TRUST Documentation

TRUST

May 29, 2024

CONTENTS

Hi there and welcome to the **TRUST documentation** page !

This page provides:

- **TRUST Generic Guide** can be found there: *TRUST Generic Guide*
- **TRUST Keyword Reference Manual** can be found there: *TRUST Keyword Reference Manual*
- **TRUST Tools Documentation** can be found there: TRUST Tools Documentation
- **TRUST C++ API Documentation** can be found there: TRUST C++ API

You can use the search bar located on the top right of your screen to lookup any keyword, or item in the generic guide. Use the dedicated Doxygen search box for C++ classes.

Here are some useful links that you can visit too:

- **TRUST Code**
  https://github.com/cea-trust-platform/trust-code
- **TRUST Website**
  https://cea-trust-platform.github.io
- **TRUST Support**
  trust@cea.fr

CHAPTER

ONE

TRUST GENERIC GUIDE

You will find here the TRUST generic guide, giving you a brief overview on how to use TRUST (working principles, brief syntax and tools overview).

Do not forget that you can use the research bar located on the top right of your screen to quickly lookup a precise element or keyword.

1.1 Table Of Contents

1.1.1 Introduction

TRUST is a High Performance Computing (HPC) thermal-hydraulic engine for Computational Fluid Dynamics (CFD) developed at the Departement of System and Structure Modelisation (DM2S) of the French Atomic Energy Commission (CEA).

The acronym **TRUST** stands for **TR**io_**U** **S**oftware for **T**hermohydraulics. This software was originally designed for conduction, incompressible single-phase, and Low Mach Number (LMN) flows with a robust Weakly-Compressible (WC) multi-species solver. However, a huge effort has been conducted recently, and now TRUST is able to simulate real compressible multi-phase flows.

TRUST is also being progressively ported to support GPU acceleration (NVidia/AMD).

The software is OpenSource with a BSD license, available on GitHub via this link.

You can easily create new project based on **TRUST** plateform. Theses projects are named **BALTIK** projects (**B**uild an **A**pplication **L**inked to **T**r**I**o_U **K**ernel).

Before TRUST: a Modular Software Named Trio_U

TRUST was born from the cutting in two pieces of **Trio_U** software. **Trio_U** was a software brick based on the **Kernel** brick (which contains the equations, space discretizations, numerical schemes, parallelism...) and used by other CEA applications (see Figure 1).

In 2015, **Trio_U** was divided in two parts: **TRUST** and **TrioCFD**.

- **TRUST** is a new platform, its name means: **TR**io_**U** **S**oftware for **T**hermohydraulics.
- **TrioCFD** is an open source BALTIK project based on **TRUST**.

Here are some other selected BALTIKS based on the TRUST platform (see Figure 2).

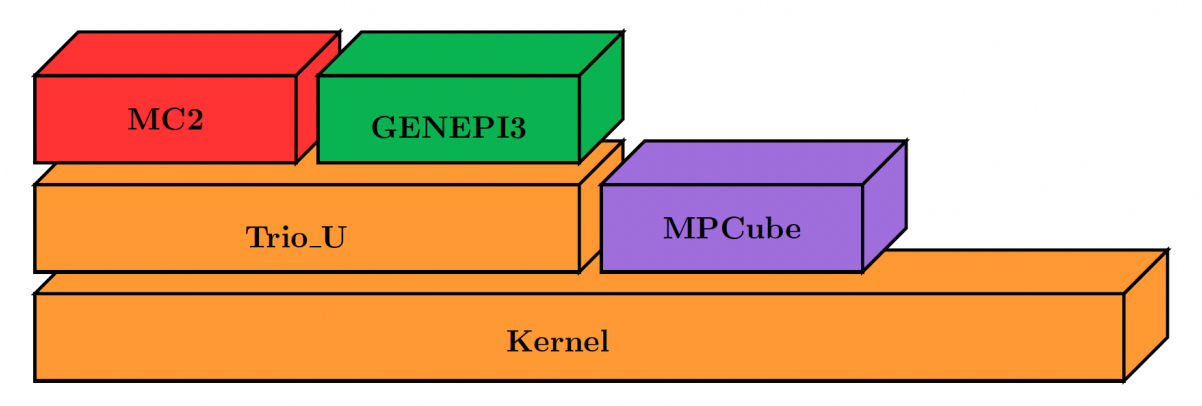

Fig. 1: Figure 1: Trio_U brick software

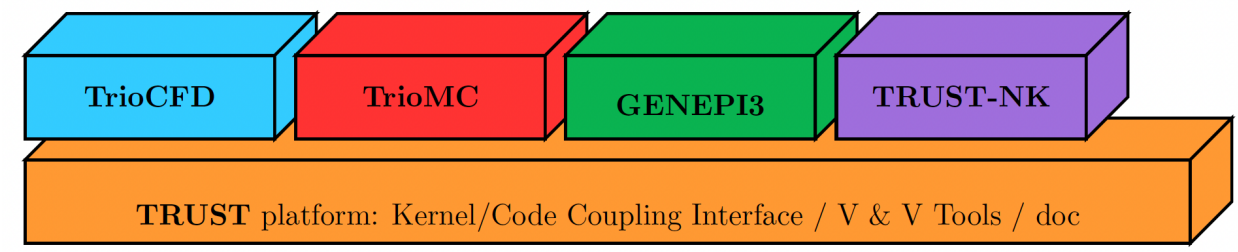

Fig. 2: Figure 2: Selected BALTIKS based on the TRUST platform.

Short History

TRUST is developed at the Laboratory of High Performance Computing and Numerical Analysis (LCAN) of the Software Engineering and Simulation Service (SGLS) in the Department of System and Structure Modeling (DM2S).

The project starts in 1994 and improved versions were built ever since:

- **1994 :** Start of the project Trio_U
- **1997 :** v1.0 - Finite Difference Volume (VDF) method only
- **1998 :** v1.1 - Finite Element Volume (VEF) method only
- **2000 :** v1.2 - Parallel MPI version
- **2001 :** v1.3 - Radiation model (TrioCFD now)
- **2002 :** v1.4 - LES turbulence models (TrioCFD now)
- **2006 :** v1.5 - VDF/VEF Front Tracking method (TrioCFD now)
- **2009 :** v1.6 - Data structure revamped
- **2015 :** v1.7 - Separation TRUST & TrioCFD + switch to open source
- **2019 :** v1.8 - New polyheadral discretization (PolyMAC)
- **2021 :** v1.8.4 - Multiphase problem + Weakly Compressible model
- **2022 ... :** Modern C++ code (templates, CRTP, ...), support GPU (NVidia/AMD), remove MACROS, ...

Data File

To launch a calculation with **TRUST**, you need to write a "data file" which is an input file for **TRUST** and will contain all the information about your simulation. Data files are written following some rules as shown below. But their language is not a programming language, users can't make loops or switch...

Note that:

- lines between `# ... #` and `/* ... */` are comments,
- **TRUST** keywords can be highlighted with your file editor via the command line:

```
trust -config gedit|vim|emacs
```

- braces `{ }` are elements that **TRUST** reads and interprets, so don't forget them and *put space before and after them*,
- elements between bracket `[ ]` are optional.

Data File Example: Base Blocks

Here is the template of a basic sequential data file:

```
# Dimension 2D or 3D #
Dimension 2
```

```
# Problem definition #
Pb_hydraulique my_problem
```

```
# Domain definition #
Domaine my_domain
```

```
# Mesh #
# BEGIN MESH #
Read_file my_mesh.geo ;
# END MESH #
```

```
# For parallel calculation only! #
# For the first run: partitioning step #
# Partition my_domain
{
   Partition_tool partitioner_name { option1 option2 ... }
   Larg_joint 2
   zones_name DOM
   ...
}
End #
```

```
# For parallel calculation only! #
# For the second run: read of the sub-domains #
# Scatter DOM .Zones my_domain #
```

```
# Discretization on hexa or tetra mesh #
VDF my_discretization
```

```
# Time scheme explicit or implicit #
Scheme_euler_explicit my_scheme
Read my_scheme
{
   # Initial time #
   # Time step #
   # Output criteria #
   # Stop Criteria #
}
```

```
# Association between the different objects #
Associate my_problem my_domain
Associate my_problem my_scheme
```

```
# Discretization of the problem #
Discretize my_problem my_discretization
```

```
# New domains for post-treatment #
# By default each boundary condition of the domain is already extracted #
# with names such as "my_dom"_boundaries_"my_BC" #
Domaine plane
extraire_surface
{
   domaine plane
   probleme my_probleme
   condition_elements (x>0.5)
   condition_faces (1)
}
```

```
# Problem description #
Read my_problem
{
```

```
{
   # hydraulic problem #
   Navier_Stokes_standard
   {
      # Physical characteristics of medium #
      Fluide_Incompressible
      {
      ...
      # Gravity vector definition #
      gravity Uniform_field 2 0 -9.81
      }

      # Choice of the pressure matrix solver #
      Solveur_Pression solver { ... }
```

(continues on next page)

(continued from previous page)

```
      # Diffusion operator #
      Diffusion { ... }

      # Convection operator #
      Convection { ... }

      # Sources #
      Sources { ... }

      # Initial conditions #
      Initial_conditions { ... }

      # Boundary conditions #
      Boundary_conditions { ... }
   }
```

```
# Post_processing description #
# To know domains that can be treated directly, search in .err #
# output file: "Creating a surface domain named" #

# To know fields that can be treated directly, search in .err #
# output file: "Reading of fields to be postprocessed" #

Post_processing
{
   # Definition of new fields #
   Definition_Champs { ... }

   # Probes #
   Probes { ... }

   # Fields #
   # format default value: lml #
   # select 'lata' for VisIt tool or 'MED' for Salomé #
   format lata
   fields dt_post 1. { ... }

   # Statistical fields #
   Statistiques dt_post 1. { ... }
}
```

```
# Saving and restarting process #
[sauvegarde binaire datafile .sauv]
[resume_last_time binaire datafile .sauv]
```

```
# End of the problem description block #
}
```

```
# The problem is solved with #
Solve my_problem
```

```
# Not necessary keyword to finish #
End
```

Basic Rules

There is no line concept in **TRUST**.

Data files uses `blocks`. They may be defined using the braces:

```
{
   a block
}
```

Objects Notion

Objects are created in the data set as follows:

```
[ export ] Type identificateur
```

- **export**: if this keyword is included, *identificateur* (identifier) will have a global range, if not, its range will be applied to the block only (the associated object will be destroyed on exiting the block).
- **Type**: must be a type of object recognised by **TRUST**, correspond to the C++ classes. The list of recognised types is given in the file hierarchie.dump.
- **identificateur**: the identifier of the object type *Type* created, correspond to an instancy of the C++ class *Type*. **TRUST** exits in error if the identifier has already been used.

There are several object types. Physical objects, for example:

- A **Fluide_incompressible** (incompressible_Fluid) object. This type of object is defined by its physical characteristics (its dynamic viscosity μ (keyword **mu**), its density ρ (keyword **rho**), etc...).
- A **Domaine**.

More abstract object types also exist:

- A **VDF**, **VEFPreP1B**, **PolyMAC_P0P1NC** or **PolyMAC_P0** according to the discretization type.
- A **Scheme_euler_explicit** to indicate the time scheme type.
- A **Solveur_pression** to denote the pressure system solver type.
- A **Uniform_field** to define, for example, the gravity field.

Interpretor Notion

Interprete (interpretor) type objects are then used to handle the created objects with the following syntax:

```
Type_interprete argument
```

- **Type_interprete**: any type derived from the **Interprete** (Interpretor) type recognised by **TRUST**.
- **argument**: an argument may comprise one or several object identifiers and/or one or several data blocks.

Interpretors allow some operations to be carried out on objects.

Currently available general interpretors include **Read**, **Read_file**, **Ecrire** (Write), **Ecrire_fichier** (Write_file), **Associate**.

