy) is the scalar number `iscalt(iphas)` in the list of the `nscal` scalars. It corresponds to the variable number `isca(iscalt(iphas))` and its value in the cell `iel` is `rtp(iel,isca(iscalt(iphas)))`. if there is no thermal scalar, `iscalt(iphas)` is equal to -1.

- A "user" scalar number `j` may represent the average 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 indicating `iscavr(j)=k` in `usini1` (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))`, and its value in the cell `iel` is `rtp(iel,isca(k))=rtp(iel,isca(iscavr(j)))`.
  The variable corresponding to the mean value of the square of the fluctuations[19] is the variable number `isca(j)` and its value in the cell `iel` is `rtp(iel,isca(j))`.

About `propce`, `propfa` and `propfb` In *Code_Saturne*, the physical properties[20] are stored in the `propce`, `propfa` and `propfb` arrays. Some properties, like the density, are only stored for cells and boundary faces. Some, like the mass flux, are only stored at the interior and boundary faces. To avoid having different index numbers for a physical property, depending on the array it is used in, the following structure is used in *Code_Saturne*:

- All the properties (used or not) have a unique and distinct index-number, given automatically by the code and stored in an integer or an integer array (its size may be the maximum number of phases, the maximum number of scalars or the maximum number of variables).

- The indexes referring to the different properties stored in the `propxx` arrays are given respectively by the following integer arrays:

  `ipproc(npromx)` [ia]: Rank `i` in `propce(.,i)` of the properties defined at the cell centers.

  `ipprof(npromx)` [ia]: Rank `i` in `propfa(.,i)` of the properties defined at the internal faces.

  `ipprob(npromx)` [ia]: Rank `i` in `propfb(.,i)` of the properties defined at the boundary faces.

For instance, the index number corresponding to the density of the phase `iphas` is `irom(iphas)`.
In the list of the properties defined at the cell center, the density of the phase `iphas` is therefore the `ipproc(irom(iphas))`$^{\text{th}}$ property: its value at the center of the cell `iel` is given by `propce(iel,ipproc(irom(iphas)))`.
In the same way, in the list of the properties defined at the boundary faces, the density of the phase `iphas` is the `ipprob(irom(iphas))`$^{\text{th}}$ property: its value at the boundary face is given by `propfb(iel,ipprob(irom(iphas)))`

The list of properties accessible in the PROPxx arrays is given below (this does not include the properties linked to the specific physics modules):

`irom(nphsmx)` [ia]: For each phase, property number corresponding to the density (*i.e.* ρ in $kg.m^{-3}$) stored at the cells and the boundary faces.

`iroma(nphsmx)` [ia]: For each phase, property number corresponding to the density (*i.e.* ρ in $kg.m^{-3}$) at the previous time step, in the case of a second-order extrapolation in time stored at the cells and the boundary faces.

[19] it is really $\overline{\varphi'\varphi'}$, and not $\sqrt{\overline{\varphi'\varphi'}}$

[20] other variables are stored in the arrays `propce`, `propfa` and `propfb`. They are not "physical properties" strictly speaking, but it is convenient to have them in the same array as the proper physical properties

`iviscl(nphsmx)` [ia]: For each phase, property number corresponding to the fluid molecular dynamic viscosity (*i.e.* μ in $kg.m^{-1}.s^{-1}$)
stored at the cells.

`ivisla(nphsmx)` [ia]: For each phase, property number corresponding to the fluid molecular dynamic viscosity (*i.e.* μ in $kg.m^{-1}.s^{-1}$) at the previous time step, in the case of a second-order extrapolation in time
stored at the cells.

`ivisct(nphsmx)` [ia]: For each phase, property number corresponding to the fluid turbulent dynamic viscosity (*i.e.* μ_t in $kg.m^{-1}.s^{-1}$)
stored at the cells.

`ivista(nphsmx)` [ia]: For each phase, property number corresponding to the fluid turbulent dynamic viscosity (*i.e.* μ_t in $kg.m^{-1}.s^{-1}$) at the previous time step, in the case of a second-order extrapolation in time
stored at the cells.

`icp(nphsmx)` [ia]: For each phase, property number corresponding to the specific heat, in case where it is variable (*i.e.* C_p in $m^2.s^{-2}.K^{-1}$). See note below
stored at the cells.

`icpa(nphsmx)` [ia]: For each phase, property number corresponding to the specific heat, in case where it is variable (*i.e.* C_p in $m^2.s^{-2}.K^{-1}$), at the previous time step, in the case of a second-order extrapolation in time. See note below
stored at the cells.

`itsnsa(nphsmx)` [ia]: For each phase, in the case of a calculation run with a second-order discretisation in time with extrapolation of the source terms, property number corresponding to the source term of Navier-Stokes at the previous time step ($kg.m^{-1}.s^{-2}$)
stored at the cells.

`itstua(nphsmx)` [ia]: For each phase, in the case of a calculation run with a second-order discretisation in time with extrapolation of the source terms, property number corresponding to the source terms of the turbulence at the previous time step
stored at the cells.

`itssca(nphsmx)` [ia]: For each phase, in the case of a calculation run with a second-order discretisation in time with extrapolation of the source terms, property number corresponding to the source terms of the equations solved for the scalars at the previous time step ($kg.m^{-1}.s^{-2}$)
stored at the cells.

`iestim(nestmx,nphsmx)` [ia]: For each phase, property number for the `nestmx` error estimators for Navier-Stokes. The estimators currently available are `iestim(iespre,iphas)`, `iestim(iesder,iphas)`, `iestim(iescor,iphas)`, `iestim(iestot,iphas)` stored at the cells.

`ifluma(nvarmx)` [ia]: Property number corresponding to the mass flow associated with each variable (*i.e.* for each face of surface S, $\rho \underline{u} \,.\, \underline{S}$ in $kg.s^{-1}$). It must be noticed that the mass flows are associated with the variables and not with the phases. This allows to have a distinct convective flow for each scalar.
stored at the internal faces and boundary faces.

`ifluaa(nvarmx)` [ia]: Property number corresponding to the mass flow associated with each variable at the previous time step, in the case of a second-order extrapolation in time
stored at the internal faces and boundary faces.

`ivisls(nscamx)` [ia]: Property number corresponding to the diffusivity of scalars for which it is variable (*i.e.* $\dfrac{\lambda}{C_p}$ for the temperature, in $kg.m^{-1}.s^{-1}$). It must be noticed that the diffusivity is associated with the scalars rather than with the variables. See note below
stored at the cells.

`ivissa(nscamx)` [ia]: Property number corresponding to the diffusivity of scalars for which it is variable (*i.e.* $\frac{\lambda}{C_p}$ for the temperature, in $kg.m^{-1}.s^{-1}$) at the previous time step, in the case of a second-order extrapolation in time
stored at the cells.

`ismago(nphsmx)` [ia]: For each phase, property number corresponding to the variable C of the dynamic model, *i.e* so that $\mu_t = \rho C \overline{\Delta}^2 \sqrt{2 S_{ij} S_{ij}}$ (with the notations of [2]). C corresponds to C_s^2 in the classical model of Smagorinsky
stored at the cells.

`icour(nphsmx)` [ia]: For each phase, CFL number in each cell at the present time step
stored at the cells.

`ifour(nphsmx)` [ia]: For each phase, Fourier number in each cell at the present time step
stored at the cells.

`iprtot(nphsmx)` [ia]: For each phase[21], total pressure in each cell
stored at the cells.

`ivisma(1 or 3)` [ia]: When the ALE method for deformable meshes is activated, `ivisma` corresponds to the "mesh viscosity", allowing to limit the deformation in certain areas. This mesh viscosity can be isotropic or be taken as a diagonal tensor (depending on the value of the parameter `iortvm`.
stored at the cells.

`icmome(nbmomx)` [ia]: Property number corresponding to the time averages defined by the user. More precisely, it is not the time average that is stored, but a summation over time (the division by the cumulated duration is done just before the results are written)
stored at the cells.

`icdtmo(nbmomx)` [ia]: Property number corresponding to the cumulated duration associated with each time average defined by the user, when this duration is not spatially uniform (see note below)
stored at the cells.

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.

- It is the case for the specific heat C_p.
  - If C_p is constant for the phase `iphas`, it can be specified in the interface or by indicating `icp(iphas)=0` in `usini1`, and the property will be stored in the real number `cp0(iphas)`.
  - If C_p is variable, it can be specified in the interface or by indicating `icp(iphas)=1` in `usini1`. The code will then modify this value to make `icp(iphas)` refer to the effective property number corresponding to the specific heat of the phase `iphas`, in a way which is transparent for the user. For each cell `iel`, the value of C_p is then given in `usphyv` and stored in the array `propce(iel,ipproc(icp(iphas)))`.
- It is the same for the diffusivity K of each scalar `iscal`.
  - If k is constant, it can be specified in the interface or by indicating `ivisls(iscal)=0` in `usini1`, and the property will be stored in the real number `visls0(iscal)`.

[21] Although the data structure of *Code_Saturne* allows multi-phase variables, the algorithm does not allow more than one pressure

- If k is variable, it can be specified in the interface or by indicating `ivisls(iscal)=1` in `usini1`. The code will then modify this value to make `ivisls(iscal)` refer to the effective property number 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 `propce(iel,ipproc(ivisls(iscal)))` array.

NOTE: CUMULATED DURATION ASSOCIATED WITH THE AVERAGES DEFINED BY THE USER

The cumulated duration associated with the calculation of a time averages defined by the user is often a spatially uniform value. In this case, it is stored in a simple real number: for the mean value `imom`, it is the real number `dtcmom(-idtmom(imom))` (`idtmom(imom)` is negative in this case).
When this cumulated duration is not spatially uniform (for instance in the case of a spatially variable time step), it is stored in `propce`. It must be noted that the cumulated duration associated with the calculation of the average `imom` is variable in space if `idtmom(imom)` is strictly positive. The number of the associated property in `propce` is then `icdtmo(idtmom(imom))`. For instance, for the average `imom`, the cumulated duration in the cell `iel` will be `propce(iel,icdtmo(idtmom(imom)))`.
The user may have a look to the example given in `usproj` to know how to calculate a time averages in a particular cases (printing of extreme values, writing of results, ...).

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 `tridim`, 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[22]. The table is allocated only if `isvtb` is set to 1 in `tridim`, 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.

5.4 Variables related to the numerical methods

The main numerical variables and "pointers"[23] are displayed below.

BOUNDARY CONDITIONS

`coefa(nfabor,*)` [ra]: Boundary conditions: see note 2.

`coefb(nfabor,*)` [ra]: Boundary conditions: see note 2.

`iclrtp(nvarmx,2)` [ia]: For each variable `ivar` ($1 \leqslant \text{ivar} \leqslant \text{nvar} \leqslant \text{nvarmx}$), rank in `coefa` and `coefb` of the boundary conditions. See note 2.

`icoef` [i]: Rank in `iclrtp` of the rank in `coefa` and `coefb` of the "standard" boundary conditions. See note 2.

`icoeff` [i]: Rank in `iclrtp` of the rank in `coefa` and `coefb` of the "flow" type boundary conditions, reserved for developers. See note 2.

`ifmfbr(nfabor)` [ia]: Family number of the boundary faces. See note 1.

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

[23] As for the geometrical variables, some variables may be accessed to directly in sections of the unidimensional macro-arrays IA (for the integers) and RA (for the real numbers) which are present as arguments in every subroutine (apart from a few ones of very low level). The number of the first position of these sections in IA and RA is indicated by an integer stored in a "common". These integers are called "pointers"

`iprfml(nfml,nprfml)` [ia]: Properties of the families of referenced entities. See note 1.

`iisymp` [i]: Integer giving the rank in `ia` of the first element of the section allowing to mark out the "wall" (`itypfb=iparoi` or `iparug`) or "symmetry" (`itypfb=isymet`) boundary faces in order to prevent the mass flow (these faces are impermeable). For instance, for the phase `iphas`, if the face `ifac` is a wall or symmetry face, `ia(iismph+ifac-1)=0` (with `iismph=iisymp+nfabor*(iphas-1)`).
Otherwise `ia(iisymp+ifac-1)=1`.
In some subroutines, an array called `isympa(nfabor)` allows to simplify the coding with `isympa(ifac)=ia(iismph+ifac-1)`.

`iitrif` [i]: In `ia`, pointer to `itrifb`.

`iitypf` [i]: In `ia`, pointer to `itypfb`.

`itrifb(nfabor,nphas)` [ia]: Indirection array allowing to sort the boundary faces according to their boundary condition type `itypfb`.

`itypfb(nfabor,nphas)` [ia]: Boundary condition type at the boundary face `ifac` for the phase `iphas` (see user subroutine `usclim`).

`iuetbo` [i]: In `ra`, pointer to `uetbor`, used to store the friction velocity at the wall, in the case of a LES calculation with van Driest-wall damping.

Distance to the wall

`iifapa(nphsmx)` [ia]: For each phase, the pointer in `ia` which marks out the number of the wall face(type `itypfb=iparoi` or `iparug`) which is closest to the center of a given volume when necessary ($R_{ij}-\varepsilon$ with wall echo, LES with van Driest-wall damping, or SST $k-\omega$ turbulence model) and when `icdpar=2`. The number of the wall face (for the phase `iphas`) which is the closest to the center of the cell `iel` is therefore `ia(iifapa(iphas)+iel-1)`. This calculation method is not compatible with parallelism and periodicity.

`idipar` [i]: For each phase, pointer in `ra` to the section allowing to mark out the distance between the center of a given volume and the closest wall, when it is necessary ($R_{ij}-\varepsilon$ with wall echo, LES with van Driest-wall damping, or SST $k-\omega$ turbulence model) and when `icdpar=1`. The distance between the center of the cell `iel` and the closest wall is therefore `ra(idipar+iel-1)`.

`iyppar` [i]: For each phase, pointer in `ra` to the section allowing to mark out the adimensional distance y^+ between a given volume and the closest wall, when it is necessary (LES with van Driest-wall damping) and when `icdpar=1`. The adimensional distance y^+ between the center of the cell `iel` and the closest wall is therefore `ra(iyppar+iel-1)`.

Pressure drops

`iicepd(nphsmx)` [ia]: For each phase `iphas`, pointer in `ia` to `icepdc(ncepdc(iphas))`, array allowing to mark out the index-numbers of the `ncepdc(iphas)` cells in which a pressure drop is imposed.
The number of these cells is therefore given by `icepdc(ii)=ia(iicepd(iphas)+ii-1)`, with $1 \leqslant$ `ii` $\leqslant$ `ncepdc(iphas)`. See the user subroutine `uskpdc`.

`icepdc(ncepdc(iphas))` [ia]: Number of the `ncepdc(iphas)` cells in which a pressure drop is imposed. See `iicepd` and the user subroutine `uskpdc`.

`ickupd(nphsmx)` [ia]: For each phase `iphas`, pointer in `ra` to `ckupdc(ncepdc(iphas),6)`, array allowing to mark out the coefficients of the pressure drop tensor of the `ncepdc(iphas)` cells in which a pressure drop is imposed. See the user subroutine `uskpdc`.

`ckupdc(ncepdc(iphas),6)` [ra]: Value of the coefficients of the pressure drop tensor of the `ncepdc(iphas)` cells in which a pressure drop is imposed. See `ickpdc` and the user subroutine `uskpdc`.

`ncepdc(nphsmx)` [ia]: For each phase, number of cells in which a pressure drop is imposed. See the user subroutine `uskpdc`.

Mass sources

`iicesm(nphsmx)` [ia]: For each phase `iphas`, pointer in `ia` to `icetsm(ncetsm(iphas)`, array allowing to mark out the numbers of the `ncetsm(iphas)` cells in which a source of mass is imposed. The number of these cells is therefore given by `icetsm(ii)=ia(iicesm(iphas)+ii-1)`, with $1 \leqslant$ `ii` $\leqslant$ `ncetsm(iphas)`. See the user subroutine `ustsma`.

`iitpsm(nphsmx)` [ia]: For each phase `iphas`, pointer in `ia` to `itypsm` (type of mass source for each variable). See `itypsm` and the user subroutine `ustsma`.

`icetsm(ncetsm(iphas))` [ia]: Number of the `ncetsm(iphas)` cells in which a mass source term is imposed. See `iicesm` and the user subroutine `ustsma`.

`ismace(nphsmx)` [ia]: For each phase `iphas`, pointer in `ra` to `smacel` (mass source term and if necessary injection value for every variable apart from pressure). See `smacel` and the user subroutine `ustsma`.

`itypsm(ncetsm(iphas),nvar)` [ia]: Type of mass source term for each variable (0 for an injection at ambient value, 1 for an injection at imposed value). See the user subroutine `ustsma`.

`ncetsm(nphsmx)` [ia]: For each phase, number of cells with mass sources. See the user subroutine `ustsma`.

`smacel(ncetsm(iphas),nvar)` [ra]: Value of the mass source term for pressure. For the other variables, eventual imposed injection value. See the user subroutine `ustsma`.

Wall 1D thermal module

`nfpt1d` [i]: Number of boundary faces which are coupled with a wall 1D thermal module. See the user subroutine `uspt1d`.

`iifpt1` [i]: In `ia`, pointer to `ifpt1d(nfpt1d)`, array allowing to mark out the numbers of the `nfpt1d` boundary faces which are coupled with a wall 1D thermal module. The number of these boundary faces is therefore given by `ifpt1d(ii)=ia(iifpt1+ii-1)`, with $1 \leqslant$ `ii` $\leqslant$ `nfpt1d`. See the user subroutine `uspt1d`.

`inppt1` [i]: In `ia`, pointer to `nppt1d(nfpt1d)`, array giving the number of discretisation cells in the 1D wall for the `nfpt1d` boundary faces which are coupled with a wall 1D thermal module. The number of cells for these boundary faces is therefore given by `nppt1d(ii)=ia(inppt1+ii-1)`, with $1 \leqslant$ `ii` $\leqslant$ `nfpt1d`. See the user subroutine `uspt1d`.

`ieppt1` [i]: In `ia`, pointer to `eppt1d(nfpt1d)`, array giving the thickness of the 1D wall for the `nfpt1d` boundary faces which are coupled with a wall 1D thermal module. The wall thickness for these boundary faces is therefore given by `eppt1d(ii)=ia(ieppt1+ii-1)`, with $1 \leqslant$ `ii` $\leqslant$ `nfpt1d`. See the user subroutine `uspt1d`.

Others

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

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

`is2kw(nphsmx)` [ia]: For each phase `iphas`, pointer in `ra` to the section storing the square of the norm of the deformation rate tensor. In the cell `iel`, for the phase `iphas`, $S^2 = 2S_{ij}S_{ij}$ is therefore given by `ra(is2kw(iphas)+iel-1)`. This array is defined only when the phase `iphas` is treated with the SST $k-\omega$ turbulence model.

`idvukw(nphsmx)` [ia]: For each phase `iphas`, pointer in `ra` to the section storing the divergence of the velocity. In the cell `iel`, for the phase `iphas`, $div(\underline{u})$ is therefore given by `ra(idvukw(iphas)+iel-1)`. This array is defined only when the phase `iphas` is treated with the SST $k-\omega$ turbulence model (because in this case it may be calculated at the same time as S^2).

`ngrmmx` [i]: upper limit of the number of grid levels in the case of a multigrid solving (see `ngrmax`).

`ia(longia)` [ia]: Integer work array.

`ra(longra)` [ra]: Real work array.

NOTE: BOUNDARY CONDITIONS

The boundary conditions in *Code_Saturne* boil down to determine a value for the current variable ϕ at the boundary faces, that is to say ϕ_f, value expressed as a function of $\phi_{I'}$, value of ϕ in I', projection of the center of the adjacent cell on the straight line perpendicular to the boundary face and crossing its center: $\phi_f = A_{\phi,f} + B_{\phi,f}\phi_{I'}$.
For a face `ifac`, the pair of coefficients $A_{\phi,f}$, $B_{\phi,f}$ is stored in `coefa(ifac,iclvar)` and `coefb(ifac,iclvar)`, where the integer `iclvar=iclrtp(ivar,ijcl)` determines the rank in `coefa` and `coefb` of the set of boundary conditions of the variable `ivar`.
The second index of the array `iclrtp` allows to have several sets of boundary conditions for each variable. The "standard" boundary conditions are determined by `ijcl=icoef`, where `icoef` is a parameter which is fixed automatically by the code, and can be accessed to in the "common" file `numvar.h`. More specificic or advanced boundary conditions can be accessed to with `ijcl=icoeff`.
In practice, for a variable `ivar` whose value $\phi_{I'}$ in a boundary cell is known, the value at the corresponding boundary face `ifac` is:
ϕ_f=`coefa(ifac,iclvar)+coefb(ifac,iclvar)` $\phi_{I'}$ with `iclvar=iclrtp(ivar,icoef)`

5.5 User arrays

The code allows to define two user arrays, one integer array and one real array. The default size of these arrays is zero, and may be changed in `usini1`. The two arrays are then passed as arguments in every user subroutine of the code. For instance, a local variable calculated during the determination of the physical properties (user subroutine `usphyv`) may be stored in these arrays and sent to the post-processor at the end of the time step (user subroutine `usvpst`).

`nituse` [i]: Size of the user integer array.

`nrtuse` [i]: Size of the user real array.

`ituser(nituse)` [ia]: User integer array.

`rtuser(nrtuse)` [ra]: User real array.

5.6 Developer arrays

The code allows to define two developer arrays (similar to the user arrays `ituser` and `rtuser`), one integer array a one real array. The default size of these arrays is zero, and may be changed in `ustbus`. The two arrays are then passed as arguments in the rest of the code. They are designed to be used during the transitory development phases, in order to ease the tests (transfer of pieces of informations without consequence on the arguments of the subroutines).

`nideve` [i]: Size of the developer integer array.

`nrdeve` [i]: Size of the developer real array.

`idevel(nideve)` [ia]: Complementary integer array, used during development and test phases.

`rdevel(nrdeve)` [ra]: Complementary real array, used during development and test phases.

5.7 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 neigboring 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 neighboring cells, as the values in the "halo" can not be computed correctly (since the halo does not have access to all its neighbors), so halo values must be updated by copying values from the corresponding cells on the neighboring 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 neighborhood. Most updates need only operate on the standard halo, requiring less data communication than those on the extended halos.

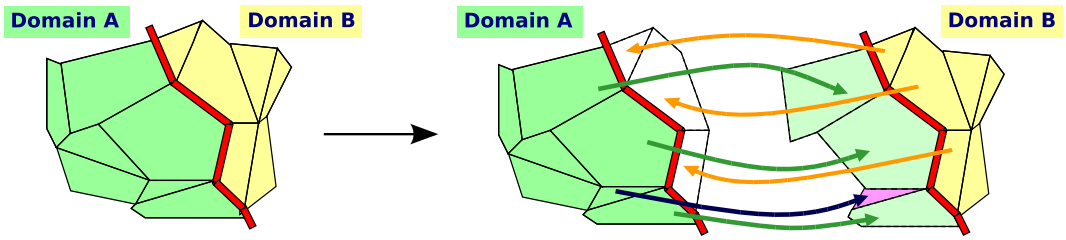


Figure 4: Parallel domain partitioning: halos

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 appying the rotation transformation. Ghost cells may be parallel, periodic, or both. The example of a pump combining parallelism and periodicity is given figure 5. In this example, all periodic boundaries match with boundaries on the same domain, so halos are either parallel or periodic.

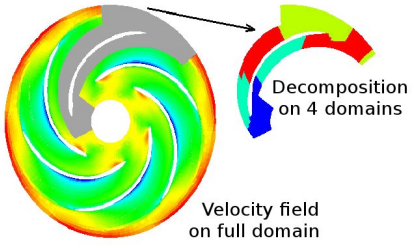


Figure 5: Combined parallelism and periodicity

Activation

Parallelism and periodicity are activated by means of the launch script in the standard cases:

- On clusters with PBS batch systems, the launching of a parallel run requires to complete the `PBS` batch cards located in the beginning of `runcase`, and particularly to set the number of physical nodes (`nodes`) and the number of physical processors per node (`ppn`) wanted. This can be done through the Graphical Interface or by editing the `runcase` file directly. The number of processors

used for the calculation will then be set automatically to the number of processors reserved and the variable `NUMBER_OF_PROCESSORS` can be left empty (see also §2.7).

- On clusters with LSF batch systems (like the CCRT machines), the launching of a parallel run requires to complete the `LSF` batch cards located in the beginning of `runcase`, and particularly to set the number of processors (`#BSUB -n`) wanted and the limit CPU time (`#BSUB -W`). As for now, this can only be done by editing the `runcase` file directly. The number of processors used for the calculation will then be set automatically to the number of processors reserved and the variable `NUMBER_OF_PROCESSORS` can be left empty (see also §2.7).
- On clusters with other 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.
- Although on batch systems the `NUMBER_OF_PROCESSORS` variable in the script (indicating the number of processors used for the calculation) is filled automatically to the number of processors reserved, the user can still choose to specify another value for it. This might only happen in very specific conditions and is not advised, as it will probably not be compatible with the batch system. Indeed, batch systems forbid to launch a calculation on more processors than the number of processors reserved, and some batch systems also forbid to launch a calculation on less processors than the number of processors reserved (automatic timeout on the idle processors that will stop the whole calculation).
- Periodicity is activated through the Graphical Interface or by completing the `COMMAND_PERIO` of the launch script `runcase`. The transformation allowing to pass from a boundary to the other one must be defined (the direction does not matter) and the set of periodic faces should be (optional but strongly advised) marked out (for instance by means of a color).
- Periodicity is compatible with parallelism.
- Periodicity can also work when the periodic boundaries are meshed differently (periodicity of non-conforming faces), *apart* from the case of a 180 degree rotation periodicity with faces coupled on the rotation axis.
- A parallel calculation may be stopped in the same manner as a sequential one using a `ficstp` file (see praragraph 2.2.4).
- The standard pieces 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 notice 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 conventionnal subroutines like `usini1` or `usclim` usually do not cause any problem, some kind of developments are more complicated. The most usual cases are dealt with below.
Examples are given in the subroutine `usproj`.

- **Access to information related to neighboring cells in parallel and periodic cases**.
  When periodicity or parallelism are brought into use, some cells of the mesh become physically distant from their neighbors. Concerning parallelism, the calculation domain is split and distributed between the processors: a cell located at the "boundary" of a given processor may have neighbors on different processors.
  In the same way, in case of periodicity, the neighboring cells of cells adjacent to a periodic face are generally distant.
  When data concerning neighboring 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 `grdcel`);
  - calculation of an internal face value from the values of A in the neighboring cells (use of `ifacel`).

  The variable A needs to be exchanged before these operations can be made: to allow it, the subroutines `parcom` and `percom` need to be called **in this order**.

- **Global operations in parallel mode**.
  In parallel mode, the user must pay attention during the realisation of 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 user subroutine `usproj`.
  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 overevaluing 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

Some particular points should be reminded:

- 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 LRR $R_{ij} - \varepsilon$ model. They would come from the way of taking into account the orthotropic viscosity (however, this term usually has a low influence).

5.8 Geometry and particule arrays related to Lagrangian modeling

In this section is given a non-exhaustive list of the main variables which may be seen by the user in the Lagrangian module. Most of them should not be modified by the user. They are calculated automatically from the data. However it may be useful to know their meaning.

These variables are listed in the alphabetical index in the end of this document.

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

Size of the Lagrangian arrays

`lndnod` [i]: Size of the array `icocel` concerning the cells → faces connectivity (the faces → nodes connectivity needs to be given to allow the construction of this connectivity. See note 3 of section 5.1).

`nbpmax` [i]: Maximum number of particles simultaneously acceptable in the calculation domain.

`nvp` [i]: Number of variables describing the particles for which a stochastic differential equation (SDE) is solved.

`nvls` [i]: Number of variables describing the supplementary user particles for which a SDE is solved.

`nvep` [i]: Number of real state variables describing the particles.

`nivep` [i]: Number of integer state variables describing the particles.

`ntersl` [i]: Number of source terms representing the backward coupling of the dispersed phase on the continuous phase.

`nvlsta` [i]: Number of volumetric statistical variables .

`nvlsts` [i]: Number of supplementary user volumetric statistical variables.

`nvisbr` [i]: Number of boundary statistical variables.

`nusbor` [i]: Number of supplementary user boundary statistical variables.

`nvgaus` [i]: Number of gaussian random variables.

Note: continuous eulerian phase number

The current version of Lagrangian module is planned to work with only one eulerian phase. This phase carries inclusions, and source terms of backward coupling are applied to it, if necessary. The number of this phase is stored in the variable `ilphas`. The standard value is `ilphas = 1`.

Lagrangian arrays

`icocel(lndnod)` [ia]: Cells *rightarrow* internal/boundary faces connectivity. The numbers of the boundary faces are marked out in `icocel` with a negative sign.

`itycel(ncelet+1)` [ia]: Array containing the position of the first face surrounding every cell in the array `icocel` (see subroutine `lagdeb` for more details).

`ettp(nbpmax,nvp)` [ra]: Variables forming the state vector related to the particles: either at the current stage if the Lagrangian scheme is a second-order, or at the current time step if the

scheme is a first-order. These variables are marked out by "pointers" whose value can vary between 1 and `nvp`:

→ `jmp`: particle mass

→ `jdp`: particle diameter

→ `jxp`, `jyp`, `jzp`: particle coordinates

→ `jup`, `jvp`, `jwp`: particle velocity components

→ `juf`, `jvf`, `jwf`: locally undisturbed fluid flow velocity components

→ `jtp`, `jtf`: particle and locally undisturbed fluid flow temperature (°C)

→ `jcp`: particle specific heat

→ `jhp`: coal particle temperature (°C)

→ `jmch`: mass of reactive coal of the coal particle

→ `jmck`: mass of coke of the coal particle

→ `jvls(ii)`: `ii`*th* supplementary user variable

`ettpa(nbpmax,nvp)` [ra]: Variables forming the state vector related to the particles: either at the previous stage if the Lagrangian scheme is a second-order, or at the previous time step if the Lagrangian scheme is a first-order.

`itepa(nbpmax,nivep)` [ia]: Integer state variables related to the particles. They are marked out by the following "pointers":

→ `jisor`: Number of the current cell containing the particle; this number is reactualised during the trajectography step

→ `jinch`: Number of the coal particle

`tepa(nbpmax,nvep)` [ra]: Real state variables related to the particles. They are marked out by the following "pointers":

→ `jrtsp`: particle residence time

→ `jrpoi`: particle statistic weight

→ `jrdck`: coal particle shrinking core diameter

→ `jrd0p`: coal particle initial diameter

→ `jrr0p`: coal particle initial density

`indep(nbpmax)` [ia]: Storage of the cell number of every particle at the beginning of a Lagrangian iteration; this data is not modified during the iteration.

`vitpar(nbpmax,3)` [ra]: At the beginning of the trajectography, `vitpar` contains the particle velocity vector components; the modifications of the particle velocity following every particle/boundary interaction are saved in this array; after the trajectography and backward coupling steps, `ettp` is updated with `vitpar`.

`vitflu(nbpmax,3)` [ra]: At the beginning of the trajectography, `vitflu` contains the locally undisturbed fluid flow velocity vector components; the modifications of the locally undisturbed fluid flow velocity following every particle/boundary interaction are saved in this array; after the trajectography and backward coupling steps, `ettp` is updated with `vitflu`.

`gradpr(ncelet,3)` [ra]: Pressure gradient of the continuous phase.

`gradvf(ncelet,9)` [ra]: Gradient of the continuous phase fluid velocity (useful if the complete model is activated: see `modcpl`).

`cpgd1(nbpmax)` [ra]: First devolatilisation term (light volatile matters) of the coal particles (useful in the case of backward coupling on the continuous phase).

`cpgd2(nbpmax)` [ra]: Second devolatilisation term (heavy volatile matters) of the coal particles (useful in the case of backward coupling on the continuous phase).

`cpght(nbpmax)` [ra]: Heterogeneous combustion term of the coal particles (useful in the case of backward coupling on the continuous phase).

`statis(ncelet,nvlsta)` [ra]: Volumetric statistics related to the dispersed phase; these statistics are the kind of results expected with the Lagrangian module. It is from these statistics that we obtain information concerning the particle cloud (the particle trajectories should only be observed on "pedagogical" account); they are marked out by the following "pointers":

→ `ilvx,ilvy,ilvz`: mean dispersed phase velocity

→ `ilvx2,ilvy2,ilvz2`: dispersed phase velocity standard deviation

→ `ilfv`: dispersed phase volumetric concentration

→ `ilpd`: sum of the statistical weights

→ `iltp`: dispersed phase temperature (°C)

→ `ildp`: dispersed phase mean diameter

→ `ilmp`: dispersed phase mean mass

→ `ilhp`: temperature of the coal particle cloud (°C)

→ `ilmch`: mass of reactive coal of the coal particle cloud

→ `ilmck`: mass of coke of the coal particle cloud

→ `ilmdk`: shrinking core diameter of the coal particle cloud

→ `ilvu(ii)`: `ii`*th* supplementary user volumetric statistics

`parbor(nfabor,nvisbr)` [ra]: Boundary statistics related the dispersed phase; after every particle/boundary interaction it is possible to save some data and to calculate averages; the boundary statistics are marked out by the following "pointers":

→ `inbr`: number of particle/boundary interactions

→ `iflm`: particle mass flow at the boundary faces

→ `iang`: mean interaction angle with the boundary faces (see example in `uslabo`)

→ `ivit`: mean interaction velocity with the boundary faces

→ `ienc`: mass of coal deposit at the walls

→ `iusb(ii)`: `ii`*th* supplementary user boundary statistics

`tslagr(ncelet,ntersl)` [ra]: Source terms corresponding to the backward coupling of the dispersed phase on the continuous phase. These source terms are marked out by the following "pointers":

→ `itsvx`, `itsvy`, `itsvz`: explicit source terms for the continuous phase velocity

→ `itsli`: implicit source term for the continuous phase velocity and for the turbulent energy if the $k-\varepsilon$ model is used

→ `itske`: explicit source term for the turbulent dissipation and the turbulent energy if the $k-\varepsilon$ turbulence model is used for the continuous phase

→ `itsr11`,... `itsr33`: source terms for the Reynolds stress and the turbulent dissipation if the $R_{ij}-\varepsilon$ turbulence model is used for the continuous phase

→ `itsmas`: mass source term

→ `itste`, `itsti`: explicit and implicit thermal source terms for the thermal scalar of the continuous phase

→ `itsmv1(icha)`, `itsmv2(icha)`: source terms respectively for the light and heavy volatile matters

→ `itsco`: source term for the carbon released during heterogeneous combustion

→ `itsf`: source term for the air variance (not used at the present time)

`croule(ncelet)` [ra]: Importance function for the technique of variance reduction (cloning/fusion of particles).

`vagaus(nbpmax,nvgaus)` [ra]: Vectors of gaussian random variables.

`auxl(nbpmax,3)` [ra]: Auxiliary work array.

5.9 Variables saved to allow calculation restarts

The directory `RESTART*` contains:

- `suiava`: main restart file,
- `suiavx`: auxiliary restart file (see `ileaux`, `iecaux`),
- `rayava`: restart file for the radiation module,
- `lagava`: main restart file for the Lagrangian module,
- `lasava`: auxiliary restart file for the Lagrangian module (mainly for the statistics),
- `t1dava`: restart file for the 1D wall thermal module,
- `vorava`: 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 scalars. Its content is sufficient for a calculation restart, but the complete continuity of the solution at restart is not ensured[24].

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

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[25]). 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 subgrid scale viscosity, specific heat, scalar diffusivities); for the Joule effect, the specific heat is stored automatically (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,
- 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`, `timpat`, `x20` (in case of coupling with the Lagrangian module, `iencp` and `x20` are not saved),
- for the electric module: the tuned potential difference `dpot` and, for the electric arc module, the tuning coefficient `coejou` (when the boundary conditions are tuned), the Joule source term for the enthalpy (with 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[26](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

[25] imposing a periodicity changes boundary faces into internal faces
[26] such a relaxation only makes sense for a stationary calculation

are stored in the restart file only when they are extrapolated in time, or with the Joule effect for the specific heat).

Apart from `vorava` which has a different structure and is always in text format, all the restart files are binary files. Nonetheless, they may be dumped by the `cs_io_dump` tool provided with the Preprocessor.

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 `suiava` file, whatever the number of processors used. The data in the file are written according to the initial full domain index-numbers for the cells, faces and nodes. This allows in particular to continue with p processors a calculation begun with n processors, or to make the restart files independent of any vectorial renumbering that may be carried out in each domain.

On the other hand, if the numbering of the initial full domain mesh is modified, the restart files will not be compatible. This may be the case if the mesh is composed of different elements that are pasted by the Preprocessor module and the order of the different elements has been changed in the Preprocessor command line between two calculations.

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[27].

NOTE: when meshes are pasted by the Preprocessor module with potential hanging nodes, two nodes closer than a certain (small) tolerance will be merged. Hence, due to numerical round-up errors, two different machines may yield different results. This might change the number of faces in the global domain[28] *and make restart files incompatible. Should that problem arise when making a calculation restart on a different architecture, the solution is to discard the* ***`suiavx`*** *file and use only the* ***`suiava`*** *file.*

6 User subroutines

6.1 Preliminary comments

The user can run the calculations with or without an interface, with or without the user subroutine. Without interface, some user subroutines are needed. With interface, all the user subroutines are optional.

The parameters can be read in the interface and then in the user subroutine. In the case that a parameter is specified in the interface and in the user subroutine, it is the value in the user subroutine that is taken into acount. It is for that reason that all the examples of user subroutines are placed in the `REFERENCE` directory by the case preparer `cs create`.