Example

A data set to write Ok on screen:

```
Nom a_name        # Creation of an object type. Name identifier a_name #
Read a_name Ok    # Allocates the string "Ok" to a_name #
Ecrire a_name     # Write a_name on screen #
```

Important Remarks

1. To insert *comments* in the data set, use # .. # (or /* ... */), the character # must always be enclosed by blanks.
2. The comma separates items in a list (a comma must be enclosed with spaces or a new line).
3. Interpretor keywords are recognised indiscriminately whether they are written in lower and/or upper case.
4. **On the contrary, object names (identifiers) are recognised differently if they are written in upper or lower case.**
5. In the following description, items (keywords or values) enclosed by [and] are *optional*.

Running a Data File

To use **TRUST**, your shell must be "bash". So ensure you are in the right shell:

```
> echo $0
/bin/bash
```

To run your data file, you must initialize the TRUST environment using the following command:

```
> source $my_path_to_TRUST_installation/env_TRUST.sh
TRUST vX.Y.Z support : trust@cea.fr
Loading personal configuration /$path_to_my_home_directory/.perso_TRUST.env
```

Sequential Calculation

You can run your sequential calculation:

```
> cd $my_test_directory
> trust [-evol] my_data_file
```

where "trust" command call the "trust" script. You can have the list of its options with:

```
> trust -help
```

or

```
> trust -h
```

Here is a panel of available options:

```
Usage: trust [option] datafile [nb_cpus] [1>file.out] [2>file.err]
Where option may be:
-help|-h                        : List options.
-baltik [baltik_name]           : Instanciate an empty Baltik project.
-index                          : Access to the TRUST ressource index.
-doc                            : Access to the TRUST manual (Generic Guide).
-html                           : Access to the doxygen documentation.
-config nedit|vim|emacs|gedit : Configure nedit or vim or emacs or gedit with TRUST␣
↪keywords.
-edit                           : Edit datafile.
-xcheck                         : Check the datafile's keywords with xdata.
-xdata                          : Check and run the datafile's keywords with xdata.
-partition                      : Partition the mesh to prepare a parallel calculation␣
↪(Creation of the .Zones files).
-mesh                           : Visualize the mesh(es) contained in the data file.
-eclipse-trust                  : Generate Eclipse configuration files to import TRUST␣
↪sources.
-eclipse-baltik                 : Generate Eclipse configuration files to import BALTIK␣
↪sources (TRUST project should have been configured under Eclipse).
-probes                         : Monitor the TRUST calculation only.
-evol                           : Monitor the TRUST calculation (GUI).
-prm                            : Write a prm file (deprecated).
-jupyter                        : Create basic jupyter notebook file.
-clean                          : Clean the current directory from all the generated files␣
↪by TRUST.
-search keywords                : Know the list of test cases from the data bases which␣
↪contain keywords.
-copy                           : Copy the test case datafile from the TRUST database␣
↪under the present directory.
-check|-ctest all|testcase|list            : Check|ctest the non regression of all the␣
↪test cases or a single test case or a list of tests cases specified in a file.
-check|-ctest function|class|class::method : Check|ctest the non regression of a list of␣
↪tests cases covering a function, a class or a class method.
-gdb                            : Run under gdb debugger.
-valgrind                       : Run under valgrind.
-valgrind_strict                : Run under valgrind with no suppressions.
-callgrind                      : Run callgrind tool (profiling) from valgrind.
-massif                         : Run massif tool (heap profile) from valgrind.
-heaptrack                      : Run heaptrack (heap profile). Better than massif.
-advisor                        : Run advisor tool (vectorization).
-vtune                          : Run vtune tool (profiling).
-perf                           : Run perf tool (profiling).
-trace                          : Run traceanalyzer tool (MPI profiling).
-create_sub_file                : Create a submission file only.
-prod                           : Create a submission file and submit the job on the main␣
↪production class with exclusive resource.
-bigmem                         : Create a submission file and submit the job on the big␣
↪memory production class.
-queue queue                    : Create a submission file with the specified queue and␣
```

(continues on next page)

(continued from previous page)

```
↪submit the job.
-c ncpus                      : Use ncpus CPUs allocated per task for a parallel␣
↪calculation.
datafile -help_trust          : Print options of TRUST_EXECUTABLE [CASE[.data]]␣
↪[options].
-convert_data datafile        : Convert a data file to the new 1.9.1 syntax (milieu,␣
↪interfaces, read_med and champ_fonc_med).
```

Parallel Calculation

To run a parallel calculation, you must do two runs:

- the first one, to partition and create your 'n' sub-domains (two methods: "By hand" method (see below) and "Assisted" method (see sections *The different blocks* & *Partitionning: "Assisted" method*).
- the second one, to read your 'n' sub-domains and run the calculation on 'n' processors.

We will explain here how to do such work:

- **Partitioning: "By hand" method**

  You have to make two data files:

  1. `BuildMeshes.data`
  2. `Calculation.data`

  The `BuildMeshes.data` file only contains the same information as the begining of the sequential data file and partitioning information. This file will create the sub-domains (cf .Zones files).

```
Dimension 2
Domaine my_domain

# BEGIN MESH #
Read_file my_mesh.geo ;
# END MESH #

# BEGIN PARTITION #
Partition my_domain
{
   Partition_tool partitioner_name { option1 option2 ... }
   Larg_joint 2
   zones_name DOM
   ...
}
End
# END PARTITION #
```

  Run the `BuildMeshes.data` with **TRUST**:

```
> trust BuildMeshes
```

  You may have obtained files named `DOM_000n*.Zones` which contains the 'n' sub-domains.

- **Read the sub-domains**

The `Calculation.data` file contains the domain definition, the block which will read the sub-domains and the problem definition

```
Dimension 2
Domaine my_domain

Pb_Hydraulique my_problem

# BEGIN SCATTER #
Scatter DOM .Zones my_domain
# END SCATTER #

VDF my_discretization

Scheme_euler_explicit my_scheme
Read my_scheme { ... }

Associate my_problem my_domain
Associate my_problem my_scheme
Discretize my_problem my_discretization

Read my_problem
{
   Fluide_Incompressible { ... }
   ...
}
Solve my_problem
End
```

Run the `Calculation.data` file with **TRUST**:

```
> trust Calculation procs_number
```

This will read your `DOM_000n*.Zones` files. You can see the documentation of the **scatter** keyword in Reference Manual which is available here.

For more information, have a look on the first exercise of the TRUST Tutorial; Flow around an Obstacle, Parallel calculation section !

Interactive Evolution

To learn how to use the "**-evol**" option, you can see the first exercise of the **TRUST** tutorial: Flow around an obstacle available on this link.

1.1.2 Data setting

We will now explain how to fill a data file. First you must specify some basic information like the dimension of your domain, its name, the problem type…

To define the problem dimension, we use the following keyword:

```
Dimension 2
```

or

```
Dimension 3
```

Problems

You have to define the problem type that you wish to solve.

```
Pb_type my_problem
```

You can find all the available **TRUST** problems via this link.

Here are some of the available **TRUST** problem types.

- for Incompressible flow: **Pb_[Thermo]Hydraulic[_Concentration]**
- for Quasi-Compressible flow: **Pb_Thermohydraulique_QC**
- for Weakly-Compressible flow: **Pb_Thermohydraulique_WC**
- for Multi-Phase flow: **Pb_Multiphase**
- for solid: **Pb_Conduction**

where:

- **hydraulique**: means that we will solve Navier-Stokes equations without energy equation.
- **Thermo**: means that we will solve Navier-Stokes equations with energy equation.
- **Concentration**: that we will solve multiple constituent transportation equations.
- **Conduction**: resolution of the heat equation.
- **QC**: Navier-Stokes equations with energy equation for quasi-compressible fluid under low Mach approach.
- **WC**: Navier-Stokes equations with energy equation for weakly-compressible fluid under low Mach approach.

Domain Definition

To define the domain, you must name it. This is done thanks to the following block:

```
Domaine my_domain
```

Then you must add your mesh to your simulation.

Mesh

Notice the presence of the tags

```
# BEGIN MESH #
...
# END MESH #
```

in the data file of section *Data File Example: Base Blocks*. This is useful for parallel calculation if well placed in datafile (see section *Parallel Calculation*).

Allowed meshes

TRUST allows all types of meshes if the appropriate spatial discretization is used. See the Discretizations section on the TRUST's website.

Import a mesh file

If your mesh was generated with an external tool like SALOME (open source software), ICEM (commercial software), Gmsh (open source software, included in **TRUST** package) or Cast3M (CEA software), then you must use one of the following keywords into your data file:

- **Read_MED** for a MED file from SALOME or Gmsh.
- **Read_File** for a binary mesh file from ICEM.
- for another format, see the TRUST Reference Manual.

If you want to learn how to build a mesh with SALOME or Gmsh and read it with **TRUST**, you can look at the exercises of the TRUST Tutorial; Exo Salome and Exo Gmsh.

You can have a look too at the Pre-Processing section of the TRUST's website.

Quickly create a mesh

Here is an example of a simple geometry (of non complex channel type) using the internal tool of **TRUST**:

```
Mailler my_domain
{
   # Define the domain with one cavity #
   # cavity 1m*2m with 5*22 cells #
   Pave box
   {
      Origine 0. 0.
      Longueurs 1 2

      # Cartesian grid #
      Nombre_de_Noeuds 6 23

      # Uniform mesh #
      Facteurs 1. 1.
   }
   {
```

(continues on next page)

(continued from previous page)

```
    # Definition and names of boundary conditions #
    bord Inlet  X = 0.   0. <= Y <= 2.
    bord Outlet X = 1.   0. <= Y <= 2.
    bord Upper  Y = 2.   0. <= X <= 1.
    bord Lower  Y = 0.   0. <= X <= 1.
  }
}
```

To use this mesh in your data file, you just have to add the previous block in your data file or save it in a file named for example `my_mesh.geo` and add the line:

```
Read_file my_mesh.geo ;
```

Note: Do not forget the semicolon at the end of the line!

Transform mesh within the data file

You can also make transformations on your mesh after the **"Mailler"** or **"Read_"** command, using the following keywords:

- **Trianguler** to triangulate your 2D cells and create an unstructured mesh (doc here).
- **Tetraedriser** to tetrahedralise 3D cells and create an unstructured mesh (doc here).
- **Raffiner_anisotrope** or **Raffiner_isotrope** to triangulate/tetrahedralise elements of an untructured mesh (doc here).
- **ExtrudeBord** to generate an extruded mesh from a boundary of a tetrahedral or an hexahedral mesh (doc here).

  Note: ExtrudeBord in VEF generates 3 or 14 tetrahedra from extruded prisms.

- **RegroupeBord** to build a new boundary with several boundaries of the domain (doc here).
- **Transformer** to transform the coordinates of the geometry (doc here).

For other commands, see the section `interprete` of the TRUST Reference Manual available here.

Note: All theses keywords work on all mesh file formats (i.e. also for ***.geo** or ***.bin** or ***.med** files).

Test your mesh

The keyword **Discretiser_domaine** (doc here) is useful to discretize the domain (faces will be created) without defining a problem. Indeed, you can create a minimal data file, post-process your mesh in lata format (for example) and visualize it with VisIt.

Note: You must name all the boundaries to descretize!

Here is an example of this kind of data file (say `my_data_file.data` for example):

```
dimension 3
Domaine my_domain

Mailler my_domain
{
   Pave box
   {
      Origine 0. 0. 0.
      Longueurs 1 2 1
      Nombre_de_Noeuds 6 23 6
      Facteurs 1. 1. 1.
   }
   {
      bord Inlet X = 0. 0. <= Y <= 2. 0. <= Z <= 1.
      bord Outlet X = 1. 0. <= Y <= 2. 0. <= Z <= 1.
      bord Upper Y = 2. 0. <= X <= 1. 0. <= Z <= 1.
      bord Lower Y = 0. 0. <= X <= 1. 0. <= Z <= 1.
      bord Front Z = 0. 0. <= X <= 1. 0. <= Y <= 2.
      bord Back Z = 1. 0. <= X <= 1. 0. <= Y <= 2.
   }
}

discretiser_domaine my_domain
postraiter_domaine { domaine my_domain fichier file format lata }
End
```

To use it, launch in a bash terminal:

```
# Initialize TRUST env if not already done
> source $my_path_to_TRUST_installation/env_TRUST.sh

# Run you data file
> trust my_data_file
> visit -o file.lata &
```

To see how to use VisIt, look at the first TRUST Tutorial exercise; Flow around an Obstacle.

Spatial Discretization

You have to specify a discretization type to run a simulation. See the Discretizations section on the TRUST's website.

Time Schemes

Now you can choose your time scheme to solve your problem. For this you must specify the time scheme type wanted and give it a name. then you have to specify its parameters by filling the associated **Read** block.

```
Scheme_type my_time_scheme
Read my_time_scheme { ... }
```

Some available time schemes

The time schemes available in the TRUST platform are summarized on the TRUST's website in the Temporal schemes section.

Here are some available types of explicit schemes:

- **Scheme_Euler_explicit** (doc here).
- **Schema_Adams_Bashforth_order_2** (doc here).
- **Runge_Kutta_ordre_3** (doc here).

And also some available types of implicit schemes:

- **Scheme_Euler_implicit** (doc here).
- **Schema_Adams_Moulton_order_3** (doc here).

For other schemes, see doc here of the Reference Manual.

Note: You can treat implicitly the diffusion/viscous operators in a TRUST calculation. For that, you should activate the **diffusion_implicite** keyword in your explicit time scheme.

Calculation stopping condition

You must specify at least one stopping condition for you simulation. It can be:

- the final time: **tmax**
- the maximal allowed cpu time: **tcpumax**
- the number of time step: **nb_pas_dt_max**
- the convergency treshold: **seuil_statio**

Note: If the time step reaches the minimal time step **dt_min**, **TRUST** will stop the calculation.

If you want to stop properly your running calculation (i.e. with all saves), you may use the `my_data_file.stop` file.

When the simulation is running, you can see the **0** value in that file.

To stop it, put a **1** instead of the **0**, save the file and at the next iteration the calculation will stop properly.

When you don't change anything in that file, at the end of the calculation, you can see that it is writen **Finished correctly**.

Medium/Type of Fluide

To specify the medium or fluid, you must add the following block.

```
Fluid_type { ... }
```

Fluid_type can be one of the following:

- **Fluide_incompressible** (doc here).
- **Fluide_Quasi_compressible** (doc here).
- **Fluide_Weakly_Compressible** (doc here).

- **Solide** (doc here).
- **Constituant** (doc here).
- **Milieu_Composite** (for Multi-Phase problems)

For other types and more information see the TRUST Reference Manual.

Note: Since TRUST v1.9.1, the medium should be read in the begining of the problem definition (before equations).

If you want to solve a coupled problem, each medium should be read in the corresponding problem.

Add Gravity

If needed, you can add a gravity term to your simulation. This is done by adding a uniform field, in the medium block since V1.9.1.

For example in 2D:

```
Gravity Uniform_field 2 0 -9.81
```

Objects association and discretization

Association

Until now, we have created some objects, now we must associate them together. For this, we must use the **Associate** interpretor (doc here):

```
# Association between the different objects #
Associate my_problem my_domain
Associate my_problem my_time_scheme
```

Discretization

Then you must discretize your domain using the **Discretize** interpretor (doc here):

```
Discretize my_problem my_discretization
```

The problem *my_problem* is discretized according to the *my_discretization* discretization.

IMPORTANT:

A number of objects must be already associated (a domain, time scheme, ...) prior to invoking the **Discretize** keyword.

Note: When the discretization step succeeds, the mesh is validated by the code.

1.1.3 Problem definition

Set of Equations

Depending on your choosed problem type, you will have a different set of equations.

Here is a summary of some selected problems. For documentation and for complete problem sets, see the TRUST Reference Manual.

Incompressible problems

TRUST solves Navier-Stokes equations with/without the heat equation for an incompressible fluid:

$$\begin{cases} \nabla \cdot \vec{u} = 0 \\ \dfrac{\partial \vec{u}}{\partial t} + \nabla \cdot (\vec{u} \otimes \vec{u}) = \nabla \cdot (\nu \nabla \vec{u}) - \nabla P^* \\ \dfrac{\partial T}{\partial t} + \vec{u} \nabla T = \nabla \cdot (\alpha \nabla T) + \dfrac{Q}{\rho C_p} \end{cases}$$

where: $P^* = \dfrac{P}{\rho} + gz$, Q is the heat source term, and:

- ρ: density,
- μ: dynamic viscosity,
- $\nu = \dfrac{\mu}{\rho}$: kinematic viscosity,
- $\vec{g} = gz$: gravity vector in cartesian coordinates,
- $\alpha = \dfrac{\lambda}{\rho C_p}$: thermal diffusivity.
- C_p: specific heat capacity at constant pressure,
- λ: thermal conductivity,

Note: Red terms are convective terms and blue terms are diffusive terms.

In your data file, you will have:

```
Pb_Thermohydraulique_Concentration my_problem
...
Read my_problem
{
   # Define medium and its properties + gravity if any #
   Fluide_incompressible { ... }

   # Navier Stokes equations #
   Navier_Stokes_Standard
   {
      Solveur_Pression my_solver { ... }
      Diffusion { ... }
      Convection { ... }
      Initial_conditions { ... }
      Boundary_conditions { ... }
```

(continues on next page)

(continued from previous page)

```
      Sources { ... }
      ...
   }

   # Energy equation #
   Convection_Diffusion_Temperature
   {
      Diffusion { ... }
      Convection { ... }
      Initial_conditions { ... }
      Boundary_conditions { ... }
      Sources { ... }
      ...
   }

   # Constituent transportation equations #
   Convection_Diffusion_Concentration
   {
      Diffusion { ... }
      Convection { ... }
      Initial_conditions { ... }
      Boundary_conditions { ... }
      Sources { ... }
      ...
   }
}
```

Quasi-Compressible problem

TRUST solves Navier-Stokes equations with/without heat equation for quasi-compressible fluid:

$$\begin{cases} \dfrac{\partial \rho}{\partial t} + \nabla \cdot (\rho \vec{u}) = 0 \\ \dfrac{\partial \rho \vec{u}}{\partial t} + \nabla \cdot (\rho \vec{u} \vec{u}) = \nabla \cdot (\mu \nabla \vec{u}) - \nabla P - \rho \vec{g} \\ \rho C_p \left(\dfrac{\partial T}{\partial t} + \vec{u} \nabla T \right) = \nabla \cdot (\lambda \nabla T) + \dfrac{dP_0}{dt} + Q \end{cases}$$

where: $P_0 = \rho RT$, Q is a heat source term, and:

- ρ: density,
- μ: dynamic viscosity,
- $\vec{g} = gz$: gravity vector in cartesian coordinates,
- C_p: specific heat capacity at constant pressure,
- λ: thermal conductivity.

Note: Red terms are convective terms and blue terms are diffusive terms.

In your data file, you will have:

```
Pb_Thermohydraulique_QC my_problem
...
Read my_problem
{
   # Define medium and its properties + gravity if any #
   Fluide_Quasi_compressible { ... }

   # Navier Stokes equations for quasi-compressible fluid under low Mach numbers #
   Navier_Stokes_Turbulent_QC
   {
      Solveur_Pression my_solver { ... }
      Diffusion { ... }
      Convection { ... }
      Initial_conditions { ... }
      Boundary_conditions { ... }
      Sources { ... }
      ...
   }

   # Energy equation for quasi-compressible fluid under low Mach numbers #
   Convection_Diffusion_Chaleur_QC
   {
      Diffusion { ... }
      Convection { ... }
      Initial_conditions { ... }
      Boundary_conditions { ... }
      Sources { ... }
      ...
   }
}
```

Weakly-Compressible problem

TRUST solves Navier-Stokes equations with/without heat equation for weakly-compressible fluid:

$$\begin{cases} \dfrac{\partial \rho}{\partial t} + \nabla \cdot (\rho \vec{u}) = 0 \\ \dfrac{\partial \rho \vec{u}}{\partial t} + \nabla \cdot (\rho \vec{u} \vec{u}) = \nabla \cdot (\mu \nabla \vec{u}) - \nabla P - \rho \vec{g} \\ \rho C_p \left(\dfrac{\partial T}{\partial t} + \vec{u} \nabla T \right) = \nabla \cdot (\lambda \nabla T) + \dfrac{dP_{tot}}{dt} + Q \end{cases}$$

where: $P_{tot} = \rho RT$, Q is a heat source term, and:

- ρ: density,
- μ: dynamic viscosity,
- $\vec{g} = gz$: gravity vector in cartesian coordinates,
- C_p: specific heat capacity at constant pressure,
- λ: thermal conductivity.

Note: Red terms are convective terms and blue terms are diffusive terms.

In your data file, you will have:

```
Pb_Thermohydraulique_WC my_problem
...
Read my_problem
{
   # Define medium and its properties + gravity if any #
   Fluide_Weakly_compressible { ... }

   # Navier Stokes equations for weakly-compressible fluid under low Mach numbers #
   Navier_Stokes_Turbulent_WC
   {
      Solveur_Pression my_solver { ... }
      Diffusion { ... }
      Convection { ... }
      Initial_conditions { ... }
      Boundary_conditions { ... }
      Sources { ... }
      ...
   }

   # Energy equation for weakly-compressible fluid under low Mach numbers #
   Convection_Diffusion_Chaleur_WC
   {
      Diffusion { ... }
      Convection { ... }
      Initial_conditions { ... }
      Boundary_conditions { ... }
      Sources { ... }
      ...
   }
}
```

Conduction problem

For this kind of problems, **TRUST** solves the heat equation:

$$\rho C_p \frac{\partial T}{\partial t} = \nabla \cdot (\lambda \nabla T) + Q$$

where:

- ρ: density,
- C_p: specific heat capacity at constant pressure,
- λ: thermal conductivity,
- Q is a heat source term.

Note: The term in blue is the diffusive term.

In your data file, you will have:

```
Pb_Conduction my_problem
...
Read my_problem
{
   # Define medium and its properties #
   Solide { ... }

   # Resolution of the heat equation #
   Conduction
   {
      Diffusion { ... }
      Convection { ... }
      Initial_conditions { ... }
      Boundary_conditions { ... }
      Sources { ... }
      ...
   }
}
```

Coupled problems

With **TRUST**, we can couple problems. We will explain here the method for two problems but you can couple as many problems as you want.

To couple two problems, we define two problems *my_problem_1* and *my_problem_2* each one associated to a separate domain *my_domain_1* and *my_domain_2*, and to a separate medium *my_medium_1* and *my_medium_2* (associated or not to the gravity).

```
Dimension 2
Pb_ThermoHydraulique my_problem_1
Pb_ThermoHydraulique my_problem_2

Domaine my_domain_1
Read_file my_mesh_1.geo ;

Domaine my_domain_2
Read_file my_mesh_2.geo ;

Associate my_problem_1 my_domain_1
Associate my_problem_2 my_domain_2
```

Then we define a coupled problem associated to a single time scheme like for example:

```
Probleme_Couple my_coupled_problem

VEFPreP1B my_discretization

Scheme_euler_explicit my_scheme
Read my_scheme { ... }

Associate my_coupled_problem my_problem_1
Associate my_coupled_problem my_problem_2
```

(continues on next page)

(continued from previous page)

```
Associate my_coupled_problem my_scheme
```

Then we discretize and solve everything:

```
Discretize my_coupled_problem my_discretization

Read my_problem_1
{
   Fluide_Incompressible { ... }
   ...
}

Read my_problem_2
{
   Fluide_Incompressible { ... }
   ...
}

Solve my_coupled_problem
End
```

You can see the documentation of this kind of problem in the TRUST Reference Manual.