6.2 Using selection criteria in user subroutines

In order to use selection criteria (cf. §2.9) 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. `usclim`, `uscpcl`, `usray2`, `uslag2`,...),
- volumic initialization (cf. `usiniv`,...),
- head-loss region (cf. `uskpdc`),
- source terms region (cf. `ustsns`, `ustssc`),
- advanced post-processing (cf. `usdpst`), `usproj`, ...),

[27] when uncertain, the user can check the saved copy of the launch script in the `RESU` directory, or the head of the `listpre` file, which repeats the command line passed to the Preprocessor module

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

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 from 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 `usclim` 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)`.

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 adpated 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 lenght `lngnom` of the type `integer`) may be used.

6.3 Initialisation of the main key words: `usini1`

Subroutine only called during calculation initialisation.

This subroutine is used to indicate the value of different calculation basic parameters: constant and uniform physical values, parameters of numerical schemes, input-output management ...

In the case of a calculation launched using the interface, it is only used to modify high-level parameters which can not be managed by the interface. In the case of a code utilisation without interface, this subroutine is compulsory and all the headings must be completed.

For more details about the different parameters, please refer to the key word list (§7).

`usini1.f90` is in fact a gouping of 6 sperate subroutines: `usipph`, `usinsc`, `usipsc`, `usipgl`, `usipsu`and `usipes`. Each one controls the management of various specific parameters. The key words that dont' feature in the supplied example can be provided by the user in `SRC/REFERENCE/base`; in this case, understanding of the comments is needed to add the key words in the appropriate subroutine (the most widely used is `iphas`, it will assure that the value has been well defined). The modifiable parameters in each of the subroutines of `usini1.f90` are:

- `usipph`: `iturb` and `icp` (don't modify these parameters anywhere else)
- `usinsc`: `nscaus` (don't modify these parameters anywhere else)
- `usipsc`: `iscavr` and `ivisls` (don't modify these parameters anywhere else)
- `usipgl`: `idtvar`, `ipucou`, `iphydr` and the parameters related to the error estimators(don't modify these parameters anywhere else).
- `usipsu`: physical parameters of the calculation (thermal scalar, physical properties,...), numeric parameters (time steps, number of iterations, ...), definition of the time averages.
- `usipes`: post processing output parameters (periodicity, variable names, probe positions,...)

For more details of the different parameters, see the list of key words (§7). The names of the key words can also be seen in the helps sections of the interface.

NOTES

• Determined in the list of `nscaus` user scalars, representing the mean square fluctuations of another whilst informing the `iscavr` array (warning, this was not the case in version 1.0). For the other scalars, `iscavr` does not need to be completed (by default, `iscavr(ii)`$\leqslant$0). For instance, if the scalar `jj` represents the average of the square of the fluctuations of the scalar `kk`, the user must indicate `iscavr(jj)=kk` (1$\leqslant$`kk`$\leqslant$`nscaus`).

• When using the interface, only the supplementary parameters (which can not be defined in the interface) should appear in `usini1`. To spare the user the necessity to delete the other parameters appearing as examples in the subroutine, the utility program `cs create` comments automatically all the example lines of `usini1` with a code `!ex`. The user needs then only to uncomment the lines which are useful in his case. This function of `cs create` can be inactivated with the option `--nogui` (useful if the user knows that he will not use the interface).

6.4 Management of boundary conditions: `usclim`

Subroutine called every time step.

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

When the interface is used, `usclim` is used to define complex boundary conditions (input profiles, conditions varying in time, ...) which could not be specified by means of the interface, and only these need to be defined. In the case of a calculation launched without the interface, all the boundary conditions must appear in `usclim`.
`usclim` is essentially constituted of loops on boundary face subsets. Several sequences of `call getfbr ('criterion', nlelt, lstelt)` (cf. §6.2) allow to differentiate the boundary faces according 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 differentiate 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 the wall conditions, see `clptur`) (to access to this document on a workstation, use `cs info --guide theory`).

From the user point of view, the boundary conditions are totally determined by three arrays[29]: `itypfb(nfabor,nphas)`, `icodcl(nfabor,nvar)` and `rcodcl(nfabor,nvar,3)`.

- `itypfb(ifac,iphas)` defines the type of the face `ifac` (input, wall, ...) for the phase `iphas`.
- `icodcl(ifac,ivar)` defines the type of boundary condition for the variable `ivar` at the face `ifac` (Dirichlet, flux ...).
- `rcodcl(ifac,ivar,.)` gives the numerical values associated with the type of boundary condition (value of the Dirichlet, of the flux ...).

In the case of standard boundary conditions (see §6.4.1), it is enough to complete `itypfb(ifac,iphas)` and some boxes of the array `rcodcl`, the array `icodcl` and most of the boxes of `rcodcl` are completed automatically. For non-standard boundary conditions (see §6.4.2), the arrays `icodcl` and `rcodcl` must be totally completed.

6.4.1 Coding of standard boundary conditions

The standard values taken by the indicator `itypfb` are: `ientre`, `iparoi`, `iparug`, `isymet`, `isolib` and `iindef`.

- If `itypfb=ientre`: inlet face.

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

- If `itypfb=iparoi`: smooth 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(iphas)`, `iv(iphas)` or `iw(iphas)`). The initial value of `rcodcl(ifac,ivar,1)` is zero for the three velocity components (and therefore needs to 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.

  → Concerning the 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

[29] except with Lagrangian

⇝ Imposed flux at the wall. The user must write
`icodcl(ifac,ivar)`=3
`rcodcl(ifac,ivar,3)`=flux imposed value (for the flux definition according to the variable, the user may 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=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(iphas)`, `iv(iphas)` or `iw(iphas)`). The initial value of `rcodcl(ifac,ivar,1)` is zero for the three velocity components (and therefore needs to 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 `rcdocl(ifac,iu(iphas),3)`. The values of `rcdocl(ifac,iv(iphas),3)` and `rcdocl(ifac,iw(iphas),3)` are 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
  `rcodcl(ifac,ivar,3)`=thermal roughness value

  ⇝ Imposed flux at the wall. The user must write
  `icodcl(ifac,ivar)`=3
  `rcodcl(ifac,ivar,3)`=flux imposed value (for the flux definition according to the variable, the user may 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 write 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 in order to have $\frac{d}{dn}\left(\frac{dP}{d\tau}\right) = 0$. The pressure is given the value P_0 at the first face `isolib` met. The pressure drop is always linked to just one face, even if there are several outlets.

  → If the mass flow is coming in, the "infinite" velocity is retained and 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 for the velocity, the scalars and the turbulent quantities.

  → Nothing to write in `icodcl` or `rcodcl` for pressure or velocity. An optional Dirichlet condition can be specified for the scalars and turbulent quantities.

- If `itypfb=iindef`: non-defined type face (non-standard case)

  → The coding is done by completing every array `rcodcl` and `icodcl` (see §6.4.2).

NOTES

- Whatever the value of the indicator `itypfb(ifac,iphas)`, if the array `icodcl(ifac,ivar)` is

modified by the user (*i.e.* filled in by a value different from zero), the code will not use the default conditions for the variable `ivar` at the face `ifac`, but will take into account the values of `icodcl` and `rcodcl` given by the user (these arrays must then be totally completed, like in the non-standard case). For instance, for a symmetry face at which the scalar 1 is given a Dirichlet condition equal to 23.8 (with an infinite exchange coefficient):

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

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

• The user may define new types of wall faces. He only needs to choose a value N and to specify completely the boundary conditions corresponding to this new wall face type (see §6.4.2). He must then specify `itypfb(ifac,iphas)`$=N$. The value of N must be between 1 and `ntypmx` (maximum number of boundary face types), and of course different from the values `ientre`, `iparoi`, `iparug`, `isymet`, `isolib` and `iindef` (the value of these variables is given in the file `paramx.h`). This allows to easily isolate some boundary faces, in order to calculate balances.

6.4.2 Coding of non-standard boundary conditions

In the case of a face not corresponding to a standard type, the user must complete all of the arrays `itypfb`, `icodcl` and `rcodcl`. `itypfb(ifac,iphas)` is then worth `iindef` or another value defined by the user (see note in the end of paragraph 6.4.1). The arrays `icodcl` and `rcodcl` must be completed 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 a perfect transfer between the outside and the fluid (default case).
  - → `rcodcl(ifac,ivar,3)` is not used.
  - → `rcodcl(ifac,ivar,1)` is expressed in the unit 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 the temperature fluctuations
    - ⇝ $J^2.kg^{-2}$ for the enthalpy fluctuations
  - → `rcodcl(ifac,ivar,2)` is expressed in the following unit (defined so that by multiplying the exchange coefficient and the variable, the obtained flux has the same unit as the flux defined below for `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)=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:

⇝ $-C_p(\frac{\lambda_T}{C_p}+\frac{\mu_t}{\sigma_T})\underline{\text{grad}}\ T\cdot\underline{n}$ in the case of a temperature (in W/m^2).
$-(\lambda_h+\frac{\mu_t}{\sigma_h})\underline{\text{grad}}\ h\cdot\underline{n}$ in the case of an enthalpy (in W/m^2).
$-(\lambda_\varphi+\frac{\mu_t}{\sigma_\varphi})\underline{\text{grad}}\ \varphi\cdot\underline{n}$ in the case of another scalar φ (in $kg.m^{-2}.s^{-1}.[\varphi]$, where $[\varphi]$ is the unit of φ).

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

⇝ $-(\mu+\mu_t)\underline{\text{grad}}\ U_i\cdot\underline{n}$ in the case of a velocity component (in $kg.m^{-1}.s^{-2}$).

⇝ $-\mu\underline{\text{grad}}\ R_{ij}\cdot\underline{n}$ in the case of 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 the velocity components ($\underline{U}\cdot\underline{n}=0$) and the R_{ij} tensor components (for the other variables, a zero-flux condition type is generally used).

- If `icodcl(ifac,ivar)`=5: friction condition, for the smooth-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 moving wall, the three components of the slipping velocity are given by (`rcodcl(ifac,iu(iphas),1)`, `rcodcl(ifac,iv(iphas),1)`, and `rcodcl(ifac,iw(iphas),1)`).
  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.

  ⇝ For the other scalars, the condition `icodcl`=5 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].

- If `icodcl(ifac,ivar)`=6: friction condition, for the rough-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 moving wall, the three components of the slipping velocity are given by (`rcodcl(ifac,iu(iphas),1)`, `rcodcl(ifac,iv(iphas),1)`, and `rcodcl(ifac,iw(iphas),1)`).
  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 height is given by `rcodcl(ifac,iu(iphas),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 can only be applied to the velocity components.
  If the mass flow at the face is going out, this condition is equivalent to a zero-flux condition.
  If the mass flow at the face is coming in, the value zero is imposed to the velocity at the face (but not to the mass flow).
  `rcodcl` is not used.

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 three components of the velocity must belong to the same type. The same must be true for the different components of the R_{ij} tensor.
• If the boundary conditions for the velocity belong to the "slipping" 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 conditions for the turbulent variables must belong to the "friction" type, too.
• If the boundary condition for a scalar belongs to the "friction" type, the boundary condition for the velocity must belong to the "friction" type, too.

In case of error, if post-processing output is activated (which is the default), a special error output to the same mesh format occurs, so as 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. In order to ease this kind of search, an array of sorted faces is automatically completed (and updated at every time step) for each phase `iphas`: `itrifb(nfabor,iphas)`.
`ifac=itrifb(i,iphas)` is the number of the i$^{\text{th}}$ face of type 1.
`ifac=itrifb(i+n,iphas)` is the number of the i$^{\text{th}}$ face de type 2, if there are n faces of type 1.
... etc.

Two auxiliary arrays of size `ntypmx` are also defined.
`idebty(ityp,iphas)` is the number of the first box corresponding to the faces of type `ityp` in the array `itrifb`.
`ifinty(ityp,iphas)` is the number of the last box corresponding to the faces of type `ityp` in the array `itrifb`.

Therefore, a number `ifac0` between `idebty(ityp,iphas)` and `ifinty(ityp,iphas)` corresponds to each face of type `ityp=itypfb(ifac,iphas)`, so that `ifac=itrifb(ifac0,iphas)`.

If there is no face of type `ityp`, the code imposes
`ifinty(ityp,iphas)=idebty(ityp,iphas)-1`,
which allows to bypass, for all the missing `ityp`, the loops like
`do ii=idebty(ityp,iphas),ifinty(ityp,iphas)`.

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

6.5 Management of the boundary conditions with LES: `usvort`

This subroutine allows to generate the non-stationary 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 parametres of this method are given.

subroutine called for each time step

To allow the application of the vortex method, an indicator must be informed of the method in the

user subroutine `usini1`(ivrtex=1)

The subroutine `usvort` contains 3 seperate 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 independant 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 rectangluar 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 interation 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 3.

  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`) which 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 6:

  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 `usclim` for fully developed flow in a pipe.

  The case independant 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. They 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 co-efficients ($L_T = (5\nu\frac{k}{\varepsilon})^{\frac{1}{2}}$, $L_K = 200(\frac{\nu^3}{\varepsilon})^{\frac{1}{4}}$).
* `idepvo` gives the vortex displacement method in the 2D inlet plane (the vortex method is a langrangian method in which the eddy centers 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 principle directions, $r\text{ud}\Delta t$ where Δt is the time step of the calcualtion). 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 flucutations 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 `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 were 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`.

6.6 Management of the variable physical properties: `usphyv`

Subroutine called every time step.

If necessary, all the variation laws related to the fluid physical parameters (density, viscosity, thermal diffusivity, ...) are written in this subroutine.

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 one wishes to impose a density or viscosity variable in `usphyv`, it can be done either in the interface or in `usini1`(`irovar(iphas)`=1, `ivivar(iphas)`=1).
- In order to impose a physical property (ρ, μ, λ, C_p)[30] a reference value must be inputted to the interface or in `usini1`(in particular for ρ, the pressure will contain 1 part as $\rho_0 gz$)
- By default, the C_p coefficient of the phase `iphas` and the diffusivity of the scalars `iscal` (λ/C_p for the temperature) are considered as constant in time and uniform in space, with the values `cp0(iphas)` and `visls0(iscal)` specified in the interface or in `usini1`.
  To give a variable value to C_p, the user must specify it in the interface or give the value 1 to `icp(iphas)` in `usini1`, and complete for each cell `iel` the array `propce(iel,ipccp)` in `usphyv`. Completing the array `propce(iel,ipccp)` while `icp(iphas)=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 or give the value 1 to `ivisls(iscal)` in `usini1`, and complete for each cell `iel` the array `propce(iel,ipcvsl)` in `usphyv`. Completing `propce(iel,ipcvsl)` while `ivisls(iscal)`=0 induces memory overwriting problems and produces wrong results.

  Example: If the scalars 1 and 3 have a constant and uniform viscosity, and if the scalars 2 and 4 have a variable viscosity, the following values must be imposed in `usini1`:
  `ivisls(1)`=0, `ivisls(2)`=1, `ivisls(3)`=0 and `ivisls(4)`=1.
  The indicators `ivisls(2)` and `ivisls(4)` are then modified automatically by the code in order to reflect the rank corresponding to the diffusivity of each scalar in the list of physical properties[31]. The arrays `propce(iel,ipcvsl)` in `usphyv` must then be completed with `ipcvsl=ipproc(ivisls(2))` and `ipcvsl=ipproc(ivisls(4))`.

  Note: The indicators `ivisls` must not be completed in the case of 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.

6.7 Non-default variables initialisation: `usiniv`

Subroutine only called during calculation initialisation.

At the calculation beginning, the variables are initialised automatically by the code. Velocities and scalars are set to the value 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 in $k-\varepsilon$, $R_{ij}-\varepsilon$, v2f or $k-\omega$ model:
`rtp(iel,ikiph) = 1.5*(0.02*uref(iphas))**2` (in $R_{ij}-\varepsilon$, $R_{ij}=\frac{2}{3}k\delta_{ij}$)
For ε in $k-\varepsilon$, $R_{ij}-\varepsilon$ or v2f model:
`rtp(iel,ieiph) = rtp(iel,ikiph)**1.5*cmu/almax(iphas)`
For ω in $k-\omega$ model:
`rtp(iel,iomgip) = rtp(iel,ikiph)**0.5/almax(iphas)`

[30] except for some specific physics
[31] they are no longer worth 1 but stay positive so that `ivisls`>0 is synonymous with variable property

For φ and $\overline{f}$ in v2f model:
`rtp(iel,iphiph) = 2/3`
`rtp(iel,ifbiph) = 0`

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

This subroutine allows also to make 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.

NOTE: VALUE OF THE TIME STEP

- In the case of a calculation 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 `usini1`, the calculation being whether a restart (`isuite`=1) or not (`isuite`=0).
- In the case of a calculation with non-constant time step (`idtvar`=1 or 2) which is not a calculation restart (`isuite`=0), the value of `dtref` given in the parametric file of the interface or in `usini1` is used to initialise the time step.
- In the case of a calculation with non-constant time step (`idtvar`=1 or 2) which is a restart (`isuite`=1) 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 `usini1` is used to initialise the time step.
- In the case of a calculation with non-constant time step (`idtvar`=1 or 2) which is a restart (`isuite`=1) 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 `usini1` is not used.

It follows that for a calculation with non-constant time step (`idtvar`=1 or 2) which is a restart (`isuite`=1) of a calculation in which `idtvar` had the same value, `dtref` does not allow to modify the time step. The user subroutine `usiniv` allows to modify 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 centers whatever the chosen time step type).

WARNING: to initialise the variables in the framework of a specific physics module (`nscapp.gt.0`) *one of the subroutines* ***`usebui`****,* ***`usd3pi`****,* ***`uslwci`*** *or* ***`uscpiv`*** *should be used instead of* ***`usiniv`*** *(depending on the activated module).*

6.8 Non-standard management of the chronological record files: `ushist`

Subroutine called every time step

The interface and the subroutine `usini1` allow to manage the "automatic" chronological record files in an autonomous way: position of the probes, printing frequency and concerned variables. The results are written in a different file for each variable. These files are written in *xmgrace or* gnuplot format and contain the profiles corresponding to every probe. This type of output format may not be well adapted if, for instance, the number of probes is too high. The subroutine `ushist` allows then to personalise the output format of the chronological record files. The version given as example in the directory works as follows:

- Positionning of the probes (only at the first passage): the index `ii` varies between 1 and the number of probes. The coordinates `xx`, `yy` and `zz` of each probe are given. The subroutine `findpt` gives then the number `icapt(ii)` of the cell center which is the closest to the defined probe.

- Opening of the output files (only at the first pass): in the version given as example, the program opens a different file for all the `nvar` variables. `ficush(j)` contains the name of the J^{th} file and `impush(j)` its unit number (`impush` is initialised by default so that the user has at his disposal specific unit numbers and does not run the risk to overwrite an already open file).
- Writing to the files: in the version given as example, the program writes the time step number, the physical time step (based on the standard time step in the case of a variable time step) and the value of the selected variable at the different probes.
- Closing of the files (only at the last time step).

WARNING: The use of `ushist` *neither erases nor replaces the parameters given in the interface or in* `usini1`*. Therefore, in the case of the use of* `ushist`*, and to avoid the creation of useless files, the user should set* `ncapt`=0 *in the interface or in* `usini1` *to deactivate the automatic production of chronological records.*
In addition, ***`ushist`*** generates supplementary result files. The user shoud remember to add in the launch script the necessary command to copy them in the directory `RESU` at the end of the calculation. The interface allows the specification of the name of the copied user results files. For the calculations without interface, the variable must be inputted in USER_OUTPUT_FILES in the launch script.

6.9 User source terms in Navier-Stokes: `ustsns`

Subroutine called every time step

This subroutine is used to add user source terms to the Navier-Stokes equations. For each phase `iphas`, it is called three times every time step, once for each velocity component (`ivar` is successively worth `iu(iphas)`, `iv(iphas)` and `iw(iphas)`). At each passage, the user must complete if necessary the arrays `crvimp` and `crvexp` expressing respectively the implicit and explicit part of the source term. If no other source terms apart from `ivar=iu(iphas)` for example, are required, `crvimp` and `crvexp` must be read over and their 2 other components, `ivar=iv(ihpas)` and `ivar=iw(iphas)` must be cancelled.

Let us assume that the user source terms modify the equation of a variable φ in the following way:

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

φ is here a velocity component, but the examples are also valid 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 the subroutines `ustske`, `ustsri`, `ustsv2`, `ustskw` and `ustssc` is similar.

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

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

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

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

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

To make the "implicitation" effective, the source term decomposition between implicit and explicit parts will be done by the user who must make sure $\texttt{crvimp}_i=\Omega_iS_{impl,i}$ is always negative (otherwise the solved equation remains right, but there is no "implicitation").

WARNING: When the second-order in time with extrapolation of the source terms[32] *is activated, 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.*

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.
Let us consider an equation of the type:

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

We want 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:
$\texttt{crvimp}_i = \Omega_i\dfrac{dF}{d\varphi}(\varphi_i^{(n)})$
$\texttt{crvexp}_i = \Omega_i\left[F(\varphi_i^{(n)})-\dfrac{dF}{d\varphi}(\varphi_i^{(n)})\times\varphi_i^{(n)}\right]$

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

6.10 User source terms for k and ε: `ustske`

Subroutine called every time step, in $k-\varepsilon$ and in v2f.

This subroutine is used to add source terms to the transport equations related to the turbulent kinetics energy k and to the turbulent dissipation ε (for each phase `iphas`). This subroutine is called every time step, once for each phase (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 `ustsns`. The way of making implicit the resulting source terms is the same as the one presented in `ustsns`. For φ and $\bar{f}$ in v2f, see `ustsv2`, §6.12.

6.11 User source terms for R_{ij} and ε: `ustsri`

Subroutine called every time step, in $R_{ij}-\varepsilon$.

This subroutine is used to add source terms to the transport equations related to the Reynolds stress variables R_{ij} and to the turbulent dissipation ε (for each phase `iphas`). This subroutine is called 7 times every time step and for each phase (once for each Reynolds stress component and once for the dissipation). The user must provide the arrays `crvimp` and `crvexp` for the variable `ivar` (referring

[32] indicator `isno2t` for the velocity, `ISTO2T` for the turbulence and `isso2t` for the scalars

successively to `ir11(iphas)`, `ir22(iphas)`, `ir33(iphas)`, `ir12(iphas)`, `ir13(iphas)`, `ir23(iphas)` and `iep(iphas)`). These arrays are similar to the arrays `crvimp` and `crvexp` given for the velocity in the user subroutine `ustsns`. The method for impliciting the resulting source terms is the same as that presented in `ustsns`.

6.12 User source terms for φ and $\overline{f}$: `ustsv2`

Subroutine called every time step, in v2f.

This subroutine is used to add source terms to the transport equations related to the variables φ and $\overline{f}$ of the v2f φ-model (for each phase `iphas`). This subroutine is called twice every time step and for each phase (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(iphas)` and `ifb(iphas)`. Concerning φ, these arrays are similar to the arrays `crvimp` and `crvexp` given for the velocity in the user subroutine `ustsns`. Concerning $\overline{f}$, the equation is slightly different:

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

In finite volume formulation, the solved system is written:

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

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 `ustsns`.

6.13 User source terms for k and ω: `ustskw`

Subroutine called every time step, in $k-\omega$.

This subroutine is used to add source terms to the transport equations related to the turbulent kinetics energy k and to the specific dissipation rate ω (for each phase `iphas`). This subroutine is called every time step, once for each phase (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 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 `ustsns`. The way of impliciting the resulting source terms is the same as the one presented in `ustsns`.

6.14 User source terms for the user scalars: `ustssc`

Subroutine called every time step.

This subroutine is used to add source terms to the transport equations related to the user scalars (passive or not, average of the square of the fluctuations of a scalar, ...). In the same way as `ustsns`, this subroutine is called every time step, once for each user scalar. The user needs to 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 term to be applied (the arrays are initially at 0 at each inlet in the subroutine.)

6.15 Management of the pressure drops: `uskpdc`

Subroutine called every time step.

This subroutine is called three times every time step and for each phase `iphas`.

The tensor representing the pressure drops is supposed to be symmetric and positive.

- During the first call, all the cells are checked to know the number of cells in which a pressure drop is present for the phase `iphas`. This number is called `ncepdp` in `uskpdc` (and corresponds to `ncepdc(iphas)`). It is used to lay out the arrays related to the pressure drops. If there is no pressure 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`$^{\text{th}}$ cell containing pressure drops (for the current phase).
- During the third call, all the cells containing pressure drops (for the current phase) 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`, `k13`, `k23` with `k12`, `k13` and `k23` being zero if the tensor is diagonal.

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

6.16 Management of the mass sources: `ustsma`

Subroutine called every time step.

This subroutine is used to add a density source term in some cells of the domain. 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 a 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 (for each phase).

- During the first call, all the cells are checked to know the number of cells containing a mass source term for the current phase `iphas`. This number is called `ncesmp` in `ustsma` (and corresponds to `ncetsm(iphas)`). 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`$^{\text{th}}$ cell containing a mass source (for the current phase).
- 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`$^{\text{th}}$ 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(iphas)`
  - `(ieltsm,ipr(iphas))` is the value of the mass source term Γ, in $kg.m^{-3}.s^{-1}$.
  - `smacel(ieltsm,ivar)`, for `ivar` different from `ipr(iphas)`, is the value of φ_i for the variable `ivar` in the `ielstm`$^{\text{th}}$ cell containing a mass source.

  NOTES
  • If `itypsm(ieltsm,ivar)=0`, `smacel(ieltsm,ivar)` is not used.
  • If Γ=`smacel(ieltsm,ipr(iphas))`<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.

6.17 Thermal module in a 1D wall

subroutine called at every time step

This subroutine takes into account the affected thermal inertia by a wall. Some boundary faces are treated as a solid wall with a given thickness, on which the code resolves an undimensional 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. In 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 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 theres is only 1 phase (`nphas=1`) and that the thermal scalar is defined as (`iscalt`> 0).

WARNING: The 1D thermal module is developped assuming the thermal scalar as a temperature. If the thermal scalar is an enthalpy, the code calls the subroutine **`usthht`** *for each transfer of information 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.*

This procedure is called twice,on initialisation and again at each time step.

- The 1st call (initialisation) all the boundary faces that will be treated as a coupled wall are marked out. This figure is written noted as `nfkpt1d`. It applies dimension to the arrays in the thermal module. `nfkpt1d` will be at 0 if there are no coupled faces (it is in fact the default value, the remainer of the subroutine is not used in this case). The parameter `isuit1` also need to be defined, this indicates if the temperature of the wall must be initialised or written in the file (stored in the variable `filmt1`).
- The 2nd call (initialisation) again concern the wall faces, it completes the `ifpt1d` array of dimension `nfpt1d`. `ifpt1d(ifbt1d)` is the number `ifbt1d`$^{\text{th}}$ boundary faces coupled with the thermal module of a 1D wall. The directional parameters are then completed for a pseudo wall associated to each face
  - `nppt1d(nfpt1d)`: number of cells in the 1D mesh associated to the pseudo wall.
  - `eppt1d(nfpt1d)`: thickness of the pseudo wall.
  - `rgpt1d(nfpt1d)`: geometery of the pseudo wall mesh (refined as a fluid if `rgt1d` is smaller than 1)
  - `tppt1d(nfpt1d)`: initialisation temperature of the wall (uniform in thickness). In the course of the calculation, the array stores the temperature of the solid at the fluid/solid interface.

  Other than for re-reading a file (`ficmt1`), `tppt1d` is not used. `nppt1d`, `ifpt1d`, `rgpt1d` and `eppt1d` are compared to data from the follow-up file and they must be identical.

  WARNING: The test in ***`ifpt1d`*** *implicilty assumes that the array is completed in ascending order (i.e* ***`ifpt1d(ii)>ifpt1d(jj)`*** *if ii>jj. This will be the case if the coupled faces are defined starting from the unique loop on the boundary faces (as in the example). If this is not the case, contact the development team to short circuit the test.*
- The 3rd call (at each time step) is for the confirmation that all the arrays involving physical parameter and external boundary conditions have been completed.
  - `iclt1d(nfpt1d)`:Typical boundary condition at the external (pseudo) wall: Dirichlet condition (`iclt1d`=1) or flux condition (`iclt1d`=3)
  - `tept1d(nfpt1d)`: External temperature of the pseudo wall in the Dirichlet case.
  - `hept1d(nfpt1d)`: External coefficient of transfer in the pseudo wall under Dirichlet conditions(en $W.m^{-2}.K^{\cdot}$).
  - `fept1d(nfpt1d)`: External heat flux in the pseudo wall under the flux conditions(en $W.m^{-2}$,negative value for energy entering the wall)
  - `xlmt1d(nfpt1d)`: Conductivityλ of the wall uniform in thickness, (in $W.m^{-1}.K^{-1}$).
  - `rcpt1d(nfpt1d)`: Volumetric heat capacity ρC_p of the wall uniform in thickness in $J.m^{-3}.K^{-1}$)
  - `dtpt1d(nfpt1d)`: Physical time step ascociated with the solved 1D equation of the pseudo wall(which can be different from the time step in the calculation)

The 3rd call, done at each time step, allows the imposition of boundary conditions and physical values in time.

6.18 Initialization of the options of the variables related to the ale module: `usalin` and `usstr1`

Subroutine called at the start.

SUBROUTINE **`USALIN`**
This subroutine completes `usini1`.

`usalin` allows to set option for the ale module, and in particular to active the ale module

SUBROUTINE `USSTR1`

`usstr1` allows to specify for the structure module the following pieces of information:

- number of structure (`nbstru`).
- initial value of deplacement, velocity and acceleration (`xstr0`, `xstreq` and `vstr0`).

Below is a list of the different variables that might be modified:

- `nbstru`
  the number of structures
- `idfstr(i)`
  index of the structure, where I is the index of the face
- `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)`
  position of balance 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

6.19 Management of the boundary conditions of velocity mesh related to the ale module: `usalcl`

Subroutine called every time step.

SUBROUTINE `USALCL`

The use of `usalcl` is obligatory to run a calculation using the ale module just as it is in `usini1`. The way of using it is the same as the way of using `usclim` 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 boundary condition of velocity mesh for each variable are defined.

The main numerical variables are described below.

`ialtyb(nfabor)` [ia]: In the ale module, the user defines the velocity mesh from the colour of the boundary faces, or more generally from their properties (colours, groups, ...), from the boundary conditions defined in `usclim`, or even from their coordinates. To do so, the array `ialtyb(nfabor)` gives for each face `ifac` the velocity mesh boundary condition types marked out by the key words `ivimpo`, `igliss`, `ibfixe`

- If `ialtyb=ivimpo`: imposed velocity.

  → In the case where all the nodes of a face have a imposed displacement, it is not necessary to fill the tables with boundary conditions velocity mesh for this face, they 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 velocity mesh is taken like a tape speed under the boundary conditions of wall for the fluid, except if wall velocity was specified by the user in the interface or usclim (in which case it is this speed which is considered).

- if `ialtyb(nfac) = ibfixe`: fixed wall

  → the velocity is null.

- if `ialtyb(nfac) = igliss`: sliding wall

  → the tangential velocity is not used.

.

6.20 Management of the structure property: `usstr2`

Subroutine called every time step.

The use of `usstr2` is obligatory to run a calculation using the ale module with a structure module.

For each structure, the system that will be solved is:

$$M.x'' + C.x'' + K.(x - x_0 = 0 \tag{1}$$

where

- M is the mass stucture (`xmstru`).
- C is the dumping coefficient of the stucture (`xcstru`).
- K is the spring constant or force constant of the stucture (`xkstru`).
- x_0 is the initial position

Below is a list of the different variables that might be modified:

- `xmstru(i,j,k)`
  the mass stucture of the structure, where `i`,`j` is the array of mass structure and `k` the index of the structure.
- `xcstru(i,j,k)`
  dumping coefficient of the stucture, where `i`,`j` is the array of dumping coefficient and `k` the index of the structure.
- `xkstru(i,j,k)`
  spring constant of the stucture, where `i`,`j` is the array of spring constant and `k` the index of the structure.
- `forstr(i,k)`
  force vector of the stucture, where `i` is the force vector and `k` the index of the structure.

6.21 Modification of the turbulent viscosity: `usvist`

Subroutine called every time step.

This subroutine is used to modify the calculation of the turbulent viscosity of the phase `iphas`, *i.e.* μ_t in $kg.m^{-1}.s^{-1}$ (this piece of information, at the mesh cell centers, is conveyed by the variable `propce(iel,ipcvst)`, with `ipcvst = ipproc(ivisct(iphas))`). 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(iphas)`).
WARNING: The calculation of the turbulent viscosity being a particularly sensible stage, a wrong use of `usvist` *may seriously distort the results.*

6.22 Modification of the variable C of the dynamic LES model: `ussmag`

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

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

Let us first remind 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 [2], to which the user may refer 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 implicit filter (of width $\overline{\Delta}$) is called $\overline{a}$. We define:

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

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

$$\begin{array}{lcll} \mu_{\text{total}} & = & \mu + \mu_{\text{sub-grid}} & \text{if } \mu_{\text{sub-grid}} > 0 \\ & = & \mu & \text{otherwise} \\ \text{with } \mu_{\text{sub-grid}} & = & \rho C \overline{\Delta}^2 ||\overline{S}|| & \end{array} \tag{3}$$

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

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

In the case of the dynamic model (`iturb(iphas)=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 neighboring cells (this average uses the extended neighborhood 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.*

6.23 Temperature-enthalpy and enthalpy-temperature conversions: `usthht`

Subroutine optionally called.

This subroutine is used to encapsulate a simple enthalpy-temperature conversion law and its inverse. This subroutine is called in `usray4`, user subroutine from the radiation module.

6.24 Modification of the mesh geometry: `usmodg`

Subroutine called only during the calculation initialisation.

This subroutine may be used to modify "manually" the mesh vertices coordinates, *i.e.* the array:

- `xyznod(3,nnod)` (vertex coordinates)

WARNING: Caution must be exercised when using this subroutine along with periodicity. Indeed, the periodicity parameters are not 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.

6.25 Management of the post-processing intermediate outputs: `usnpst`

Subroutine called every time step(even if the user hasn't moved it to the SRC directroy).

This subroutine is used to determine when post-processing outputs will be generated. By default, it tests if the current time step number (`ntcabs`) is a multiple of the chosen output frequency (`ntchr`). If it is the case, the indicator `iipost` turns to 1, which triggers the writing of an intermediate output. If the frequency is given a negative value, the test is not done.

For instance, a user who wants to generate post-processing outputs (also called "chronological outputs") at the time step number 36 and around the physical time t=12 seconds may use the following test:

```
iipost = 0                                  No output by default.
if (ntcabs.eq.36) then                      If the current time step is the 36th,
   iipost=1                                   generate an output.
endif                                       End of the test on the time step number.
if (abs(ttcabs-12.d0).le.0.01d0) then       If the physical time is 12s +/- 0.01s,
   iipost=1                                   generate an output.
endif                                       End of the test on the physical time.
```

In any case, a post-processing output is generated after the last time step, `usnpst` being used or not.

6.26 Definition of post-processing and mesh zones: `usdpst`

Subroutine called at the calculation beginning..

This subroutine allows for the definition of surface or volume sections, in the form of lists of `nlfac` internal faces (`lstfac`) and `nlfab` boundary faces (`lstfab`), or of `nlcel` cells (`lstcel`), in order to generate chronological outputs in *EnSight*, *MED* or *CGNS* format.

One or several "writers" can be associated with each visualization mesh, or "part" created. The arguments of the function `pstcwr` defining a "writer" are as follows:

- `nomcas`: basic name of the associated case.
  WARNING: depending on the chosen format, this name may be shortened (maximum number of characters: 32 for *MED*, 19 for *EnSight*) or modified automatically (whitespaces or forbidden characters will be replaced by '_')
- `nomrep`: name of the output directory
- `nomfmt`: choice of the output format:
  - → *EnSight Gold* (*EnSight* also accepted)
  - → *MED_fichier* (*MED* also accepted)
  - → *CGNS*
  - → *text* (mesh output, no variables output, for debug only).

  The options are not case-sensitive, so *ensight* or *cgns* are valid, too.
- `optfmt`: character string containing a list of options related to the format, separated by commas; for the *EnSight Gold* format, these options are:
  - → *binary* for a binary format version (by default)
  - → *text* for a text format version
  - → *discard_polygons* to prevent from exporting faces with more than four edges (which may not be recognized by some post-processing tools); such faces will therefore not appear in the post-processing mesh.
  - → *discard_polyhedra* to prevent from exporting elements which are neither tetrahedra, prisms, pyramids nor hexahedra (which may not be recognized by some post-processing tools); such elements will therefore not appear in the post-processing mesh.
  - → *divide_polygons* to divide faces with more than four edges into triangles, so that any post-processing tool can recognize them
  - → *divide_polyhedra* to divide elements which are neither tetrahedra, prisms, pyramids nor hexahedra into simpler elements (tetrahedra and pyramids), so that any post-processing tool can recognize them
  - → *split_tensor* to export the components of a tensor variable as a series of independent variables (a variable is recognised as a tensor if its dimension is 6 or 9); not implemented yet.
- `indmod`: indicates if the post-processing (i.e. visualization) meshes (or "parts") are:
  - → 0 fixed (usual case)
  - → 1 deformable (the vertex positions may vary over time)
  - → 2 modifiable: (the lists of cells or faces defining these "parts" can be changed over time)
- `ntchrl`: default output frequency associated with this "writer" (the output may be forced or prevented at every time step using the subroutine `usnpst`)

In order to allow the user to add an output format to the main output format, or to add a mesh to the default output, the lists of standard and user meshes and "writers" are not separated. Negative numbers are reserved for the non-user items. For instance,the mesh numbers -1 and -2 correspond respectively to the global mesh and to boundary faces, generated by default, and the "writer" -1 corresponds to the usual post-processing case defined *via* `usini1` or *via* the interface.