Pressure Solvers

Then you may indicate the choice of pressure solver using the following syntax (see the Solvers section on the TRUST's website)

```
Solveur_pression my_solver { ... }
```

The *my_solver* may be:

- **GCP** (doc here).
- **PETSc PETSc_solver_name** (doc here).
- **Cholesky** (doc here).
- **Gmres** (doc here).
- **Gen** (doc here).

Reminder: in CFD, a separate solver is used to solve the pressure. For more details, you can have a look at the section **Time and space schemes** of the **TRUST**& **TrioCFD** user slides.

Convection

There is no default convective scheme so you must choose a specific scheme and specify as follows:

```
convection { convective_scheme }
```

Have a look at the Spatial Schemes section for a list of schemes available in the TRUST platform.

In short, you can use the following convective scheme, following the recommendations of the user training session (see section **Time and space schemes** of the **TRUST& TrioCFD** user slides and the section **Recommendations for schemes**) following your discretization type:

- **Amont** (doc here).
- **Muscl** (doc here).
- **EF_stab** (doc here).

Note: There is no default convective scheme and if you don't want convection in your problem, you may use:

```
convection { negligeable }
```

Diffusion

The diffusion term is more or less a Laplacien operator and is thus always discretized by a second order centered difference scheme. So you just need to do this:

```
diffusion { }
```

Note: If you don't want diffusion in your problem, you may use:

```
diffusion { negligeable }
```

Initial Conditions

For each equation, you **must** set initial conditions:

```
initial_conditions { ... }
```

See the TRUST Reference Manual to see the syntax of each available initial condition. Here are the most used initial conditions:

- **Vitesse** field_type *bloc_lecture_champ*
- **Temperature** field_type *bloc_lecture_champ*

We list here some "field_type":

- **Uniform_Field** for a uniform field (doc here).
- **Champ_Fonc_Med** to read a data field in a MED-format file .med at a specified time (doc here).
- **Champ_Fonc_txyz** for a field which depends on time and space (doc here).

- **Champ_Fonc_Fonction_txyz** for a field which is a function of another field and time and/or space coordinates (doc here).
- **Champ_Fonc_Reprise** to read a data field in a saved file (.xyz or .sauv) at a specified time (doc here).

Boundary Conditions

Then you may specify your boundary conditions like:

```
boundary_conditions { ... }
```

It is important to specify here that **TRUST will not accept any boundary conditions by default.**

You can find help for boundary conditions in the Boundary Conditions section on the TRUST's website.

Here is a list of the most used boundary conditions:

- Bord **Frontiere_ouverte_vitesse_imposee** boundary_field_type *bloc_lecture_champ* (doc here).
- Bord **Frontiere_ouverte_pression_imposee** boundary_field_type *bloc_lecture_champ* (doc here).
- Bord **Paroi_fixe** (doc here).
- Bord **Symetrie** (doc here).
- Bord **Periodique** (doc here).
- Bord **Frontiere_ouverte_temperature_imposee** boundary_field_type *bloc_lecture_champ* (doc here).
- Bord **Frontiere_ouverte T_ext** boundary_field_type *bloc_lecture_champ* (doc here).
- Bord **Paroi_adiabatique** (doc here).
- Bord **Paroi_flux_impose** boundary_field_type *bloc_lecture_champ* (doc here).

To choose your *boundary_field_type* parameters, refer to the TRUST Reference Manual.

Source Terms

To introduce a source term into an equation, add the following line into the block defining the equation. The list of source keyword is described below.

```
Sources { source_keyword }
```

To introduce several source terms into the same equation, the blocks corresponding to the various terms need to be separated by a comma:

```
Sources { source_keyword1 , source_keyword2 , ... }
```

Here are some available source terms. For a complete list, refer to the TRUST Reference Manual.

- **Perte_Charge_Reguliere** type_perte_charge bloc_definition_pertes_charges (doc here).
- **Perte_Charge_Singuliere KX | KY | KZ** coefficient_value { ... } (doc here).
- **Canal_perio** { ... } (doc here).
- **Boussinesq_temperature** { ... } (doc here).

> **Note:** Defined as $\rho(T) = \rho(T_0)(1 - \beta_{th}(T - T_0))$

- **Boussinesq_concentration** { ... } (doc here).
- **Puissance_thermique** field_type bloc_lecture_champ (doc here).

Post-Processings

Before post-processing fields, during a run, **TRUST** creates several files which contain information about the calculation, the convergence, fluxes, balances... See section *Output Files* for more information.

Several keywords can be used to create a post-processing block, into a problem. First, you can create a single post-processing task (**Post_processing** keyword). Generally, in this block, results will be printed with a specified format at a specified time period.

```
Post_processing
{
   Postraitement_definition
   ...
}
```

But you can also create a list of post-processings with **Post_processings** keyword (named with Post_name1, Post_name2, etc...), in order to print results into several formats or with different time periods, or into different results files:

```
Post_processings
{
   Post_name1 { Postraitement_definition }
   Post_name2 { Postraitement_definition }
   ...
}
```

Have a look at the Post-Processing section on the TRUST's website.

Field names

- **Existing & predefined fields**

  You can post-process predefined fields and already existing fields. Here is a list of post-processable fields, but it is not the only ones.

Physical values	Keyword for field_name	Unit
Velocity	Vitesse or Velocity	m.s1
Velocity residual	Vitesse_residu	m.s2
Kinetic energy per elements	Energie_cinetique_elem	kg.m1.s2
Total kinetic energy	Energie_cinetique_totale	kg.m1.s2
Vorticity	Vorticite	s1
Pressure in incompressible flow (P/ + gz)	Pression	Pa.m3.kg1
Pressure in incompressible flow (P+gz)	Pression_pa or Pressure	Pa
Pressure in compressible flow	Pression	Pa
Hydrostatic pressure (gz)	Pression_hydrostatique	Pa

continues on next page

Table 1 – continued from previous page

Totale pressure	Pression_tot	Pa
Pressure gradient	Gradient_pression	m.s2
Velocity gradient	gradient_vitesse	s1
Temperature	Temperature	C or K
Temperature residual	Temperature_residu	C.s1 or K.s1
Temperature variance	Variance_Temperature	K2
Temperature dissipation rate	Taux_Dissipation_Temperature	K2.s1
Temperature gradient	Gradient_temperature	K.m1
Heat exchange coefficient	H_echange_Tref	W.m2.K1
Turbulent viscosity	Viscosite_turbulente	m2.s1
Turbulent dynamic viscosity	Viscosite_dynamique_turbulente	kg.m.s1
Turbulent kinetic	Energy	K m2.s2
Turbulent dissipation rate	Eps	m3.s1
Constituent concentration	Concentration	–
Constituent concentration residual	Concentration_residu	–
Component velocity along X	VitesseX	m.s1
Component velocity along Y	VitesseY	m.s1
Component velocity along Z	VitesseZ	m.s1
Mass balance on each cell	Divergence_U	m3.s1
Irradiancy	Irradiance	W.m2
Q-criteria	Critere_Q	s1
Distance to the wall Y +	Y_plus	–
Friction velocity	U_star	m.s1
Void fraction	Alpha	–
Cell volumes	Volume_maille	m3
Source term in non Galinean referential	Acceleration_terme_source	m.s2
Stability time steps	Pas_de_temps	s
Volumetric porosity	Porosite_volumique	–
Distance to the wall	Distance_Paroi	m
Volumic thermal power	Puissance_volumique	W.m3
Local shear strain rate	Taux_cisaillement	s1
Cell Courant number (VDF only)	Courant_maille	–
Cell Reynolds number (VDF only)	Reynolds_maille	–
Viscous force	Viscous_force	kg.m2.s1
Pressure force	Pressure_force	kg.m2.s1

continues on next page

Table 1 – continued from previous page

Total force	Total_force	kg.m2.s1
Viscous force along X	Viscous_force_x	kg.m2.s1
Viscous force along Y	Viscous_force_y	kg.m2.s1
Viscous force along Z	Viscous_force_z	kg.m2.s1
Pressure force along X	Pressure_force_x	kg.m2.s1
Pressure force along Y	Pressure_force_y	kg.m2.s1
Pressure force along Z	Pressure_force_z	kg.m2.s1
Total force along X	Total_force_x	kg.m2.s1
Total force along Y	Total_force_y	kg.m2.s1
Total force along Z	Total_force_z	kg.m2.s1

Note: Physical properties (conductivity, diffusivity,...) can also be post-processed.

Note: The name of the fields and components available for post-processing is displayed in the error file after the following message: "Reading of fields to be postprocessed". Of course, this list depends of the problem being solved.

- **Creating new fields**

  The **Definition_champs** keyword is used to create new or more complex fields for advanced post-processing.

```
Definition_champs { field_name_post field_type { ... } }
```

  field_name_post is the name of the new created field and **field_type** is one of the following possible type:

  - **refChamp** (doc here).
  - **Reduction_0D** using for example the **min**, **max** or **somme** methods (doc here).
  - **Transformation** (doc here).

  Refer to the TRUST Reference Manual for more information.

  Note: You can combine several **field_type** keywords to create your field and then use your new fields to create other ones.

  Here is an example of new field named *max_temperature*:

```
Read my_problem
{
   ...
   Postraitement
   {
      Definition_champs
      {
         # Creation of a 0D field: maximal temperature of the domain #
         max_temperature Reduction_0D
         {
            methode max
```

(continues on next page)

(continued from previous page)

```
            source refChamp { Pb_champ my_problem temperature }
         }
      }

      Probes
      {
         # Print max(temperature) into the datafile_TMAX.son file #
         tmax max_temperature periode 0.01 point 1 0. 0.
      }

      Champs dt_post 1.0 { ... }
   }
}
```

You can find other examples in the **TRUST**& **TrioCFD** user slides in the section "Post processing description".

Post-processing blocks

There are three methods to post-process in **TRUST**: using probes, fields or making statistics.

- **Probes**

  Probes refer to sensors that allow a value or several points of the domain to be monitored over time. The probes are a set of points defined:

  - one by one: **Points** keyword

    or

  - by a set of points evenly
    - distributed over a straight segment: **Segment** keyword

      or

    - arranged according to a layout: **Plan** keyword

      or

    - arranged according to a parallelepiped **Volume** keyword.

  Here is an example of 2D **Probes** block:

```
Probes
{
   pressure_probe [loc] pressure Periode 0.5 Points 3 1. 0. 1. 1. 1. 2.
   velocity_probe [loc] velocity Periode 0.5 Segment 10 1. 0. 1. 4.
}
```

  where the use of *loc* option allow to specify the wanted location of the probes. The available values are **grav** for gravity center of the element, **nodes** for faces and **som** for vertices. There is not default location. If the point does not coincide with a alculation node, the value is extrapolated linearly according to neighbouring node values.

  For complete syntax, see the TRUST Reference Manual.

- **Fields**

  This keyword allows to post-process fields on the whole domain, specifying the name of the backup file, its format, the post-processing time step and the name (and location) of the post-processed fields.

Here is an example of **Fields** block:

```
Fichier results
Format lata
Fields dt_post 1.
{
   velocity [faces] [som] [elem]
   pressure [elem] [som]
   temperature [elem] [som]
}
```

where **faces** , **elem** and **som** are keywords allowed to specify the location of the field.

Note: When you don't specify the location of the field, the default value is **som** for values at the vertices. So fields are post-processed at the vertices of the mesh.

To visualize your post-processed fields, you can use open source softwares like:

VisIt (included in **TRUST** package) or SALOME.

For complete syntax, see the TRUST Reference Manual.

- **Statistics**

Using this keyword, you will compute statistics on your unknows. You must specify the begining and ending time for the statistics, the post-processing time step, the statistic method, the name (and ocation) of your post-processed field.

Here is an example of **Statistiques** block:

```
Statistiques dt_post 0.1
{
   t_deb 1. t_fin 5.
   moyenne velocity [faces] [elem] [som]
   ecart_type pressure [elem] [som]
   correlation pressure velocity [elem] [som]
}
```

This block will write at every **dt_post** the average of the velocity $\overline{V(t)}$:

$$\overline{V(t)} = \begin{cases} 0 & , \text{ for } t \leq t_{deb} \\ \frac{1}{t-t_{deb}} \int_{t_{deb}}^{t} V(t)dt & , \text{ for } t_{deb} < t \leq t_{fin} \\ \frac{1}{t_{fin}-t_{deb}} \int_{t_{deb}}^{t_{fin}} V(t)dt & , \text{ for } t > t_{fin} \end{cases}$$

the standard deviation of the pressure $\langle P(t) \rangle$:

$$\langle P(t) \rangle = \begin{cases} 0 & , \text{ for } t \leq t_{deb} \\ \frac{1}{t-t_{deb}} \sqrt{\int_{t_{deb}}^{t} \left[P(t) - \overline{P(t)} \right]^2 dt} & , \text{ for } t_{deb} < t \leq t_{fin} \\ \frac{1}{t_{fin}-t_{deb}} \sqrt{\int_{t_{deb}}^{t_{fin}} \left[P(t) - \overline{P(t)} \right]^2 dt} & , \text{ for } t > t_{fin} \end{cases}$$

and correlation between the pressure and the velocity $\langle P(t).V(t)\rangle$ like:

$$\langle P(t).V(t)\rangle = \begin{cases} 0 & , \text{ for } t \leq t_{deb} \\ \frac{1}{t-t_{deb}} \int_{t_{deb}}^{t} \left[P(t) - \overline{P(t)}\right] \cdot \left[V(t) - \overline{V(t)}\right] dt & , \text{ for } t_{deb} < t \leq t_{fin} \\ \frac{1}{t_{fin}-t_{deb}} \int_{t_{deb}}^{t_{fin}} \left[P(t) - \overline{P(t)}\right] \cdot \left[V(t) - \overline{V(t)}\right] dt & , \text{ for } t > t_{fin} \end{cases}$$

Remark: Statistical fields can be plotted with probes with the keyword "operator_field_name" like for example: Moyenne_Vitesse or Ecart_Type_Pression or Correlation_Vitesse_Vitesse. For that, it is mandatory to have the statistical calculation of this fields defined with the keyword **Statistiques**.

For complete syntax, see the TRUST Reference Manual.

Post-process location

You can use location keywords to specify where you want to post-process your fields in order to avoid interpolations on your post-processed fields.

For that, recall the variables localisation from the Discretizations section available the TRUST's website.

Note: If you are in P0+P1 discretization (default option) and you post-process the pressure field at the element (or at the vertices), you will have an **interpolation** because the field is computed at the element **and** at the vertices.

Note: Non-main variables (like the viscosity, conductivity, cp, density, y+, ...) are always located at the element gravity center.

1.1.4 End of the data file

Solve

Now that you have finished to specify all your computation parameters, you may add the **Solve** keyword at the end of your data file, in order to solve your problem.

You may also add the **End** keyword to specify the end of your data file.

```
Solve my_problem
End
```

You can see methods to run your data file in section *Running a Data File*.

Stop a Running Calculation

Your calculation will automatically stop if it has reached:

- the end of the calculation time.
- the maximal allowed cpu time.
- the maximal number of iterations.
- the threshold of convergence.

You may use the `my_data_file.stop` file, if you want to stop properly your running calculation (i.e. with all saves).

When the simulation is running, you can see the **0** value in that file. To stop it, put a **1** instead of the **0** and at the next iteration the calculation will stop properly.

When you don't change anything to that file, at the end of the calculation, you can see that it is writen **Finished correctly**.

Save

TRUST makes automatic backups during the calculation. The unknowns (velocity, temperature,...) are saved in:

- one **.xyz** file, happening:
- at the end of the calculation.
- but, user may disable it with the specific keyword **EcritureLectureSpecial 0** added just before the **Solve** keyword.
- one (or several in case of parallel calculation) **.sauv** files, happening:
- at the start of the calculation.
- at the end of the calculation.
- each 23 hours of CPU, to change it, uses **periode_sauvegarde_securite_en_heure** keyword (default value 23 hours).
- user may also specify a time physical period with **dt_sauv** keyword.
- periodically using **tcpumax** keyword for which calculation stops after the specified time (default value 10^{30}). Use it for calculation on CCRT/TGCC and CINES clusters for example.

Note: By default, the name for the **.sauv** files is **filename_problemname.sauv** for sequential calculation, **filename_problemname_000n.sauv** for parallel calculation (one per process). The format of theses files is binary and the files are completed during successive saves.

You can change the behaviour using the following keywords just before the **solve** instruction:

```
sauvegarde binaire|xyz filename .sauv|filename .xyz
```

with **xyz**: the **.xyz** file is written instead of the **.sauv** files.

Note: You can use **sauvegarde_simple** instead of **sauvegarde** where the .sauv or .xyz file is deleted before saves, in order to keep disk space:

```
sauvegarde_simple binaire|xyz filename .sauv|filename .xyz
```

For more information, see the TRUST Reference Manual.

Resume

To resume your calculation, you may:

- change your initial time, the new initial time will be the real final calculation time of the previous calculation (see the .err file).
- change your final calculation time to the new wanted value.
- add the following block just before the **Solve** keyword:

```
reprise binaire|xyz filename .sauv|filename .xyz
```

Note: Instead of **reprise** keyword, you can use **resume_last_time** where **tinit** is automatically set to the last time of saved files (but you may change **tmax**):

```
resume_last_time binaire|xyz filename .sauv|filename .xyz
```

You can resume your calculation:

- from .sauv file(s) (one file per process): you can only resume the calculation with the **same number of equations** on **the same number of processes**.
- or from a .xyz file: here you can resume your calculation by **changing the number of equations solved** and/or with a **different number of processes**.

For examples, see the TRUST tutorial.

Note: You can run a calculation with initial condition read into a save file (.xyz or .sauv) from a previous calculation using **Champ_Fonc_reprise** or read a into a MED file with **Champ_Fonc_MED**.

1.1.5 Post-processing

Output Files

After running, you will find different files in your directory. Here is a short explaination of what you will find in each type of file depending on its extension.

Even if you don't post-process anything, you will have output files which are listed here:

File	Contents
my_data_file.**dt_ev**	Time steps, facsec, equation residuals
my_data_file.**stop**	Stop file ('0', '1' or 'Finished correctly')
my_data_file.**log**	Journal logging
my_data_file.**TU**	CPU performances
*my_data_file*_**detail.TU**	Statistics of execution
my_da ta_file_problem_name.**sauv** or **.xyz**	Saving 2D/3D results for resume
or *specified_name*.**sauv** or **.xyz**	(binary files)

and the listing of boundary fluxes where:

- *my_data_file*_**Contrainte_visqueuse.out** correspond to the friction drag exerted by the fluid.

- *my_data_file*_**Convection_qdm.out** contains the momentum flow rate.
- *my_data_file*_**Debit.out** is the volumetric flow rate.
- *my_data_file*_**Force_pression.out** correspond to the pressure drag exerted by the fluid.

If you add post-processings in your data files, you will find:

File	Contents
my_data_file.**sons**	1D probes list
my_data_file_probe_name.**son**	1D results with probes
my_data_file_probe_name.**plan**	3D results with probes
my_data_file.**lml** *(default format)*	
my_data_file.**lata** *(with all *.lata.* files)*	
my_data_file.**med**	2D/3D results
or *specified_name*.**lml** or **.lata** or **.med**	

The sceen outputs are automatically redirected in *my_data_file*.**out** and *my_data_file*.**err** files if you run a parallel calculation or if you use the "**-evol**" option of the "trust" script.

Else you can redirect them with the following command:

```
# Source TRUST env if not already done
> source $my_path_to_TRUST_installation/env_TRUST.sh

# then
> trust my_data_file 1>file_out.out 2>file_err.err
```

In the .out file, you will find the listing of physical infos with mass balance and in the .err file, the listing of warnings, errors and domain infos.

Tools

To open your 3D results in **lata** format, you can use VisIt which is an open source software included in **TRUST** package. For that you may "source" **TRUST** environment and launch VisIt:

```
# Source TRUST env if not already done
> source $my_path_to_TRUST_installation/env_TRUST.sh

# then
> visit -o my_data_file.lata &
```

To learn how to use it, you can do the first exercise of the TRUST Tutorial; Flow around an Obstacle.

To open your 3D results in **med** format, you can also use VisIt, SALOME or Paraview.

Here are some actions that you can perform when your simulation is finished:

- To visualize the positions of your probes in function of the 2D/3D mesh, you can open your .son files at the same time of the .lata file in VisIt.
- If you need more probes, you can create them with VisIt (if you have post-processed the good fields) or with MEDCoupling.
- You can use the option **-evol** of the trust script, like:

```
trust -evol my_data_file
```

and access to the probes or open VisIt for 2D/3D visualizations via this tool.

1.1.6 Parallel Simulations

TRUST is a plateform which allows to make parallel calculation following some basic rules:

- **S**ingle **P**rogram, **M**ultiple **D**ata model: tasks are split up and run simultaneously on multiple processors with different input in order to obtain results faster.
- messages exchange by **M**essage **P**assing **I**nterface.
- from a PC to a massively parallel computer, with shared or distributed memory.

Basic Notions

To make a parallel calculation, you have to partition your domain. Each sub-domain will be treated by one processor. In order to have good performances, ensure you that:

- sub-domains have almost the same number of cells.
- joint lengths (boundaries between sub-domains) are minimal.

Performances

You have to choose a number of processors which is in agreement with the number of cells. So, you can evaluate your speed-up (sequential time/parallel time which must be as close as possible of the number of processors) or efficiency (=1/SpeedUp) to choose the right number of processors.

From our experience, for good performance with **TRUST**, each processor has to treat between 20000 and 30000 cells.

Partitioning

To run a parallel calculation, you must:

- make some modifications on your *my_data_file.data* file,
- do two runs:
  - the first one, to partitioning and create your 'n' sub-domains (two methods will by presented).
  - the second one, to read your 'n' sub-domains and run the calculation on 'n' processors.

The different blocks

Different blocks appear in the data file.

- **Modifications on the mesh block**

  First you may add the tags `# BEGIN MESH #` and `# END MESH #` before and after your mesh block, for example:

```
# BEGIN MESH #
Read_file my_mesh.geo ;
[Trianguler_h my_domain ]
# END MESH #
```

You can refer to section *Mesh* to have more information.

- **Adding a partitioning block**

  You may now add the partitioning block which contains the cutting instruction, after your mesh block:

```
# BEGIN PARTITION
Partition my_domain
{
   Partition_tool partitioner_name { option1 option2 ... }
   Larg_joint 2
   zones_name DOM
   ...
}
End
END PARTITION #
```

  Where *partitioner_name* is the name of the chosen partitioner, one of **METIS**, **Tranche**, **Sous_Zones**, **Partition** or **Fichier_Decoupage** (see section *TRUST available partitioning tools*).

  The **Larg_joint** keyword allows to specify the overlapping width value.

  Note the **End** before the last line. It will be useful for the cutting step.

  This block will make the partitioning of your domain into the specified number of sub-domains during the partitioning run.

- **Adding a block to read the sub-domains**

  At last, you will add a block which will be activated during the parallel calculation and will allow to read the sub-domains:

```
# BEGIN SCATTER
Scatter DOM .Zones my_domain
END SCATTER #
```

Partitionning: "Assisted" method

Here we will use the **trust -partition datafile** command line to make the partitioning step. We consider that you have correctly add the "#" in your *my_data_file.data* file with the partitioning block and cutting block.

Be careful with the hashtags "#", they are interpreted by the script!

To automatically perform the partitioning step and obtain the parallel data file, you have to run:

```
> trust -partition my_data_file [parts_number]
```

Note: Here parts_number is the number of sub-domains created but it is also the number of processors which will be used.