The user chooses the numbers corresponding to the post-processing meshes and "writers" he wants to create. These numbers must be positive integers. It is possible to assocate a user mesh with the standard post-processing case (-1), or to ask for outputs regarding the boundary faces (-2) associated with a user "writer".

For safety, the output frequency and the possibility to modify the post-processing meshes are associated with the "writers" rather than with with the "parts". This logic avoids unwanted generation of inconstitent post-processing outputs. For instance EnSight would not be able to read a case in which one field is output to a given part every 10 time steps while another field is output to the same part every 200 time steps.

The possibility to modify a mesh over time is limited by the more restrictive "writer" which is associated with it. For instance, if the "writer" 1 allows the modification of the mesh topology (argument `indmod = 2` in the call to `pstcwr`) and the "writer" 2 allows no modification (`indmod = 0`), a user post-processing mesh associated with the "writers" 1 and 2 will not be modifiable, but a mesh associated only with the "writer" 1 will be modifiable. The modification is done by means of the user subroutine `usmpst`, which is called only for the currently modifiable meshes.

It is possible to output variables which are normally automatically output on the main volume or boundary meshes to a user mesh which is a subset of one of these by assigning the corresponding category to the user mesh. By default, a meshe's category is identical to its number, so the category associated with the main volume output is -1, and that associated with the main boundary output is -2. A category may be assigned using the `pstcat` subroutine.

It is also possible to define an alias of a post-processing mesh. An alias shares all the attributes of a "part" (without duplication), except its number. This may be used to output different variables on a same "part" with 2 different writers: the choice of output variables is based on the "part", so if P_a is associated with writer W_a, all that is needed is to define an alias P_b to P_a and associate it with writer W_b to allow a different output variable selection with each writer. An alias may be created using the `pstalm` subroutine.

Modification of a part or it's alias over time is always limited by the most restrictive "writer" to which it's meshes have been asscoiated (parts of the structures being shared in memory). It is possible to define as many alias' as are required for a "part", but an alias cannot be defined for another alias.

It is not possible to mix cells and faces in the same "part" (most of the post-processing tools being perturbed by such a case)[33]. If the user defines lists of faces and cells simultaneously, only the higher dimension entities (the cells) will be taken into account.

For a better understanding, the user may refer to the example given in `usdpst`. We can note that the whitespaces in the beginning or in the end of the character strings given as arguments of the functions called are suppressed automatically.

The variables to post-process on the defined "parts" will be specified in the subroutine `usvpst`. "

WARNING In the parallel case, some "parts" may not contain any local elements on a given processor. This is not a problem at all, as long as the "part" is defined for all processors (empty or not). It would in fact not be a good idea at all to define a "part" only if it contains local elements, global operations on the "part" would become impossible, leading to probable deadlocks or crashes.

[33] in thr future, it will probably be possible to automatically add faces bearing group or attribute characteristics to a cell mesh, but those faces will only be written for formats supporting this (such as MED 2.3), and will only bear attributes, not variable fields

6.27 Modification of the mesh zones to post-process: usmpst

Subroutine called only for each modifiable "part", at every active time step of an associated "writer".

For the user "parts" defined *via* the user subroutine **usdpst** and associated only with "writers" allowing the "part" modification over time (*i.e.* created with the parameter **indmod** = 2), this subroutine is used to modify the lists of cells, internal and boundary faces defining this "part" (or post-processing mesh).

At first, the corresponding lists contain the previously defined values. If these lists are modified for a given post-processing mesh, the argument **imodif** must be given the value 1.If this argument maintains it's initial value of 0, the code will not consider this "part" to have been modified away from that call and it will offer to bring it upto date. It is in fact at the end of an optimisation so there is no need to modify these "parts" within the definate and modifiable assembly (if in doubt, let imodif=1).

Note that the **itypps** flag can be used to determine whether the current post-processing mesh contains cells (**itypps(1)** = 1), internal faces (**itypps(2)** = 1), or boundary faces (**itypps(2)** = 1) globally (as the number of local cells or faces of a processor could be 0, it doesn't provide sufficient information). If at any time, a given part contains no element of any type, all the values of itypes will be 0 and that number cannot be put in the part (**nummai**) to determine if it will affect the cells or faces[34].

The user may refer to the example, in which cells are selected according to a given criterion:
- For a volume "part", cells for which the velocity exceeds a certain value.
- For a surface "part", interior faces which are between a cell in which the velocity exceeds a certain value and a cell in which the velocity is lower than this value (and boundary faces neighboring a cell in which the velocity exceeds this value). This surface post-processing mesh corresponds therefore to an approximation of a velocity isosurface.

6.28 Definition of the variables to post-process: usvpst

*Subroutine called for each "part", at every active time step of an associated "writer" (see **usdpst**).*

For the parts defined in **usdpst**, the subroutine **usvpst** is used to specify the variables to post-process.

The output of a given variable is generated by means of a call to **psteva**, whose arguments are:

- **nummai**: current "part" number (input argument in **usvpst**).
- **namevr**: name to give to the variable.
- **idimt**: dimension of the variable (3 for a vector, 1 for a scalar).
- **ientla**: indicates if the stored arrays are "interlaced" or not:
  - → 0: not interlaced, in the form $\{x_1, x_2, ..., x_n, y_1, y_2, ..., y_n, z_1, z_2, ..., z_n\}$ (case of all variables defined in **rtp**).
  - → 1: interlaced, in the form $\{x_1, y_1, z_1, x_2, y_2, z_2, ..., x_n, y_n, z_n\}$ (case of the geometric parameters, like **xyzcen**, **surfbo**, ...).

  For a scalar variable, this argument does not matter.
- **ivarpr**: indicates if the variable is defined on the "parent" mesh or locally:
  - → 0: variable generated by the user in the given work arrays **tracel**, **trafac**, and **trafbr** (whose size is respectively the number of cells, internal faces and boundary faces of the "part", ×3). The arrays **lstcel**, **lstfac**, and **lstfbr** can be used to get the numbers corresponding to the cells, internal faces and boundary faces associated with the "part" and to generate the appropriate post-processing variable.

[34] It is not expressly forbidden to associate cells with the "part" at a certain timestep and faces at another, but this has not been tested

→ 1: variable already defined in the main mesh ("parent" mesh of the "parts"), for example the variables in the rtp array. Instructions in the report which list`lstcel`, `lstfac`, and `lstfbr`will be treated directly by the sub routine, avoiding unused copies and simplifying hte code

- `ntcabs`: absolute current time step number. If a negative value is given (usually -1), the variable will be regarded as time-independent (and we will have to make sure this call is only made once).
- `ttcabs`: current physical time value. It is not taken into account if `ntcabs` < 0.
- `tracel`: array containing the values of the variable at the cells. If `ivarpr` $= 1$, this argument will be replaced by the position of the beginning of the array on which the variable in defined, for instance `rtp(1, iu(1))` for the velocity of the phase 1.
- `trafac`: equivalent of `tracel` for the internal faces.
- `trafbr`: equivalent of `tracel` for the boundary faces.

The user may refer to the example, which presents the different ways of generating an output of a variable.

WARNING: Apart from the time-independent variables, it is not recommended not to generate the same variables at every call (corresponding to an active time step) for a given mesh, because the post-processing tool may have difficulties to deal with such a case. To generate outputs of different variables on the same mesh with different frequancies, it is recommended to create an alias of this mesh and to associate it with a different "writer" in the subroutine ***`usdpst`****.*

6.29 Modification of the variables at the end of a time step: `usproj`

Subroutine called every time step.

This subroutine 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:

- 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
- Printing of a moment
- Utilisation of utility subroutines useful 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 subdomains 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 cells neighboring a face).

- Finally it must not be forgotten that the solving with temperature as a solved variable is questionable when the specific heat is not constant.

6.30 Radiative thermal transfers in semi-transparent gray media

6.30.1 Initialisation of the radiation main key words: `usray1`

Subroutine called only during calculation initialisation.

This subroutine is one of the two which must be completed by the user for all calculations including radiative thermal transfers. This subroutine 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 run 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 key word list (§7).

6.30.2 Management of the radiation boundary conditions: `usray2`

Subroutine called every time step.

This is the second subroutine is necessary for every calculation including radiative thermal transfers. It is used to give all the necessary parameters concerning, in the one case, the wall temperature calculation, and in the other, the coupling between the termal 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 **`usclim`** will be modified by the radiation module according to the data given in **`usray2`** (cf. §6.2).
A zone number must be given to each boundary face [35]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 to assign 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 `usray1`.
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 gray or black wall face, calculation of the temperature by means of a flux balance,

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

- `iprefl`: for a reflecting wall face, calculation of the temperature by means of a flux balance. This is fixed at 2000 in `radiat.h` and cannot be modified.
- `ifgrno`: gray 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.

Depending on the selected boundary condition type at every wall face, the code needs to be given some supplementary pieces of 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, 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 obligatory 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.

6.30.3 Absorption coefficient of the medium, boundary conditions for the luminance and calcualtion of the net radiative flux: `usray3`

Subroutine called every time step.

This subroutine 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, it is calculated automatically, or given 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.

6.30.4 Encapsulation of the temperature-enthalpy conversion: `usray4`

Subroutine called every time step.

This subroutine is used to call the subroutine `usthht`. The user can implement his own conversion formulas into it.
This subroutine is useless when the thermal scalar is the temperature.

WARNING: when a specific physics is activated, it is forbidden to use this subroutine. In this case, `usray4` *is replaced by* `ppray4`*, which is not a user subroutine.*

The value of the argument `mode` allows to know in which direction the conversion will be made:

- `mode = 1`: the fluid enthalpy in the cell must be converted into temperature (in Kelvin),
- `mode = -1`: the wall temperature (`text` or `tparoi`, in Kelvin) must be converted into enthalpy.

WARNING: the value of `mode` *is passed as argument and must not be modified by the user.*

6.31 Utilisation of a specific physics: `usppmo`

Subroutine called only during calculation initialisation.

This is one of the three subroutines which must be obligatory completed by the user in order to use a specific physics module. At the moment, *Code_Saturne* allows to use two "pulverised coal" modules (Lagrangian coupling or not), two "gas combustion" modules, two "electric" modules and one "compressible" module. To activate one of these modules, the user needs to 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 condintions.

→ `ippmod(icolwc)`=-1 module not activated.

- Multi-coals and multi-classes pulverised coal combustion: indicator `ippmod(icp3pl)` The number of different coals must be inferior or equal to `ncharm` = 3. The number of particle size classes `nclpch(icha)` for the coal `icha`, must be inferior or equal to `ncpcmx` = 10.

  → `ippmod(icp3pl)` = 0 imbalance between the temperature of the continuous and the solid phases

  → `ippmod(icp3pl)` = 1 otherwise

  → `ippmod(icp3pl)` =-1 module not activated

- Lagrangian modeling of multi-coals and multi-classes pulverised coal combustion: indicator `ippmod(icpl3c)` The number of different coals must be inferior or equal to `ncharm` = 3. The number of particle size classes `nclpch(icha)` for the coal `icha`, must be inferior 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 arc 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

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

In the framework of the gas combustion modeling, 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 (thermo-chemical files).

NOTE: THE THERMO-CHEMICAL FILES

The user must not forget to place in the directory DATA the thermo-chemical file `dp_FCP`, `dp_C3P`, `dp_C3PSJ` or `dp_ELE` (depending on the specific physics module he activated) and to specify the name of this file in the variable THERMOCHEMISTRY_DATA in the launch script (for instance: THERMOCHEMISTRY_DATA"dp_C3P"). Some example files are placed in the directory `DATA/THCH` at the creation of the study case. Their content is described below.

- Example of file for the gas combustion:

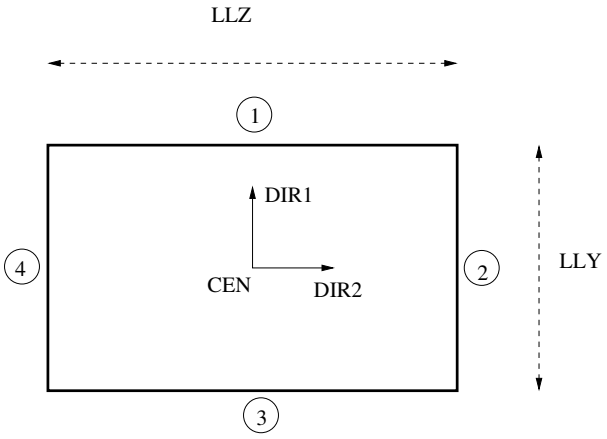


Figure 6: Numbering of the edges of a rectangular inlet(`icas`=1) treated by the vortex method

Lines	Examples of values	Variables	Observations
1	5	`ngaze`	Number of current species
2	10	`npo`	Number of points for the enthalpy-temperature tabulation
3	300.	`tmin`	Temperature inferior limit for the tabulation
4	3000.	`tmax`	Temperature superior limit 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 fonction of the current species of the 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 reaction global species. 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 enthalpy 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 mass 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 coefficient of fuel, oxidiser and products
11	1. 2.	`xco2`, `xh2o`	Molar coefficents of CO_2 and H_2O in the products (radiation using Modak)

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

→ if the enthalpy-temperature conversion data base JANAF is used: `dp_C3P` (see array1).

→ 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` (see array 3).
- Example of file for the electric arc: `dp_ELE` (see array 4).

Lines	Examples of values	Variables	Observations
1	THERMOCHIMIE		Comment line
2	8	ncoel	Number of current species
3	8	npo	Number of points for the enthalpy-temperature tabulation
4	ESPECES COURANTES		Comment line
5	CH4 C2H4 CO O2 CO2 H2O N2 C(S)	nomcoel(ncoel)	List of the current species
6	300.	tmin	Temperature inferior limit (Kelvin) for the enthalpy-temperature tabulation
7	2400.	tmax	Temperature superior limit (Kelvin) for the enthalpy-temperature tabulation
8	4	nato	Number of elemental species
9 10 11 12	.012 1 2 1 0 1 0 0 1 .001 4 4 0 0 0 2 0 0 .016 0 0 1 2 2 1 0 0 .014 0 0 0 0 0 0 2 0	wmolat(nato), atcoel(ncoel,nato)	Molar mass of the elemental species (first column) and composition of the current species as a function of the elemental species
13	RAYONNEMENT		Comment line
14	0.1	ckabs1	Constant absorption coefficient for the gas mixture
15	CARACTERISTIQUES CHARBONS		Comment line
16	2	ncharb	Number of coal types
17	1 1	nclpch(ncharb)	Number of classes for each coal (each column corresponding to one coal type)
18	50.E-6 50.E-6	diam20(nclacp)	Initial diameter of each class (m) nclacp is the total number of classes. All the diameters are written on the same line (sucessively for each coal, we give the diameter corresponding to each class)
19	74.8 60.5	cch(ncharb)	Composition in C (mass.-%, dry) of each coal
20	5.1 4.14	hch(ncharb)	Composition in H (mass.-%, dry) of each coal
21	12.01 5.55	och(ncharb)	Composition in O (mass.-%, dry) of each coal
22	0 31524000. 0 31524000.	ipci(ncharb) pcich(ncharb)	Value of the PCI (Jkg^{-1}) for each coal, the first integer indicating if this value refers to pure (0) or dry coal (1)
23	1800. 1800.	cp2ch(ncharb)	Heat-storage capacity at constant pressure ($Jkg^{-1}K^{-1}$) for each coal
24	1200. 1200.	rho0ch(ncharb)	Initial density (kgm^{-3}) of each
25	Coke		Comment line
26	0. 0.	cck(ncharb)	Composition in C (mass.-%, dry) of the coke for each coal
27	0. 0.	hck(ncharb)	Composition in H (mass.-%, dry) of the coke for each coal
28	0. 0.	ock(ncharb)	Composition in O (mass.%, dry) of the coke for each coal
29	0. 0.	pcick(ncharb)	PCI of the dry coke (Jkg^{-1}) for each coal
30	Cendres		Comment line
31	6.3 6.3	xashch(ncharb)	Ash mass fraction (mass.-%, dry) in each coal
32	0. 0.	h0ashc(ncharb)	Ash formation enthalpy (Jkg^{-1}) for each coal
33	0. 0.	cpashc(ncharb)	CP of the ashes ($Jkg^{-1}K^{-1}$) for each coal
34	Dévolatilisation (Kobayashi)		Comment line
35	1 0.37 0 0.37	iy1ch(ncharb), y1ch(ncharb)	For each coal, pairs (iy1ch, y1ch). The real y1ch is the adimensional stoich. coefficient If the integer iy1ch is worth 1, the provided value of y1ch is adopted and the composition of the light volatile matters is calculated automatically. If the integer iy1ch is worth 0, the provided value of y1ch is ignored: y1ch is calculated automatically (the light volatiles are then composed of CH_4, CO).
36	1 0.74 1 0.74	iy2ch(ncharb), y2ch(ncharb)	For each coal, pairs (iy2ch, y2ch). The real y2ch is the adimensional stoich. coefficient If the integer iy2ch is worth 1, the provided value of y2ch is adopted and the composition of the heavy volatile matters is calculated automatically. If the integer iy2ch is worth 0, the provided value of y2ch is ignored: y2ch is calculated automatically (the heavy volatiles are then composed of C_2H_4, CO).
37	370000. 410000.	a1ch(ncharb)	Devolatilisation pre-exponential factor A1 (s^{-1}) for each coal (light volatile matters)
38	1.3E13 1.52E13	a2ch(ncharb)	Devolatilisation pre-exponential factor A2 (s^{-1}) for each coal (heavy volatile matters)
39	74000. 80000.	e1ch(ncharb)	Devolatilisation activation energy E1 ($Jmol^{-1}$) for each coal (light volatile matters)
40	250000. 310000.	e2ch(ncharb)	Energie d'activation E2 ($Jmol^{-1}$) de dévolatilisation for each coal (heavy volatile matters)
41	Combustion hétérogène		Ligne de commentaire
42	17.88 17.88	ahetch(ncharb)	Char burnout pre-exponential constant ($kgm^{-2}s^{-1}atm^{-1}$) for each coal
43	16.55 16.55	ehetch(ncharb)	Char burnout activation energy ($kcalmol-1$) for each coal
44	1 1	iochet(ncharb)	Char burnout reaction order for each coal 0.5 if iochet = 0 and 1 if iochet = 1

Table 3: Example of file for the pulverised coal combustion: `dp_FCP`

Li nes	Examples of values	Variables	Observations
1	# Fichier ASCII format libre ...		Free comment
2	# Les lignes de commentaires ...		Free comment
3	# ...		Free comment
4	# Proprietes de l'Argon ...		Free comment
5	# ...		Free comment
6	# Nb d'especes NGAZG et Nb ...		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.h`) So there will be `ngazg` blocks of `npo` lines each
9	# ...		Free comment
10	# Proprietes ...		Free comment
11	# T H ...		Free comment
12	# Temperature Enthalpie ...		Free comment
13	# ...		Free comment
14	# K J/kg ...		Free comment
15	# ...		Free comment

16	300. 14000. ...	
		
		`h`
		`roel`
		`cpel`
		`sigel`
		`visel`
		`xlabel`
		`xkabel`
			Tabulation in line 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 4: Example of file for the electric arc module: `dp_ELE`

6.32 Management of the boundary conditions related to pulverised coal and gas combustion: `usebuc, usd3pc, uslwcc, uscpcl and uscplc`

Subroutines called every time step.

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

As are `usini1` and `usppmo`, the use of `usebuc`, `usd3pc`, `uslwcc`, `uscpcl` or `uscplc` is obligatory to run a calculation concerning a specific physics modeling. The way of using them is the same as the way of using `usclim` in the framework of standard calculations, that is to say several loops on the boundary faces lists (cf. §6.2) 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 modeling, all the boundary conditions for every variable must be defined here, even for the eventual user scalars: `usclim` *is not used at all.*

In the case of a specific physics modeling, a zone number `izone` [36] (for instance the color `icoul`) is associated with every boundary face, in order to gather together all the boundary faces of the same type. In comparison to `usclim`, 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(iphas))`, `rcodcl(ifac,iv(iphas))` and `rcodcl(ifac,iw(iphas)`

      WARNING:
      - *the variable* ***`qimp(izone)`*** *refers to the mass flow across the whole zone* ***`izone`*** *and not across a boundary face (specifically for the axisymetric calculations, the inlet suface 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 recomended 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(iphas))`, `rcodcl(ifac,iv(iphas))` and `rcodcl(ifac,iw(iphas))`
  - → 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 "oxydiser inlet" type marked out by `ientox(izone)`=1 and the "fuel inlet" type marked out by `ientfu(izone)`=1

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

→ 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 oxydiser inlet and `tinfue`, for each fuel inlet

Note: In the standard version, only the cases with only one oxydising 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(iphas))`, `rcodcl(ifac,iv(iphas))` and `rcodcl(ifac,iw(iphas))` and the incoming air temperature `timpat(izone)` in Kelvin. If the velocity is imposed, he has to set `rcodcl(ifac,iu(iphas))`, `rcodcl(ifac,iv(iphas))` and `rcodcl(ifac,iw(iphas))`.

→ 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)`

6.33 Initialisation of the variables related to pulverised coal and gas combustion: `usebui, usd3pi, uslwci and uscpiv`

Subroutines called only during the calculation initialisation.

In this paragraph, "specific physics" refers 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.

WARNING: in the case of a specific physics modeling, all the variables will be initialised here, even the eventual user scalars: `usiniv` *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 `rtp(iel,isca(ifm))` in variable richness, the fresh gas mass fraction `rtp(iel,isca(iygfm))` and the mixture enthalpy `rtp(iel,isca(ihm))` 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 `rtp(iel,isca(ifm))`, its variance `rtp(iel,isca(ifp2m))` and the mixture mass enthalpy `rtp(iel,isca(ihm))` 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

`rtp(iel,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)

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

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

`rtp(iel,isca(ih2(icla)))` the mass enthalpy related to the class `icla` in permeatic conditions

→ `rtp(iel,isca(ihm))` the mixture enthalpy

→ the transport variables related to the gas phase

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

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

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

`rtp(iel,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)

`rtp(iel,isca(ifp3m))` the variance associated with the tracer 3

6.34 Initialisation of the options of the variables related to pulverised coal and gas combustion: `usebu1, usd3p1, uslwc1, uscpi1 and uscpl1`

Subroutines called at calculation beginning.

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

These 3 subroutines are used to complete `usini1` for the considered specific physics. They allow to:

- generate, for the variables which are specific to the activated specific physics module, chronological outputs (indicators `ichrvr(ipp)`), follow-ups in the listings (indicator `ilisvr(ipp)`) and to activate chronological records at the probes defined in `usini1` (indicators `ihisvr(ipp)`).
  The way of doing it is the same as in `usini1` and the writing frequencies of these ouputs are set by `usini1`. The values of the indicators `ipp` are `ipp=ipppro(ipproc(ivar))`, with `ivar` the number of the specific physics variable. Concerning the main variables (velocity, pressure, etc ...) the user must still complete `usini1` 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 calculation variables `ivar` (defined at the cell `iel` by `rtp(iel,ivar)`) and the properties `iprop` (defined at the cell `iel` by `propce(iel,ipproc(iprop))`) are listed now:

  → EBU pre-mixed flame modeling:

  - Calculation variables `rtp(iel,ivar)`
    `ivar` = `isca(iygfm)` fresh gas mass fraction
    `ivar` = `isca(ifm)` mixing rate
    `ivar` = `isca(ihm)` enthalpy, if transported
  - Properties `propce(iel,ipproc(iprop))`
    `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 modeling is activated
    `iprop` = `it3m` and `it4m` "T^3" and "T^4" terms, when the radiation modeling is activated

  → rapid complete chemistry diffusion flame modeling:

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

→ pulverised coal modeling with 3 comustables:

variables shared by the two phases:

- Calculation variables `rtp(iel,ivar)`

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

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

variables specific to the dispersed phase:

- Calculation variables `rtp(iel,ivar)`

  `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 `propce(iel,ipproc(iprop))`

  `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:

- Calculation variables `rtp(iel,ivar)`

  `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 `propce(iel,ipproc(iprop))`

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

- 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 `usebu1`. By default `cebu`=2.5)

6.35 Management of Boundary Conditions of the electric arc: `uselcl`

sub routine called at each time step.

As in the `usini1` and `usppmo`, the use of `usecl` is required to run an electric calculation. The main use is the same as occurs in `usclim` for the standard *Code_Saturne* calculations, for which different loops on the boundary faces is defined. Each faces list is built with the use of selection criteria (cf. §6.2), and is referenced by their group(s), their color(s) or geometrical criterions. The face type, the boundary conditions for each variable, and finally the value of each variable or imposed flow are fixed.

WARNING:for the electric module, , the boundary conditions of all the variables are defined here, even those of the eventual user scalars: `usclim` *is not used at all.*

For the electric module, each boundary face is associated with a number `izone` [37](the color `icoul` for example) in order to group together all the boundary faces of the same type. In the report `usclim`, 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).

Whats more, if one wishes to slow down the power dissipation(Joule module effect) or the current (electric arc module) from the imposed values (`puismp` and `couimp` respectively), they can be changed by the potential scalar as shown below:

- For the electric arc, 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 `useli1` by an estimated potential difference. If `ielcor`=1 (see `useli1`), `dpot` is updated automatically during the calculation to obtain the required current.

- For the Joule module effect, `dpot` is again used with the same signification as in the electric arc module. If `dpot` is not wanted in the setting of the boundary condtions, 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 `useli1`), multiply the imposed potentials in `uselcl` by `coejou` at each time step to achieive the desired power dissipation.

WARNING: In alternative 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 someitmes needed (`ippmod(ieljou)`=2): this is the case in particular where the current has 3 phases. To have access to the phase of the potential, and not just its amplitude, the 2 variables must be deleted: in *Code_Saturne*, there are 2 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 respecively $Pp_{\max}\, cos\phi$ and $Pp_{\max}\, sin\phi$.

[37] `izone` must be less than the maximum value allowed by the code, `nozzppm`. This is fixed at 2000 in `ppvar.h` and cannot be modified.

- 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 continous current, but also with single phase alternative current. In *Code_Saturne* there is only 1 varialbe 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 continous current there is no such ambiguity).

6.36 Initialisation of the variables in the electric module

subroutine called only at the initialisation of the calculation

This subroutine allows the user to initialise some of the specific physics variables prompted via `usppmo` . The user has access, as usual, to many geometric variables so that the zones can be differentiated if needed.

WARNING: For the specific physics, it is here that all varialbes are initialsed: ***`usiniv`*** *is not used*

This subroutine works like `usiniv`. The values of potential and its constiuents are initialised if required.

It should be noted that the enthalpy is important.

- For the electric arc module, the enthalpy value is taken from the temperature of reference `t0(iphas)` (given is `usini1`) 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 (IPHAS)(given in `usini1`), the the temperature-enthalpy low must be supplied. A code is suggested in the sub routine `usthht`(which is there for the determination of physical properties).

6.37 Initialisation of the variable options in the electric module

subroutine called at each time step

This subroutine is completed in `usini1` for the specific physics. It allows:

- Activates the variables in the specific physics module, the chronolgical outputs (`ichrvr(ipp)` indicators), the listings (`ilisvr(ipp)` indicators) and the historical exits at the probes defined in `usini1` (`ihisvr(ipp)` indicators). The functions are the same as in `usini1` and the script frequency of the exits are fixed using `usini1`. The indicators `ipp` are for the value `ipp=ipppro` (`ipproc(ivar)`, with `ivar`, the number of specific physics varibles. With the main variables which concern the user (velocity, pressure, etc), the user must always use `usini1` if the history,the listings or the chronological files are required. The variables which the user can activate are marked out. The number of variables in the calculation is given in `ivar` (defined by `propce(iel,ipproc(iprop)` for cell `iel`):

  → Electric Arc Module:

  - Calculation variables `rtp(iel,ivar)`

    `ivar` = `isca(ihm)` enthalpy

    `ivar` = `isca(ipotr)` real potentiel

    `ivar` = `isca(ipotva(i))` solved components of the potential vector.

    `ivar` = `isca(iycoel(iesp))` the mass fraction of `ngazg` composites if there are more than 1
  - Properties `propce(iel,ipproc(iprop))`

    `iprop` = `itemp` temperature

`iprop` = `iefjou` power dissipation by the Joule effect.

`iprop` = `ilapla(i)` components of the laplace forces.

→ Joule Module effect :

- Calculation variables `rtp(iel,ivar)`

  `ivar` = `isca(ihm)` enthalpy

  `ivar` = `isca(ipotr)` real potential

  `ivar` = `isca(ipoti)` imaginary potential if its to be taken into account

  `ivar` = `isca(iycoel(iesp))` the mass fraction of `ngazg` composites if there are more than 1

- Properties `propce(iel,ipproc(iprop))`

  `iprop` = `itemp` temperature

  `iprop` = `iefjou` volumic power dissipation by Joule effect.

- 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 arc, the sub-relaxation is taken into account during the 2nd time step; for the Joule effect the sub relaxation is not accounted for unless the user specifies in `uselph`

- indicates if the data will be fixed in the power dissipation or in the current, done in `ielcor`.

- target current fixed as `couimp` (electric arc module) or the power dissipation `puism` (Joule module effect).

- Fix the initial value of potential difference `dpot`, the for the calculations with a single fixed parameter as `couimp` or `puism`.

6.38 Management of variable physical properties in the electric module

Subroutine called at each time step

All the laws of the variation of physical data of the fluid are written (where neccessary) in this subroutine... The subroutine replaces `usphyyv` and a similar component.

WARNING: For the electric module, it is here that all the physical variables are defined (including the relative cells and the eventuel user scalars):`usepelph` is not used.

The user should ensure that the defined variation laws are valid for the whole range of variables. Particular attention should be taken with the non-linear laws (for example, a 3rd degree polynomial law giving negative values of density)

WARNING: with the electric module, all the physical propertie are assumed as variables and so are stored in the `propce` array. `cp0`, `viscls0`, `visclO` are not used

For the Joule effect, the user is required to supply the physical properties in the sub- routine. 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 preffered temperature-enthalpy law can be selected in the subruotine `usthht` (an example of the interpolation is given from the law table. This subroutine can be re-used for the initialisation of the variables(`useliv`)) For the elecrtic arc module, the physical properties are intepolated from the data file `dp_ELE` supplied by the user. Modification is not generally necessary.

6.39 Management of the post-processing output in the electric module: `uselen`

Subroutine called at each chronological output

This subroutine allows the addition on n variables in the preprocessing output and works like the subroutine `usvpst` (with the electric module, it is however also possible to use `usvpst`.

The algebraic variables related to the electric module are provided by default provided that they are not explicitely contained in the `propce` array:

- gradient of real potential in Vm^{-1} ($\underline{\text{grad}}\ Pot_R = -\underline{E}$)
- 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 arc module with `ippmod(ielarc)`=2 :

- magnetic field in T ($\underline{\text{B}} = \underline{\text{rot}}\,\underline{\text{A}}$)

If it is convenient for the user, there is no need to add this subroutine into the SRC directory: the post-processing will be done automatically (at the same frequency (`NTCHR`) as the other calculation variables)

6.40 Compressible module

When the compressible module[38] 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 `usini1`
- keep the convective numerical schemes proposed by default.

6.40.1 Initialisation of the options of the variables related to the compressible module: `uscfx1` and `uscfx2`

Subroutine called every time step.

These subroutines complete `usini1`.

`uscfx1` allows to set non standard calculation options related to the compressible module, and in particular to fill in the key word `icfgrp` allowing to take into account the hydrostatic equilibrium in the boundary conditions.

`uscfx2` allows to specify for the molecular thermal conductivity and the volumetric viscosity the following pieces of information:

- variable or not (`iviscv`)
- reference value (`viscv0`)

[38] For more details concerning the compressible version, the user may refer to 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.

6.40.2 Management of the boundary conditions related to the compressible module: `uscfcl`

Subroutine called every time step.

The use of `uscfcl` is obligatory to run a calculation using the compressible module just as it is in both `usini1` and `usppmo` . The way of using it is the same as the way of using `usclim` in the framework of standard calculations, that is to say several loops on the boundary faces lists (cf. §6.2) 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 calculation using the compressible module, the boundary conditions of all the variables are defined here, even those of the eventual user scalars: `usclim` *is not used at all.*

In the compressible module, the different available boundary conditions are the followings:

- inlet/outlet for which everything is known
- supersonic outlet
- subsonic inlet
- subsonic wall
- wall
- symmetry

6.40.3 Ininitialisation of the variables related to the compressible module: `uscfxi`

Subroutine called only during calculation initialisation.

This subroutine is used 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.

WARNING: in the case of a specific physics modeling, all the variables are initialised here: `usiniv` *is not used at all.*

This subroutine works like `usiniv` for velocity, turbulence and passive scalars. Concerning pressure, density, temperature and specific total energy, only 2 variables out of the 4 are independant. The user may also initialise the variable pair he wants (apart from temperature-energy) and the two other variables will be calculated automatically by giving the right value to the variable `iccfth` used for the call to `uscfth`.

6.40.4 Compressible module thermodynamics: `uscfth`

This subroutine is called several times every time step (boundary conditions, physical properties, solving of the energy equation, ...).

This subroutine is used to set the thermodynamics parameters. By default, the perfect gas laws are implemented. If the user needs to use other laws (perfect gas with variable Gamma, Van der Waals), he must modify this subroutine.

6.40.5 Management of the variable physical properties in the compressible module: `uscfpv`

Subroutine called every time step.

If necessary, all the variation laws of the fluid physical properties (viscosity, specific heat, ...) are described here. This subroutine replaces and is similar to `usphyv`.

The user should make sure that the defined variation laws are valid for the whole variation range of the variables.

6.41 Lagrangian modeling of multiphasic flows with dipersed inclusions

6.41.1 Initialisation of the main key words in the Lagrangian modeling: `uslag1`

Subroutine called only during calculation initialisation.

This is one of the two subroutines which must be completed in the case of a calculation modeling a Lagrangian multiphasic flow. This subroutine gathers in different headings all the key word 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
- the boundary statistics
- the postprocessing in trajectory mode

For more details about the different parameters, the user may refer to the key word list (§7.8).

The results of the lagangian module consist in some information about the particle cloud. These pieces of information are displayed in the form of statistics. It is therefore necessary to activate the calculation of the statistics at a given instant during the simulation. To do so, there are different strategies which are strongly related to the flow nature, stationary or not.
Except from the cases where the injection conditions depend on the time, it is generally recommended to realise a first Lagrangian calculation whose aim is to get a nearly constant particle number in the calculation domain. In a second step, a calculation restart is done to calculate the statistics.
When the single-phase flow is steady and the inclusion presence rate is low enough to neglect their influence on the continuous phase behaviour, it is better to realise a Lagrangian calculation on a fixed field. It is then possible to calculate stationary volumetric statistics and to give a statistical weight higher than 1 to the particles, in order to reduce the number to treat while keeping the right concentrations.
Otherwise, when the continuous phase flow is stationary, but the backward coupling must be taken into consideration, it is still possible to activate stationary statistics.
When the continuous phase flow is non-stationary, it is no longer possible to use stationary 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 complete model is used for the turbulent dispersion modeling, 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 (temperature or enthalpy). If the thermal scalar associated with the continuous phase is the temperature in Kelvin, the unit change is done automatically. 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.

WARNING: Up to now, parallelism and periodicity are not compatible with the Lagrangian module. This compatibility will be soon implemented. It is however possible, in the framework of a Lagrangian calculation on a fixed field, to realise in a first step the calculation of the continuous phase using parallelism, and to conduct in a second step the Lagrangian calculation by doing a restart on only one processor.

6.41.2 Management of the boundary conditions related to the particles: `uslag2` and `uslain`

In the framework of the multiphasic lagrangrian modeling, 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 (they are of course generally coherent). The boundary condition zones are actually redefined by the Lagrangian module (cf. §6.2), and a type of particle behaviour is associated with each one.
The management of the Lagrangian boundary conditions is done by means of several user subroutines: `uslag2` for the classic conditions and `uslain` to specify profiles if necessary. Otherwise, the subroutine `uslabo` allows to define the type of particle/wall interaction. It will be described in a specific paragraph.

SUBROUTINE `USLAG2`

Subroutine called every time step.

It is the second indispensable subroutine for every calculation using the Lagrangian module. The main numerical variables and "pointers" are described below.

`ifrlag(nfabor)` [ia]: In the Lagrangian module, the user defines `nfrlag` boundary zones from the color of the boundary faces, or more generally from their properties (colors, groups ...), from the boundary conditions defined in `usclim`, or even from their coordinates. To do so, the array `ifrlag(nfabor)` giving for each face `ifac` the number `ifrlag(ifac)` corresponding to the zone to which it belongs, is completed. The zone numbers (*i.e.* the values of `ifrlag(ifac)`) are chosen freely by the user, but must be strictly positive integers inferior or equal to `nflagm` (parameter stored in lagpar.h, whose default value is 100). A zone type is associated with every zone; it will be used to impose global boundary conditions. *WARNING: it is essential that every boundary face belongs to a zone.*.

`iusncl(nflagm)` [ia]: For all the `nfrlag` boundary zones previously identified, the number of classes `nbclas`[39] of entering particles is given: `iusncl(izone)` = `nbclas`. By default, the number of particle classes is zero. The maximum number of classes is `nclagm` (parameter stored in lagpar.h, whose default value is 20)..

`iusclb(nflagm)` [ia]: For all the `nfrlag` boundary zones previously identified, a particle boundary condition type is given. There are two categories of particle boundary condition types: those predefined in the subroutine `uslabo` (marked out by the key words `ientrl`, `isortl`, `irebol`, `idepo1`, `idepo2`, `idepo3`, `iencrl`) and the user boundary condition types (marked out by the key words `jbord1` to `jbord5`), whose corresponding particle behaviour must be defined in the subroutine `uslabo`.