This command creates:

- a *SEQ_my_data_file.data* file which is a backup file of *my_data_file.data* the sequential data file,
- a *DEC_my_data_file.data* file which is the first data file to be run to make the partitioning.

  It is immediately run by the command line **trust -partition datafile** to create a partition, located in the *DOM_000n*.**Zones** files.

  Note: The code stops reading this file at the **End** keyword just before the **# END PARTITION #** block.

- a *PAR_my_data_file.data* file which is the data file for the parallel calculation. It reads the *DOM_000n*.**Zones** files through the instruction **"Scatter"**.

  Note that the meshing and cut of the mesh are commented here.

To see your sub-domains, you can run:

```
> trust -mesh PAR_my_data_file
```

For more information, you can do the exercise of the TRUST Tutorial.

TRUST available partitioning tools

In **TRUST**, you can make partitioning with:

- the external partitionning library METIS (open source).

  It is a general algorithm that will generate a partition of the domain

  ```
  Partition_tool Metis
  {
     nb_parts N
     [use_weights]
     [pmetis | kmetis]
     [nb_essais N]
  }
  ```

- internal **TRUST** partitioning tool **Tranche** which makes parts by cutting the domain following x, y and/or z directions.

  ```
  partition_tool Tranche
  {
     tranches nx ny [nz]
  }
  ```

Figure 3 is an example of what you can obtain by cutting a 1m x 1m square, divided in three parts using METIS and the same square divided in Figure 4 into three slices following the x direction with **Tranche**.

For more information, see the TRUST Reference Manual.

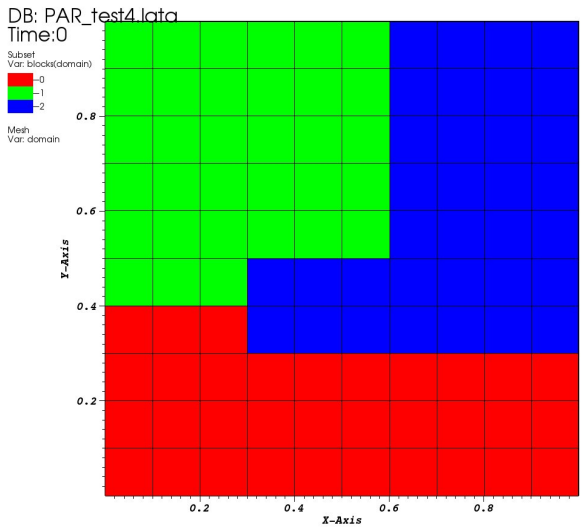

Fig. 3: Figure 3: METIS partition.

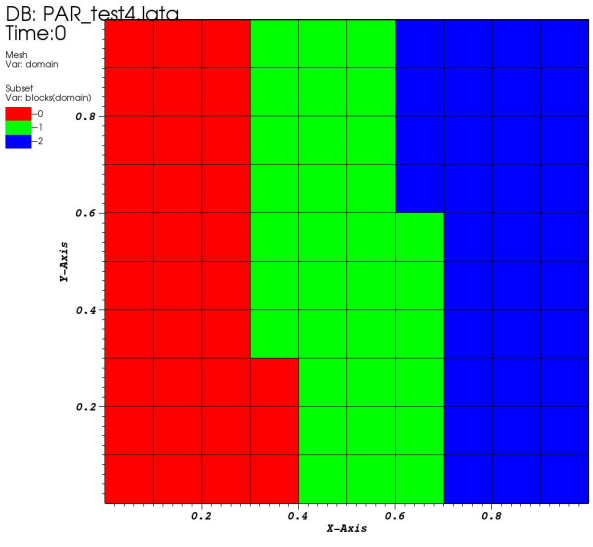

Fig. 4: Figure 4: TRANCH partition.

Overlapping width value

To make the partitioning, you will have to specify the *overlapping width value*. This value corresponds to the thickness of the virtual ghost zone (data known by one processor though not owned by it) i.e. the number of vertices or elements on the remote sub-domain known by the local sub-domain (see Figure 5).

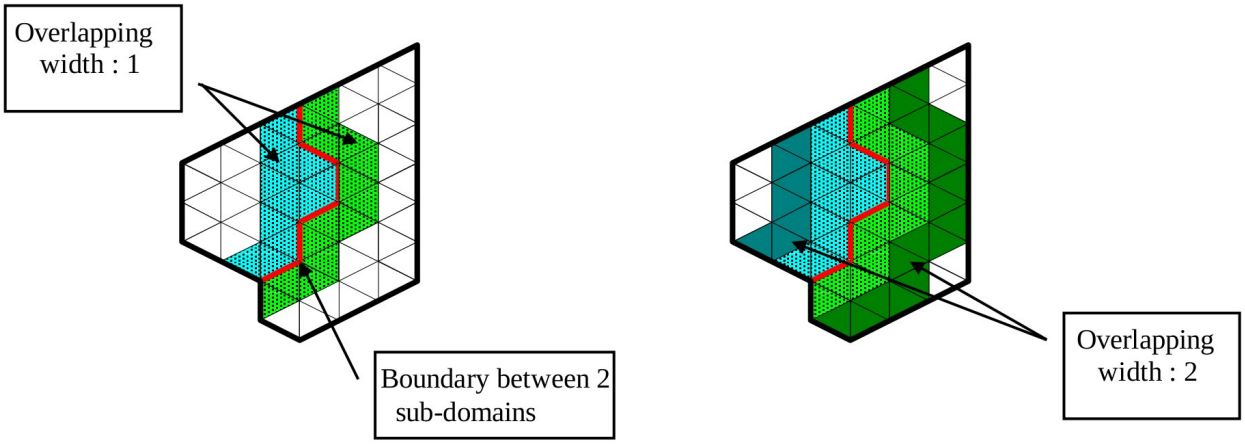

Fig. 5: Figure 5: Overlapping width.

This value depends on the space discretization scheme orders:

- 1 if 1st or 2nd order.
- 2 if 3rd or 4th order.

Note that in general, you will use "2"!

Running a Parallel Calculation

On a PC

To launch the calculation, you have to run the calculation by the usual command completed by the number of processors needed:

```
> trust my_parallel_data_file procs_number
```

and *procs_number* is the number of processors used. In fact it is the same as the number of sub-domains.

You can see the **TRUST**& **TrioCFD** user slides in the "Parallel calculation" section for more information. Also work the exercises in the TRUST Tutorial.

On a cluster

You must submit your job in a queue system. For this, you must have a submission file. **TRUST** can create a submission file for you **on clusters on which the support team has done installations**.

To create this file, run:

```
> trust -create_sub_file my_parallel_data_file
```

You obtain a file named **sub_file**, you can open it and verify/change values(for example the name of the job, the name of the exe, ...).

Then you must submit you calculation with:

```
> sbatch sub_file
```

or

```
> ccc_msub sub_file
```

following the queue system of the cluster.

You can see the **TRUST**& **TrioCFD** user slides in the "Parallel calculation" section for more information.

Visualization

To visualize your probes, you can use the CurvePlot tool, with the command line:

```
> trust -evol my_parallel_data_file
```

or use Gnuplot or any software which reads values in columns in a file.

There are three ways to visualize your parallel results with VisIt:

- HPCDrive or Nice DCV on CCRT/TGCC clusters: opens a deported graphic session on dedicated nodes with more memory (on TGCC cluster: HPCDrive),
- local mode: copy your results from the cluster to your local computer and open it with a local parallel version of VisIt with:

  ```
  > visit -np 4 &
  ```

You can have a look at the **TRUST**& **TrioCFD** user slides in the "Parallel calculation description" section.

Useful Information

Modify the mesh

If you want to modify your mesh, you have two possibilities:

- modify the *my_data_file.data* file and run:

  ```
  > trust -partition my_data_file [parts_number]
  ```

Be carefull it will erase the *SEQ_my_data_file.data*, *DEC_my_data_file.data* and *PAR_my_data_file.data* files and creates new ones.

Then it will run the new *DEC_my_data_file.data* file which gives your new *DOM_000n***.Zones** files.

- modify the meshing part of file *DEC_my_data_file.data* and run it with:

```
> trust DEC_my_data_file
```

Then run the parallel calculation normally, on the new *DOM_000n***.Zones** files.

```
> trust PAR_my_data_file procs_number
```

Modify calculation parameters

If you want to modify the calculation parameters, you can modify:

- the file *my_data_file.data* and run:

```
> trust -partition data_file_name [parts_number]
```

But it will erase the *SEQ_my_data_file.data*, *DEC_my_data_file.data* and *PAR_my_data_file.data* files and create new ones.

Then it will run the new *DEC_my_data_file.data* file which gives your new *DOM_000n***.Zones** files.

Note: In that case, you don't need to re-create the mesh so you can use the second point below:

- modify the *PAR_my_data_file.data* file *without* running **trust -partition datafile** command line.

Then run the *PAR_my_data_file.data* file with:

```
> trust PAR_my_data_file procs_number
```

Note: If after a certain time, you want to reopen an old case and understand want you did in it without any doubts, you may create two files by hands:

- one "BuildMeshes.data" file only for the mesh and the cut of the mesh.
- one "calculation.data" file for the parallel calculation.

You will run it like:

```
> trust BuildMeshes
> trust calculation processors_number
```

1.1.7 Important references

For details and practices, see:

- The **TRUST** Tutorial available here.
- The **TRUST** Reference Manual available here.
- The **TRUST** user & development slides.

Other references:

- The **TRUST** website available here.
- Methodology for incompressible single phase flow (Models_Equations_TRUST.pdf).
- Trio_U code validation data base & best practice guidelines (Best_Practice_TRUST.pdf).
- To access **TRUST** test case list and most documentation:

```
> trust -index
```

Note: Remember that you can attend to:

- a user training session (2 days)
- a developper training session (2 days)

provided by the support team. To request session, send an email to trust@cea.fr.

CHAPTER

TWO

TRUST KEYWORD REFERENCE MANUAL

This is the TRUST keyword reference manual, listing all the available TRUST keywords you can use in a dataset.

Do not forget that you can use the research bar located on the top right of your screen to quickly lookup a precise keyword.

For each keyword:

- its **synonyms** are given;
- its **parent class** is indicated;
- the **list of attributes** is provided. For each attribute in turn:
  - its name and synonyms are given
  - its type is given parenthesis
  - when the attribute is surrounded by square brackets, it is optional.

2.1 Table Of Contents

2.1.1 Keywords derived from bloc_comment

bloc_comment

Synonyms: /*

bloc of Comment in a data file.

Parameters are:

- **comm** (*type:* string) Text to be commented.