[39] a class is a set of particles sharing the same physical properties and the same characteristics concerning the injection in the calculation domain

- if `iusclb(izone) = ientrl, izone` is a particle injection zone. For each particle class associated with this zone, some pieces of information must be given (see below). If a particle trajectory crosses an injection zone, then we consider that this particle leaves the calculation domain.
- if `iusclb(izone) = isortl`, the particles interacting with the zone `izone` leave the calculation domain.
- if `iusclb(izone) = irebol`, the particles undergo an elastic rebound on the boundary zone `izone`.
- if `iusclb(izone) = idepo1`, the particles settle definitevely on the boundary zone `izone`. These particles can not be put in suspension again, and we consider that they leave the calculation domain.
- if `iusclb(izone) = idepo2`, the particles settle definitevely on the boundary zone `izone`, but they are kept in the calculation domain. This distinction with the type `idepo1` is useful only when post-processings in movement mode (`ifensi2` = 1) are realised: the particles do not disappear after touching the boundary zone. However, using `idepo2` type zones necessitates more memory than using `idepo1` type zones.
- if `iusclb(izone) = idepo3`, the particles settle on the boudary zone `izone`, but can be put in suspension again depending on the local description of the continuous phase flow.
- if `iusclb(izone) = iencrl`, the particles which are coal particles (if `iphyla` = 2) can become fouled up on the zone `izone`. The slagging is a `idepo1` type deposit of the coal particle if a certain criterion is respected. Otherwise, the coal particle rebounds (`irebol` type behaviour). This boundary condition type is available if `iencra` = 1. A limit temperature `tprenc`, a critical viscosity `visref` and the coal composition in mineral matters must be given in the subroutine `uslag1`. The slagging criterion given by default may be modified in the subroutine `uslabo`.
- if `iusclb(izone) = jbord1` to `jbord5`, then the particle interaction with the boundary zone `izone` is given by the user. The particle behaviour associated with each type `jbord*` must be defined in the subroutine `uslabo`.

`iuslag(nclagm, nflagm, ndlaim)` [ia]: Some pieces of information must be given for each particle class associated with an injection zone. The first part consists in integers contained in the array `iuslag`. There are at the most `ndlaim` integers. These pieces of information must be provided for each class `iclas` and each particle injection zone `izone`. They are marked out by means of "pointers":

→ `iuslag(iclas,izone,ijnbp)`: number of particles to inject in the calculation domain per class and per zone.

→ `iuslag(iclas,izone,ijfre)`: injection period (expressed in number of time steps). If the period is null, then there is injection only at the first absolute Lagrangian time step (including the restart calculations).

→ `iuslag(iclas,izone,ijuvw)`: type of velocity condition:

- if `iuslag(iclas,izone,ijuvw)` = 1, the particle velocity vector is imposed, and its components must be given in the array `ruslag` (see below).
- if `iuslag(iclas,izone,ijuvw)` = 0, the particle velocity is imposed perpendicular to the injection boundary face and with the norm `ruslag(iclas,izone,iuno)`.
- if `iuslag(iclas,izone,ijuvw)` = -1, the particle injection velocity is equal to the fluid velocity at the center of the cell neighboring the injection boundary face.

→ `iuslag(iclas,izone,inuchl)`: when the particles are coal particles (`iphyla` = 2), this part of the array contains the coal index-number, between 1 and `ncharb` (defined by the user in the thermo-chemical file dp_FCP, with `ncharb`⩽`ncharm = 3)`.

`ruslag(nclagm, nflagm, ndlagm)` [ra]: Some pieces of information must be given for each particle class associated with an injection zone. The second and last part consists in real numbers contained in the array `ruslag`. There are at the most `ndlagm` such real numbers. These pieces of information must be provided for each class `iclas` and each particle injection zone `izone`. They are marked out by means of "pointers":

→ `ruslag(iclas,izone,iuno)`: norm of the injection velocity,
useful if `iuslag(iclas,izone,ijuvw)` = 0.

→ `ruslag(iclas,izone,iupt)`, `ruslag(iclas,izone,ivpt)`,
`ruslag(iclas,izone,iwpt)`: components of the particle injection vector,
useful if `iuslag(iclas,izone,ijuvw)` = 1.

→ `ruslag(iclas,izone,idebt)`: allows to impose a particle mass flow. According to the number of injected particles, the particle statistical weight `tepa(npt,jrpoi)` is recalculated in order to respect the required mass flow (the number of injected particles does not change). When the mass flow is null, it is not taken into account.

→ `ruslag(iclas,izone,ipoit)`: particle statistical weight per class and per zone.

→ `ruslag(iclas,izone,idpt)`: particle diameter. When the particles are coal particles (`iphyla` = 2), this diameter is provided by the thermo-chemical file dp_FCP *via* the array `diam20(iclg)`, where `iclg` is the "pointer" on the total class number (*i.e.* for all the coal types). When the standard deviation of the particle diameter is different from zero, this diameter becomes a mean diameter.

→ `ruslag(iclas,izone,ivdpt)`: standard deviation of the injection diameter. To impose this standard deviation allows to respect granulometric distribution: the diameter of each particle is calculated from the mean diameter, the standard deviation and a gaussian random number. In this case, it is strongly recommended to intervene in the subroutine `uslain` to restrict the diameter variation range, in order to avoid aberrant values. If this standard deviation is null, then the particle diameter is constant per class and per zone.

→ `ruslag(iclas,izone,iropt)`: particle density. When the particles are coal particles (`iphyla` = 2), this density is set in the thermo-chemical file dp_FCP *via* the array `rho0ch(icha)`, where `icha` is the coal number.

→ `ruslag(iclas,izone,itpt)`: particle injection temperature in °C. Useful if `iphyla` = 1 and if `itpvar` = 1.

→ `ruslag(iclas,izone,icpt)`: particle injection specific heat. Useful if `iphyla` = 1 and if `itpvar` = 1. When the particles are coal particles (`iphyla` = 2), the specific heat is set in the thermo-chemical file dp_FCP *via* the array `cp2ch(icha)`.

→ `ruslag(iclas,izone,iepsi)`: particle emissivity. Useful if `iphyla` = 1 and if `itpvar` = 1, and if the radiation module is activated for the continuous phase (note: when `iphyla` = 2, the coal particle emissivity is given the value 1).

→ `ruslag(iclas,izone,ihpt)`: particle injection temperature in °C when these particles are coal particles. The array `ruslag(iclas,izone,itpt)` is then no longer active. Useful if `iphyla` = 2.

→ `ruslag(iclas,izone,imcht)`: mass of reactive coal. Useful if `iphyla` = 2.

→ `ruslag(iclas,izone,imckt)`: mass of coke. This mass is null if the coal did not begin to burn before its injection. Useful if `iphyla` = 2.

`iusvis(nflagm)` [ia]: In order to display the variables at the boundaries defined in the subroutine `uslag1`, this array allows to select the boundary zones on which a display is wanted. To do so, a number is associated with each zone `izone`. If this number is strictly positive, the corresponding zone is selected; if it is null, the corresponding zone is eliminated. If several zones are associated with the same number, they will be displayed together in the same selection with *EnSight*. Each selection will be split in *EnSight* parts according to the geometric types of the present boundary faces ((*i.e.* 'tria3', 'quad4' and 'nsided')..

SUBROUTINE `USLAIN`

Subroutine called every time step.

It is not obligatory to intervene in this subroutine.

`uslain` is used to complete `uslag2` when the particles must be injected in the domain according to fine constraints (profile, position, ...): the arrays `ettp`, `tepa` and `itepa` can be modified here for the new particles (these arrays were previously completed automatically by the code from the data provided by the user in `uslag2`).

In the case of a more advanced utilisation, it is possible to modify here all the arrays `ettp`, `tepa` and `itepa`. The particles already present in the calculation domain are marked out by an index varying between 1 and `nbpart`. The particles entering the calculation domain at the current iteration are marked out by an index varying between `nbpart`+1 and `nbpnew`.

6.41.3 Treatment of the particle/boundary interaction: `uslabo`

Subroutine called at every particle/boundary interaction.

It is not obligatory to intervene in this subroutine, but it is required in four different cases.

Firstly, an intervention is required when `jbord`* type boundary conditions are used: it is then necessary to code in this subroutine the corresponding particle/boundary interactions.

Secondly, it is possible to select the particle/boundary interaction types (`irebol`, `idepo1`, ...) for which the user wants to save the wall statistics activated in the subroutine `uslag1`.

Thirdly, if user boundary statistics are activated *via* the key word `nusbor` in the subroutine `uslag1`, it is then necessary to program them in the subroutine `uslabo`. When the boundary statistics are stationary, these new boundary statistics are added using the array `parbor`. When they are non-stationary (number of Lagrangian iterations lower than `nstbor`, or `isttio` = 0), the array `parbor` is reset at every iteration.

Fourthly, when the user wants to modify the formulation of the wall slagging by the coal particles, it is then necessary to program the new laws in the subroutine `uslabo`.

CONSTRUCTION RULES OF A NEW PARTICLE/BOUNDARY INTERACTION

1. The real numbers `kx`, `ky`, `kz` provide the coordinates of the intersection point between the current particle trajectory and the interacting boundary face.
2. If the user wants to modify the particle position, it can be done directly *via* the arrays `ettp` and `ettpa`:

- new departure point of the current trajectory segment:
  `ettpa(npt,jxp)`, `ettpa(npt,jyp)`, `ettpa(npt,jzp)`
- new arrival point of the current trajectory segment:
  `ettp(npt,jxp)`, `ettp(npt,jyp)`, `ettp(npt,jzp)`

3. The particle and the fluid velocities may be modified according to the desired interaction *via* the arrays `vitpar` and `vitflu`, they **must not** be modified *via* `ettp` and `ettpa` in this subroutine.

4. For a given interaction, it is necessary to specify the key word `isuivi`:

   - `isuivi` = 0 if the particle does not need to be followed in the mesh after the interaction between its trajectory and the boundary face (by default, it is the case for `ientrl`, `isortl`, `idepo1`, `idepo2`);
   - `isuivi` = 1 to continue to follow the particle in the mesh after its interaction (by default, it is the case for `irebol` and `idepo3`). The value of `isuivi` may be a function of the particle and boundary state (for instance, `isuivi` = 0 or 1 depending on the physical properties for the interaction type `iencrl`).

5. The array zone `itepa(npt,jisor)`, containing the index-number of the cell where the particle is, must be updated. Generally:

   - `itepa(npt,jisor)` = `ifabor(kface)` when the particle stays in the calculation domain (`kface` is the number of the interacting boundary face).
   - `itepa(npt,jisor)` = 0 to eliminate definitively the particle from the calculation domain.

NOTE: ORDER OF THE NUMERICAL SCHEME AFTER A PARTICLE/BOUNDARY INTERACTION

When a particle interacts with a boundary face, the integration order of the associated stochastic equations is always a first-order, even if a second-order scheme is used elsewhere.

6.41.4 Option for particle cloning/merging: `uslaru`

Subroutine called every Lagrangian iteration.

An intervention in this subroutine is required if the particle cloning/merging option is activated *via* the key word `iroule`. The importance function `croule` must then be completed.
The aim of this technique is to reduce the number of particles to treat in the whole flow and to refine the description of the particle cloud only where the user wants to get volumetric statistics more accurate than in the rest of the calculation domain.
The values given to the importance function are strictly positive real numbers allowing to classify the zones according to their importance. The higher the value given to the importance function, the more important the zone.

For instance, when a particle moves from a zone of importance 1 to a zone of importance 2, it undergoes a cloning: the particle is replaced by two identical particles, whose statistical weight is the half of the initial particle. When a particle moves from a zone of importance 2 to a zone of importance 1, it undergoes a fusion: the particle survives to its passing through with a probability of 1/2. A random dawing is used to determine if the particle will survive or disappear.
In the same way, when a particle moves from a zone of importance 3 to a zone of importance 7, it undergoes a cloning. The particle is cloned in Int(7/3)=2 or Int(7/3)+1=3 particles with a probability of respectively 1-(7/3-Int(7/3))=2/3 and 7/3-Int(7/3)=1/3. If the particle moves from a zone of importance 7 to a zone of importance 3, it undergoes a fusion: it survives with a probability of 3/7.

WARNING: The importance function must be a strictly positive real number in every cell

6.41.5 Manipulation of particulate variables at the end of an iteration and user volumetric statistics: `uslast` and `uslaen`

`uslast`*: subroutine called at the end of every Lagrangian iteration*

`uslaen`*: subroutine called at every chronological output and every listing printing*

The subroutine `uslast` is called at the end of every Lagrangian iteration, it allows therefore the modification of variables related to the particles, or the extraction and preparation of data to display in the listing or the post-processing.

An intervention in both subroutines `uslast` and `uslaen` is required if supplementary user volumetric statistics are wanted.

User volumetric statistics:

The volumetric statistics are calculated by means of the array `statis`. Two situations may happen:

- the calculation of the statistics is not stationary: `statis` is reset at every Lagrangian iteration;
- the calculation of the statistics is stationary: the array `statis` is used to store cumulated values of variables, which will be averaged at the end of the calculation in the subroutine `uslaen`.

According to the user parameter settings, it may happen that during the same calculation, the statistics will be non-stationary in a first part and stationary in second part.

- User volumetric statistics: subroutine `USLAST`

  In this subroutine, the variable whose volumetric statistic is wanted is stored in the array `statis`. In the framework of stationary statistics, the average itself is calculated in the subroutine `uslaen`. This average is obtained through the division of the cumulated value by:
  - either the duration of the stationary statistics calculation stored in the variable `tstat`,
  - or the number of particles in statistical weight.

  This method of averaging is applied to every piece in the listing and to the post-processing outputs.

- User volumetric statistics: subroutine `USLAEN`

  In this subroutine is calculated the average corresponding to the cumulated value obtained in the subroutine `uslast`. This subroutine is also used for the standard volumetric statistics. Several examples are therefore described.

6.41.6 User stochastic differential equations: `uslaed`

Subroutine called every Lagrangian sub-step.

An intervention in this subroutine is required if supplementary user variables are added to the particle state vector (arrays `ettp` and `ettpa`).

The integration of the stochastic differential equations associated with supplementary particulate variables is done in this subroutine.
When the integration scheme of the stochastic differential equations is a first-order (`nordre` = 1), this 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 I*th* supplementary user variable (`nvls` in total) available in `ettp(nbpmax, jvls(i))` and in `ettpa(nbpmax,jvls(i))`, τ_ϕ is a quantity homogen to a characteristic time, and Π is a coefficient which may be expressed as a function of the other particulate variables contained in `ettp` and `ettpa`. 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 `ettpa`. 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 `ettpa`), then at the second call (correction step `nor` $= 2$) as a function of the predicted variables stored in the array `ettp`.

If necessary, the thermal characteristic time τ_c, whose calculation can be modified by the user in the subroutine `uslatc`, is stored for each particle in the part `tempct(nbpmax,1)` of the array `tempct`.

6.41.7 Particle relaxation time: `uslatp`

Subroutine called every Lagrangian sub-step.

An intervention in this subroutine is not obligatory.

In this subroutine, the particle relaxation time may be modified 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`.

6.41.8 Particle thermal characteristic time: `uslatc`

Subroutine called every Lagrangian sub-step.

An intervention in this subroutine is not obligatory.

In this subroutine, the particle thermal characteristic time may be modified according to the chosen correlation for the calculation of the Nusselt number.
The thermal characteristic time, modified or not by the user, is available in the zone `tempct(nbpmax,1)` of the array `tempct`.

7 Key word list

The key words are classified under headings. For each key word of the Kernel of *Code_Saturne*, the following data are given:

Variable name Type Allowed values [Default] O/C Level
Description
Potential dependences

- **Variable name**: Name of the variable containing the key word.
- **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 key word has one). In some cases, a non-allowed value is given (generally -999 or -10^{12}), to force the user to specify a value. If he does not do it, the code may:
  - automatically use a recommended value (for instance, automatical choice of the variables for which chronological records will be generated).
  - stop, if the key word is essential (for instance, value of the time step).
- **O/C**: Optional/Compulsory
  - O: optional key word, whose default value may be enough.
  - C: key word which must imperatively be specified (for instance, the time step).
- **Level**: L1, L2 or L3
  - L1 (level 1): the users will have to modify it in the framework of standard applications. The L1 key words are written in bold.
  - L2 (level 2): the users may have to modify it in the framework of advanced applications. The L2 key word are all optional.
  - L3 (level 3): the developers may have to modify it ; it keeps its default value in any other case. The L3 key word are all optional.
- **Description**: key word description, with its potential dependences.

The L1 key words can be modifed through the Graphical Use Interface or in the `usini1` subroutine. L2 and L3 key words can only be modified through the `usini1` subroutine, 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 key words which do not belong to the L1 level.

The alphabetical key word 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}.

7.1 Input-output

Notes

- Two different files can have neither the same unit number nor the same name.

7.1.1 "Calculation" files

General

`impgeo` i strictly positive integer [10] O L3
unit of the geometric file (if the Preprocessor is not used)
useful if and only if `SOLCOM = 1`

`ficgeo` c string of 6 characters [`geomet`] O L3
name of the geometric file (if the Preprocessor is not used)
useful if and only if `SOLCOM = 1`

`impstp` i strictly positive integer [12] O L3
unit of the calculation interactive stop file
always useful (because of the interactive character)

`ficstp` c string of 6 characters [`ficstp`] O L3
name of the calculation interactive stop file (see p.16)
always useful (because of the interactive characteristic)

`ficamo` c string of 13 characters [`suiamo`] O L3
name of the main upstream restart file. As with all restart files, its "format" (current or version 1.3) is automatically determined by the code.
useful if and only if `isuite = 1`

`ficamx` c string of 13 characters [`suiamx`] O L3
name of the auxiliary upstream restart file.
useful if and only if `isuite = 1`

`ficava` c string of 13 characters [`suiava`] O L3
name of the main downstream restart file
always useful

`ficavx` c string of 13 characters [`suiavx`] O L3
name of the auxiliary downstream restart file
always useful

1D wall thermal module

`ficmt1` c string of 13 characters [`t1damo`] O L3
name of the upstream restart file for the 1D wall thermal module.
useful if and only if `isuit1 = 1` and `nfpt1d`>0

`ficvt1` c string of 13 characters [`t1dava`] O L3
name of the upstream restart file for the 1D wall thermal module
useful if and only if `nfpt1d`>0

Vortex method for LES

`impmvo` i strictly positive integer [`impmvo`] O L3
unit of the upstream restart file for the vortex method
useful if and only if `isuivo = 1` and `ivrtex=1`

`ficmvo` c string of 13 characters [`voramo`] O L3
name of the upstream restart file for the vortex method
This is always a text file (this file has a different structure from the other restart files)
useful if and only if `isuivo = 1` and `ivrtex=1`

`impvvo` i strictly positive integer [`impvvo`] O L3
unit of the downstream restart file for the vortex method
useful if and only if `ivrtex=1`

`ficvvo` c string of 13 characters [`vorava`] O L3
name of the upstream restart file for the vortex method
This is always a text file (this file has a different structure from the other restart files)
useful if and only if `ivrtex=1`

`impdvo` i strictly positive integer [`impdvo`] O L3
unit of the `ficvor` data files for the vortex method. These files are text files. Their number and names are specified by the user in the `usvort` subroutine.
(Although it corresponds to an "upstream" data file, `impdvo` is initialized to 20 because, in case of multiple vortex entries, it is opened at the same time as the `ficmvo` upstream restart file, which already uses unit 11)
useful if and only if `ivrtex=1`

Radiation

`ficamr` c string of 13 characters [`rayamo`] O L3
name of the radiation upstream restart file.
useful if and only if `isuird = 1`

`ficavr` c string of 13 characters [`rayava`] O L3
name of the radiation downstream restart file
always useful in case of radiation modeling

Thermochemistry

`impfpp` i strictly positive integer [25] O L3
unit of the thermochemical data file
useful in case of gas or pulverised coal combustion or electric arc

`ficfpp` c string of 6 characters [`dp_tch`] O L3
name of the thermochemical data file. The launch script is designed to copy the user specified thermochemical data file in the temporary execution directory under the name `dp_tch`, for *Code_Saturne* to open it properly. Should the value of `ficfpp` be

changed, the launch script would have to be adapted.
useful in case of gas or pulverised coal combustion

`impjnf` i strictly positive integer [`impfpp`] O L3
unit of the JANAF data file
useful in case of gas or pulverised coal combustion

`ficjnf` c string of 5 characters [`JANAF`] O L3
name of the JANAF data file. The launch script is designed to copy the user specified JANAF data file in the temporary execution directory under the name `JANAF`, for *Code_Saturne* to open it properly. Should the value of `ficjnf` be changed, the launch script would have to be adapted.
useful in case of gas or pulverised coal combustion

LAGRANGIAN

`ficaml` c string of 6 characters [`lagamo`] O L3
name of the upstream restart file in case of Lagrangian modeling.
useful if and only if `isuila = 1`

`ficmls` c string of 13 characters [`lasamo`] O L3
name of the upstream restart file for the statistics in case of Lagrangian modeling.
useful if and only if `isuist = 1`

`ficavl` c string of 13 characters [`lagava`] O L3
name of the downstream restart file in case of Lagrangian modeling
always useful in case of Lagrangian modeling

`ficvls` c string of 6 characters [`lasava`] O L3
name of the downstream restart file for the statistics in case of Lagrangian modeling
useful in case of Lagrangian modeling with statistics

`impla1` i strictly positive integer [50] O L3
unit of a file specific to Lagrangian modeling
useful in case of Lagrangian modeling

`impla2` i strictly positive integer [51] O L3
unit of a file specific to Lagrangian modeling
useful in case of Lagrangian modeling

`impla3` i strictly positive integer [52] O L3
unit of a file specific to Lagrangian modeling
useful in case of Lagrangian modeling

`impla4` i strictly positive integer [53] O L3
unit of a file specific to Lagrangian modeling
useful in case of Lagrangian modeling

`impla5` ia strictly positive integer [54 to 68] O L3
units of files specific Lagrangian modeling, 15-dimension array
useful in case of Lagrangian modeling

7.1.2 Post-processing for *EnSight* or other tools

NOTES

- The format depends on the user choices.
- The post-processing files, directly generated by the Kernel through the FVM library, can be of the following formats: *Ensight Gold*, *MED_fichier* or *CGNS*. The use of the two latter formats depends on the installation of the corresponding external libraries.
- For each quantity (problem unknow, preselected numerical variable or preselected physical parameter), the user specifies if a post-processing output is wanted. The output frequency can be set.

`ichrvl` i 0 or 1 [1] O L3
indicates whether post-processing outputs are wanted (=1) or not (=0) on the 3D volume mesh
always useful

`ichrbo` i 0 or 1 [0] O L2
indicates whether post-processing outputs are wanted (=1) or not (=0) on the 2D boundary mesh
always useful

`ichrsy` i 0 or 1 [0] O L2
indicates whether post-processing outputs are wanted (=1) or not (=0) on the 2D boundary mesh patches coupled with the SYRTHES conjugate heat trabsfer code
always useful

`ichrmd` i 0, 1, 2, 10, 11 or 12 [0] O L2
indicates whether the post-processing geometry varies with time:
= 0: time independent
= 1: deforming or moving mesh
= 2: changing vertex coordinates and topology
= 10: time independent base, with time dependent nodal displacement field
= 11: deforming or moving mesh, plus nodal displacement field
= 12: changing vertex coordinates and topology, plus nodal displacement field

fmtchr c string of less than 32 characters [Ensight Gold] O L1
name of the output format, among the following:
- "Ensight Gold"
- "MED_fichier" (if available)
- "CGNS" (if available)

`optchr` c string of less than 96 characters [binary] O L2
options associated to the selected output format. The string is given as a series of key words, separated by a comma (and optional spaces). The key words are among the following:
- *text* for a text format (for *EnSight*)
- *binary* for a binary format (default choice)
- *big_endian* to force outputs to be in *big-endian* mode; this can be useful when using *ParaView*, which uses this mode by default.
- *discard_polygons* to prevent from exporting faces with more than four edges (which may not be recognised by some post-processing tools); such faces will therefore appear as "holes" in the post-processing mesh.

- *discard_polyhedra* to prevent from exporting elements which are neither tetrahedra, prisms, pyramids nor hexahedra (which may not be recognised by some post-processing tools); such elements will therefore appear as "holes" in the post-processing mesh
- *divide_polygons* to divide faces with more than four edges into triangles, so that any post-processing tool can recognise them
- *divide_polyhedra* to divide elements which are neither tetrahedra, prisms, pyramids nor hexahedra into simpler elements (tetrahedra and pyramids), so that any post-processing tool can recognise them
- *split_tensors* to export the components of a tensor variable as a series of independent variables (always the case for now)

ntchr i -1 or strictly positive integer [-1] O L1
output period for the post-processing
= -1: only at the end of the calculation
> 0: period (every `ntchr` time step)
always useful

ichrvr ia -999, 0 or 1 [-999] O L1
for each quantity defined at the cell centers (physical or numerical variable), indicator of whether it should be post-processed or not
= -999: not initialised. By default, the post-processed quantities are the unknowns (pressure, velocity, k, ε, R_{ij}, ω, φ, $\overline{f}$, scalars), density, turbulent viscosity and the time step if is not uniform
= 0: not post-processed
= 1: post-processed
useful if and only if the variable is defined at the cell centers: calculation variable, physical property (time step, density, viscosity, specific heat) or turbulent viscosity if `iturb(iphas)` $\geqslant 10$

ipstdv i integer $\geqslant 1$: see below [`ipstyp*ipstcl*ipstft`] O L1
indicates the data to post-process on the boundary mesh (the boundary mesh must have been activated with `ichrbo=1`). The value of `ipstdv` is the product of the following integers, depending on the variables that should be post-processed:
`ipstyp`: y^+ at the boundary
`ipstcl`: value of the variables at the boundary (using the boundary conditions but without reconstruction)
`ipstft`: thermal flux at the boundary ($W\,m^{-2}$), if a thermal scalar has been defined (`iscalt`)
For instance, with `ipstdv=ipstyp*ipstcl`, y^+ and the variables will be post-processed at the boundaries.
With `ipstdv=1`, none of these data are post-processed at the boundaries.
always useful if `ichrbo=1`

7.1.3 Chronological records of the variables on specific points

Standard use through Interface or `usini1`
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 produces 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).

ncapt i positive or null integer [0] O L1
total number of probes (limited to `ncaptm`=100)
always useful

xyzcap ra real numbers [0.0] O L1
3D-coordinates of the probes
the coordinates are written: `xyzcap(i,j)`, with $i = 1$, 2 or 3 and $j \leqslant$ `ncapt`
useful if and only if `ncapt` > 0

ihisvr ia -999, -1 or positive or null integer [-999] O L1
number `ihisvr(n, 1)` and index-numbers `ihisvr(n, j>1)` of the record probes to be used for each variable, *i.e.* calculation variable or physical property defined at the cell centers. With `ihisvr(n, 1)`=-999 or -1, `ihisvr(n, j>1)` is useless.

- `ihisvr(n, 1)`: number of record probes to use for the variable N

= -999: by default: chronogical records are generated on all the probes if N is one of the main variables (pressure, velocity, turbulence, scalars), the local time step or the turbulent viscosity. For the other quantities, no chronological record is generated.

= -1: chronological records are produced on all the probes

= 0: no chronological record on any probe

$>$ 0: chronological record on `ihisvr(n, 1)` probes to be specified with `ihisvr(n, j>1)`

always useful, must be inferior or equal to `ncapt`

- `ihisvr(n, j>1)`: index-numbers of the probes used for the variable n (with $j \leqslant$ `ihisvr(n,1)+1`)

= -999: by default: if `ihisvr(n, 1)` $\neq$ -999, the code stops. Otherwise, refer to the description of the case `ihisvr(n, 1)`=-999

useful if and only if `ihisvr(n, 1)` > 0
The condition `ihisvr(n, j)` $\leqslant$`ncapt` must be respected.
For an easier use, it is recommended to simply specify `ihisvr(n,1)`=-1 for all the interesting variables.

`imphis` ia strictly positive integer [30 and 31] O L3
working units for the production of chronological record files by the Kernel
useful if and only if chronological files are produced (*i.e.* there is n for which `ihisvr(n, 1)` $\neq 0$)

`emphis` c string of less than 80 characters [`./`] O L3
directory in which the potential chronological record files generated by the Kernel will be written (path related to the execution directory)
it is recommended to keep the default value and, if necessary, to modify the launch script to copy the files in the alternate destination directory
useful if and only if chronological record files are generated (*i.e.* there is `n` for which `ihisvr(n, 1)` $\neq 0$)

`exthis` c string of less than 80 characters [hst] O L3
extension of the chronological record files
useful if and only if chronological record files are generated (*i.e.* there is `n` for which `ihisvr(n, 1)` $\neq 0$)

nthist i -1 or strictly positive integer [1 or -1] O L1
output period of the chronological record files
= -1: no output
> 0: period (every `nthist` time step)
The default value is -1 if there is no chronological record file to generate (if there is no probe, `ncapt` = 0, or if `ihisvr(n, 1)`=0 for all the variables) and 1 otherwise
If chronological records are generated, it is usually wise to keep the default value `nthist`=1, in order to avoid missing any high frequency evolution (unless the total number of time steps is much too big)
useful if and only if chronological record files are generated (*i.e.* there are probes (`ncapt`>0) there is `n` for which `ihisvr(n, 1)` $\neq$ 0)

`nthsav` i -1 or positive or null integer [0] O L3
saving period the chronological record files (they are first stored in a temporary file and then saved every `nthsav` time step)
= 0: by default (4 times during a calculation)
= -1: saving at the end of the calculation
> 0: period (every `nthsav` time step)
During the calculation, the user can read the chronological record files in the execution directory when they have been saved, *i.e.* at the first time step, at the tenth time step and when the time step number is a multiple of `nthsav` (multiple of `(ntmabs-ntpabs)/4` if `nthsav`=0)
Note: using the ***`ficstp`*** *file allows to update the value of* `ntmabs`*. Hence, if the calculation is at the time step n, the saving of the chronological record files can be forced by changing* `ntmabs` *to* `ntpabs`*+4(n+1) using* ***`ficstp`****; after the files have been saved,* `ntmabs` *can be reset to its original value, still using* ***`ficstp`****.*
useful if and only if chronological record files are generated (*i.e.* there are probes (`ncapt`>0) there is `n` for which `ihisvr(n, 1)` $\neq$ 0)

NON-STANDARD USE THROUGH `USHIST`
(see p.72)

`impush` ia strictly positive integer [33 to 32+`nushmx`=49] O L3
units of the user chronological record files
useful if and only if the subroutine `ushist` is used

`ficush` ca strings of 13 characters [`ush*` or `ush*.n_*`] O L2
names of the user chronological record files. In the case of a non-parallel calculation, the suffix applied the file name is a three digit number: `ush001`, `ush002`, `ush003`... In the case of a parallel-running calculation, the processor index-number is added to the suffix. For instance, for a calculation running on two processors: `ush001.n_0001`, `ush002.n_0001`, `ush003.n_0001`... and `ush001.n_0002`, `ush002.n_0002`, `ush003.n_0002`...
The opening, closing, format and location of these files must be managed by the user.
useful if and only if the subroutine `ushist` is used

7.1.4 Time averages

The code allows the calculation of time averages of the type $< f_1 * f_2 ... * f_n >$. The variables f_i (defined at the cell centers) which may be taken into account are the followings:

- the solved calculation variables (velocity, pressure ...),

- the auxiliary variables from the array `propce` (density and physical properties when they are variable in space).

The averages are treated like auxiliary variables defined at the cell centers and stored in the `propce` array. The standard post-processing actions may therefore be activated, like the writing in the listing or the output of result files (EnSight, MED, ...). However, if the user wants to manipulate the averages in a more advanced way, it is recommended to refer first to the user subroutines `usproj` and `usvpst` which provide examples. Indeed, the `propce` array does not contain the time averages directly, but only the cumulated value of the product $f_1 * f_2 ... * f_n$ of the selected variables f_i. The division by the cumulated duration is done only before the writing of the results. See also page 49.

To calculate p time averages of the type $< f_1 * f_2 ... * f_{n(imom)} >$, the user must:

- make sure that $p \leqslant$`nbmomx` (do not overstep the maximum number of averages),
- make sure that $n(imom) \leqslant$`ndgmox` for every average `imom` (do not overstep the maximum degree, *i.e* the maximum number of variables which may compose an average),
- define every average `imom` ($1 \leqslant$`imom`$\leqslant p$, without skipping any index-number) by marking out the n(`imom`) variables which form it by means of the array `idfmom(ii,imom)` (with $1 \leqslant$`ii`$\leqslant$n(`imom`)),
- define for each average `imom` the time step number at which the calculation of the cumulated value must begin, by means of the array `ntdmom(imom)`.

The total number of averages (p=`nbmomt`) is automatically determined by the code from the values of `idtmom`. The user must not specify specify it.

idfmom ia 0, ± variable index-number [0] O L1
Index-number of the variables composing a time average of the type $< f_1 * f_2 ... * f_n >$. For every time average `imom` to calculate:
- if `idfmom(ii,imom)` is positive, it refers to the index-number of a solved variable (stored in the array `rtp`), like for instance a velocity component (`iu(iphas)`, `iv(iphas)`, `iw(iphas)`) or the pressure (`ipr(iphas)`)
- if `idfmom(ii,imom)` is negative, it refers to the index-number of an auxiliary variable (stored in `propce`), like for instance the density (`idfmom(ii,imom)=-irom(iphas)`)

useful if and only if the user wants to calculate time averages

ntdmom ia integer [-1] O L1
For every average `imom` to calculate, absolute time step number at which the calculation should begin. The value -1 means "never". Every strictly negative value (in particular -1) will considered an error and cause the calculation to stop (because the user is supposed to want to calculate the averages he has defined)
useful if and only if the user wants to calculate time averages

imoold ia -2, $1 \leqslant$ integer $\leqslant$ `jbmomt` [-2] O L1
Correspondence table of the averages in the case of a calculation restart. In this case, for every average `imom` in the current calculation ($1 \leqslant$`imom`$\leqslant$`nbmomx`), `imoold(imom)` gives the index-number of the corresponding average in the previous calculation (in which `jbmomt` averages were calculated).
- if `imoold(imom)` = -2, the user lets the code automatically determine the correspondence. By default, the average `ii` in the current calculation will correspond to the average `ii` in the previous calculation, if it existed. Otherwise, `ii` will be a new average.
- if `imoold(imom) = -1`, the average is reset to zero.
- if `imoold(imom) = kk`, the average `imom` will correspond to the average

`kk=imoold(imom)` in the previous calculation.
useful if and only if the user wants to calculate averages. Allows to add or suppress some averages, to reset them, to change their order, ...
Warning: if the calculation is not a restart, `imoold` *must not be specified (its value must remain -2)*

7.1.5 Others

impusr ia strictly positive integer [70 to 69+nusrmx=79] O L3
unit numbers for potential user specified files
useful if and only if the user needs files (therefore always useful, by security)

ficusr ca string of 13 characters [usrf* or usrf*.n_*] O L1
name of the potential user specified files. In the case of a non-parallel calculation, the suffix applied the file name is a two digit number: from `usrf01` to `usrf10`. In the case of a parallel-running calculation, the four digit processor index-number is added to the suffix. For instance, for a calculation running on two processors: from `usrf01.n_0001` to `usrf10.n_0001` and from `usrf01.n_0002` to `usrf10.n_0002`. The opening, closing, format and location of these files must be managed by the user.
useful if and only if the user needs files (therefore always useful, by security)

ilisvr ia -999, 1 or 0 [-999] O L1
for every quantity (variable, physical or numerical property ...), indicator concerning the writing in the execution report file
= -999: automatically converted into 1 if the concerned quantity is one of the main variables (pressure, velocity, turbulence, scalar), the density, the time step if `idtvar` $\neq 0$ or the turbulent viscosity. Otherwise converted into 0.
= 1: writing in the execution listing.
= 0: no writing.
always useful

iwarni ia integer [0] O L1
`iwarni(ivar)` characterises the level of detail of the outputs for the variable `ivar` (from 1 to `nvar`). The quantity of information increases with its value.
Impose the value 0 or 1 for a reasonable listing size. Impose the value 2 to get a maximum quantity of information, in case of problem during the execution.
always useful

nomvar ca string of less than 80 characters [“”] O L1
name of the variables (unknowns, physical properties ...): used in the execution listing, in the post-processing files, etc.
“”: not initialised (the code chooses the manes by default)
It is recommended not to define variable names of more than 8 characters, to get a clear execution listing (some advanced writing levels take into account only the first 8 characters).
always useful

ntlist i -1 or strictly positive integer [1] O L1
writing period in the execution report file
= -1: no writing
$>$ 0: period (every `ntlist` time step)

The value of `ntlist` must be adapted according to the number of iterations carried out in the calculation. Keeping `ntlist` to 1 will indeed provide a maximum volume of information, but if the number of time steps is too large, the execution report file might become too big and unusable (problems with disk space, memory problems while opening the file with a text editor, problems finding the desired information in the file, ...).
always useful

`ntsuit` i -1, 0 or positive or null integer [0] O L3
saving period of the restart files
= -1: only at the end of the calculation
= 0: by default (four times during the calculation)
> 0: period
always useful

7.2 Numerical options

7.2.1 Calculation management

`iecaux` i 0 or 1 [1] O L2
indicates the writing (=1) or not (=0) of the auxiliary calculation restart file
always useful

`ileaux` i 0 or 1 [1] O L2
indicates the reading (=1) or not (=0) of the auxiliary calculation restart file
useful if and only if `isuite`=1

inpdt0 i 0 or 1 [0] O L1
indicates the calculation mode: 1 for a zero time step control calculation, *i.e.* without solving the transport equations, and 0 for a standard calculation.
In case of a calculation using the control mode (`inpdt0`=1), when the calculation is not a restart, the equations are not solved, but the physical properties and the boundary conditions are calculated. When the calculation is a restart, the physical properties and the boundary conditions are those read from the restart file (note: in the case of a second-order time scheme, the mass flow is modified as if a normal time step was realised: the mass flow generated in an potential post-processing is therefore not the mass flow read from the restart file).
In the control mode (`inpdt0`=1), the variable `ntmabs` is not used.
In the standard mode (`inpdt0`=0), the code solves the equations at least once, even if `ntmabs`=0.
always useful

isuite i 0 or 1 [0] C L1
indicator of a calculation restart (=1) or not (=0)
always useful

`ntcabs` i integer [`ntpabs`] O L3
current time step number
always useful
`ntcabs` is initialised and updated automatically by the code, its value is not to be modified by the user

ntmabs i integer > ntpabs [10] C L1
number of the last time step after which the calculation stops. It is an absolute number: for the restart calculations, ntmabs takes into account the number of time steps of the previous calculations. For instance, after a first calculation of 3 time steps, a restart file of 2 time steps is realised by setting ntmabs=3+2=5
always useful

ntpabs i integer [0, read] O L3
number of the last time step in the previous calculation. In the case of a restart calculation, ntpabs is read from the restart file. Otherwhise it is initialised to 0
always useful
ntpabs is initialised automatically by the code, its value is not to be modified by the user

tmarus r -1 or strictly positive real [-1] O L3
margin in seconds on the remaining CPU time which is necessary to allow the calculation to stop automatically and write all the required results (for the machines having a queue manager)
= -1: calculated automatically
> 0: margin defined by the user
always useful, but the default value should not be changed unless absolutely necessary.

ttcabs r positive or null real number [ttpabs] O L3
physical simulation time at the current time step. For the restart calculations, ttcabs takes into account the physical time of the previous calculations.
If the time step is uniform (idtvar=0 or 1), ttcabs increases of dt (value of the time step) at each iteration. If the time step is non-uniform (idtvar=2), ttcabs increases of dtref at each time step.
always useful
ttcabs is initialised and updated automatically by the code, its value is not to be modified by the user

ttpabs r positive or null real number [0, read] O L3
simulation physical time at the last time step of the previous calculation. In the case of a restart calculation, ttpabs is read from the restart file. Otherwhise it is initialised to 0.
always useful
ttcabs is initialised automatically by the code, its value is not to be modified by the user

7.2.2 Scalar unknowns

iscold ia -999, 1⩽ integer ⩽ jscal [-999] O L1
correspondence table of the scalars in the case of a calculation restart. For a calculation restart with nscal scalars, iscold(iscal) gives, for every scalar iscal of the current calculation (1⩽iscal⩽nscal), the index-number of the corresponding scalar in the previous calculation (in which jscal scalars were taken into account).
iscold(iscal) = -999: the code automatically determines the correspondence. By default, the following rules are applied:
- the user scalar ii of the current calculation is initialised by the the user scalar ii of the previous calculation, if this scalar existed already (otherwise, ii is a new scalar).

- the particular physics scalar `jj` is initialised by the particular physics scalar `jj` of the previous calculation if this scalar existed already (otherwise, `jj` is a new scalar).

`iscold(iscal) = kk`: the scalar `iscal` (user or particular physics scalar) is initialised by the scalar `kk=iscold(iscal)` of the previous calculation.
always useful. Allows to add or remove some scalars, to change the solving order, to change the physics, ...

nscaus i 0⩽ integer ⩽ `nscmax` [0] O L1
number of user scalars solutions of an advection equation
always useful

iscavr ia 0, 1 ⩽ integer ⩽ `nscal` [0] O L1
if the scalar `iscal` is the average of the square of the fluctuations of a scalar `kk`, then `iscavr(iscal)=kk`. Otherwise `iscavr(iscal)`=0. For `iscal` and `kk`, the user can only use index-numbers refering to user scalars (⩽ `nscaus`).
always useful

`iphsca` ia 1 ⩽ integer ⩽ `nphas` [0] O L3
for every scalar `iscal`, `iphsca(iscal)` is the index-number of the associated phase
always useful

iscalt ia -1 or integer > 0 [-1] O L1
for every phase `iphas`, `iscalt(iphas)` is the index-number of the scalar representing the temperature or the enthalpy. If `iscalt(iphas)`=-1, no scalar represents the temperature nor the enthalpy. When a specific physics module is activated (gas combustion, pulverised coal, electricity or compressible), the user must not modify `iscalt` (the choice is made automatically)[40].
useful if and only if `nscal` ⩾ 1

iscsth ia -1, 0, 1, 2 or 3 [-10] O L1
type of scalar
= -10: not specified. By default, the code chooses `iscsth(iscal)`=0 for the scalars apart from `iscalt(iphas)`
= -1: temperature in degrees Celsius (use only in case of radiation modeling)
= 0: passive scalar
= 1: temperature (in Kelvin if the radiation modeling is activated)
= 2: enthalpy
= 3: total energy (this value is automatically chosen by the code when using the compressible module, it must never be used otherwise and must never be specified by the user)
useful if and only if `nscal` ⩾ 1. The distinction between `iscsth(iscal)` = -1 or 1 (respectively degrees Celsius or Kelvin) is useful only in case of radiation modeling. For calculations without radiation modeling, use `iscsth(iscal)`=1 for the temperature. When a particular physics module is activated (gas combustion, pulverised coal, electricity or compressible), the user must not modify `iscsth` (the choice is made automatically: the solved variable is the enthalpy or the total energy).
It is also reminded that, in the case of a coupling with SYRTHES, the solved thermal variable should be the temperature (`iscsth(iscalt(iphas))`=1 or -1). More precisely, everything is designed in the code to allow for the running of a calculation coupled with SYRTHES with the enthalpy as thermal variable (the correspondence

[40] in the case of the compressible module, `iscalt` does not correspond to the temperature nor enthalpy but to the total energy

and conversion is then specified by the user in the subroutine `usthht`). However this case has never been used in practice and has therefore not been tested. With the compressible model, it is possible to carry out calculations coupled with SYRTHES, although the thermal scalar represents the total energy and not the temperature.

`iclvfl` ia -1, 0, 1 or 2 [-1] O L3
for every scalar `iscal` representing the average of the square of the fluctuations of another scalar `ii=iscavr(iscal)` (noted f), indicator of the clipping method
= -1: no clipping because the scalar does not represent the average of the square of the fluctuations of another scalar
= 0: clipping to 0 for lower values
= 1: clipping to 0 for lower values and to $(f - f_{min})(f_{max} - f)$ for higher values, where f is the associated scalar, f_{min} and f_{max} its minimum and maximum values specified by the user (*i.e.* `scamin(ii)` and `scamax(ii)`)
= 2: clipping to `max(0,scamin(iscal))` for lower values and to `scamax(iscal)` for higher values. `scamin` and `scamax` are limits specified by the user
useful for the scalars `iscal` for which `iscavr(iscal)`>0.

`itbrrb` i 0 or 1 [0] O L3
Reconstruction (=1) or not (=0) of the temperature, enthalpy or total energy value in the boundary cells. Useful in the case of coupling with SYRTHES and with radiation.

`icpsyr` ia -999,0,1 [-999] O L3
For each scalar `iscal`, `icpsyr(iscal)` indicates if it is coupled with SYRTHES (=1) or not (=0). There can be only one coupled scalar per calculation.
=-999: by default
• `icpsyr(iscal)`=1 for the thermal scalar `iscal=(iscalt(iphas))` when a coupling with SYRTHES has been specified in the Interface or the launch script
• `icpsyr(iscal)`=0 otherwise
= 0: the scalar `iscal` is not coupled with SYRTHES
= 1: the scalar `iscal` is coupled with SYRTHES
useful in case of coupling with SYRTHES

7.2.3 Definition of the equations

`istat` ia 0 or 1 [1 or 0] O L2
for each unknown `ivar` to calculate, indicates if non-stationary terms are present (`istat(ivar)`=1) or not (0) in the matrices.
By default, `istat` is set to 0 for the pressure (variable `ivar=ipr(iphas)`) or $\overline{f}$ in v2f modeling (variable `ivar=ifb(iphas)`) and set to 1 for the other unknowns.
useful for all the unknowns

`iconv` ia 0 or 1 [1 or 0] O L2
for each unknown `ivar` to calculate, indicates if the convection is taken into account (`iconv(ivar)`=1) or not (0).
By default, `iconv` is set to 0 for the pressure (variable `ivar=ipr(iphas)`) or $\overline{f}$ in v2f modeling (variable `ivar=ifb(iphas)`) and set to 1 for the other unknowns.
useful for all the unknowns

`idiff` ia 0 or 1 [1] O L2
for each unknown `ivar` to calculate, indicates if the diffusion is taken into account

(`idiff(ivar)`=1) or not (0)
useful for all the unknowns

`idifft` ia 0 or 1 [1] O L3
for each unknown `ivar` to calculate, when diffusion is taken into account (`idiff(ivar)`=1), `idifft(ivar)` indicates if the turbulent diffusion is taken into account (`idifft(ivar)`=1) or not (0)
useful for all the unknowns

`idircl` ia 0 or 1 [1 or 0] O L3
for each unknown `ivar` to calculate, indicates whether the diagonal of the matrix should be slightly shifted (`idircl(ivar)`=1) or not (0) if there is no Dirichlet boundary condition and if `istat`=0. Indeed, in such a case, the matrix for the general advection/diffusion equation is singular. A slight shift in the diagonal will make it invertable again.
By default, `idircl` is set to 1 for all the unknowns, except $\overline{f}$ in v2f modeling, since its equation contains another diagonal term that ensures the regularity of the matrix.
useful for all the unknowns

`ivisse` ia 0 or 1 [1] O L3
for each phase `iphas`, indicates whether the source terms in transposed gradient and velocity divergence should be taken into account in the momentum equation. In the compressible module, these terms also account for the volume viscosity (cf. `viscv0` and `iviscv`):
$\partial_i \left[(\kappa - 2/3\,(\mu + \mu_t))\partial_k U_k\right] + \partial_j \left[(\mu + \mu_t)\partial_i U_j\right]$
= 0: not taken into account
= 1: taken into account
always useful

7.2.4 Definition of the time advancement

idtvar i -1, 0, 1, 2 [0] O L1
type of time step
= 0: constant in time and spatially uniform
= 1: variable in time and spatially uniform
= 2: variable in time and in space
= -1: steady-state algorithm
If the numerical scheme is a second-order in time, only the option 0 is allowed.
always useful

`iptlro` i 0 or 1 [0] O L2
when density gradients and gravity are present, a local thermal time step can be calculated, based on the Brunt-Vaissala frequency. In numerical simulations, it is usually wise for the time step to be lower than this limit, otherwise numerical instabilities may appear
`iptlro` indicates whether the time step should be limited to the local thermal time step (=1) or not (=0)
when `iptlro`=1, the listing shows the number of cells where the time step has been clipped due to the thermal criterium, as well as the maximum ratio between the time step and the maximum thermal time step. If `idtvar`=0, since the time step is fixed and cannot be clipped, this ratio can be larger than 1[41]. When `idtvar`>0, this ratio

[41] it is then the user's choice to decide whether he should diminish DTREF or not

will be smaller than 1, except if the constraint `dtmin` has prevented the code from reaching a sufficiently low value for `dt`
useful when density gradients and gravity are present

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 unrecommended
useful if and only if `nscal` $\geqslant 1$

`coumax` r strictly positive real number [1] O L1
target local or maximum Courant number in case of non-constant time step
useful if `idtvar` $\neq 0$

`foumax` r strictly positive real number [10] O L1
target local or maximum Fourier number in case of non-constant time step
useful if `idtvar` $\neq 0$

dtref r strictly positive real number [`-grand*10`] C L1
reference time step
always useful.
It is the time step value used in the case of a calculation run with a uniform and constant time step, *i.e.* `idtvar`=0 (restart calculation or not). It is the value used to initialise the time step in the case of an initial calculation (`isuite`=0) run with a non-constant time step (`idtvar`=1 or 2). It is also the value used to initialise the time step in the case of a restart calculation (`isuite`=1) in which the type of time step has been changed (for instance, `idtvar`=1 in the new calculation and `idtvar`=0 or 2 in the previous calculation): see `usiniv`

`dtmin` r positive or null real number [`0.1*dtref`] O L2
lower limit for the calculated time step when non-constant time step is activated
useful if `idtvar` $\neq 0$

`dtmax` r strictly positive real number [`1000*dtref`] O L2
upper limit for the calculated time step when non-constant time step is activated
useful if `idtvar` $\neq 0$

`varrdt` r strictly positive real number [0.1] O L3
maximum allowed relative increase in the calculated time step value between two succesive time steps (to ensure stability, any decrease in the time step is immediate and without limit)
useful if `idtvar` $\neq 0$

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.

7.2.5 Turbulence

iturb ia 0, 10, 20, 21, 30, 31, 40, 41, 50 or 60 [-999] O L1
for each phase `iphas`, indicator of the turbulence model `iturb(iphas)`
= -999: not initalised. This value is not allowed and must be modified by the user
= 0: laminar
= 10: mixing length (not valided)
= 20: $k-\varepsilon$
= 21: $k-\varepsilon$ with linear production (Laurence & Guimet)
= 30: $R_{ij}-\varepsilon$ "standard" LRR (Launder, Reece & Rodi)
= 31: $R_{ij}-\varepsilon$ SSG (Speziale, Sarkar & Gatski)
= 40: LES (Smagorinsky model)
= 41: LES (dynamic model)
= 50: v2-f, φ-model version
= 60: $k-\omega$, SST version
always useful

The $k-\varepsilon$ (standard and linear 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 sublayer (at the wall, $y^+ > 2.5$ is required as a minimum, and preferably between 30 and 100)[42]. If the mesh does not respect this condition, the results may be biased (particularly if thermal processes are involved). Using scalable wall-functions (cf. key word `ideuch`) 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 sublayer ($y^+ < 1$).
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 key word `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 sublayer, 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 neighborhood (`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 neighborhood).

`ideuch` ia 0, 1 or 2 [0 or 1] O L2
for each phase `iphas`, indicates the type of wall function is used for the velocity

[42] 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.

boundary conditions on a frictional wall.
= 0: one-scale model
= 1: two-scale model
= 2: scalable wall function
`ideuch` is initialised to 0 for `iturb(iphas)`=0, 10, 40 or 41 (laminar, mixing length, LES).
`ideuch` is initialised to 1 for `iturb(iphas)`=20, 21, 30, 31 or 60 ($k-\varepsilon$, $R_{ij}-\varepsilon$ LRR, $R_{ij}-\varepsilon$ SSG and $k-\omega$ SST models).
The v2f model (`iturb(iphas)`=50) is not designed to use wall functions (the mesh must be "low Reynolds").
The value `ideuch(iphas)`=1 is not compatible with `iturb(iphas)`=0, 10, 40 or 41 (laminar, mixing length and LES).
Concerning the $k-\varepsilon$ and $R_{ij}-\varepsilon$ models, the two-scales model is usually at least as satisfactory as the one-scale model.
The scalable wall function allows to virtually "shift" the wall when necessary in order to be always in a logarithmic layer. It is used to make up for the problems related to the use of High-Reynolds models on very refined meshes.
useful if `iturb(iphas)` is different from 50

`ilogpo` ia 0 or 1 [1] O L3
for each phase `iphas`, type of wall function used for the velocity: power law (`ilogpo(iphas)`=0) or logarithmic law (`ilogpo(iphas)`=1)
always useful

`ypluli` ra real number > 0 [`1/xkappa, 10.88`] O L3
for each phase `iphas`, limit value of y^+ for the viscous sublayer
`ypluli` depends on the chosen wall function: it is initialised to 10.88 for the scalable wall function (`ideuch(iphas)`=2), otherwise it is initialised to $1/\kappa \approx 2,38$
In LES, `ypluli` is taken by default to be 10.88
always useful

$k-\varepsilon$, $k-\varepsilon$ WITH LINEAR PRODUCTION, V2-F AND $k-\omega$ SST

`igrake` ia 0 or 1 [1] O L1
for each phase `iphas`, indicates if the terms related to gravity in the equations of k and ε or ω are taken into account (`igrake(iphas)`=1) or not (0)
useful if and only if `iturb(iphas)` = 20, 21, 50 or 60, (`gx, gy, gz`) $\neq$ (0,0,0) and the density is not uniform

`igrhok` ia 0 or 1 [0] O L2
for each phase `iphas`, indicates if the term $\frac{2}{3}\underline{\text{grad}}\ \rho k$ is taken into account (`igrhok(iphas)`=1) or not (0) in the velocity equation
useful if and only if `iturb(iphas)` = 20, 21, 50 or 60.
This term may generate non-physical velocities at the wall. When it is not explicitely taken into account, it is implicitely included into the pressure.

`ikecou` ia 0 or 1 [0 or 1] O L3
for each phase `iphas`, indicates if the coupling of the source terms of k and ε or k and ω is taken into account (`ikecou(iphas)`=1) or not (0)
if `ikecou`=0 in $k-\varepsilon$ model, the term in ε in the equation of k in made implicit
`ikecou(iphas)` is initialised to 0 if `iturb(iphas)` = 21 or 60, and to 1 if

`iturb(iphas)`= 20
`ikecou(iphas)`=1 is forbidden when using the v2f model (`iturb(iphas)`=50)
useful if and only if `iturb(iphas)` = 20, 21 or 60 ($k-\varepsilon$ and $k-\omega$ models)

`relaxk` ra $0 \leqslant$ real $\leqslant 1$ [0.7] O L3
for each phase `iphas`, relaxation coefficient of the turbulent variables (k and ε or ω) when `ikecou(iphas)` = 0. If `ikecou(iphas)`=1, `relaxk` is not used, whatever its value may be.
useful if and only if `iturb(iphas)` = 20, 21, 50 or 60 and `ikecou(iphas)`=0 ($k-\varepsilon$, v2f or $k-\omega$ models without coupling)

`iclkep` ia 0 or 1 [0] O L3
for each phase `iphas`, indicates the clipping method used for k and ε, for the $k-\varepsilon$ and v2f models
= 0: clipping in absolute value
= 1: clipping from physical relations
useful if and only if `iturb(iphas)` = 20, 21 or 50 ($k-\varepsilon$ and v2f models). The results obtained with the method corresponding to `iclkep(iphas)`=1 showed in some cases a substantial sensitivity to the values of the length scale `almax(iphas)`.
The option `iclkep(iphas)`=1 is therefore not recommended, and, if chosen, must be used cautiously.

$R_{ij}-\varepsilon$ (LRR and SSG)

`iclptr` ia 0 or 1 [0] O L3
for each phase `iphas`, indicates if R_{ij} is made partially implicit (`iclptr(iphas)`=1) or not (0) in the wall boundary conditions.
useful if and only if `iturb(iphas)` = 30 or 31 ($R_{ij}-\varepsilon$ model)

`iclsyr` ia 0 or 1 [0] O L3
for each phase `iphas`, indicates if R_{ij} is made partially implicit (`iclsyr(iphas)`=1) or not (0) in the symmetry boundary conditions.
useful if and only if `iturb(iphas)` = 30 or 31 ($R_{ij}-\varepsilon$ model)

`idifre` ia 0 or 1 [1] O L3
for each phase `iphas`, complete (`idifre(iphas)=1`) or simplified (0) taking into account of the diagonals of the diffusion tensors of R_{ij} and ε, for the LLR model.
useful if and only if `iturb(iphas)` = 30 (LLR $R_{ij}-\varepsilon$ model)

`igrari` ia 0 or 1 [1] O L1
for each phase `iphas`, indicates if the terms related to gravity are taken into account (`igrari(iphas)`=1) or not (0) in the equations of $R_{ij}-\varepsilon$.
useful if and only if `iturb(iphas)` = 30 or 31 and (`gx, gy, gz`) $\neq$ (0,0,0) ($R_{ij}-\varepsilon$ model with gravity) and the density is not uniform

`irijec` ia 0 or 1 [0] O L2
for each phase `iphas`, indicates if the wall echo terms in $R_{ij}-\varepsilon$ LRR model are taken into account (`irijec(iphas)`=1) or not (0).
useful if and only if `iturb(iphas)` = 30 ($R_{ij}-\varepsilon$ LRR).
It is not recommended to take these terms into account: they have an influence only

near the walls, their expression is hardly justifiable according to some authors and, in the configurations studied with *Code_Saturne*, they did not bring any improvement in the results.
In addition, their use induces an increase in the calculation time.
The wall echo terms imply the calculation of the distance to the wall for every cell in the domain. See `icdpar` for potential restrictions due to this.

`irijnu` ia 0 or 1 [0] O L3
for each phase `iphas`, addition (`irijnu(iphas)`=1) or not (0) of a turbulent viscosity in the matrix of the incermental system solved for the velocity in $R_{ij}-\varepsilon$ models. The goal is to improve the stability of the calculation. The usefulness of `irijnu(iphas)`=1 has however not been clearly demonstrated.
Since the system is solved in incremental form, this extra turbulent viscosity does not change the final solution for steady flows. However, for unsteady flows, the parameter `nswrsm` should be increased.
useful if and only if `iturb(iphas)` = 30 or 31 ($R_{ij}-\varepsilon$ model).

`irijrb` ia 0 or 1 [0] O L3
for each phase `iphas`, reconstruction (`irijrb(iphas)`=1) or not (0) of the boundary conditions at the walls for R_{ij} and ε.
useful if and only if `iturb(iphas)` = 30 or 31 ($R_{ij}-\varepsilon$ model)

LES

`ivrtex` i 0 or 1 [0] O L1
activates (=1) or not (=0) the generation of synthetic turbulence at the different inlet boundaries with the LES model (generation of unsteady synthetic eddies)
useful if `iturb(iphas)`=40 or 41
this key word requires the completion of the routine `usvort`

`isuivo` i 0 or 1 [isuite] O L1
for the vortex method, indicates whether the synthetic vortices at the inlet should be initialised (=0) or read form the restart file `ficmvo`.
useful if `iturb(iphas)`=40 or 41 and `ivrtex`=1

`idries` ia 0 or 1 [0,1] O L2
for each phase `iphas`, `idries(iphas)` activates (1) or not (0) the van Driest wall-damping for the Smagorinsky constant (the Smagorinsky constant is multiplied by the damping function $1-e^{-y^+/\texttt{cdries(iphas)}}$, where y^+ designates the adimensional distance to the nearest wall). The default value is 1 for the Smagorinsky model and 0 for the dynamic model.
the van Driest wall-damping requires the knowledge of the distance to the nearest wall for each cell in the domain. Refer to key word `icdpar` for potential limitations
useful if and only if `iturb(iphas)` = 40 or 41

`cdries` ra real number >0 [26] O L3
for each phase `iphas`, `cdries(iphas)` is the constant appearing in the van Driest damping function applied to the Smagorinsky constant: $1-e^{-y^+/\texttt{cdries(iphas)}}$
useful if and only if `iturb(iphas)` = 40 or 41

`csmago` ra real number > 0 [0.065] O L2
for each phase `iphas`, `csmago(iphas)` is the Smagorinsky constant used in the Smagorinsky model for LES
the sub-grid scale viscosity is calculated by $\mu_{sg} = \rho C_{smago}^2 \bar{\Delta}^2 \sqrt{2\bar{S}_{ij}\bar{S}_{ij}}$ where $\bar{\Delta}$ is the width of the filter and $\bar{S}_{ij}$ the filtered strain rate
useful if and only if `iturb(iphas)` $= 40$

`smagmx` ra real number > 0 [10*csmago] O L3
for each phase `iphas`, `smagmx(iphas)`^{2} is the maximum allowed value for the variable C appearing in the LES dynamic model (the "square" comes from the fact that the variable of the dynamic model corresponds to the square of the constant of the Smagorinsky model). Any larger value yielded by the calculation procedure of the dynamic model will be clipped to `smagmx(iphas)`^{2}
useful if and only if `iturb(iphas)` $= 41$

`xlesfl` ra real number > 0 [2] O L3
for each phase `iphas`, `xlesfl(iphas)` is a constant used to define, for each cell Ω_i, the width of the (implicit) filter:
$\overline{\Delta} = xlesfl(iphas)(ales(iphas) * |\Omega_i|)^{bles(iphas)}$
useful if and only if `iturb(iphas)` $= 40$ or 41

`ales` ra real number > 0 [1] O L3
for each phase `iphas`, `ales(iphas)` is a constant used to define, for each cell Ω_i, the width of the (implicit) filter:
$\overline{\Delta} = xlesfl(iphas)(ales(iphas) * |\Omega_i|)^{bles(iphas)}$
useful if and only if `iturb(iphas)` $= 40$ or 41

`bles` ra real number > 0 [1/3] O L3
for each phase `iphas`, `bles(iphas)` is a constant used to define, for each cell Ω_i, the width of the (implicit) filter:
$\overline{\Delta} = xlesfl(iphas)(ales(iphas) * |\Omega_i|)^{bles(iphas)}$
useful if and only if `iturb(iphas)` $= 40$ or 41

`xlesfd` ra real number > 0 [1.5] O L3
for each phase `iphas`, `xlesfd(iphas)` is the constant used to define, for each cell Ω_i, the width of the explicit filter used in the framework of the LES dynamic model:
$\widetilde{\overline{\Delta}} = xlesfd(iphas)\overline{\Delta}$
useful if and only if `iturb(iphas)` $= 41$

7.2.6 Time scheme

By default, the standard time scheme is a first-order. A second-order scheme is activated automatically with LES modeling. 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 modeling 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.

`ischtp` ia 1 or 2 [1 or 2] O L2

for each phase `iphas`, `ischtp(iphas)` indicates the order of the activated time scheme (this indicator allows the code to automatically complete the other indicators related to the time scheme)

= 1: first-order

= 2: second-order

when `ischtp(iphas)`=2, the physical properties are by default not second-order. It it possible to modify this by means of the following indicators.

due to specific coupling between certain variables, the source terms in the turbulence equations (except convection and diffusion) cannot be second order, except with the R_{ij} models (cf. key word `isto2t`)

by default, `ischtp(iphas)` is initialised to 2 with the LES model and 1 otherwise

always useful

`istmpf` ia 0, 1 or 2 [0 or 1] O L3

for each phase `iphas`, `istmpf(iphas)` specifies the time scheme activated for the mass flow. The chosen value for `istmpf(iphas)` will automatically determine the value given to the variable `thetfl(iphas)`

= 0: "explicit" first-order: the mass flow calculated at the previous time step ("n") is used in the convective terms of all the equations (momentum, turbulence and scalars

= 1: "standard" first-order: the mass flow calculated at the previous time step ("n") is used in the convective terms of the momentum equation, and the updated mass flow (time "n+1") is used in the equations of turbulence and scalars

= 2: second-order: the mass flow used in the momentum equations is extrapolated at "n+`thetfl`" (=n+1/2) from the values at the two former time steps (Adams Bashforth); the mass flow used in the equations for turbulence and scalars is interpolated at time "n+`thetfl`" (=n+1/2) from the values at the former time step and at the newly calculated "n+1" time step.

by default, `istmpf(iphas)`=2 is used in the case of a second-order time scheme (if `ischtp(iphas)`=2) and `istmpf(iphas)`=1 otherwise

always useful

`isno2t` ia 0, 1 or 2 [0 or 1] O L3

for each phase `iphas`, `isno2t(iphas)` specifies the time scheme activated for the source terms of the momentum equation, apart from convection and diffusion (for instance: head loss, transposed gradient, ...).

= 0: "standard" first-order: the terms which are linear functions of the solved variable are implicit and the others are explicit

= 1: second-order: the terms of the form $S_i\phi$ which are linear functions of the solved variable ϕ are expressed as second-order terms by interpolation (according to the formula $(S_i\phi)^{n+\theta} = S_i^n[(1-\theta)\phi^n + \theta\phi^{n+1}]$, θ being given by the value of `thetav` associated with the variable ϕ) ; the other terms S_e are expressed as second-order terms by extrapolation (according to the formula $(S_e)^{n+\theta} = [(1+\theta)S_e^n - \theta S_e^{n-1}]$, θ being given by the value of `thetsn(iphas)`=0.5)

= 2: the linear terms $S_i\phi$ are treated in the same way as when `isno2t`=1; the other terms S_e are extrapolated according to the same formula as when `isno2t`=1, but with θ=`thetsn(iphas)`=1

by default, `isno2t(iphas)` is initialised to 1 (second-order) when the selected time scheme is second-order (`ischtp`=2), otherwise to 0.

always useful

`isto2t` ia 0, 1 or 2 [0] O L3
for each phase `iphas`, `isto2t(iphas)` specifies the time scheme activated for the source terms of the turbulence equations (related to k, R_{ij}, ε, ω, φ, $\overline{f}$), apart from convection and diffusion.
= 0: "standard" first-order: the terms which are linear functions of the solved variable are implicit and the others are explicit
= 1: second-order: the terms of the form $S_i\phi$ which are linear functions of the solved variable ϕ are expressed as second-order terms by interpolation (according to the formula $(S_i\phi)^{n+\theta} = S_i^n[(1-\theta)\phi^n + \theta\phi^{n+1}]$, θ being given by the value of `thetav` associated with the variable ϕ); the other terms S_e are expressed as second-order terms by extrapolation (according to the formula $(S_e)^{n+\theta} = [(1+\theta)S_e^n - \theta S_e^{n-1}]$, θ being given by the value of `thetst(iphas)`=0.5)
= 2: the linear terms $S_i\phi$ are treated in the same way as when `isto2t`=1; the other terms S_e are extrapolated according to the same formula as when `isto2t`=1, but with θ=`thetst(iphas)`=1
due to certain specific couplings between the turbulence equations, `isto2t(iphas)` is allowed the value 1 or 2 only for the R_{ij} models (`iturb(iphas)`=30 or 31); hence, it is always initialised to 0.
always useful

`isso2t` ia 0, 1 or 2 [0 or 1] O L3
for each scalar `iscal`, `isso2t(iscal)` specifies the time scheme activated for the source terms of the equation for the scalar, apart from convection and diffusion (for instance: variance production, user-specified terms, ...).
= 0: "standard" first-order: the terms which are linear functions of the solved variable are implicit and the others are explicit
= 1: second-order: the terms of the form $S_i\phi$ which are linear functions of the solved variable ϕ are expressed as second-order terms by interpolation (according to the formula $(S_i\phi)^{n+\theta} = S_i^n[(1-\theta)\phi^n + \theta\phi^{n+1}]$, θ being given by the value of `thetav` associated with the variable ϕ) ; the other terms S_e are expressed as second-order terms by extrapolation (according to the formula $(S_e)^{n+\theta} = [(1+\theta)S_e^n - \theta S_e^{n-1}]$, θ being given by the value of `thetss(iscal)`=0.5)
= 2: the linear terms $S_i\phi$ are treated in the same way as when `isso2t`=1; the other terms S_e are extrapolated according to the same formula as when `isso2t`=1, but with θ=`thetss(iscal)`=1
by default, `isso2t(iscal)` is initialised to 1 (second-order) when the selected time scheme is second-order (`ischtp`=2), otherwise to 0.
always useful

`iroext` ia 0, 1 or 2 [0] O L3
for each phase `iphas`, `iroext(iphas)` specifies the time scheme activated for the physical property ϕ "density".
= 0: "standard" first-order: the value calculated at the beginning of the current time step (from the variables known at the end of the previous time step) is used
= 1: second-order: the physical property ϕ is extrapolated according to the formula $\phi^{n+\theta} = [(1+\theta)\phi^n - \theta\phi^{n-1}]$, θ being given by the value of `thetro(iphas)`=0.5
= 2: first-order: the physical property ϕ is extrapolated at $n+1$ according to the same formula as when `iroext`=1 but with θ=`thetro(iphas)`=1
always useful

`iviext` ia 0, 1 or 2 [0] O L3
for each phase `iphas`, `iviext(iphas)` specifies the time scheme activated for the physical property ϕ "total viscosity" (molecular+turbulent or sub-grid viscosities).

= 0: "standard" first-order: the value calculated at the beginning of the current time step (from the variables known at the end of the previous time step) is used

= 1: second-order: the physical property ϕ is extrapolated according to the formula $\phi^{n+\theta} = [(1+\theta)\phi^n - \theta\phi^{n-1}]$, θ being given by the value of `thetvi(iphas)`=0.5

= 2: first-order: the physical property ϕ is extrapolated at $n+1$ according to the same formula as when `iviext`=1, but with θ=`thetvi(iphas)`=1

always useful

`icpext` ia 0, 1 or 2 [0] O L3

for each phase `iphas`, `icpext(iphas)` specifies the time scheme activated for the physical property ϕ "specific heat".

= 0: "standard" first-order: the value calculated at the beginning of the current time step (from the variables known at the end of the previous time step) is used

= 1: second-order: the physical property ϕ is extrapolated according to the formula $\phi^{n+\theta} = [(1+\theta)\phi^n - \theta\phi^{n-1}]$, θ being given by the value of `thetcp(iphas)`=0.5

= 2: first-order: the physical property ϕ is extrapolated at $n+1$ according to the same formula as when `icpext`=1, but with θ=`thetcp(iphas)`=1

always useful

`ivsext` ia 0, 1 or 2 [0] O L3

for each scalar `iscal`, `ivsext(iscal)` specifies the time scheme activated for the physical property ϕ "diffusivity".

= 0: "standard" first-order: the value calculated at the beginning of the current time step (from the variables known at the end of the previous time step) is used

= 1: second-order: the physical property ϕ is extrapolated according to the formula $\phi^{n+\theta} = [(1+\theta)\phi^n - \theta\phi^{n-1}]$, θ being given by the value of `thetvs(iscal)`=0.5

= 2: first-order: the physical property ϕ is extrapolated at $n+1$ according to the same formula as when `ivsext`=1, but with θ=`thetvs(iscal)`=1

always useful

`thetav` ra $0 \leqslant$ real $\leqslant 1$ [1 or 0.5] O L3

for each variable `ivar`, `thetav(ivar)` is the value of θ used to express at the second-order the terms of convection, diffusion and the source terms which are linear functions of the solved variable (according to the formula $\phi^{n+\theta} = (1-\theta)\phi^n + \theta\phi^{n+1}$). Generally, only the values 1 and 0.5 are used. The user is not allowed to modify this variable.

= 1: first-order

= 0.5: second-order

Concerning the pressure, the value of `thetav` is always 1. Concerning the other variables, the value `thetav`=0.5 is used when the second-order time scheme is activated by `ischtp`=2 (standard value for LES calculations), otherwise `thetav` is set to 1.

always useful

`thetfl` ra $0 \leqslant$ real $\leqslant 1$ [0 or 0.5] O L3

for each phase `iphas`, `thetfl(iphas)` is the value of θ used to interpolate the convective fluxes of the variables when a second-order time scheme has been activated for the mass flow (see `istmpf`)

generally, only the value 0.5 is used. The user is not allowed to modify this variable.

= 0.0: "explicit" first-order (corresponds to `istmpf(iphas)`=0 or 1)

= 0.5: second-order (corresponds to `istmpf(iphas)`=2). The mass flux will be interpolated according to the formula $Q^{n+\theta} = \frac{1}{2-\theta}Q^{n+1} + \frac{1-\theta}{2-\theta}Q^{n+1-\theta}$).

always useful

`thetsn` ra $0 \leqslant$ real $\leqslant 1$ [0, 0.5 or 1] O L3

for each phase `iphas`, `thetsn(iphas)` is the value of θ used to extrapolate the non linear explicit source terms S_e of the momentum equation, when the source term extrapolation has been activated (see `isno2t`), following the formula
$(S_e)^{n+\theta} = (1+\theta)S_e^n - \theta S_e^{n-1}$
the value of θ=`thetsn(iphas)` is deduced from the value chosen for `isno2t(iphas)`. Generally, only the value 0.5 is used. The user is not allowed to modify this variable.
= 0: first-order (unused, corresponds to `isno2t(iphas)`=0)
= 0.5: second-order (used when `isno2t(iphas)`=1)
= 1: first-order (used when `isno2t(iphas)`=2)
always useful

`thetst` ra $0 \leqslant$ real $\leqslant 1$ [0, 0.5 or 1] O L3

for each phase `iphas`, `thetst(iphas)` is the value of θ used to extrapolate the non linear explicit source terms S_e of the turbulence equations, when the source term extrapolation has been activated (see `isto2t`), following the formula
$(S_e)^{n+\theta} = (1+\theta)S_e^n - \theta S_e^{n-1}$
the value of θ=`thetsn(iphas)` is deduced from the value chosen for `isto2t(iphas)`. Generally, only the value 0.5 is used. The user is not allowed to modify this variable.
= 0: first-order (unused, corresponds to `isto2t(iphas)`=0)
= 0.5: second-order (used when `isto2t(iphas)`=1)
= 1: first-order (used when `isto2t(iphas)`=2)
always useful

`thetss` ra $0 \leqslant$ real $\leqslant 1$ [0, 0.5 or 1] O L3

for each scalar `iscal`, `thetss(iscal)` is the value of θ used to extrapolate the non linear explicit source terms S_e of the scalar equation, when the source term extrapolation has been activated (see `isso2t`), following the formula
$(S_e)^{n+\theta} = (1+\theta)S_e^n - \theta S_e^{n-1}$
the value of θ=`thetss(iscal)` is deduced from the value chosen for `isso2t(iscal)`. Generally, only the value 0.5 is used. The user is not allowed to modify this variable.
= 0: first-order (unused, corresponds to `isso2t(iscal)`=0)
= 0.5: second-order (used when `isso2t(iscal)`=1)
= 1: first-order (used when `isso2t(iscal)`=2)
useful if `nscal`>1

`thetro` ra $0 \leqslant$ real $\leqslant 1$ [0, 0.5 or 1] O L3

for each phase `iphas`, `thetro(iphas)` is the value of θ used to extrapolate the physical property ϕ "density" when the extrapolation has been activated (see `iroext`),according to the formula $\phi^{n+\theta} = (1+\theta)\phi^n - \theta\phi^{n-1}$
the value of θ=`thetro(iphas)` is deduced from the value chosen for `iroext(iphas)`. Generally, only the value 0.5 is used. The user is not allowed to modify this variable.
= 0: first-order (unused, corresponds to `iroext(iphas)`=0)
= 0.5: second-order (corresponds to `iroext(iphas)`=1)
= 1: first-order (corresponds to `iroext(iphas)`=2)
always useful

`thetvi` ra $0 \leqslant$ real $\leqslant 1$ [0, 0.5 or 1] O L3

for each phase `iphas`, `thetvi(iphas)` is the value of θ used to extrapolate the physical property ϕ "total viscosity" when the extrapolation has been activated (see `iviext`),according to the formula $\phi^{n+\theta} = (1+\theta)\phi^n - \theta\phi^{n-1}$
the value of θ=`thetvi(iphas)` is deduced from the value chosen for `iviext(iphas)`. Generally, only the value 0.5 is used. The user is not allowed to modify this variable.