2.1.2 Keywords derived from champ_generique_base

champ_generique_base

not_set

---

champ_post_de_champs_post

Inherits from: *champ_generique_base*

not_set

Parameters are:

- **[source]** (*type:* *champ_generique_base*) the source field.
- **[nom_source]** (*type:* string) To name a source field with the nom_source keyword
- **[source_reference]** (*type:* string) not_set
- **[sources_reference]** (*type:* *list_nom_virgule*) not_set
- **[sources]** (*type:* *listchamp_generique*) sources { Champ_Post.... { ... } Champ_Post.. { ... }}

---

champ_post_operateur_base

Inherits from: *champ_generique_base*

not_set

Parameters are:

- **[source]** (*type:* *champ_generique_base*) the source field.
- **[nom_source]** (*type:* string) To name a source field with the nom_source keyword
- **[source_reference]** (*type:* string) not_set
- **[sources_reference]** (*type:* *list_nom_virgule*) not_set
- **[sources]** (*type:* *listchamp_generique*) sources { Champ_Post.... { ... } Champ_Post.. { ... }}

---

champ_post_operateur_eqn

Synonyms: operateur_eqn

Inherits from: *champ_generique_base*

Post-process equation operators/sources

Parameters are:

- **[numero_source]** (*type:* int) the source to be post-processed (its number). If you have only one source term, numero_source will correspond to 0 if you want to post-process that unique source

- **[numero_op]** (*type:* int) numero_op will be 0 (diffusive operator) or 1 (convective operator) or 2 (gradient operator) or 3 (divergence operator).
- **[numero_masse]** (*type:* int) numero_masse will be 0 for the mass equation operator in Pb_multiphase.
- **[sans_solveur_masse]** (*type:* flag) not_set
- **[compo]** (*type:* int) If you want to post-process only one component of a vector field, you can specify the number of the component after compo keyword. By default, it is set to -1 which means that all the components will be post-processed. This feature is not available in VDF disretization.
- **[source]** (*type: champ_generique_base*) the source field.
- **[nom_source]** (*type:* string) To name a source field with the nom_source keyword
- **[source_reference]** (*type:* string) not_set
- **[sources_reference]** (*type: list_nom_virgule*) not_set
- **[sources]** (*type: listchamp_generique*) sources { Champ_Post.... { ... } Champ_Post.. { ... }}

---

champ_post_statistiques_base

Inherits from: *champ_generique_base*

not_set

Parameters are:

- **t_deb** (*type:* double) Start of integration time
- **t_fin** (*type:* double) End of integration time
- **[source]** (*type: champ_generique_base*) the source field.
- **[nom_source]** (*type:* string) To name a source field with the nom_source keyword
- **[source_reference]** (*type:* string) not_set
- **[sources_reference]** (*type: list_nom_virgule*) not_set
- **[sources]** (*type: listchamp_generique*) sources { Champ_Post.... { ... } Champ_Post.. { ... }}

---

correlation

Synonyms: champ_post_statistiques_correlation

Inherits from: *champ_generique_base*

to calculate the correlation between the two fields.

Parameters are:

- **t_deb** (*type:* double) Start of integration time
- **t_fin** (*type:* double) End of integration time
- **[source]** (*type: champ_generique_base*) the source field.
- **[nom_source]** (*type:* string) To name a source field with the nom_source keyword

- **[source_reference]** (*type:* string) not_set
- **[sources_reference]** (*type: list_nom_virgule*) not_set
- **[sources]** (*type: listchamp_generique*) sources { Champ_Post.... { ... } Champ_Post.. { ... }}

---

divergence

Synonyms: champ_post_operateur_divergence

Inherits from: *champ_generique_base*

To calculate divergency of a given field.

Parameters are:

- **[source]** (*type: champ_generique_base*) the source field.
- **[nom_source]** (*type:* string) To name a source field with the nom_source keyword
- **[source_reference]** (*type:* string) not_set
- **[sources_reference]** (*type: list_nom_virgule*) not_set
- **[sources]** (*type: listchamp_generique*) sources { Champ_Post.... { ... } Champ_Post.. { ... }}

---

ecart_type

Synonyms: champ_post_statistiques_ecart_type

Inherits from: *champ_generique_base*

to calculate the standard deviation (statistic rms) of the field nom_champ.

Parameters are:

- **t_deb** (*type:* double) Start of integration time
- **t_fin** (*type:* double) End of integration time
- **[source]** (*type: champ_generique_base*) the source field.
- **[nom_source]** (*type:* string) To name a source field with the nom_source keyword
- **[source_reference]** (*type:* string) not_set
- **[sources_reference]** (*type: list_nom_virgule*) not_set
- **[sources]** (*type: listchamp_generique*) sources { Champ_Post.... { ... } Champ_Post.. { ... }}

---

extraction

Synonyms: champ_post_extraction

Inherits from: *champ_generique_base*

To create a surface field (values at the boundary) of a volume field

Parameters are:

- **domaine** (*type:* *domaine*) name of the volume field
- **nom_frontiere** (*type:* string) boundary name where the values of the volume field will be picked
- **[methode]** (*type:* string into ["trace", "champ_frontiere"]) name of the extraction method (trace by_default or champ_frontiere)
- **[source]** (*type:* *champ_generique_base*) the source field.
- **[nom_source]** (*type:* string) To name a source field with the nom_source keyword
- **[source_reference]** (*type:* string) not_set
- **[sources_reference]** (*type:* *list_nom_virgule*) not_set
- **[sources]** (*type:* *listchamp_generique*) sources { Champ_Post.... { ... } Champ_Post.. { ... }}

---

gradient

Synonyms: champ_post_operateur_gradient

Inherits from: *champ_generique_base*

To calculate gradient of a given field.

Parameters are:

- **[source]** (*type:* *champ_generique_base*) the source field.
- **[nom_source]** (*type:* string) To name a source field with the nom_source keyword
- **[source_reference]** (*type:* string) not_set
- **[sources_reference]** (*type:* *list_nom_virgule*) not_set
- **[sources]** (*type:* *listchamp_generique*) sources { Champ_Post.... { ... } Champ_Post.. { ... }}

---

interpolation

Synonyms: champ_post_interpolation

Inherits from: *champ_generique_base*

To create a field which is an interpolation of the field given by the keyword source.

Parameters are:

- **localisation** (*type:* string) type_loc indicate where is done the interpolation (elem for element or som for node).
- **[methode]** (*type:* string) The optional keyword methode is limited to calculer_champ_post for the moment.

- **[domaine]** (*type:* string) the domain name where the interpolation is done (by default, the calculation domain)
- **[optimisation_sous_maillage]** (*type:* string into ["default", "yes", "no",]) not_set
- **[source]** (*type: champ_generique_base*) the source field.
- **[nom_source]** (*type:* string) To name a source field with the nom_source keyword
- **[source_reference]** (*type:* string) not_set
- **[sources_reference]** (*type: list_nom_virgule*) not_set
- **[sources]** (*type: listchamp_generique*) sources { Champ_Post.... { ... } Champ_Post.. { ... }}

---

morceau_equation

Synonyms: champ_post_morceau_equation

Inherits from: *champ_generique_base*

To calculate a field related to a piece of equation. For the moment, the field which can be calculated is the stability time step of an operator equation. The problem name and the unknown of the equation should be given by Source refChamp { Pb_Champ problem_name unknown_field_of_equation }

Parameters are:

- **type** (*type:* string) can only be operateur for equation operators.
- **[numero]** (*type:* int) numero will be 0 (diffusive operator) or 1 (convective operator) or 2 (gradient operator) or 3 (divergence operator).
- **[unite]** (*type:* string) will specify the field unit
- **option** (*type:* string into ["stabilite", "flux_bords", "flux_surfacique_bords"]) option is stability for time steps or flux_bords for boundary fluxes or flux_surfacique_bords for boundary surfacic fluxes
- **[compo]** (*type:* int) compo will specify the number component of the boundary flux (for boundary fluxes, in this case compo permits to specify the number component of the boundary flux choosen).
- **[source]** (*type: champ_generique_base*) the source field.
- **[nom_source]** (*type:* string) To name a source field with the nom_source keyword
- **[source_reference]** (*type:* string) not_set
- **[sources_reference]** (*type: list_nom_virgule*) not_set
- **[sources]** (*type: listchamp_generique*) sources { Champ_Post.... { ... } Champ_Post.. { ... }}

---

moyenne

Synonyms: champ_post_statistiques_moyenne

Inherits from: *champ_generique_base*

to calculate the average of the field over time

Parameters are:

- **[moyenne_convergee]** (*type: field_base*) This option allows to read a converged time averaged field in a .xyz file in order to calculate, when resuming the calculation, the statistics fields (rms, correlation) which depend on this average. In that case, the time averaged field is not updated during the resume of calculation. In this case, the time averaged field must be fully converged to avoid errors when calculating high order statistics.
- **t_deb** (*type:* double) Start of integration time
- **t_fin** (*type:* double) End of integration time
- **[source]** (*type: champ_generique_base*) the source field.
- **[nom_source]** (*type:* string) To name a source field with the nom_source keyword
- **[source_reference]** (*type:* string) not_set
- **[sources_reference]** (*type: list_nom_virgule*) not_set
- **[sources]** (*type: listchamp_generique*) sources { Champ_Post.... { ... } Champ_Post.. { ... }}

---

predefini

Inherits from: *champ_generique_base*

This keyword is used to post process predefined postprocessing fields.

Parameters are:

- **pb_champ** (*type: deuxmots*) { Pb_champ nom_pb nom_champ } : nom_pb is the problem name and nom_champ is the selected field name. The available keywords for the field name are: energie_cinetique_totale, energie_cinetique_elem, viscosite_turbulente, viscous_force_x, viscous_force_y, viscous_force_z, pressure_force_x, pressure_force_y, pressure_force_z, total_force_x, total_force_y, total_force_z, viscous_force, pressure_force, total_force

---

reduction_0d

Synonyms: champ_post_reduction_0d

Inherits from: *champ_generique_base*

To calculate the min, max, sum, average, weighted sum, weighted average, weighted sum by porosity, weighted average by porosity, euclidian norm, normalized euclidian norm, L1 norm, L2 norm of a field.

Parameters are:

- **methode** (*type:* string into ["min", "max", "moyenne", "average", "moyenne_ponderee", "weighted_average", "somme", "sum", "somme_ponderee", "weighted_sum", "somme_ponderee_porosite", "weighted_sum_porosity", "euclidian_norm", "normalized_euclidian_norm", "l1_norm", "l2_norm", "valeur_a_gauche", "left_value"]) name of the reduction method: - min for the minimum value, - max for the maximum value, - average (or moyenne) for a mean, - weighted_average (or moyenne_ponderee) for a mean ponderated by integration volumes, e.g: cell volumes for temperature and pressure in VDF, volumes around faces for velocity and temperature in VEF, - sum (or somme) for the sum of all the values of the field, - weighted_sum (or somme_ponderee) for a weighted sum (integral), - weighted_average_porosity (or moyenne_ponderee_porosite) and weighted_sum_porosity (or somme_ponderee_porosite) for the mean and sum weighted by the volumes of the elements, only for ELEM localisation, - euclidian_norm for the euclidian norm, - normalized_euclidian_norm for the euclidian norm normalized, - L1_norm for norm L1, - L2_norm for norm L2

- **[source]** (*type: champ_generique_base*) the source field.
- **[nom_source]** (*type:* string) To name a source field with the nom_source keyword
- **[source_reference]** (*type:* string) not_set
- **[sources_reference]** (*type: list_nom_virgule*) not_set
- **[sources]** (*type: listchamp_generique*) sources { Champ_Post.... { ... } Champ_Post.. { ... }}

---

refchamp

Synonyms: champ_post_refchamp

Inherits from: *champ_generique_base*

Field of prolem

Parameters are:

- **pb_champ** (*type: deuxmots*) { Pb_champ nom_pb nom_champ } : nom_pb is the problem name and nom_champ is the selected field name.
- **[nom_source]** (*type:* string) The alias name for the field

---

tparoi_vef

Synonyms: champ_post_tparoi_vef

Inherits from: *champ_generique_base*

This keyword is used to post process (only for VEF discretization) the temperature field with a slight difference on boundaries with Neumann condition where law of the wall is applied on the temperature field. nom_pb is the problem name and field_name is the selected field name. A keyword (temperature_physique) is available to post process this field without using Definition_champs.

Parameters are:

- **[source]** (*type: champ_generique_base*) the source field.
- **[nom_source]** (*type:* string) To name a source field with the nom_source keyword
- **[source_reference]** (*type:* string) not_set
- **[sources_reference]** (*type: list_nom_virgule*) not_set
- **[sources]** (*type: listchamp_generique*) sources { Champ_Post.... { ... } Champ_Post.. { ... }}

---

transformation

Synonyms: champ_post_transformation

Inherits from: *champ_generique_base*

To create a field with a transformation using source fields and x, y, z, t. If you use in your datafile source refChamp { Pb_champ pb pression }, the field pression may be used in the expression with the name pression_natif_dom; this latter is the same as pression. If you specify nom_source in refChamp bloc, you should use the alias given to pressure field. This is avail for all equations unknowns in transformation.