= 0: first-order (unused, corresponds to `iviext(iphas)`=0)
= 0.5: second-order (corresponds to `iviext(iphas)`=1)
= 1: first-order (corresponds to `iviext(iphas)`=2)
always useful

`thetcp` ra $0 \leqslant$ real $\leqslant 1$ [0, 0.5 or 1] O L3
for each phase `iphas`, `thetcp(iphas)` is the value of θ used to extrapolate the physical property ϕ "specific heat" when the extrapolation has been activated (see `icpext`),according to the formula $\phi^{n+\theta} = (1+\theta)\phi^n - \theta\phi^{n-1}$
the value of θ=`thetcp(iphas)` is deduced from the value chosen for `icpext(iphas)`. Generally, only the value 0.5 is used. The user is not allowed to modify this variable.
= 0: first-order (unused, corresponds to `icpext(iphas)`=0)
= 0.5: second-order (corresponds to `icpext(iphas)`=1)
= 1: first-order (corresponds to `icpext(iphas)`=2)
always useful

`thetvs` ra $0 \leqslant$ real $\leqslant 1$ [0, 0.5 or 1] O L3
for each scalar `iscal`, `thetvs(iscal)` is the value of θ used to extrapolate the physical property ϕ "diffusivity" when the extrapolation has been activated (see `ivsext`),according to the formula $\phi^{n+\theta} = (1+\theta)\phi^n - \theta\phi^{n-1}$
the value of θ=`thetvs(iscal)` is deduced from the value chosen for `ivsext(iscal)`. Generally, only the value 0.5 is used. The user is not allowed to modify this variable.
= 0: first-order (unused, corresponds to `ivsext(iscal)`=0)
= 0.5: second-order (corresponds to `ivsext(iscal)`=1)
= 1: first-order (corresponds to `ivsext(iscal)`=2)
useful if `nscal`>1

7.2.7 Gradient reconstruction

`imrgra` i 0, 1, 2, 3 or 4 [0] O L2
indicates the type of gradient reconstruction (one method for all the variables)
= 0: iterative reconstruction of the non-orthogonalities
= 1: least squares method based on the first neighbor cells (cells which share a face with the treated cell)
= 2: least squares method based on the extended neighborhood (cells which share a node with the treated cell)
= 3: least squares method based on a partial extended neighborhood (all first neighbors plus the extended neighborhood cells that are connected to a face where the non-orthogonality angle is larger than parameter `anomax`)
= 4: iterative reconstruction with initialisation using the least squares method (first neighbors)
if `imrgra` fails due to probable mesh quality problems, it is usually effective to use `imrgra`=3. Moreover, `imrgra`=3 is usually faster than `imrgra`=0 (but with less feedback on its use).
it should be noted that `imrgra`=1, 2 or 3 automatically triggers a gradient limitation procedure. See `imligr`.
useful if and only if there is `n` so that `nswrgr(n)` > 1

`nswrgr` ia positive integer [100] O L3
for each unknown `ivar`, `nswrgr(ivar)` $\leqslant 1$ indicates that the gradients are not reconstructed
if `imrgra` = 0 or 4, `nswrgr(ivar)` is the number of iterations for the gradient

reconstruction
if `imrgra` = 1, 2 or 3, `nswrgr(ivar)` > 1 indicates that the gradients are reconstructed (but the method is not iterative, so any value larger than 1 for `nswrgr` yields the same result)
useful for all the unknowns

`epsrgr` ra real number > 0 [10^{-5}] O L3
for each unknown `ivar`, relative precision for the iterative gradient reconstruction: `epsrgr(ivar)`
useful for all the unknowns when `imrgra` = 0 or 4

`imligr` ia -1, 0 or 1 [-1 or 1] O L3
for each unknown `ivar`, indicates the type of gradient limitation: `imligr(ivar)`
=-1: no limitation
= 0: based on the neighbors
= 1: superior order
for all the unknowns, `imligr` is initialised to -1 if `imrgra`=0 or 4 and to 1 if `imrgra` = 1, 2 or 3
useful for all the unknowns

`climgr` ra real number > 0 [1.5] O L3
for each unknown `ivar`, factor of gradient limitation: `climgr(ivar)` (high value means little limitation)
useful for all the unknowns `ivar` for which `imligr(ivar)` $\neq$ -1

`extrag` ra 0, 0.5 or 1 [0] O L3
for the variable "pressure" `ivar=ipr(iphas)`, extrapolation coefficient of the gradients at the boundaries. It affects only the Neumann conditions. The only possible values of `extrag(ipr(iphas))` are:
= 0: homogeneous Neumann calculated at first-order
= 0.5: improved homogeneous Neumann, calculated at second-order in the case of an orthogonal mesh and at first-order otherwise
= 1: gradient extrapolation (gradient at the boundary face equal to the gradient in the neighbor cell), calculated at second-order in the case of an orthogonal mesh and at first-order otherwise
`extrag` often allows to correct the non-physical velocities that appear on horizontal walls when density is variable and there is gravity. It is strongly advised to keep `extrag`=0 for the variables apart from pressure. See also `iphydr`.
In practice, only the values 0 and 1 are allowed. The value 0.5 isn't allowed by default (but the lock can be overridden if necessary, contact the development team).
always useful

`anomax` r $0 \leqslant$ real $\leqslant \pi/2$ [$\pi/4$] O L3
limit non-orthogonality angle used to restrict the extended neighborhood for the gradient calculation with `imrgra`=3.
`anomax`=0 will yield the same result as `imrgra`=2 (full extended neighborhood). `anomax`=$\pi/2$ will yield the same result as `imrgra`=2 (first neighbors only)[43]
useful if and only if `imrgra`=3

[43] except for pathological cases where the non-orthogonality angle of a face would be larger than $\pi/2$

7.2.8 Solution of the linear systems

`iresol` ia -1, `ipol*1000+j` [-1] O L3
for each unknown `ivar`, `iresol(ivar)` determines the method used for the solution of the linear system
= -1: automatically managed by the code (conjugate gradient for the pressure `ivar=ipr(iphas)` or any variable which is not convected, Jacobi for the others. Diagonal preconditioning with conjugate gradient).
= `ipol*1000+j` with `j`= 0: conjugate gradient
`j`= 1: Jacobi
`j`= 2: stabilised bi-conjugate gradient (BI-CGSTAB)
`ipol` is the degree of the Neumann polynomial used for the preconditioning[44].
`ipol` is necessarily 0 with the Jacobi algorithm.
Concerning the computational time, the performance depends on the case. If a preconditioning method different from the diagonal preconditioning is to be used, it seems to be better to restrict to a first-order preconditioning (`ipol`=1). This preconditioning may slightly increase performance in some cases but may decrease it in others.
always useful

`nitmax` ia integer > 0 [10000] O L3
for each unknown `ivar`, maximum number of iterations for the solution of the linear systems: `nitmax(ivar)`
when the algebraic multigrid option is activated for the variable `ivar` (`imgr(ivar)`=1), `nitmax(ivar)` is the maximum number of iterations for the solution on the coarsest mesh
always useful

`epsilo` ra real number > 0 [10^{-8},10^{-5}] O L3
for each unknown `ivar`, relative precision for the solution of the linear system. The default value is `epsilo(ivar)`=10^{-8}. This value is set low on purpose. When there are enough iterations on the reconstruction of the right-hand side of the equation, the value may be increased (by default, in case of second-order in time, with `nswrsm` = 5 or 10, `epsilo` is increased to 10^{-5}).
always useful

`imgr` ia 0 or 1 [0] O L3
for each unknown `ivar`, indicates the use (`imgr(ivar)`=1) or not (=0) of the algebraic multigrid method for the solution of the linear systems
`imgr(ivar)` can be set independently for every variable
always useful. Generally, its use is designed for the variable "pressure" in case of meshes with strongly stretched cells. It is recommended not to modify `imgr`

`ncegrm` i integer > 0 [30] O L3
for the multigrid method, maximum number of cells on the coarsest grid
useful if and only if `imgr(ivar)` = 1 for at least one variable `ivar`

`ncymax` ia integer > 0 [100] O L3
for each unknown `ivar`, `ncymax(ivar)` is the maximum number of cycles when using

[44] D being the diagonal part of A and X its extra-diagonal part, it can be written $A = D(Id + D^{-1}X)$. Therefore $A^{-1} = (Id + D^{-1}X)^{-1}D^{-1}$. A series development of $Id + D^{-1}X$ can then be used which yields, symbolically, $Id + \sum_{I=1}^{IPOL} (-D^{-1}X)^I$.

the multigrid method.
useful if and only if `imgr(ivar)` = 1

`ngrmax` i 1⩽ integer ⩽`ngrmmx` [`ngrmmx`] O L3
when using the multigrid method, maximum number of grid levels
useful if and only if `imgr(ivar)` = 1 for at least one variable `ivar`

`ncymax` ia integer > 0 [10] O L3
for each unknown `ivar`, `ncymax(ivar)` is the maximum number of multigrid cycles.
useful if and only if `imgr(ivar)` = 1

`nitmgf` ia integer > 0 [10] O L3
for each unknown `ivar`, `nitmgf(ivar)` is the maximum number of iterations on all grids except for the coarsest when the multigrid method is used; the resolution on the coarsest grid uses `nitmax`.
useful if and only if `imgr(ivar)` = 1

WARNING
The algebraic multigrid method has only been tested for the "pressure" variable (`imgr(ipr(iphas))`=1).

7.2.9 Convective scheme

blencv ra $0 \leqslant$ real $\leqslant 1$ [0 or 1] O L1
for each unknown `ivar` to calculate, `blencv(ivar)` indicates the proportion of second-order convective scheme (0 corresponds to an "upwind" first-order scheme) ; in case of LES calculation, a second-order scheme is recommended and activated by default (`blencv=1`)
useful for all the unknowns `ivar` for which `iconv(ivar)` = 1

`ischcv` ia 0 or 1 [1] O L2
for each unknown `ivar` to calculate, `ischcv(ivar)` indicates the type of second-order convective scheme
= 0: Second Order Linear Upwind
= 1: Centered
useful for all the unknowns `ivar` which are convected (`iconv(ivar)`=1) and for which a second-order scheme is used (`blencv(ivar)` > 0)

`isstpc` ia 0 or 1 [0] O L2
for each unknown `ivar` to calculate, `isstpc(ivar)` indicates whether a "slope test" should be used to switch from a second-order to an "upwind" convective scheme under certain conditions, to ensure stability.
= 0: "slope test" activated for the considered unknown
= 1: "slope test" deactivated for the considered unknown
useful for all the unknowns `ivar` which are convected (`iconv(ivar)`=1) and for which a second-order scheme is used (`blencv(ivar)` > 0).
the use of the "slope test" stabilises the calculation but may bring the order in space to decrease quickly.

7.2.10 Pressure-continuity step

`iprco` i 0 or 1 [1] O L3
indicates if the pressure-continuity step is taken into account (1) or not (0)
always useful

`arak` ra $0 <$ real $\leqslant 1$ [1] O L3
for each phase `iphas`, `arak(iphas)` is the Arakawa coefficient before the Rhie& Chow filter
always useful

`relaxp` ra $0 <$ real $\leqslant 1$ [1] O L2
for each phase `iphas`, relaxation of the pressure increment during the solution of the system (`relaxp(iphas)`=1: no relaxation)
can improve the convergence in case of meshes of insufficient quality
always useful

`irevmc` ia 0, 1 or 2 [0] O L3
for each phase `iphas`, method used to update the velocity after the pressure correction:
- standard gradient of pressure increment (`irevmc(iphas)`=0)
- least squares on the pressure increment (`irevmc(iphas)`=1)
-"`rt0`" *i.e.* least squares on the updated mass flux (`irevmc(iphas)`=2)

the method `irevmc(iphas)`=2 is generally not recommended
always useful

`iphydr` i 0 or 1 [0] O L2
method for taking into account the balance between the pressure gradient and the source terms (gravity and head losses): by extension it will be referenced as "taking into account of the hydrostatic pressure"
= 0: standard algorithm
= 1: improved algorithm
always useful
When the density effects are important, the choice of `iphydr`=1 allows to improve the interpolation of the pressure and correct the non-physical velocities which may appear in highly stratified areas or near horizontal walls (thus avoiding the use of `extrag` if the non-physical velocities are due only to gravity effects).
The improved algorithm also allows to eradicate the velocity oscillations which tend to appear at the frontiers of areas with high head losses.
In the case of a stratified flow, the calculation cost is higher when the improved algorithm is used (about 30% depending on the case) because the hydrostatic pressure must be recalculated at the outlet boundary conditions: see `icalhy`.
On meshes of insufficient quality, in order to improve the convergence, it may be useful to increase the number of iterations for the reconstruction of the pressure right-hand member, *i.e.* `nswrsm(ipr(iphas))`.
If head losses are present just along an outlet boundary, it is necessary to specify `icalhy`=0 in order to deactivate the recalculation of the hydrostatic pressure at the boundary, which may otherwise cause instabilities.

`icalhy` i 0 or 1 [0 or 1] O L3
activates the calculation of hydrostatic pressure boundary conditions at outlet boundaries
= 0: no calculation of the hydrostatic pressure at the outlet boundary
= 1: calculation of the hydrostatic pressure at the outlet boundary

always useful
This option is automatically specified depending on the choice of `iphydr` and the value of gravity (`icalhy`=1 if `iphydr`=1 and gravity is different from 0; otherwise `icalhy`=0). The activation of this option generates an additional calculation cost (about 30% depending on the case).
If head losses are present just along an outlet boundary, it is necessary to specify `icalhy`=0 in order to deactivate the recalculation of the hydrostatic pressure at the boundary, which may otherwise cause instabilities

7.2.11 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[45]. These scalars indicate the areas (cells) in which some error types may be important. They are stored in the array `propce` containing the properties at the cells (see `iestim`). 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[46] ; 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 $\underline{u}^*$), the estimator $\eta_{i,k}^{pred}(\underline{u}^*)$, local variable calculated at every cell Ω_i, is created from $\underline{\mathcal{R}}^{pred}(\underline{u}^*)$, which represents the residual of the equation solved during this step:

$$\begin{aligned}\underline{\mathcal{R}}^{pred}(\underline{u}^*) &= \rho^n \frac{\underline{u}^* - \underline{u}^n}{\Delta t} + \rho^n \underline{u}^n \cdot \underline{\underline{grad}}(\underline{u}^*) - div\left((\mu+\mu_t)^n \underline{\underline{grad}}(\underline{u}^*)\right) + \underline{\text{grad}}\ (P^n) \\ &- \text{ rest of the right-hand member } (\underline{u}^n, P^n, \text{other variables}^n)\end{aligned}$$

By definition:

$$\eta_{i,k}^{pred}(\underline{u}^*) = |\Omega_i|^{(k-2)/2}\ ||\underline{\mathcal{R}}^{pred}(\underline{u}^*)||_{I\!L^2(\Omega_i)}$$

• The first family, $k = 1$, suppresses the volume $|\Omega_i|$ which intrinsicly appears with the norm $I\!L^2(\Omega_i)$.
• The second family, $k = 2$, exactly represents the norm $I\!L^2(\Omega_i)$. The size of the cell therefore appears in its calculation and induces a weighting effect.
$\eta_{i,k}^{pred}(\underline{u}^*)$ 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):

$$\begin{aligned}\eta_{i,k}^{der}(\underline{u}^{n+1}) &= |\Omega_i|^{(k-2)/2} ||div(\text{corrected mass flow after the pressure step}) - \Gamma||_{L^2(\Omega_i)} \\ &= |\Omega_i|^{(1-k)/2} |div(\text{corrected mass flow after the pressure step}) - \Gamma|\end{aligned} \quad (4)$$

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:

$$\eta_{i,k}^{corr}(\underline{u}^{n+1}) = |\Omega_i|^{\delta_{2,k}}\ |div(\rho^n \underline{u}^{n+1}) - \Gamma|$$

[45] choice made by the user
[46] indeed, all the first-order in space differential terms have to be recalculated at the time t^{n+1}

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:

$$\begin{aligned}\underline{\mathcal{R}}^{tot}(\underline{u}^{n+1}) &= \rho^n \frac{\underline{u}^{n+1}-\underline{u}^n}{\Delta t} + \rho^n \underline{u}^{n+1}\cdot \underline{\underline{grad}}(\underline{u}^{n+1}) - div\left((\mu+\mu_t)^n \underline{\underline{grad}}(\underline{u}^{n+1})\right) + \underline{\text{grad}}\ (P^{n+1}) \\ &- \text{ rest of the right-hand member } (\underline{u}^{n+1}, P^{n+1}, \text{other variables}^n)\end{aligned}$$

By definition:

$$\eta_{i,k}^{tot}(\underline{u}^{n+1}) = |\Omega_i|^{(k-2)/2}\ ||\underline{\mathcal{R}}^{tot}(\underline{u}^{n+1})||_{I\!L^2(\Omega_i)}$$

The mass flux in the convective term is recalculated from $\underline{u}^{n+1}$ expressed at the cell centers (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 $I\!L^2(\Omega_i)$.

• The second family, $k = 2$, exactly represents the norm $I\!L^2(\Omega_i)$. 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`.

iescal ia 0, 1 or 2 [0] O L1
for each phase `iphas`, `iescal(iest,iphas)` indicates the calculation mode for the error estimator `iest` (`iespre`, `iesder`, `iescor` or `iestot`), for the Navier-Stokes equation:
`iescal` = 0: estimator not calculated,
`iescal` = 1: the estimator $\eta_{i,1}^*$ is calculated, without contribution of the volume,
`iescal` = 2: the estimator $\eta_{i,2}^*$ is calculated, with contribution of the volume ("norm L^2"), except for `iescor`, for which $|\Omega_i|\ \eta_{i,1}^{corr}$ is calculated.

The name of the estimators appearing in the listing and the post-processing is made up of the default name (given before), followed first by the value of `iescal`, then by the phase number. For instance, EsPre201 is the estimator `iespre` calculated with `iescal`=2 for the phase 01.
always useful

7.2.12 Calculation of the distance to the wall

`icdpar` i -1, 1, -2 or 2 [-1] O L2
specifies the method used to calculate the distance to the wall y and the adimensional distance y^+ for all the cells of the calculation domain (when necessary):
= 1: standard algorithm (based on a Poisson equation for y and convection equation for y^+), with reading of the distance to the wall from the restart file if possible

=-1: standard algorithm (based on a Poisson equation for y and convection equation for y^+), with systematic recalculation of the distance to the wall in case of calculation restart

= 2: former algorithm (based on geometrical considerations), with reading of the distance to the wall from the restart file if possible

=-2: former algorithm (based on geometrical considerations) with systematic recalculation of the distance to the wall in case of calculation restart

In case of restart calculation, if the position of the walls haven't changed, reading the distance to the wall from the restart file can save a fair amount of CPU time.

Useful in $R_{ij} - \varepsilon$ model with wall echo (`iturb(iphas)`=30 and `irijec`=1), in LES with van Driest damping (`iturb(iphas)`=40 and `idries(iphas)`=1) and in $k - \omega$ SST (`iturb(iphas)`=60).

By default, `icdpar` is initialised to -1, in case there has been a change in the definition of the boundary conditions between two computations (change in the number or the positions of the walls). Yet, with the $k - \omega$ SST model, the distance to the wall is needed to calculate the turbulent viscosity, which is done before the calculation of the distance to the wall. Hence, when this model is used (and only in that case), `icdpar` is set to 1 by default, to ensure total continuity of the calculation at restart.

As a consequence, with the $k - \omega$ SST model, if the number and positions of the walls are changed at a calculation restart, it is mandatory for the user to set `icdpar` explicitly to -1, otherwise the distance to the wall used will not correspond to the actual position of the walls.

The former algorithm is not compatible with parallelism nor periodicity. Also, whatever the value chosen for `icdpar`, the calculation of the distance to the wall is made at the most once for all a the beginning of the calculation. It is therefore not compatible with moving walls. Please contact the development team if you need to override this limitation.

The following options are related to `icdpar`=1 or -1. The options of level 2 are described first. Some options are used only in the case of the calculation of the adimensional distance to the wall y^+ (LES model with van Driest damping). Most of these key words are simple copies of the key words 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.

`iwarny` i integer [0] O L2

specifies the level of the output writing concerning the calculation of the distance to the wall with `icdpar`=1 or -1. The higher the value, the more detailled the outputs

useful when `icdpar`=1 or -1

`ntcmxy` i positive integer [1000] O L2

number of pseudo-time iterations for the calculation of the adimensional distance to the wall y^+

useful when `icdpar`=1 or -1 for the calculation of y^+

`nitmay` i integer > 0 [10000] O L3

maximum number of iterations for the solution of the linear systems

useful when `icdpar`=1 or -1

`nswrsy` i positive integer [1] O L3

number of iterations for the reconstruction of the right-hand members: corresponds to `nswrsm`

useful when `icdpar`=1 or -1

`nswrgy` i positive integer [100] O L3
number of iterations for the gradient reconstruction: corresponds to `nswrgr`
useful when `icdpar`=1 or -1

`imligy` i -1, 0 or 1 [-1 or 1] O L3
type of gradient limitation: corresponds to `imligr`
useful when `icdpar`=1 or -1

`ircfly` i 0 or 1 [1] O L3
indicates the reconstruction of the convective and diffusive fluxes at the faces: corresponds to `ircflu`
useful when `icdpar`=1 or -1

`ischcy` i 0 or 1 [1] O L3
type of second-order convective scheme: corresponds to `ischcv`
useful when `icdpar`=1 or -1 for the calculation of y^+

`isstpy` i 0 or 1 [0] O L3
indicates if a "slope test" should be used for a second-order convective scheme: corresponds to `isstpc`
useful when `icdpar`=1 or -1 for the calculation of y^+

`imgrpy` i 0 or 1 [0] O L3
indicates whether the algebraic multigrid method should be used (`imgr(ivar)`=1) or not (0): corresponds to `imgr`
useful when `icdpar`=1 or -1

`blency` r $0 \leqslant$ real $\leqslant 1$ [0] O L3
proportion of second-order convective scheme: corresponds to `blencv`
useful when `icdpar`=1 or -1 for the calculation of y^+

`epsily` r real number > 0 [10^{-8}] O L3
relative precision for the solution of the linear systems: corresponds to `epsilo`
useful when `icdpar`=1 or -1

`epsrgy` r real number > 0 [10^{-5}] O L3
relative precision for the iterative gradient reconstruction: corresponds to `epsrgr`
useful when `icdpar`=1 or -1

`climgy` r real number > 0 [1.5] O L3
limitation factor of the gradients: corresponds to `climgr`
useful when `icdpar`=1 or -1

`extray` r 0, 0.5 or 1 [0] O L3
extrapolation coefficient of the gradients at the boundaries: corresponds to `extrag`
useful when `icdpar`=1 or -1

`coumxy` r strictly positive real number [5000] O L3
Target Courant number for the calculation of the adimensional distance to the wall
useful when `icdpar`=1 or -1 for the calculation of y^+

`epscvy` r strictly positive real number [10^{-8}] O L3
relative precision for the convergence of the pseudo-transient regime for the calculation of the adimensional distance to the wall
useful when `icdpar`=1 or -1 for the calculation of y^+

`yplmxy` r real number [200] O L3
value of the adimensional distance to the wall above which the calculation of the distance is not necessary (for the damping)
useful when `icdpar`=1 or -1 for the calculation of y^+

7.2.13 Others

iccvfg i 0 or 1 [0] O L1
indicates whether the dynamic field should be frozen (1) or not (0)
in such a case, the values of velocity, pressure and the variables related to the potential turbulence model (k, R_{ij}, ε, φ, $\bar{f}$, ω, turbulent viscosity) are kept constant over time and only the equations for the scalars are solved
also, if `iccvfg`=1, the physical properties modified in `usphyv` will keep being updated. Beware of non-consistencies if these properties would normally affect the dynamic field (modification of density for instance)
useful if and only if `nscal` >0 and `isuite`=1

ipucou i 0 or 1 [0] O L1
indicates the algorithm for velocity/pressure coupling
= 0: standard algorithm
= 1: reinforced coupling in case calculation with long time steps
always useful (it is seldom advised, but it can prove very useful, for instance, in case of flows with weak convection effects and highly variable viscosity)

isuit1 i 0 or 1 [0] O L1
for the 1D wall thermal module, activation (1) or not(0) of the reading of the mesh and of the wall temperature from the `ficmt1` restart file
useful if `nfpt1d`>0.

`imvisf` i 0 or 1 [0] O L3
indicates the interpolation method used to project variables from the cell centers to the faces
= 0: linear
= 1: harmonic
always useful

`ircflu` ia 0 or 1 [1] O L2
for each unknown `ivar`, `ircflu(ivar)` indicates whether the convective and diffusive fluxes at the faces should be reconstructed:
= 0: no reconstruction
= 1: reconstruction
deactivating the reconstruction of the fluxes can have a stabilising effect on the calculation. It is sometimes useful with the $k-\varepsilon$ model, if the mesh is strongly non-orthogonal in the near-wall region, where the gradients of k and ε are strong. In such a case, setting `ircflu(ik(iphas))`=0 and `ircflu(iep(iphas))`=0 will probably help (switching to a first order convective scheme, `blencv=0`, for k and ε might also help

in that case)
always useful

`nswrsm` ia positive integer [1, 2, 5 or 10] O L3
for each unknown `ivar`, `nswrsm(ivar)` indicates the number of iterations for the reconstruction of the right-hand members of the equations
with a first-order scheme in time (standard case), the default values are 2 for pressure and 1 for the other variables. With a second-order scheme in time (`ischtp`=2) or LES, the default values are 5 for pressure and 10 for the other variables.
useful for all the unknowns

`epsrsm` ra real number > 0 [10^{-8},10^{-5}] O L3
for each unknown `ivar`, relative precision on the reconstruction of the right hand-side. The default value is `epsrsm(ivar)`=10^{-8}. This value is set low on purpose. When there are enough iterations on the reconstruction of the right-hand side of the equation, the value may be increased (by default, in case of second-order in time, with `nswrsm` = 5 or 10, `epsrsm` is increased to 10^{-5}).
always useful

7.3 Numerical, physical and modeling parameters

7.3.1 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`).

`zero` r 0 [0] O L3
Parameter containing the value 0

`epzero` r 10^{-12} [10^{-12}] O L3
"Small" real parameter, used for the comparisons of real numbers (absolute value of the difference lower than `epzero`)

`pi` r 3.141592653589793 [3.141592653589793] O L3
Parameter containing an approximate value of π

`grand` r 10^{12} [10^{12}] O L3
"Large" real parameter, generally used by default as a non physical value for the initialisations of variables which have to be modified by the user

`rinfin` r 10^{30} [10^{30}] O L3
Real parameter used to represent "infinity"

7.3.2 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`).

`tkelvi` r 273.15 [273.15] O L3
Temperature in Kelvin correponding to 0 degrees Celsius.

`tkelvn` r -273.15 [-273.15] O L3
Temperature in degrees Celsius corresponding to 0 Kelvin.

`rr` r 8.31434 [8.31434] O L3
Perfect gas constant in $J/mol/K$

`trefth` r 25 + tkelvi [25 + tkelvi] O L3
Reference temperature for the specific physics, in K

`prefth` r 101325 [101325] O L3
Reference pressure for the specific physics, in Pa

`volmol` r $22.41.10^{-3}$ [$22.41.10^{-3}$] O L3
Molar volume under normal pressure and temperature conditions (1 atmosphere, 0°C) in m^{-3}

`stephn` r $5.6703.10^{-8}$ [$5.6703.10^{-8}$] O L3
Stephan constant for the radiative module σ in $W.m^{-2}.K^{-4}$

`permvi` r $1.2566.10^{-6}$ [$1.2566.10^{-6}$] O L3
Vacuum magnetic permeability μ_0 ($=4\pi.10^{-7}$) in $kg.m.A^{-2}.s^{-2}$

`epszer` r $8.854.10^{-12}$ [$8.854.10^{-12}$] O L3
Vacuum permittivity ε_0 in $F.m^{-1}$

7.3.3 Physical variables

gx,gy,gz r 3 real numbers [0,0,0] O L1
gravity components
always useful

irovar ia 0 or 1 [`-1`] C L1
for each phase `iphas`, `irovar(iphas)`=0 indicates that the density is constant. Its value is the reference density `ro0(iphas)`.
`irovar(iphas)`=1 indicates that the density is variable: its variation law must be given in the user subroutine `usphyv`
negative value: not initialised
always useful

ivivar ia 0 or 1 [`-1`] C L1
for each phase `iphas`, `ivivar(iphas)`=0 indicates that the molecular dynamic viscosity is constant. Its value is the reference molecular dynamic viscosity `viscl0(iphas)`.
`ivivar(iphas)`=1 indicates that the molecular dynamic viscosity is variable: its variation law must be given in the user subroutine `usphyv`
negative value: not initialised
always useful

ro0 ra real number $\geqslant 0$ [`-grand*10`] C L1
for each phase `iphas`, `ro0(iphas)` is the reference density

negative value: not initialised
its value is not used in gas or coal combustion modeling (it will be calculated following the perfect gas law, with $P0$ and $T0$). With the compressible module, it is also not used by the code, but it may be (and often is) referenced by the user in user subroutines; it is therefore better to specify its value.
always useful otherwise, even if a law defining the density is given by the user subroutine `usphyv` or `uselph`
indeed, except with the compressible module, *Code_Saturne* does not use the total pressure P when solving the Navier-Stokes equation, but a reduced pressure
$P^* = P - \rho_0 \underline{g}.(\underline{x} - \underline{x}_0) + P_0^* - P_0$
where $\underline{x}_0$ is a reference point (see `xyzp0`) and P_0^* and P_0 are reference values (see `pred0` and `p0`). Hence, the term $-\underline{\text{grad}}\ P + \rho \underline{g}$ in the equation is treated as $-\underline{\text{grad}}\ P^* + (\rho - \rho_0)\underline{g}$. The closer `ro0` is to the value of ρ, the more P^* will tend to represent only the dynamic part of the pressure and the faster and more precise its solution will be. Whatever the value of `ro0`, both P and P^* appear in the listing and the post-processing outputs.
with the compressible module, the calculation is made directly on the total pressure

viscl0 ra real number > 0 [`-grand*10`] C L1
for each phase `iphas`, `viscl0(iphas)` is the reference molecular dynamic viscosity
negative value: not initialised
always useful, it is the used value unless the user specifies the viscosity in the subroutine `usphyv`

srrom r $0 \leqslant$ réel < 1 [`-grand or 0`] c or O L1
With gas combustion, pulversied coal or the electric module, `srrom` is the sub-relaxation coefficient for the density, following the formula:
$\rho^{n+1} =$ srrom $\rho^n +$(1-srrom) ρ^{n+1}
hence, with a zero value, there is no sub-relaxation. With combustion and pulversied coal, `srrom` is initialised to `-grand` and the user must specify a proper value through the Interface or the initialisation subroutines (`usd3p1`, `usebu1`, `uslwc1`, `uscpi1` or `uscpl1`).With the electric module, `srrom` is initialised in to 0 and may be modified by the user in `useli1`.
With gas combustion, pulverised coal or electric arc, `ssrom` is automatically used after the second time-step. With Joule effect, the user decides whether or not it will be used in `uselph` from the coding law giving the density.

always useful with gas combustion, pulversized coal or the electric module.

p0 ra real number [$1.013e-5$] O L1
for each phase `iphas`, `p0(iphas)` is the reference pressure for the total pressure
except with the compressible module, the total pressure P is evaluated from the reduced pressure P^* so that P is equal to `p0` at the reference position $\underline{x}_0$ (given by `xyzp0`)
with the compressible module, the total pressure is solved directly
always useful

`pred0` ra real number [0] O L3
for each phase `iphas`, `pred0(iphas)` is the reference value for the reduced pressure P^* (see `ro0`)
it is especially used to initialise the reduced pressure and as a reference value for the outlet boundary conditions
for an optimised precision in the resolution of P^*, it is wiser to keep `pred0` to 0
with the compressible module, the "pressure" variable appearing in the equations

directly represents the total pressure. It is therefore initialised to `p0` and not `pred0` (see `ro0`)
always useful, except with the compressible module

xyzp0 ra 3 real numbers [0,0,0] O L1
for each phase `iphas`, `xyzp0(ii,iphas)` is the `ii` coordinate ($1 \leqslant II \leqslant 3$) of the reference point $\underline{x}_0$ for the total pressure
when there are no Dirichlet conditions for the pressure (closed domain), `xyzp0` does not need to be specified (unless the total pressure has a clear physical meaning in the configuration treated)
when Dirichlet conditions on the pressure are specified but only through stantard outlet conditions (as it is in most configurations), `xyzp0` does not need to be specified by the user, since it will be set to the coordinates of the reference outlet face (*i.e.* the code will automatically select a reference outlet boundary face and set `xyzp0` so that P equals `p0` at this face). Nontheless, if `xyzp0` is pecified by the user, the calculation will remain correct
when direct Dirichlet conditions are specified by the user (specific value set on specific boundary faces), it is better to specify the corresponding reference point (*i.e.* specifiy where the total pressure is `p0`). This way, the boundary conditions for the reduced pressure will be close to `pred0`, ensuring an optimal precision in the resolution. If `xyzp0` is not specified, the reduced pressure will be shifted, but the calculations will remain correct.
with the compressible module, the "pressure" variable appearing in the equations directly represents the total pressure. `xyzp0` is therefore not used.
always useful, except with the compressible module

t0 ra real number [0] O L1
for each phase `iphas`, `t0(iphas)` is the reference temperature
useful for the specific physics gas or coal combustion (initialisation of the density), for the electricity modules to initialise the domain temperature and for the comperssible module (initialisations). It must be given in Kelvin.

cp0 ra real number > 0 [`-grand*10`] O L1
for each phase `iphas`, `cp0(iphas)` is the reference specific heat
useful if there is $1 \leqslant n \leqslant nscaus$[47] so that `iscsth(n)`=1 (there is a scalar "temperature"), unless the user specifies the specific heat in the user subroutine `usphyv`[48] (`icp(iphas)` > 0)
with the compressible module or coal combustion, `cp0` is also needed even when there is no user scalar

icp ia 0 or 1 [0] O L1
for each phase `iphas`, indicates if the specific heat C_p is variable (`icp(iphas)`=1) or not (0)
When gas or coal combustion is activated, `icp` is automatically set to 0 (constant C_p). With the electric module, it is automatically set to 1. The user is not allowed to modify these default choices.
When `icp(iphas)`=1 is specified, the code automatically modifies this value to make `icp(iphas)` designate the effective index-number of the property "specific heat of the phase `iphas`". For each cell `iel`, the value of C_p is then specified by the user in the appropriate subroutine (`usphyv` for the standard physics) and stored in the array

[47] none of the scalars from the specific physics is a temperature
[48] when using the Graphical Interface, `cp0` is also used to calculate the diffusivity of the thermal scalars, based on their conductivity; it is therefore needed, unless the diffusivity is also specified in `usphyv`

`propce(iel,ipproc(icp(iphas)))` (*see p.71 for specific conditions of use*)
useful if there is 1⩽N⩽`nscal` so that `iscsth(n)`=1 (there is a scalar "temperature") or with the compressible module for non perfect gases

visls0 ra real number > 0 [`-grand*10`] C L1
`visls0(j)`: reference molecular diffusivity related to the scalar J ($kg.m^{-1}.s^{-1}$)
negative value: not initialised
useful if 1⩽J⩽ `nscal`, unless the user specifies the molecular diffusivity in the appropriate user subroutine (`usphyv` for the standard physics) (`ivisls(iscal)` > 0)
Warning: `visls0` *corresponds to the diffusivity. For the temperature, it is therefore defined as* λ/C_p *where* λ *and* C_p *are the conductivity and specific heat. When using the Graphical Inteface,* λ *and* C_p *are specified separately, and* `visls0` *is calculated automatically*
With the compressible module, `visls0` *(given in* ***uscfxi2****) is directly the thermal conductivity* $W.m^{-1}.K^{-1}$
With gas or coal combustion, the molecular diffusivity of the enthalpy ($kg.m^{-1}.s^{-1}$) *must be specified by the user in the variable* `diftl0` *(****usebu1****,* ***usd3p1****,* ***uslwc1****,* ***uscpi1****,* ***uscpl1****)*
With the electric module, for the Joule effect, the diffusivity is specified by the user in ***uselph*** *(even if it is constant). For the electric arc, it is calculated from the thermochemical data file*

ivisls ia positive or zero integer [0] O L1
indicates if the viscosity related to the scalar `iscal` is variable (`ivisls(iscal)`=1) or not (0). The user must specify `ivisls` only for the user scalars (`iscal` ⩽ `nscaus`).
When `ivisls(iscal)`=1 is specified, the code automatically modifies this value to make `ivisls(iscal)` designate the effective index-number of the property "diffusivity of the scalar `iscal`". For each cell `iel`, the value is then specified by the user in the appropriate subroutine (`usphyv` for the standard physics) and stored in the array `propce(iel,ipproc(ivisls(iphas)))` (*see p.71 for specific conditions of use*)
useful if 1⩽`n`⩽`nscal`

diftl0 r real number > 0 [`-grand`] C L1
molecular diffusivity for the enthalpy ($kg.m^{-1}.s^{-1}$) for gas or coal combustion (the code then automatically sets `visls0` to `diftl0` for the scalar representing the enthalpy)
always useflu for gas or coal combustion

scamin ra real number [`grand`] O L1
`scamin(iscal)` is the lower limit value for the scalar `iscal`. At each time step, in every cell where the calculated value for `rtp(iel,isca(iscal))` is lower than `scamin(iscal)`, `rtp(iel,isca(iscal))` will be reset to `scamin(iscal)`
there is no limitation if `scamin(iscal)`>`scamax(iscal)`
`scamin` shall not be specified for non-user scalars (specific physics) or for scalar variances
useful if and only if 1⩽`iscal`⩽ `nscaus`

scamax ra real number [`-grand`] O L1
`scamax(iscal)` is the higher limit value for the scalar `iscal`. At each time step, in every cell where the calculated value for `rtp(iel,isca(iscal))` is higher than `scamax(iscal)`, `rtp(iel,isca(iscal))` will be reset to `scamax(iscal)`
there is no limitation if `scamin(iscal)`>`scamax(iscal)`

`scamax` shall not be specified for non-user scalars (specific physics) or for scalar variances
useful if and only if 1⩽`iscal`⩽ `nscaus`

`sigmas` ra real number > 0 [1] O L2
`sigmas(iscal)`: turbulent Prandtl (or Schmidt) number for the scalar `iscal`
useful if and only if 1⩽`iscal`⩽ `nscaus`

`rvarfl` ra real number > 0 [0.8] O L2
when `iscavr(iscal)`>0, `rvarfl(iscal)` is the coefficient R_f in the dissipation term $-\frac{\rho}{R_f}\frac{\varepsilon}{k}$ of the equation concerning the scalar `iscal`, which represents the root mean square of the fluctuations of the scalar `iscavr(iscal)`
useful if and only if there is 1⩽`iscal`⩽ `nscal` such as `iscavr(iscal)`>0

7.3.4 Modeling parameters

xlomlg ra real number > 0 [`-grand*10`] O L1
for each phase `iphas`, `xlomlg(iphas)` is the mixing length
useful if and only if there is a phase `iphas` so that `iturb(iphas)`= 10 (mixing length)

`almax` ra `-grand`, real number > 0 [`-grand*10`] O L2
for each phase `iphas`, `almax(iphas)` is a characteristic macroscopic length of the domain, used for the initialisation of the turbulence and the potential clipping (with `iclkep(iphas)`=1)
negative value: not initialised (the code then uses the cubic root of the domain volume)
useful if and only if there is a phase `iphas` such as `turb(iphas)`= 20, 21, 30, 31, 50 or 60 (RANS models)

uref ra real number > 0 [`-grand*10`] C L1
for each phase `iphas`, `uref(iphas)` is the characteristic flow velocity, used for the initialisation of the turbulence
negative value: not initialised
useful if and only if there is a phase `iphas` such that `iturb(iphas)`= 20, 21, 30, 31, 50 or 60 (RANS model) and the turbulence is not initialised somewhere else (restart file or subroutine `usiniv`)

Basic constants of the $k-\varepsilon$ and the other RANS models

`xkappa` r real number > 0 [0.42] O L3
Kármán constant
useful if and only if there is a phase `iphas` such as `iturb(iphas)`⩾10 (mixing length, $k-\varepsilon$, $R_{ij}-\varepsilon$, LES, v2f or $k-\omega$)

`cstlog` r real number > 0 [5.2] O L3
constant of the logarithmic wall function
useful if and only if there is a phase `iphas` such as `iturb(iphas)`⩾10 (mixing length, $k-\varepsilon$, $R_{ij}-\varepsilon$, LES, v2f or $k-\omega$)

`cmu` r real number > 0 [0.09] O L3
constant C_μ for all the RANS turbulence models except for the v2f model (see `cv2fmu`

for the value of C_μ in case of v2f modeling)
useful if and only if there is a phase `iphas` such as `iturb(iphas)`= 20, 21, 30, 31 or 60 ($k-\varepsilon$, $R_{ij}-\varepsilon$ or $k-\omega$)

`ce1` r real number > 0 [1.44] O L3
constant $C_{\varepsilon 1}$ for all the RANS turbulence models except for the v2f and the $k-\omega$ models
useful if and only if there is a phase `iphas` such as `iturb(iphas)`= 20, 21, 30 or 31 ($k-\varepsilon$ or $R_{ij}-\varepsilon$)

`ce2` r real number > 0 [1.92] O L3
constant $C_{\varepsilon 2}$ for the $k-\varepsilon$ and $R_{ij}-\varepsilon$ LRR models
useful if and only if there is a phase `iphas` such as `iturb(iphas)`= 20, 21 or 30 ($k-\varepsilon$ or $R_{ij}-\varepsilon$ LRR)

`ce4` r real number > 0 [1.2] O L3
constant $C_{\varepsilon 4}$ for the interfacial term (Lagrangian module) in case of two-way coupling
useful in case of Lagrangian modeling, in $k-\varepsilon$ and $R_{ij}-\varepsilon$ with two-way coupling

`sigmak` r real number > 0 [1.0] O L3
Prandtl number for k with $k-\varepsilon$ and v2f models
useful if and only if there is a phase `iphas` such as `iturb(iphas)`=20, 21 or 50 ($k-\varepsilon$ or v2f)

`sigmae` r real number > 0 [1.3] O L3
Prandtl number for ε
useful if and only if there is a phase `iphas` such as `iturb(iphas)`= 20, 21, 30, 31 or 50 ($k-\varepsilon$, $R_{ij}-\varepsilon$ or v2f)

Constants specific to the $R_{ij}-\varepsilon$ LRR model (`iturb`=30)

`crij1` r real number > 0 [1.8] O L3
constant C_1 for the $R_{ij}-\varepsilon$ LRR model
useful if and only if there is a phase `iphas` such as `iturb(iphas)`=30 ($R_{ij}-\varepsilon$ LRR)

`crij2` r real number > 0 [0.6] O L3
constant C_2 for the $R_{ij}-\varepsilon$ LRR model
useful if and only if there is a phase `iphas` such as `iturb(iphas)`=30 ($R_{ij}-\varepsilon$ LRR)

`crij3` r real number > 0 [0.55] O L3
constant C_3 for the $R_{ij}-\varepsilon$ LRR model
useful if and only if there is a phase `iphas` such as `iturb(iphas)`=30 ($R_{ij}-\varepsilon$ LRR)

`crijep` r real number > 0 [0.18] O L3
constant C_ε for the $R_{ij}-\varepsilon$ LRR model
useful if and only if there is a phase `iphas` such as `iturb(iphas)`=30 ($R_{ij}-\varepsilon$ LRR)

`csrij` r real number > 0 [0.22] O L3
constant C_s for the $R_{ij}-\varepsilon$ LRR model
useful if and only if there is a phase `iphas` such as `iturb(iphas)`=30 ($R_{ij}-\varepsilon$ LRR)

`crijp1` r real number > 0 [0.5] O L3
constant C_1' for the $R_{ij} - \varepsilon$ LRR model, corresponding to the wall echo terms
useful if and only if there is a phase `iphas` such as `iturb(iphas)`=30 and `irijec(iphas)`=1 ($R_{ij} - \varepsilon$ LRR)

`crijp2` r real number > 0 [0.3] O L3
constant C_2' for the $R_{ij} - \varepsilon$ LRR model, corresponding to the wall echo terms
useful if and only if there is a phase `iphas` such as `iturb(iphas)`=30 and `irijec(iphas)`=1 ($R_{ij} - \varepsilon$ LRR)

Constants specific to the $R_{ij} - \varepsilon$ SSG model

`cssgs1` r real number > 0 [1.7] O L3
constant C_{s1} for the $R_{ij} - \varepsilon$ SSG model
useful if and only if there is a phase `iphas` such as `iturb(iphas)`=31 ($R_{ij} - \varepsilon$ SSG)

`cssgs2` r real number > 0 [-1.05] O L3
constant C_{s2} for the $R_{ij} - \varepsilon$ SSG model
useful if and only if there is a phase `iphas` such as `iturb(iphas)`=31 ($R_{ij} - \varepsilon$ SSG)

`cssgr1` r real number > 0 [0.9] O L3
constant C_{r1} for the $R_{ij} - \varepsilon$ SSG model
useful if and only if there is a phase `iphas` such as `iturb(iphas)`=31 ($R_{ij} - \varepsilon$ SSG)

`cssgr2` r real number > 0 [0.8] O L3
constant C_{r2} for the $R_{ij} - \varepsilon$ SSG model
useful if and only if there is a phase `iphas` such as `iturb(iphas)`=31 ($R_{ij} - \varepsilon$ SSG)

`cssgr3` r real number > 0 [0.65] O L3
constant C_{r3} for the $R_{ij} - \varepsilon$ SSG model
useful if and only if there is a phase `iphas` such as `iturb(iphas)`=31 ($R_{ij} - \varepsilon$ SSG)

`cssgr4` r real number > 0 [0.625] O L3
constant C_{r4} for the $R_{ij} - \varepsilon$ SSG model
useful if and only if there is a phase `iphas` such as `iturb(iphas)`=31 ($R_{ij} - \varepsilon$ SSG)

`cssgr5` r real number > 0 [0.2] O L3
constant C_{r1} for the $R_{ij} - \varepsilon$ SSG model
useful if and only if there is a phase `iphas` such as `iturb(iphas)`=31 ($R_{ij} - \varepsilon$ SSG)

`cssge2` r real number > 0 [1.83] O L3
constant $C_{\varepsilon 2}$ for the $R_{ij} - \varepsilon$ SSG model
useful if and only if there is a phase `iphas` such as `iturb(iphas)`=31 ($R_{ij} - \varepsilon$ SSG)

Constants specific to the v2f φ-model

`cv2fa1` r real number > 0 [0.05] O L3
constant a_1 for the v2f φ-model
useful if and only if there is a phase `iphas` such as `iturb(iphas)`=50 (v2f φ-model)

`cv2fe2` r real number > 0 [1.85] O L3
constant $C_{\varepsilon 2}$ for the v2f φ-model
useful if and only if there is a phase `iphas` such as `iturb(iphas)`=50 (v2f φ-model)

`cv2fmu` r real number > 0 [0.22] O L3
constant C_μ for the v2f φ-model
useful if and only if there is a phase `iphas` such as `iturb(iphas)`=50 (v2f φ-model)

`cv2fc1` r real number > 0 [1.4] O L3
constant C_1 for the v2f φ-model
useful if and only if there is a phase `iphas` such as `iturb(iphas)`=50 (v2f φ-model)

`cv2fc2` r real number > 0 [0.3] O L3
constant C_2 for the v2f φ-model
useful if and only if there is a phase `iphas` such as `iturb(iphas)`=50 (v2f φ-model)

`cv2fct` r real number > 0 [6] O L3
constant C_T for the v2f φ-model
useful if and only if there is a phase `iphas` such as `iturb(iphas)`=50 (v2f φ-model)

`cv2fcl` r real number > 0 [0.25] O L3
constant C_L for the v2f φ-model
useful if and only if there is a phase `iphas` such as `iturb(iphas)`=50 (v2f φ-model)

`cv2fet` r real number > 0 [110] O L3
constant C_η for the v2f φ-model
useful if and only if there is a phase `iphas` such as `iturb(iphas)`=50 (v2f φ-model)

Constants specific to the $k-\omega$ SST model

`ckwsk1` r real number > 0 [1/0.85] O L3
constant σ_{k1} for the $k-\omega$ SST model
useful if and only if there is a phase `iphas` such as `iturb(iphas)`=60 ($k-\omega$ SST)

`ckwsk2` r real number > 0 [2] O L3
constant σ_{k2} for the $k-\omega$ SST model
useful if and only if there is a phase `iphas` such as `iturb(iphas)`=60 ($k-\omega$ SST)

`ckwsw1` r real number > 0 [2] O L3
constant $\sigma_{\omega 1}$ for the $k-\omega$ SST model
useful if and only if there is a phase `iphas` such as `iturb(iphas)`=60 ($k-\omega$ SST)

`ckwsw2` r real number > 0 [1/0.856] O L3
constant $\sigma_{\omega 2}$ for the $k-\omega$ SST model
useful if and only if there is a phase `iphas` such as `iturb(iphas)`=60 ($k-\omega$ SST)

`ckwbt1` r real number > 0 [0.075] O L3
constant β_1 for the $k-\omega$ SST model
useful if and only if there is a phase `iphas` such as `iturb(iphas)`=60 ($k-\omega$ SST)

`ckwbt2` r real number > 0 [0.0828] O L3
constant β_2 for the $k-\omega$ SST model
useful if and only if there is a phase `iphas` such as `iturb(iphas)`=60 ($k-\omega$ SST)

`ckwgm1` r real number > 0 $[\frac{\beta_1}{C_\mu} - \frac{\kappa^2}{\sqrt{C_\mu}\sigma_{\omega 1}}]$ O L3
constant γ_1 for the $k-\omega$ SST model
useful if and only if there is a phase `iphas` such as `iturb(iphas)`=60 ($k-\omega$ SST)
Warning: γ_1 is calculated before the call to ***`usini1`****. Hence, if β_1, C_μ, κ or $\sigma_{\omega 1}$ is modified in* ***`usini1`****, CKWGM1 must also be modified in accordance*

`ckwgm2` r real number > 0 $[\frac{\beta_2}{C_\mu} - \frac{\kappa^2}{\sqrt{C_\mu}\sigma_{\omega 2}}]$ O L3
constant γ_2 for the $k-\omega$ SST model
useful if and only if there is a phase `iphas` such as `iturb(iphas)`=60 ($k-\omega$ SST)
Warning: γ_2 is calculated before the call to ***`usini1`****. Hence, if β_2, C_μ, κ or $\sigma_{\omega 2}$ is modified in* ***`usini1`****,* ***`ckwgm2`*** *must also be modified in accordance*

`ckwa1` r real number > 0 [0.31] O L3
constant a_1 for the $k-\omega$ SST model
useful if and only if there is a phase `iphas` such as `iturb(iphas)`=60 ($k-\omega$ SST)

`ckwc1` r real number > 0 [10] O L3
constant c_1 for the $k-\omega$ SST model
useful if and only if there is a phase `iphas` such as `iturb(iphas)`=60 ($k-\omega$ SST)

7.4 ALE

iale i 0 or 1 [C] O L1
activates (=1) or not (=0), activate the ALE module

nalinf i 0 or positive integer [0] C L2
The number of sub-iterations of initialization of the fluid

nbstr i 0 or positive integer [0] C L1
number of structures

alpnmk r real [0] C L3
alpha newmark's method

betnmk r real [-grand] C L3
beta newmark's method

gamnmk r real [-grand] C L3
gamma newmark's method

nalimx i positive integer [15] C L2
maximum number of imlicitation iterations of of the structure displacement

epalim r positive real [1.10^{-5}] C L2
Relative precision of implicitation of the structure displacement

7.5 Thermal radiative transfers: global settings

All the following key words may be modified in the user subroutines `usray*` (or, for some of them, by through the thermochemical data files). It is however not recommended to modify those which do not belong to level L1.

irayon ia 0, 1, 2 [0] O L1
for each phase `iphas`, `irayon(iphas)` activates (> 0) or deactivates (=0) the radiation module
if a specific physics is activated (in that case, `nscapp`>0), `irayon(iphas)` must be kept to 0 (see `iraypp`)
The different values correspond to the following modelings:
= 1 discrete ordinates (standard option for radiation in semi-transparent media)
= 2 "P-1" model
Warning: the P-1 model allows faster computations, but it may only be applied to media with uniform large optical thickness, such as some cases of pulverised coal combustion

iraypp i 0, 1, 2, 3 or 4 [0] O L1
when a specific physics is activated[49] (`nphas`=1, compulsory) `iraypp` indicates if thermal radiative transfers are calculated (> 0) or not (=0).
The value of `iraypp` is given *via* a data file (gas combustion: *dp_C3P*, *dp_C3PSJ*, or *dp_C4P*; pulverised coal combustion: *dp_FCP*; electric module: *dp_ELE*)
`iraypp` allows to choose between the discrete ordinates method and the P-1 method (see `irayon`) and to choose the method used to calculate the absorption coefficient. The absorption coefficient may be set by the user in the data file (then, `imodak`=0) or calculated using "Modak[50]" (then, `imodak`=1). The options are the followings:
= 1 discrete ordinates method with the absorption coefficient given by the user in the data file (`imodak`=0)
= 2 discrete ordinates method using Modak for the calculation of the absorption coefficient (`imodak`=1)
= 3 "P-1" model with the absorption coefficient given by the user in the data file (`imodak`=0)
= 4 "P-1" model using Modak for the calculation of the absorption coefficient (`imodak`=1)
For the electric module, `iraypp` is not set directly in the data file, but deduced from the type of `xkabel` specified in the file (given by `ixkabe`). In that case, `iraypp` can only be equal to 0 (`ixkabe`=0 or 2) or 1 (`ixkabe`=1)

`imodak` i 0 or 1 [0] O L3
when gas or coal combustion is activated, `imodak` indicates whether the absorption coefficient shall be calculated "automatically" (=1) or read from the data file (=0) (see `iraypp`)
useful if the radiation module is activated; `imodak` is then automatically set from the value of IRAYPP, without intervention of the user

isuird i 0 or 1 [isuite] C L1
indicates whether the radiation variables should be initialised (=0) or read from a restart file (=1)
useful if and only if the radiation module is activated (in this case, a restart file *rayamo* must be available)

[49] except with the compressible module, which is not compatible with radiation
[50] for details about the calculation of the absorption coefficient, please refer to Modak A.T., "Radiation from products of combustion"

nfreqr i strictly positive integer [1] O L1
period of the radiation module
the radiation module is called every `nfreqr` time steps (more precisely, every time `ntcabs` is a multiple of `nfreqr`). Also, in order to have proper initialisation of the variables, whatever the value of `nfreqr`, the radiation module is called at the first time step of a calculation (restart or not)
useful if and only if the radiation module is activated

ndirec i 32 or 128 [32] O L1
number of directions for the angular discretisation of the radiation propagation with the DOM model (`irayon`=1)
no other possible value, because of the way the directions are calculated
the calculation with 32 directions may break the symmetry of physically axisymmetric cases (but the cost in CPU time is much lower than with 128 directions)
useful if and only if the radiation module is activated with the DOM method

`xnp1mx` r real number [10] O L3
with the P-1 model (`irayon`=2), `xnp1mx` is the percentage of cells of the calculation domain for which it is acceptable that the optical thickness is lower than unity[51], although it is not to be desired
useful if and only if the radiation module is activated with the P-1 method

idiver i 0, 1 or 2 [2] C L1
indicates the method used to calculate the radiative source term:
= 0: semi-analytic calculation (compulsory with transparent media)
= 1: conservative calculation
= 2: semi-analytic calculation corrected in order to be globally consevative
useful if and only if the radiation module is activated
Note: if the medium is transparent, the choice has no effect on the calculation

iimpar i 0, 1 or 2 [1] O L1
choice of the display level in the listing concerning the calculation of the wall temperatures:
= 0: no display
= 1: standard
= 2: complete
useful if and only if the radiation module is activated

iimlum i 0, 1 or 2 [1] O L1
choice of the display level in the listing concerning the solution of the radiative transfer equation:
= 0: no display
= 1: standard
= 2: complete
useful if and only if the radiation module is activated

nbrvap ca string of less than 80 characters [name_iphas] O L1
name associated for the post-processing to each of the following variables, defined at the cell centers (*see* [5] for more details concerning their definitions):
`nbrvap(itsray,iphas)`: radiative source term (W/m^3)

[51] more precisely, where KL is lower than 1, where K is the absorption coefficient of the medium and L is a characteristic length of the domain

`nbrvap(iqrayp,iphas)`: radiative flux density vector (W/m^2)
`nbrvap(iabsp,iphas)`: absorption part in the source term (W/m^3)
`nbrvap(iemip,iphas)`: emission part in the source term (W/m^3)
`nbrvap(icakp,iphas)`: absorption coefficient of the medium (m^{-1})
the default values are:
`nbrvap(itsray,iphas)` = Srad_iphas
`nbrvap(iqrayp,iphas)` = Qrad_iphas
`nbrvap(iabsp,iphas)` = Absorp_iphas
`nbrvap(iemip,iphas)` = Emiss_iphas
`nbrvap(icakp,iphas)` = CoefAb_iphas
useful if and only if the radiation module is activated

irayvp ia -1 or 1 [-1] O L1
activates (=1) or deactivates (=-1) the post-processing for the each of the followiing variables defined at the cell centers:
`irayvp(itsray,iphas)`: radiative source term (W/m^3)
`irayvp(iqrayp,iphas)`: radiative flux density vector (W/m^2)
`irayvp(iabsp,iphas)`: absorption part in the source term (W/m^3)
`irayvp(iemip,iphas)`: emission part in the source term (W/m^3)
`irayvp(icakp,iphas)`: absorption coefficient of the medium (m^{-1})
useful if and only if the radiation module is activated

nbrvaf ca string of less than 80 characters [name_iphas] O L1
name associated for the post-processing to each of the following variables, defined at the boundary faces (*see* [5] for more details concerning their definitions):
`nbrvaf(itparp,iphas)`: wall temperature at the boundary faces (K)
`nbrvaf(iqincp,iphas)`: radiative incident flux density (W/m^2)
`nbrvaf(ixlamp,iphas)`: thermal conductivity of the boundary faces ($W/m/K$)
`nbrvaf(iepap,iphas)`: wall thickness (m)
`nbrvaf(iepsp,iphas)`: wall emissivity
`nbrvaf(ifnetp,iphas)`: net radiative flux density (W/m^2)
`nbrvaf(ifconp,iphas)`: convective flux density (W/m^2)
`nbrvaf(ihconp,iphas)`: convective exchange coefficient ($W/m^2/K$)
The default values are:
`nbrvaf(itparp)` = Wall_temp
`nbrvaf(iqincp)` = Incident_flux
`nbrvaf(ixlamp)` = Th_conductivity
`nbrvaf(iepap)` = Thickness
`nbrvaf(iepsp)` = Emissivity
`nbrvaf(ifnetp)` = Net_flux
`nbrvaf(ifconp)` = Convective_flux
`nbrvaf(ihconp)` = Convective_exch_coef
useful if and only if the radiation module is activated

irayvf ia -1 or 1 [-1] O L1
activates (=1) or deactivates (=-1) the post-processing for each of the followiing variables defined at the boundary faces:
`irayvf(itparp)`: wall temperature at the boundary faces (K)
`irayvf(iqincp)`: radiative incident flux density (W/m^2)
`irayvf(ixlamp)`: thermal conductivity of the boundary faces ($W/m/K$)
`irayvf(iepap)`: wall thickness (m)
`irayvf(iepsp)`: wall emissivity
`irayvf(ifnetp)`: net radiative flux density (W/m^2)
`irayvf(ifconp)`: convective flux density (W/m^2)

`irayvf(ihconp)`: convective exchange coefficient ($W/m^2/K$)
useful if and only if the radiation module is activated

`tmin`	r	real number positif	[0]	O	L3

minimum allowed value for the wall temperatures in Kelvin
useful if and only if the radiation module is activated

`tmax`	r	real number positif	[`grand` + 273.15]	O	L3

maximum allowed value for the wall temperatures in Kelvin
useful if and only if the radiation module is activated

7.6 Electric module (Joule effect and electric arc): specificities

The electric module is composed of a Joule effect module (`ippmod(ieljou)`) and an electric arc 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 impluse momentum equation. Specific boundary conditions can be applied to account for the coupled effect of transformers (offset) in glass furnaces.

The electric arc 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 key words used in the global settings are quite few. They are found in the subroutine `useli1` (see the description of this user subroutine §6.37).

ielcor	i	0, 1	[0]	O	L1

when `ielcor`=1, the boundary conditions for the potential will be tuned at each time step in order to reach a user-specified target dissipated power `puisim` (Joule effect) or a user-specified target current intensity `couimp` (electric arc)
the boundary condition tuning is controlled by the subroutine `uselrc`
alway useful

couimp	r	real number $\geqslant 0$	[0]	O	L1

with the electric arc module, `couimp` is the target current intensity (A) for the calculations with boundary condition tuning for the potential
the target intensity will be reached if the boundary conditions are expressed using the variable `dpot` or if the initial boundary conditions are multiplied by the variable `coejou`
useful with the electric arc module if `ielcor`=1

puisim	r	real number $\geqslant 0$	[0]	O	L1

with the Joule effect module, `puisim` is the target dissipated power (W) for the calculations with boundary condition tuning for the potential
the target power will be reached if the boundary conditions are expressed using the variable `dpot` or if the initial boundary conditions are multiplied by the variable `coejou`
useful with the Joule effect module if `ielcor`=1

dpot	r	real number $\geqslant 0$	[0]	O	L1

`dpot` is the potential difference (V) which generates the current (and the Joule effect) for the calculations with boundary conditions tuning for the potential. This value is initialised set by the user (`useli1`). It is then automatically tuned depending on the

value of dissipated power (Joule effect module) or the intensity of current (electric arc module). In order for the correct power or intensity to be reached, the boundary conditions for the potential must be expressed with `dpot` (`uselcl`). The tuning can be controlled in `uselrc`
useful if `ielcor`=1

`coejou` r real number $\geqslant 0$ [1] O L2
only with the Joule effect, `coejou` can be used if the user does not wish to use `dpot`; `coejou` is the coefficient to be applied to the initial potential difference to reach the target dissipated power. Its value is automatically initialised to 1 and is updated during the calculation. In order for the correct power to be reached, the boundary conditions for the potential must be expressed with `coejou` (`uselcl`). The tuning can be controlled in `uselrc`
Useful if `ielcor`=1

7.7 Compressible module: specificities

The key words used in the global settings are quite few. They are found in the subroutines `uscfx1` and `uscfx2` (see the description of these user subroutines, §6.40.1).

icfgrp ia 0 or 1 [1] C L1
for each phase `iphas`, `icfgrp(iphas)` indicates if the boundary conditions should take into account (=1) or not (=0) the hydrostatic balance.
always useful.
In the cases where gravity is predominant, taking into account the hydrostatic pressure allows to get rid of the disturbances which may appear near the horizontal walls when the flow is little convective.
Otherwise, when `icfgrp`=0, the pressure condition is calculated from the solution of the unidimensional Euler equations for a perfect gas near a wall, for the variables "normal velocity", "density" and "pressure":

Case of an expansion ($M \leqslant 0$):

$$\begin{cases} P_p = 0 & \text{if} \quad 1 + \dfrac{\gamma-1}{2} M < 0 \\ P_p = P_i \left(1 + \dfrac{\gamma-1}{2} M\right)^{\frac{2\gamma}{\gamma-1}} & \text{otherwise} \end{cases}$$

Case of a shock ($M > 0$):

$$P_p = P_i \left(1 + \frac{\gamma(\gamma+1)}{4} M^2 + \gamma M \sqrt{1 + \frac{(\gamma+1)^2}{16} M^2}\right)$$

with $M = \dfrac{\underline{u}_i \cdot \underline{n}}{c_i}$, internal Mach number calculated with the variables taken in the cell adjacent to the wall.

iviscv ia 0 or 1 [0] C L1
for each phase `iphas`, `iviscv(iphas)=0` indicates that the volume viscosity is constant and equal to the reference volume viscosity `viscv0(iphas)`.
`iviscv(iphas)`=1 indicates that the volume viscosity is variable: its variation law must be specified in the user subroutine `uscfpv`.

always useful
The volume viscosity κ is defined by the formula expressing the stress:

$$\underline{\underline{\sigma}} = -P\,\underline{\underline{Id}} + \mu(\underline{\underline{\text{grad}}}\ \underline{u} + {}^t\underline{\underline{\text{grad}}}\ \underline{u}) + (\kappa - \frac{2}{3}\mu)\,\text{div}(\underline{u})\,\underline{\underline{Id}} \quad (5)$$

viscv0 ra real number ≥ 0 [0] O L1
for each phase `iphas`, `viscv0(iphas)` is the reference volume viscosity (noted κ in the equation expressing $\underline{\underline{\sigma}}$ in the paragraph dedicated to `iviscv`)
always useful, it is the used value, unless the user specifies the volume viscosity in the user subroutine `uscfpv`

igrdpp i 0 or 1 [0] O L3
indicates whether the pressure should be updated (=1) or not (=0) after the solution of the acoustic equation
always useful

7.8 Lagrangian multiphase flows

Most of these key words may be modified in the user subroutines `uslag1`, `uslag2`, `uslabo`, `uslaen`, `uslast` and `uslaed`. It is however strongly recommended not to modify those belonging to the level L3.

First of all, it should be noted that the Lagrangian module is compliant with all the RANS turbulence models and with laminar flows. However, the particule turbulent diffusion is not specially adapted to the second order $R_{ij} - \varepsilon$ models. The same isotropic model is used as in the $k - \varepsilon$ models, with k calculated from the trace of R_{ij}. Also, two-way coupling is not compatible with the $k - \omega$ SST model.

7.8.1 Global settings

iilagr I 0, 1, 2, 3 [0] C L1
activates (>0) or deactivates (=0) the Lagrangian module
the different values correspond to the following modelings:
= 1 Lagrangian two-phase flow in one-way coupling (no influence of the particles on the continuous phase)
= 2 Lagrangian two-phase flow with two-way coupling (influence of the particles on the dynamics of the continuous phase). It must be noted that the two-way coupling is taken into account only for the first eulerian phase. Dynamics, temperature and mass may be coupled independently
= 3 Lagrangian two-phase flow on frozen continuous phase. This option can only be used in case of a calculation restart (`isuite` = 1). All the eulerian fields are frozen (including the scalar fields). This option automatically implies `iccvfg` = 1
always useful

isuila i 0, 1 [0] C L1
activation (=1) or not (=0) of a Lagrangian calculation restart. The calculation restart file read when this option is activated (`ficaml`) only contains the data related to the particles (see also `isuist`)
the global calculation must also be a restart calculation (`isuite`=1)
always useful

isuist i 0, 1 [0] C L1
during a Lagrangian calculation restart, indicates whether the particle statistics (volume and boundary) and two-way coupling terms are to be read from a restart file (=1) or reinitialised (=0). The file to be read is `ficmls`
useful if `isuila` = 1

nbpmax i positive or null integer [1000] C L1
maximum number of particles allowed simultaneously in the calculation domain. It must be reminded that the required memory evolves accordingly

`nbpart` i positive or null integer [0] O L3
number of particles treated during one Lagrangian time step
`nbpart` must always be lower than `nbpmax`
always useful, but initialised and updated without intervention of the user

`nvls` i integer between 0 and 10 [0] O L2
number of additional variables related to the particles
the additional variables can be accessed in the arrays `ettp` and `ettpa` by means of the pointer `jvls`: `ettp(nbpt,jvls(ii))` and `ettpa(nbpt,jvls(ii))` (`nbpt` is the index-number of the treated particle, and `ii` an integer between 1 and `nvls`)

isttio i 0, 1 [0] C L1
indicates the steady (=1) or unsteady (=0) state of the continuous phase flow
in particular, `isttio` = 1 is needed in order to:
calculate stationary statistics in the volume or at the boundaries (starting respectively from the Lagrangian iterations `nstist` and `nstbor`)
calculate time-averaged two-way coupling source terms (from the Lagrangian iteration `nstits`)
useful if `iilagr`=1 or `iilagr`=2 (if `iilagr`=3, then `isttio`=1 automatically)

injcon i 0, 1 [0] O L1
activates (=1) or not (=0) the continuous injection of particles
this option allows to inject particles continuously during the duration of the Lagrangian time step `dtp` rather than only once at the beginning of the Lagrangian iteration. It helps avoiding the fractioning of the particle cloud close to the injection areas

iroule i 0, 1 [0] O L1
activates (=1) or not (=0) of the particle cloning/fusion technique (option also called "Russian roulette")
when `iroule` = 1, the importance function must be specified *via* the array `croule` in the user subroutine `uslaru`

`isuivi` i 0, 1 [0 or 1] O L2
specifies if a particle should be followed (=1) or will disappear from the domain (=0) after an interaction with a boundary:
= 0: the particle must not be followed in the calculation domain after an iteraction between its trajectory and a boundary face, for instance entry (`ientrl`), outlet (`isortl`), definitive deposition on a wall (`idepo1`, `idepo2`)
= 1: the particle must still be followed in the calculation domain after an iteraction between its trajectory and a boundary face, for instance rebound (`irebol`), deposition with potential resuspension (`idepo3`)
the value of `isuivi` (`isuivi` = 0 or `isuivi` = 1) for a type of interaction can be

defined as a function of the particle behaviour or properties. It is for example the default case for the fouling interaction type (`iencrl`)
always useful

`ttclag` r positive real number [0] O L3
physical time of the Lagrangian simulation
always useful

`iplas` i integer > 0 [1] O L3
absolute iteration number (including the restarts) in the Lagrangian module (*i.e.* Lagrangian time step number)
always useful

7.8.2 Specific physics models associated with the particles

iphyla i 0, 1, 2 [0] C L1
activates (>0) or deactivates (=0) the physical models associated to the particles:
= 1: allows to associate with the particles evolution equations on their temperature (in degrees Celsius), their diameter and their mass
= 2: the particles are pulverised coal particles. Evolution equations on temperature (in degree Celsius), mass of reactive coal, mass of char and diameter of the shrinking core are associated with the particles. This option is available only if the continuous phase represents a pulverised coal flame
always useful

idpvar i 0, 1 [0] O L1
activation (=1) or not (=0) of an evolution equation on the particle diameter
useful if `iphyla` = 1

itpvar i 0, 1 [0] O L1
activation (=1) or not (=0) of an evolution equation on the particle temperature (in degrees Celsius)
useful if `iphyla` = 1 and if there is a thermal scalar associated with the continuous phase

impvar i 0, 1 [0] O L1
activation (=1) or not (=0) of an evolution equation on the particle mass
useful if si `iphyla` = 1

tpart r real number > `tkelvn` [700] O L1
initialisation temperature (in degree Celsius) for the particles already present in the calculation domain when an evolution equation on the particle temperature is activated during a calculation (`iphyla` = 1 and `itpvar` = 1)
useful if `isuila` = 1 and `itpvar` = 0 in the previous calculation

cppart r positive real number [5200] O L1
initialisation value for the specific heat ($J.kg^{-1}.K^{-1}$) of the particles already present in the calculation domain when an evolution equation on the particle temperature is activated during a calculation (`iphyla` = 1 and `itpvar` = 1)
useful if `isuila` = 1 and `itpvar` = 0 in the previous calculation

iencra i 0, 1 [0] O L1
activates (=1) or not (=0) the option of coal particle fouling. It then is necessary to specify the domain boundaries on which fouling may take place.
useful if `iphyla` = 2

tprenc r real number > `tkelvn` [600] O L1
limit temperature (in degree Celsius) below which the coal particles do not cause any fouling (if the fouling model is activated)
useful if `iphyla` = 2 and `iencra` = 1

visref r positive real number [10000] O L1
ash critical viscosity in $kg.m^{-1}.s^{-1}$, in the fouling model [52]
useful if `iphyla` = 2 and `iencra` = 1

7.8.3 Options for two-way coupling

nstits i strictly positive integer [1] O L1
number of absolute Lagrangian iterations (including the restarts) after which a time-average of the two-way coupling source terms is calculated
indeed, if the flow is steady (`isttio`=1), the average quantities that appear in the two-way coupling source terms can be calculated over different time steps, in order to get a better precision
if the number of absolute Lagrangian iterations is strictly inferior to `nstits`, the code considers that the flow has not yet reached its steady state (transition period) and the averages appearing in the source terms are reinitialised at each time step, as it is the case for unsteady flows (`isttio`=0)
useful if `iilagr` = 2 and `isttio` = 1

ltsdyn i 0, 1 [0] O L1
activation (=1) or not (=0) of the two-way coupling on the dynamics of the continuous phase
useful if `iilagr` = 2 and `iccvfg` = 0

ltsmas i 0, 1 [0] O L1
activation (=1) or not (=0) of the two-way coupling on the mass
useful if `iilagr` = 2, `iphyla` = 1 and `impvar` = 1

ltsthe i 0, 1 [0] O L1
if `iphyla` = 1 and `itpvar` = 1, `ltsthe` activates (=1) or not (=0) the two-way coupling on temperature
if `iphyla` = 2, `ltsthe` activates (=1) or not (=0) the two-way coupling on the eulerian variables related to pulverised coal combustion
useful if `iilagr` = 2

7.8.4 Numerical modeling

`nordre` i 1, 2 [2] O L2
order of integration for the stochastic differential equations

[52] J.D. Watt et T. Fereday (*J.Inst.Fuel*, Vol.42-p99)

= 1 integration using a first-order scheme
= 2 integration using a second-order scheme
always useful

`ilapoi` i 0, 1 [0] O L3
activation (=1) or not (=0) of the solution of a Poisson's equation for the correction of the particle instantaneous velocities (in order to obtain a null divergence)
this option is not validated and reserved to the development team. Do not change the default value

`idistu` i 0, 1 [1] O L3
activation (=1) or not (=0) of the particle turbulent dispersion
the turbulent dispersion is compatible only with the RANS turbulent models ($k-\varepsilon$, $R_{ij}-\varepsilon$, v2f or $k-\omega$)
(`iturb(iphas)`=20, 21, 30, 31, 50 or 60 with IPHAS = 1)
always useful

`idiffl` i 0, 1 [0] O L3
`idiffl`=1 suppresses the crossing trajectory effect, making turbulent dispersion for the particles identical to the turbulent diffusion of fluid particles
useful if `idistu`=1

modcpl i positive integer [0] O L1
activates (>0) or not (=0) the complete turbulent dispersion model
when `modcpl` is strictly positive, its value is interpreted as the absolute Lagrangian time step number (including restarts) after which the complete model is applied
since the complete model uses volume statistics, `modcpl` must either be 0 or be larger than `idstnt`
useful if `istala` = 1

idirla i 1, 2, 3 [1] O L1
x, y or z direction of the complete model
it corresponds to the main directions of the flow
useful if `modcpl` > 0

7.8.5 Volume statistics

istala i 0, 1 [0] C L1
activation (=1) or not (=0) of the calculation of the volume statistics related to the dispersed phase
if `istala` = 1, the calculation of the statistics is activated starting from the absolute iteration (including the restarts) `idstnt`
by default, the statistics are not stationary (reset to zero at every Lagrangian iteration). But if `isttio`=1, since the flow is steady, the statistics will be averaged overt he different time steps
the statistics represent the significant results on the particle cloud
always useful

seuil r positive real number [0] O L1
every cell of the calculation domain contains a certain quantity of particles, representing a certain statistical weight (sum of the statistical weights of all the particles

present in the cell). `seuil` is the limit statistical weight value, below which the contribution of the cell in term of statistical weight is not taken into account in the volume statistics (for the complete turbulent dispersion model, in the Poisson's equation used to correct the mean velocities or in the listing and post-processing outputs)
useful if `istala` = 1

idstnt i strictly positive integer [1] C L1
absolute Lagrangian iteration number (includings the restarts) after which the calculation of the volume statistics is activated
useful if `istala` = 1

nstist i integer $\geqslant$ `idstnt` [`idstnt`] O L1
absolute Lagrangian iteration number (includings the restarts) after which the volume statistics are cumulated over time (they are then said to be stationary)
if the absolute Lagrangian iteration number is lower than `nstist`, or if the flow is unsteady (`isttio`=0), the statistics are reset to zero at every Lagrangian iteration (the volume statistics are then said to be non-stationary)
useful if `istala`=1 and `isttio`=1

nomlag ca string of less than 50 characters [VarLagXXXX] O L1
name of the volumetric statistics, displayed in the listing and the post-processing files. The default value is given above, with "XXXX" representing a four digit number (for instance 0001, 0011 ...)
useful if `istala` = 1
Warning: this name is also used to reference information in the restart file (`isuist` =1). If the name of a variable is changed between two calculations, it will not be possible to read its value from the restart file

nvlsts i $0 \leqslant$ integer $\leqslant$ `nussta`=20 [0] O L1
number of additional user volume statistics
the additional statistics (or their cumulated value in the stationary case) can be accessed in the array `statis` by means of the pointer `ilvu`: `statis(iel,ilvu(ii))` (`iel` is the cell index-number and `ii` an integer between 1 and `nvlsts`)
useful if `istala` = 1

`npst` i positive integer [0] O L3
number of iterations during which stationary volume statistics have been cumulated
useful if `istala`=1, `isttio`=1 and if `nstist` is inferior or equal to the current Lagrangian iteration
`npst` is initialised and updated automatically by the code, its value is not to be modified by the user

`npstt` i positive integer [0] O L3
number of iterations during which volume statistics have been calculated (the potential iterations during which non-stationary statistics have been calculated are counted in `npstt`)
useful if `istala`=1
`npstt` is initialised and updated automatically by the code, its value is not to be modified by the user

`tstat` r positive real number [`dtp`] O L3
if the volume statistics are calculated in a stationary way, `tstat` represents the physical

time during which the statistics have been cumulated
if the volume statistics are calculated in a non-stationary way, then `tstat=dtp` (it is the Lagrangian time step, because the statistics are reset to zero at every iteration)
useful if `istala`=1
`tstat` is initialised and updated automatically by the code, its value is not to be modified by the user

7.8.6 Display of trajectories and particle movements

iensi1 i 0, 1 [0] O L1
activation (=1) or not (=0) of the post-processing in trajectory mode
this option generates files allowing to display the trajectory of some pre-selected particles in the *EnSight6* format
always useful
Warning: this option very expensive with regards to CPU time and may generate very large files

iensi2 i 0, 1 [0] O L1
activation (=1) or not (=0) of the post-processing in movement mode
This option generates files allowing to display the movement of some pre-selected particles in the *EnSight6* format
always useful
Warning: this option very expensive with regards to CPU time and may generate very large files

nbvis i positive integer [nliste] O L1
number of particles selected for post-processing display in trajectory or movement mode
`nbvis` must be lower than `nbpmax` and `nliste` (set to 500 in `lagpar.h` and not to be modified)
useful if `iensi1` = 1 or `iensi2` = 1

nvisla i strictly positive integer [1] O L1
output period for the post-processing in trajectory or movement mode
may be useful to diminish the size of the post-processing files
useful if `iensi1` = 1 or `iensi2` = 1

liste ia positive integers [between 1 and 500] O L1
contains the index-numbers of the particles selected for the display in trajectory or movement mode
useful if `iensi1` = 1 or `iensi2` = 1

ivisv1 i 0, 1 [0] O L1
associates (=1) or not (=0) the variable "velocity of the locally undisturbed fluid flow field" with the display in trajectory or movement mode
useful if `iensi1` = 1 or `iensi2` = 1

ivisv2 i 0, 1 [0] O L1
associates (=1) or not (=0) the variable "particle velocity" with the display in trajectory or movement mode
useful if `iensi1` = 1 or `iensi2` = 1

ivistp i 0, 1 [0] O L1
associates (=1) or not (=0) the variable "residence time" with the display in trajectory or movement mode
useful if `iensi1` = 1 or `iensi2` = 1

ivisdm i 0, 1 [0] O L1
associates (=1) or not (=0) the variable "particle diameter" with the display in trajectory or movement mode
useful if `iensi1` = 1 or `iensi2` = 1

iviste i 0, 1 [0] O L1
associates (=1) or not (=0) the variable "particle temperature" with the display in trajectory or movement mode
useful if `iensi1` = 1 or `iensi2` = 1

ivismp i 0, 1 [0] O L1
associates (=1) or not (=0) the variable "particle mass" with the display in trajectory or movement mode
useful if `iensi1` = 1 or `iensi2` = 1

ivishp i 0, 1 [0] O L1
associates (=1) or not (=0) the variable "temperature of the coal particles" with the display in trajectory or movement mode
useful if `iensi1` = 1 or `iensi2` = 1, if and only if `iphyla` = 2

ivisdk i 0, 1 [0] O L1
associates (=1) or not (=0) the variable "shrinking core diameter of the coal particles" with the display in trajectory or movement mode
useful if `iensi1` = 1 or `iensi2` = 1, if and only if `iphyla` = 2

ivisch i 0, 1 [0] O L1
associates (=1) or not (=0) the variable "mass of reactive coal of the coal particles" with the display in trajectory or movement mode
useful if `iensi1` = 1 or `iensi2` = 1, if and only if `iphyla` = 2

ivisck i 0, 1 [0] O L1
associates (=1) or not (=0) the variable "mass of char of the coal particles" with the display in trajectory or movement mode
useful if `iensi1` = 1 or `iensi2` = 1, if and only if `iphyla` = 2

7.8.7 Display of the particle/boundary interactions and the statistics at the boundaries

iensi3 i 0, 1 [0] C L1
activation (=1) or not (=0) of the recording of the particle/boundary interactions in `parbor`, and of the calculation of the statistics at the corresponding boundaries, for post-processing (*EnSight6* format)
By default, the statistics are non-stationary (reset to zero at every Lagrangian iteration). They may be stationary if `isttio`=1 (*i.e.* calculation of a cumulated value over time, and then calculation of an average over time or over the number of interactions

with the boundary)
always useful

nstbor i strictly positive integer [1] O L1
number of absolute Lagrangian iterations (including the restarts) after which the statistics at the boundaries are considered stationary and are averaged (over time or over the number of interactions)
If the number of absolute Lagrangian iterations is lower than `nstbor`, or if `isttio`=0, the statistics are reset to zero at every Lagrangian iteration (non-stationary statistics)
useful if `iensi3`=1 and `isttio`=1

seuilf r positive real number [0] O L1
every boundary face of the mesh undergoes a certain number of interactions with particles, expressed in term of statistical weight (sum of the statistical weights of all the particles which have interacted with the boundary face). `seuilf` is the limit statistical weight value, below which the contribution of the face is not taken into account in the statistics at the boundaries for post-processing
useful if `iensi3`=1

inbrbd i 0, 1 [1] O L1
activation (=1) or not (=0) of the recording of the number of particle/boundary interactions, and of the calculation of the associated boundary statistics.
`inbrd` = 1 is a compulsory condition to use the particulate average `imoybr` = 2
the selection of the type of interactions that are to be recorded is specified in the subroutine `uslabo`
useful if `iensi3`=1

iflmbd i 0, 1 [0] O L1
activation (=1) or not (=0) of the recording of the particulate mass flow related to the particle/boundary interactions, and of the calculation of the associated boundary statistics
the selection of the type of interactions that are to be recorded is specified in the subroutine `uslabo`
`inbrd` = 1 is a compulsory condition to use `iflmbd`=1
useful if `iensi3`=1 and `inbrbd`=1

iangbd i 0, 1 [0] O L1
activation (=1) or not (=0) of the recording of the angle between a particle trajectory and a boundary face involved in a particle/boundary interaction, and of the calculation of the associated boundary statistics
the selection of the type of interactions that are to be recorded is specified in the subroutine `uslabo`
useful if `iensi3`=1

ivitbd i 0, 1 [0] O L1
activation (=1) or not (=0) of the recording of the velocity of a particle involved in a particle/boundary interaction, and of the calculation of the associated boundary statistics
the selection of the type of interactions that are to be recorded is specified in the subroutine `uslabo`
useful if `iensi3`=1

iencbd i 0, 1 [0] O L1
activation (=1) or not (=0) of the recording of the mass of coal particles stuck to the wall due to fouling, on the boundary faces of the `iencrl` interaction type
useful if `iensi3`=1, `iphyla`=2, `iencra`=1, and if there is at least one boundary face of the `iencrl` interaction type

nusbor i positive integer [0] O L1
number additional user data to record for the calculation of additional boundary statistics in `parbor`
useful if `iensi3`=1

nombrd ca string of less than 50 characters [see `uslag1`] O L1
name of the boundary statistics, displayed in the listing and the post-processing files
useful if `iensi3`=1
Warning: this name is also used to reference information in the restart file (`isuist` =1). If the name of a variable is changed between two calculations, it will not be possible to read its value from the restart file

imoybr ia 0, 1, 2 [0 , 1 or 2] O L1
the recordings in `parbor` at every particle/boundary interaction are cumulated values (possibly reset to zero at every iteration in the non-stationary case). They must therefore be divided by a quantity to get boundary statistics. The user can choose between two average types:
= 0: no average is applied to the recorded cumulated values
= 1: a time-average is calculated. The cumulated value is divided by the physical duration in the case of stationary averages (`isttio`=1). The cumulated value is divided by the value of the last time step in the case of non-stationary averages (`isttio`=0), and also in the case of stationary averages while the absolute Lagrangian iteration number is inferior to `nstbor`
= 2: a particulate average is calculated. The cumulated value is divided by the number of particle/boundary interactions (in terms of statistical weight) recorded in `parbor(nfabor,inbr)`. This average can only be calculated when `inbrbd`=1. The average is calculated if the number of interactions (in statistical weight) of the considered boundary face is strictly higher than `seuilf`, otherwise the average at the face is set to zero
only the cumulated value is recorded in the restart file
useful if `iensi3`=1

`npstf` i positive integer [0] O L3
number of iterations during which stationary boundary statistics have been cumulated
useful if `iensi3`=1, `isttio`=1 and `nstbor` inferior or equal to the current Lagrangian iteration
`npstf` is initialised and updated automatically by the code, its value is not to be modified by the user

`npstft` i positive integer [0] O L3
number of iterations during which boundary statistics have been calculated (the potential iterations during which non-stationary statistics have been calculated are counted in `npstft`)
useful if `iensi3`=1
`npstft` is initialised and updated automatically by the code, its value is not to be modified by the user

`tstatp`	r	positive real number	[`dtp`]	O	L3

if the recording of the boundary statistics is stationary, `tstatp` contains the cumulated physical duration of the recording of the boundary statistics
if the recording of the boundary statisticss is non-stationary, then `tstat=dtp` (it is the Lagrangian time step, because the statistics are reset to zero at every time step)
useful if `iensi3`=1

8 Bibliography

[1] Archambeau F., *et al.*,
Note de validation de Code_Saturne version 1.1.0,
Rapport EDF HI-83/04/003/A, 2004 (in french).

[2] Benhamadouche S.,
Modélisation de sous-maille pour la LES - Validation avec la Turbulence Homogène Isotrope (THI) dans une version de développement de Code_Saturne,
Rapport EDF HI-83/01/033/A, 2001 (in french).

[3] Boucker M., Archambeau F., Méchitoua N.,
Quelques éléments concernant la structure informatique du Solveur Commun - Version 1.0_init0,
Compte-rendu express EDF I81-00-8, 2000 (in french).

[4] Boucker M., Mattéi J.D.,
Proposition de modification des conditions aux limites de paroi turbulente pour le Solveur Commun dans le cadre du modèle $k-\varepsilon$ standard,
Rapport EDF HI-81/00/019/A, 2000 (in french).

[5] Douce A., Méchitoua N.,
Mise en œuvre dans Code_Saturne des physiques particulières. Tome3 : Transfert thermique radiatif en milieu gris semi-transparent,
Rapport EDF HI-81/02/019/A, 2002 (in french).

[6] Douce A.,
Physiques particulières dans Code_Saturne 1.1, Tome 5 : modélisation stochastique lagrangienne d'écoulements turbulents diphasiques polydispersés,
Rapport EDF, HI-81/04/03/A, 2005 (in french).

[7] Escaich A., Plion P., *Mise en œuvre dans Code_Saturne des modélisations physiques particulières. Tome 1 : Combustion en phase gaz*,
Rapport EDF, HI-81/02/03/A, 2002 (in french).

[8] Escaich A., *Mise en œuvre dans Code_Saturne des modélisations physiques particulières. Tome 2 : Combustion du charbon pulvérisé*,
Rapport EDF, HI-81/02/09/A, 2002 (in french).

[9] Fournier Y.,
Code_Saturne 2.0.0-rc1 guide pratique et théorique du Preprocesseur,
on line with the release of *Code_Saturne* 2.0.0-rc1 (`info_cs ecsmu`).

[10] Méchitoua N., Archambeau F.,
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,
Rapport EDF HE-41/98/010/B, 1998 (in french).

[11] *Code_Saturne* documentation,
Code_Saturne 2.0.0-rc1 Theory and Programmer's guide,
on line with the release of *Code_Saturne* 2.0.0-rc1 (`info_cs theory`).

[12] Sakiz M., Équipe de validation,
Validation de Code_Saturne version 1.2 : note de synthèse,
Rapport EDF H-I83-2006-00818-FR, 2006 (in french).

[13] Tagorti M., Dal-Secco S., Douce A., Méchitoua N.,
Physiques particulières dans Code_Saturne, tome 4 : le modèle P-1 pour la modélisation des transferts thermiques radiatifs en milieu gris semi-transparent,
Rapport EDF HI-81/03/017/A, 2003 (in french).

[14] *Code_Saturne* DOCUMENTATION,
Code_Saturne version 2.0.0-rc1 turorial, on line with the release of *Code_Saturne* 2.0.0-rc1 (`info_cs tutorial`).

9 Appendix 1 : automatic validation procedure

9.1 Introduction

This document is the practical user guide for the autovalidation procedure associated with *Code_Saturne* version 2.0.0-rc1. The aim of this document is to guide the user through all the steps necessary for the running and the user-understanding of the autovalidation procedure. The guide describes the selected test cases, the modifiable settings and the procedure to add a case in the reference base.

The procedure is written in python language and a XML file containing the data settings is necessary.

9.2 Practical informations on the procedure

This procedure aims to run automatically all the selected cases and to compare the obtained results with those of the reference base. All the comparisons are summarized in a report file. If the discrepancies between the reference and the test overpass a determined tolerance, the procedure creates an EnSight part containing the variable differences.

For each test cases, the detailed actions are the following:

- preparation of the study with the *cs_create* utility,
- copy of all the necessary files (meshes, XML data file, user fortran files) from the reference base,
- execution of the case with the *runcase* utility,
- comparison between the reference results and the test results,
- update of the report file.

First, an empty directory named BASETEST is generated by the user. In this directory, the command to launch the script is the following:

autovalid -f [xml file name] [-d [tmp directory]]

where *xml file name* is the data file containing the settings necessary to the autovalidation. The reference base has to be easily updatable. The user has to copy this file, initially associated to the directory BASEREF, in the directory BASETEST in order to modify it (for example, if the user doesn't want to execute all the tests or if he wants to compare only some variables).

9.3 Directories architecture

The typical architecture is given in the following section:

- a directory BASEREF containing all the reference studies (five elementary tests GRADIENT and LAPLACIEN) and the XML data file *autovalid.xml*,
- the user has to create a directory BASETEST and to copy the XML data file *autovalid.xml* in this directory before launching the script,
- a directory Autovalid containing all the python source files.

9.4 Validation base

The selected cases in the reference directory are:

- GRADIENT : elementary tests of gradient calculation using the different methods proposed by *Code_Saturne*,
- LAPLACIEN : resolution of a laplacian equation.

9.4.1 Elementary tests : gradient calculations

The elementary tests are performed on a cubical mesh composed by hexahedrons and tetrahedrons with non-conforming merging. The mesh is generated using Simail-6.4 mesher (file with extension *.des*).

All the tests are contained in the fortran file *testel.F* called by the fortran file *caltri.F*. To activate the elementary tests, we use the existing parameter IVERIF in the main program *cs_main.c*. IVERIF is initialised to -1 (no action). If IVERIF takes the value 0, there is no difference with the standard version. If IVERIF > 0, *testel.F* will be called with :

- IVERIF = 1: IMRGRA = 0
- IVERIF = 2: IMRGRA = 1
- IVERIF = 3: IMRGRA = 2
- IVERIF = 4: IMRGRA = 3
- IVERIF = 5: IMRGRA = 4

A new keyword ARG_CS_VERIF refering to IVERIF is added in the universal launch script *lance*, so that the command line is: *cs14.exe -v ARG_CS_VERIF*.

The test case consists in calculating the gradient of sin(x+2y+3z) with the different methods implemented in *Code_Saturne* (boundary conditions are treated with Dirichlet condition). We compare the result to the reference solution (not to the exact solution).

9.4.2 Laplacien calculation

The mesh is the same as for the previous elementary tests.

The case consists in the resolution of a stationary equation without convection terms for a passive scalar. The source term and Dirichlet boundary conditions are specified so that the solution is sin(x+2y+3z). The source term is imposed in the fortran file *ustssc.F*. We compare the result with the reference solution (not with the exact solution).

9.5 Architecture description

In the directory Autovalid, the user finds all the python source files necessary to the execution of the procedure. The main file *autovalid* runs the autovalidation and manages the general printouts.

9.5.1 Python files in the modules directory

All the python files are listed here:

- *Common.py*: this file contains the global variables (XML file name, reference version, reference path, temporary directory and local directory),
- *CommandLine.py*: this file manages the command line usage,
- *Parser.py*: this file defines the parser class which loads and reads the XML data file,

- *Study.py*: this file defines the study class which contains case objects,
- *Case.py*: this file defines the case class which contains the launch script and the listing and chrono comparisons,
- *Listing.py*: this file defines the listing class which contains minima/maxima variables and clippings,
- *Chrono.py*: this file defines the chrono class which contains a list of values and creates, if necessary, a part EnSight (if tolerance > specified value).

9.5.2 XML file description

The XML file contains all the data to run the different cases. It is important to note the definitions of the following attributes:

- *label* refers to the name of the study, the name of the case, the name of the variable or the name of the post-treatment script,
- *status* is 'on' or 'off' to activate or not the action,
- *compute* is 'on' or 'off' to run or not the calculation,
- *tolerance* is the maximum allowed value for the norm of the variable X defined by

$$\frac{|X_{Ref} - X_{Test}|}{|X_{max_{Ref}} - X_{min_{Ref}} + \varepsilon|}$$

.

An example of XML data file is given below:

```
<?xml version="1.0"?>
<autovalid name="Validation Code_Saturne V1.4">
  <referencepath>/home/saturne/BASEREF</referencepath>
  <referenceversion>SaturneV1.4</referenceversion>

  <study label='LAPLACIEN' status='on'>
    <variable label='passif' status='on'>
      <tolerance>0.1</tolerance>
    </variable>
    <variable label='Pression' status='on'>
      <tolerance>0.1</tolerance>
    </variable>
    <variable label='VitesseX' status='on'>
      <tolerance>0.1</tolerance>
    </variable>

    <case label='CAS1' status='on' compute='on'>
      <post label='depou_elargb' status='off'> </post>
    </case>

    <case label='VERIF' status='on' compute='on'>
      <post label='depou_elargb' status='off'> </post>
    </case>

    <case label='2PROCS' status='on' compute='on'>
```

```
      <post label='depou_elargb' status='off'> </post>
      <nproc>2</nproc>
    </case>
  </study>

</autovalid>
```

Note : If *status* is 'on' and *compute* is 'off', we compare listing files and chrono files but if there isn't result available, *compute* becomes 'on'.

9.5.3 To add a new study

To add a new study in the reference base, the user has to create and run a calculation in the directory BASEREF. He also has to add the following typical section in the XML data file:

```
<study label='GRADIENT' status='on'>
  <variable label='gradient' status='on'>
    <tolerance>0.1</tolerance>
  </variable>

  <case label='CAS1' status='on' compute='on'>
  </case>
</study>
```

For example, this previous sequence means that the user wants to run (compute is 'on') and compare the variable 'gradient' with a tolerance 0.1 for the case CAS1 of the study GRADIENT.

9.5.4 Report files

There are three kinds of report file :

- *report.txt*: this general file contains just OK, NOK, 'Execution error' or 'Compilation error' for each case of each study,
- *STUDY_listing.report*: this file depends on the study and contains the listing files comparison for each selected variable at the last time step (min/max values, min/max norms, min/max clippings),

$$Norm_{X_{max}} = \frac{|X_{max_{Ref}} - X_{max_{Test}}|}{|X_{max_{Ref}} - X_{min_{Ref}} + \varepsilon|}$$

$$Norm_{X_{min}} = \frac{|X_{min_{Ref}} - X_{min_{Test}}|}{|X_{max_{Ref}} - X_{min_{Ref}} + \varepsilon|}$$

- *STUDY_chrono.report*: this file depends on the study and contains the chrono files comparison for each selected variable at the last time step (maximum difference, mean difference and norm),

$$\delta_{max} = max|X_{Ref} - X_{Test}|$$

$$\delta_{mean} = \frac{\sum |X_{Ref} - X_{Test}|}{Nb_{values}}$$

$$Norm = \frac{\delta_{max}}{|X_{max_{Ref}} - X_{min_{Ref}} + \varepsilon|}$$

Index of the main variables and keywords

– Symbols –

iortvm 48
isvhb 49
isvtb 49
isympa 50
ttclag 164
coejou 60
dpot 60
dtcmom 49
icdpar 60
icdtmo 49
icodcl 64
idtmom 49
iep 45
ifb 45
ik 45
ilphas 56
iomg 45
iphi 45
iphsca 45
ipr 45
ir11 45
ir12 45
ir13 45
ir22 45
ir23 45
ir33 45
iscapp 46
iscavr 46
isca 45
itypfb 64
iu 45
iv 45
iw 45
rcodcl 64

– A –

a1ch 93
a2ch 93
ahetch 93
ales 81, 132
almax 152
alpnmk 156
anomax 138
arak 141
atcoel 93
atgaze 91
auxl 59

– B –

betnmk 156
blencv 140
blency 145
bles 81, 132

– C –

cch 93
cck 93
cdgfac 43
cdgfbo 43
cdries 131
cdtvar 127
ce1 153
ce2 153
ce4 153
cebu 99
ckabs1 93
ckabsg 92
ckupdc 51, 76
ckwa1 156
ckwbt1 155
ckwbt2 156
ckwc1 156
ckwgm1 156
ckwgm2 156
ckwsk1 155
ckwsk2 155
ckwsw1 155
ckwsw2 155
climgr 138
climgy 145
cmu 152
coefa 49
coefb 49
coejou 161
compog 91
couimp 99, 160
coumax 127
coumxy 145
cp0 150
cp2ch 93
cpashc 93
cpgd1 58
cpgd2 58
cpght 58
cppart 164
crij1 153
crij2 153
crij3 153
crijep 153
crijp1 154
crijp2 154
croule 59, 163
csmago 81, 132
csrij 153

cssge2 154
cssgr1 154
cssgr2 154
cssgr3 154
cssgr4 154
cssgr5 154
cssgs1 154
cssgs2 154
cstlog 152
cv2fa1 154
cv2fc1 155
cv2fc2 155
cv2fcl 155
cv2fct 155
cv2fe2 155
cv2fet 155
cv2fmu 152, 155

– D –

diam20 93
diftl0 ..... 99, 151
distch 96
dpot 160
dt 51
dtmax 127
dtmin 127
dtp 163, 168, 172
dtpt1d 78
dtref 127

– E –

e1ch 93
e2ch 93
ehetch 93
ehgazg 92
emphis 118
epalim 156
eppt1d 78
epscvy 146
epsilo 139
epsily 145
epsrgr 138
epsrgy 145
epsrsm 147
epszer 148
epzero 147
ettp 56, 163
ettpa 57, 163
exthis 118
extrag 138
extray 145

– F –

ficaml 115
ficamo 113
ficamr 114
ficamx 113
ficava 113
ficavl 115
ficavr 114
ficavx 113
ficfpp 114
ficgeo 113
ficjnf 115
ficmls 115
ficmt1 113
ficmvo 114
ficstp 113
ficush 119
ficusr 121
ficvls 115
ficvt1 113
ficvvo 114
fment 95
fmtchr 116
foumax 127
fs(1) 92

– G –

gamnmk 156
gradpr 58
gradvf 58
grand 147
gx,gy,gz 148

– H –

h0ashc 93
hbord 49
hch 93
hck 93
hept1d 78

– I –

ia 52
iale 156
ialtyb 79
iangbd 170
ibfixe 79
icalhy 141
icapt 72
iccvfg 146
icdpar 128, 143
icdtmo 48
icepdc 50
icepdp 76
icetsm 51, 77
icfgrp 161
ichrbo 116
ichrmd 116
ichrsy 116
ichrvl 116
ichrvr 97, 117

ickabs 97
ickupd 50
iclkep 130
iclptr 130
iclrtp 49
iclsyr 130
iclt1d 78
iclvfl 125
iclvor 69
icmome 48
icocel 56
icod3p 89
icoebu 89
icoef 49
icoeff 49
icolwc 89
icompf 90
iconv 125
icour 48
icp 47, 150
icp3pl 90
icpa 47
icpext 135
icpl3c 90
icpsyr 125
idebty 68
idepo1 105
idepo2 105
idepo3 105
ideuch 128
idevel 53
idfmom 120
idiam2 98
idiff 125
idiffl 166
idifft 126
idifre 130
idiipb 44
idijpf 44
idipar 50
idircl 126
idirla 166
idist 44
idistb 44
idistu 166
idiver 158
idofij 44
idpvar 164
idries 131
idstnt 167
idtvar 126
idvukw 52
iecaux 59, 122
iefjou 101
ielarc 90, 160
ielcor 160
ieljou 90, 160
iencbd 171
iencra 165
iencrl 105
iensi1 168
iensi2 168
iensi3 169
ientat 96
ientcp 96
ientfu 95
ientgb 95
ientgf 95
ientox 95
ientre 64, 95
ientrl 105
ieppt1 51
iescal 143
iescor 47, 142
iesder 47, 142
iespre 47, 142
iestim 47, 142
iestot 47, 143
if1m 97, 98
if2m 97, 98
if3m 97, 98
if3p2m 98
if4p2m 97
if4pm 98
ifabor 43
ifacel 43
ifinty 68
iflmbd 170
ifluaa 47
ifluma 47
ifm 96, 97
ifmcel 51
ifmfbr 49
ifoenv 26
ifour 48
ifp2m 96, 97
ifp3m 97
ifrlag 105
igfuel 91
igliss 79
igmdch 98
igmdv1 98
igmdv2 98
igmhet 98
igoxy 91
igrake 129
igrari 130
igrdpp 162
igrhok 129
ih2 ..... 97, 98

ihisvr 118
ihm 96–98, 100, 101
iicelb 44
iicepd 50
iicesm 51
iifapa 50
iifpt1 51
iilagr 162
iimlum 158
iimpar 158
iindef 64
iisymp 50
iitpsm 51
iitrif 50
iitypf 50
ikecou 129
ilapla(i) 101
ilapoi 166
ileaux 59, 122
ilisvr 97, 121
ilogpo 129
ilvu 167
imgr 139
imgrpy 145
imligr 138
imligy 145
immel 98
imodak 157
imoold 120
imoybr 171
impdvo 114
impfpp 114
impgeo 113
imphis 118
impjnf 115
impla1 115
impla2 115
impla3 115
impla4 115
impla5 115
impmvo 114
impstp 113
impush 119
impusr 121
impvar 164
impvvo 114
imrgra 137
imvisf 146
inbrbd 170
indep 57
indjon 90
injcon 163
inp 96, 98
inpdt0 122
inppt1 51
iochet 93
iparoi 64
iparug 64
ipci 93
iphsca 124
iphydr 141
iphyla 164
iplas 164
ipnfac 43
ipnfbr 43
ipond 44
ipoti 101
ipotr 100, 101
ipotva 100
ippmod 89
ipppro 97
ipprob 46
ipproc 46, 97
ipprof 46
iprco 141
iprfml 50
iprtot 48
ipstcl 117
ipstdv 117
ipstft 117
ipstyp 117
iptlro 126
ipucou 146
iqimp 95
irayon 157
iraypp 157
irayvf 159
irayvp 159
ircflu 146
ircfly 145
irebol 105
irepvo 69
iresol 139
irevmc 141
irijec 130
irijnu 131
irijrb 131
iroext 134
irom 46
irom2 98
iroma 46
iroule 163
irovar 148
is2kw 52
iscalt 46, 124
iscavr 124
ischcv 140
ischcy 145
ischtp 133
iscold 123

iscsth ... 124
ismace ... 51
ismago ... 48
isno2t ... 133
isolib ... 64
isortl ... 105
isrfan ... 44
isrfbn ... 44
isso2t ... 134
isstpc ... 140
isstpy ... 145
istala ... 166
istat ... 125
istmpf ... 133
isto2t ... 134
isttio ... 163
isuila ... 162
isuird ... 157
isuist ... 163
isuit1 ... 146
isuite ... 122
isuivi ... 109, 163
isuivo ... 131
isymet ... 64
it3m ... 97
it4m ... 97
itbrrb ... 125
itemp ... 97, 100, 101
itemp1 ... 98
itemp2 ... 98
itepa ... 57
itpvar ... 164
itrifb ... 50, 68
itsnsa ... 47
itssca ... 47
itstua ... 47
iturb ... 128
ituser ... 52
itycel ... 56
itypfb ... 50
itypsm ... 51, 77
iu ... 95
iuetbo ... 50
iusclb ... 105
iuslag ... 106
iusncl ... 105
iusvis ... 108
iv ... 95
iviext ... 134
ivimpo ... 79
ivisch ... 169
ivisck ... 169
iviscl ... 47
ivisct ... 47
iviscv ... 161
ivisdk ... 169
ivisdm ... 169
ivishp ... 169
ivisla ... 47
ivisls ... 47, 151
ivisma ... 48
ivismp ... 169
ivissa ... 48
ivisse ... 126
ivista ... 47
iviste ... 169
ivistp ... 169
ivisv1 ... 168
ivisv2 ... 168
ivitbd ... 170
ivivar ... 148
ivrtex ... 59, 131
ivsext ... 135
iw ... 95
iwarni ... 121
iwarny ... 144
ix2 ... 98
ixch ... 96, 98
ixck ... 96, 98
iy1ch ... 93
iy2ch ... 93
iycoel ... 100, 101
iygfm ... 96, 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
iyppar ... 50
izone ... 95

– J –

jbord1 ... 105
jvls ... 163

– K –

kabse ... 91

– L –

liste ... 168
lndfac ... 41
lndfbr ... 41
lndnod ... 56
longia ... 42
longra ... 42
ltsdyn ... 165

ltsmas 165
ltsthe 165

– M –

modcpl 166

– N –

nalimx 156
nalinf 156
nato 91, 93
nbmomt 42
nbmomx 42
nbpart 163
nbpmax 56, 163
nbrvaf 159
nbrvap 158
nbstr 156
nbvis 168
ncapt 118
ncegrm 139
ncel 41
ncelbr 41
ncelet 41
ncepdc 51, 76
ncepdp 76
ncesmp 77
ncetsm 51, 77
ncharb 93, 107
ncharm 90, 107
nclacp 42, 93
nclagm 105
nclcpm 42
nclpch 90, 93
ncoel 93
ncpcmx 90
ncymax 139, 140
ndgmox 42
ndim 41
ndirec 158
ndlagm 107
ndlaim 106
nestmx 42, 142
nfabor 41
nfac 41
nflagm 105
nfml 41
nfpt1d 51, 78
nfreqr 158
nfrlag 105
ngaze 91
ngazg 91, 94
ngrmax 140
ngrmmx 52
nideve 52
nitmax 139
nitmay 144
nitmgf 140
nituse 52
nivep 56
nnent 69
nnod 41
nodfac 41, 43
nodfbr 41, 43
nombrd 171
nomcoe 91
nomcoel 93
nomlag 167
nomvar 121
nordre 165
nphas 41
nphsmx 41
npo 91, 93, 94
nppt1d 78
nprfml 41
nproce 42
nprofa 42
nprofb 42
npromx 42
npst 167
npstf 171
npstft 171
npstt 167
nrdeve 52
nrgaz 91
nrtuse 52
nscal 41
nscamx 41
nscapp 41
nscaus 42, 124
nstbor 170
nstist 167
nstits 165
nswrgr 137
nswrgy 145
nswrsm 147
nswrsy 144
ntcabs 122
ntchr 117
ntcmxy 144
ntdmom 120
ntersl 56
nthist 119
nthsav 119
ntlist 121
ntmabs 123
ntpabs 123
ntsuit 122
ntypmx 66
nusbor 56, 108, 171
nushmx 42

nvar ... 41
nvep ... 56
nvgaus ... 56
nvisbr ... 56
nvisla ... 168
nvisls ... 42
nvls ... 56, 163
nvlsta ... 56
nvlsts ... 56, 167
nvort ... 69
nvp ... 56

– O –

och ... 93
ock ... 93
optchr ... 116

– P –

p0 ... 149
parbor ... 58, 169, 171
pcich ... 93
pcick ... 93
permvi ... 148
pi ... 147
pred0 ... 149
prefth ... 148
propce ... 44
propfa ... 44
propfb ... 44
puisim ... 160
puismp ... 99

– Q –

qimp ... 95
qimpat ... 96
qimpcp ... 96

– R –

ra ... 52
rcodcl ... 95
rcpt1d ... 78
rdevel ... 53
relaxk ... 130
relaxp ... 141
rgpt1d ... 78
rho0ch ... 93
rinfin ... 147
ro0 ... 148
rr ... 148
rtp ... 44, 96
rtpa ... 44
rtuser ... 52
ruslag ... 107
rvarfl ... 152

– S –

scamax ... 151
scamin ... 151
seuil ... 166
seuilf ... 170
sigmae ... 153
sigmak ... 153
sigmas ... 152
smacel ... 51, 77
smagmx ... 132
srrom ... 98, 149
statis ... 58, 167
stephn ... 148
stoeg ... 91
surfac ... 43
surfbo ... 43

– T –

t0 ... 150
tbord ... 49
tepa ... 57
tept1d ... 78
th ... 92
thetav ... 135
thetcp ... 137
thetfl ... 135
thetro ... 136
thetsn ... 136
thetss ... 136
thetst ... 136
thetvi ... 136
thetvs ... 137
timpat ... 96
timpcp ... 96
tinfue ... 96
tinoxy ... 96
tkelvi ... 147
tkelvn ... 148
tkent ... 95
tmarus ... 123
tmax ... 91, 93, 160
tmin ... 91, 93, 160
tpart ... 164
tppt1d ... 78
tprenc ... 106, 165
trefth ... 148
tslagr ... 59
tstat ... 110, 167
tstatp ... 172
ttcabs ... 123
ttpabs ... 123

– U –

uref ... 152

– V –

vagaus ... 59
varrdt ... 127

viscl0 149
viscv0 162
visls0 151
visref 106, 165
vitflu 58, 109
vitpar 57, 109
volmol 148
volume 43

– W –

wmolat 91, 93
wmolg 92

– X –

xashch 93
xco2 92
xh2o 92
xkabel 94
xkappa 152
xlesfd 132
xlesfl 81, 132
xlmt1d 78
xlomlg 152
xnp1mx 158
xyzcap 118
xyzcen 43
xyznod 43
xyzp0 150

– Y –

y1ch 93
y2ch 93
yplmxy 146
ypluli 129

– Z –

zero 147