Parameters are:

- **methode** (*type:* string into ["produit_scalaire", "norme", "vecteur", "formule", "composante"]) methode 0 methode norme : will calculate the norm of a vector given by a source field methode produit_scalaire : will calculate the dot product of two vectors given by two sources fields methode composante numero integer : will create a field by extracting the integer component of a field given by a source field methode formule expression 1 : will create a scalar field located to elements using expressions with x,y,z,t parameters and field names given by a source field or several sources fields. methode vecteur expression N f1(x,y,z,t) fN(x,y,z,t) : will create a vector field located to elements by defining its N components with N expressions with x,y,z,t parameters and field names given by a source field or several sources fields.
- **[unite]** (*type:* string) will specify the field unit
- **[expression]** (*type:* string list) expression 1 see methodes formule and vecteur
- **[numero]** (*type:* int) numero 1 see methode composante
- **[localisation]** (*type:* string) localisation 1 type_loc indicate where is done the interpolation (elem for element or som for node). The optional keyword methode is limited to calculer_champ_post for the moment
- **[source]** (*type: champ_generique_base*) the source field.
- **[nom_source]** (*type:* string) To name a source field with the nom_source keyword
- **[source_reference]** (*type:* string) not_set
- **[sources_reference]** (*type: list_nom_virgule*) not_set
- **[sources]** (*type: listchamp_generique*) sources { Champ_Post.... { ... } Champ_Post.. { ... }}

2.1.3 Keywords derived from champs_a_post

champs_a_post

Fields to be post-processed.

2.1.4 Keywords derived from chimie

chimie

Keyword to describe the chmical reactions

Parameters are:

- **reactions** (*type: reactions*) list of reactions
- **[modele_micro_melange]** (*type:* int) modele_micro_melange (0 by default)
- **[constante_modele_micro_melange]** (*type:* double) constante of modele (1 by default)
- **[espece_en_competition_micro_melange]** (*type:* string) especе in competition in reactions

---

2.1.5 Keywords derived from class_generic

amgx

Inherits from: *class_generic*

Solver via AmgX API

Parameters are:

- **solveur** (*type:* string) not_set
- **option_solveur** (*type: bloc_lecture*) not_set
- **[atol]** (*type:* double) Absolute threshold for convergence (same as seuil option)
- **[rtol]** (*type:* double) Relative threshold for convergence

---

cholesky

Inherits from: *class_generic*

Cholesky direct method.

Parameters are:

- **[impr]** (*type:* flag) Keyword which may be used to print the resolution time.
- **[quiet]** (*type:* flag) To disable printing of information

---

class_generic

not_set

---

dt_calc_dt_calc

Synonyms: dt_calc

Inherits from: *class_generic*

The time step at first iteration is calculated in agreement with CFL condition.

---

dt_calc_dt_fixe

Synonyms: dt_fixe

Inherits from: *class_generic*

The first time step is fixed by the user (recommended when resuming calculation with Crank Nicholson temporal scheme to ensure continuity).

Parameters are:

- **value** (*type:* double) first time step.

---

dt_calc_dt_min

Synonyms: dt_min

Inherits from: *class_generic*

The first iteration is based on dt_min.

---

dt_start

Inherits from: *class_generic*

not_set

---

gcp_ns

Inherits from: *class_generic*

not_set

Parameters are:

- **solveur0** (*type:* *solveur_sys_base*) Solver type.
- **solveur1** (*type:* *solveur_sys_base*) Solver type.
- **[precond]** (*type:* *precond_base*) Keyword to define system preconditioning in order to accelerate resolution by the conjugated gradient. Many parallel preconditioning methods are not equivalent to their sequential counterpart, and you should therefore expect differences, especially when you select a high value of the final residue (seuil). The result depends on the number of processors and on the mesh splitting. It is sometimes useful to run the solver with no preconditioning at all. In particular: - when the solver does not converge during initial projection, - when comparing sequential and parallel computations. With no preconditioning, except in some particular cases (no open boundary), the sequential and the parallel computations should provide exactly the same results within fpu accuracy. If not, there might be a coding error or the system of equations is singular.
- **[precond_nul]** (*type:* flag) Keyword to not use a preconditioning method.
- **seuil** (*type:* double) Value of the final residue. The gradient ceases iteration when the Euclidean residue standard ||Ax-B|| is less than this value.
- **[impr]** (*type:* flag) Keyword which is used to request display of the Euclidean residue standard each time this iterates through the conjugated gradient (display to the standard outlet).
- **[quiet]** (*type:* flag) To not displaying any outputs of the solver.
- **[save_matrice | save_matrix]** (*type:* flag) to save the matrix in a file.
- **[optimized]** (*type:* flag) This keyword triggers a memory and network optimized algorithms useful for strong scaling (when computing less than 100 000 elements per processor). The matrix and the vectors are duplicated, common items removed and only virtual items really used in the matrix are exchanged.NL2 Warning: this is experimental and known to fail in some VEF computations (L2 projection step will not converge). Works well in VDF.
- **[nb_it_max]** (*type:* int) Keyword to set the maximum iterations number for the Gcp.

---

gen

Inherits from: *class_generic*

not_set

Parameters are:

- **solv_elem** (*type:* string) To specify a solver among gmres or bicgstab.
- **precond** (*type:* *precond_base*) The only preconditionner that we can specify is ilu.
- **[seuil]** (*type:* double) Value of the final residue. The solver ceases iterations when the Euclidean residue standard ||Ax-B|| is less than this value. default value 1e-12.
- **[impr]** (*type:* flag) Keyword which is used to request display of the Euclidean residue standard each time this iterates through the conjugated gradient (display to the standard outlet).
- **[save_matrice | save_matrix]** (*type:* flag) To save the matrix in a file.

- **[quiet]** (*type:* flag) To not displaying any outputs of the solver.
- **[nb_it_max]** (*type:* int) Keyword to set the maximum iterations number for the GEN solver.
- **[force]** (*type:* flag) Keyword to set ipar[5]=-1 in the GEN solver. This is helpful if you notice that the solver does not perform more than 100 iterations. If this keyword is specified in the datafile, you should provide nb_it_max.

---

gmres

Inherits from: *class_generic*

Gmres method (for non symetric matrix).

Parameters are:

- **[impr]** (*type:* flag) Keyword which may be used to print the convergence.
- **[quiet]** (*type:* flag) To disable printing of information
- **[seuil]** (*type:* double) Convergence value.
- **[diag]** (*type:* flag) Keyword to use diagonal preconditionner (in place of pilut that is not parallel).
- **[nb_it_max]** (*type:* int) Keyword to set the maximum iterations number for the Gmres.
- **[controle_residu]** (*type:* int into ["0", "1"]) Keyword of Boolean type (by default 0). If set to 1, the convergence occurs if the residu suddenly increases.
- **[save_matrice | save_matrix]** (*type:* flag) to save the matrix in a file.
- **[dim_espace_krilov]** (*type:* int) not_set

---

optimal

Inherits from: *class_generic*

Optimal is a solver which tests several solvers of the previous list to choose the fastest one for the considered linear system.

Parameters are:

- **seuil** (*type:* double) Convergence threshold
- **[impr]** (*type:* flag) To print the convergency of the fastest solver
- **[quiet]** (*type:* flag) To disable printing of information
- **[save_matrice | save_matrix]** (*type:* flag) To save the linear system (A, x, B) into a file
- **[frequence_recalc]** (*type:* int) To set a time step period (by default, 100) for re-checking the fatest solver
- **[nom_fichier_solveur]** (*type:* string) To specify the file containing the list of the tested solvers
- **[fichier_solveur_non_recree]** (*type:* flag) To avoid the creation of the file containing the list

---

petsc

Inherits from: *class_generic*

Solver via Petsc API

Dummy test with math!

Quand $a > 0$ alors il y a deux solutions:

$$ax^2 + bx + c = 0$$

et elles sont exactement

x = {-b pm sqrt{b^2 - 4ac} over 2a}.

Parameters are:

- **solveur** (*type:* string) not_set
- **option_solveur** (*type: bloc_lecture*) not_set
- **[atol]** (*type:* double) Absolute threshold for convergence (same as seuil option)
- **[rtol]** (*type:* double) Relative threshold for convergence

---

rocalution

Inherits from: *class_generic*

Solver via rocALUTION API

Parameters are:

- **solveur** (*type:* string) not_set
- **option_solveur** (*type: bloc_lecture*) not_set
- **[atol]** (*type:* double) Absolute threshold for convergence (same as seuil option)
- **[rtol]** (*type:* double) Relative threshold for convergence

---

solv_gcp

Synonyms: gcp

Inherits from: *class_generic*

Preconditioned conjugated gradient.

Parameters are:

- **[precond]** (*type: precond_base*) Keyword to define system preconditioning in order to accelerate resolution by the conjugated gradient. Many parallel preconditioning methods are not equivalent to their sequential counterpart, and you should therefore expect differences, especially when you select a high value of the final residue (seuil). The result depends on the number of processors and on the mesh splitting. It is sometimes useful to run the solver with no preconditioning at all. In particular: - when the solver does not converge during initial projection, - when comparing sequential and parallel computations. With no preconditioning, except in some particular cases (no open boundary), the sequential and the parallel computations should provide exactly the same results within fpu accuracy. If not, there might be a coding error or the system of equations is singular.

- **[precond_nul]** (*type:* flag) Keyword to not use a preconditioning method.
- **seuil** (*type:* double) Value of the final residue. The gradient ceases iteration when the Euclidean residue standard ||Ax-B|| is less than this value.
- **[impr]** (*type:* flag) Keyword which is used to request display of the Euclidean residue standard each time this iterates through the conjugated gradient (display to the standard outlet).
- **[quiet]** (*type:* flag) To not displaying any outputs of the solver.
- **[save_matrice | save_matrix]** (*type:* flag) to save the matrix in a file.
- **[optimized]** (*type:* flag) This keyword triggers a memory and network optimized algorithms useful for strong scaling (when computing less than 100 000 elements per processor). The matrix and the vectors are duplicated, common items removed and only virtual items really used in the matrix are exchanged.NL2 Warning: this is experimental and known to fail in some VEF computations (L2 projection step will not converge). Works well in VDF.
- **[nb_it_max]** (*type:* int) Keyword to set the maximum iterations number for the Gcp.

---

solveur_sys_base

Inherits from: *class_generic*

Basic class to solve the linear system.

2.1.6 Keywords derived from coarsen_operators

coarsen_operators

not_set

2.1.7 Keywords derived from comment

comment

Synonyms: #

Comments in a data file.

Parameters are:

- **comm** (*type:* string) Text to be commented.

2.1.8 Keywords derived from condinits

condinits

Initial conditions.

2.1.9 Keywords derived from condlim_base

condlim_base

Basic class of boundary conditions.

---

dirichlet

Inherits from: *condlim_base*

Dirichlet condition at the boundary called bord (edge) : 1). For Navier-Stokes equations, velocity imposed at the boundary; 2). For scalar transport equation, scalar imposed at the boundary.

---

echange_couplage_thermique

Inherits from: *condlim_base*

Thermal coupling boundary condition

Parameters are:

- **[temperature_paroi]** (*type: field_base*) Temperature
- **[flux_paroi]** (*type: field_base*) Wall heat flux

---

echange_interne_global_impose

Synonyms: paroi_echange_interne_global_impose

Inherits from: *condlim_base*

Internal heat exchange boundary condition with global exchange coefficient.

Parameters are:

- **h_imp** (*type:* string) Global exchange coefficient value. The global exchange coefficient value is expressed in W.m-2.K-1.
- **ch** (*type: front_field_base*) Boundary field type.

---

echange_interne_global_parfait

Synonyms: paroi_echange_interne_global_parfait

Inherits from: *condlim_base*

Internal heat exchange boundary condition with perfect (infinite) exchange coefficient.

---

echange_interne_impose

Synonyms: paroi_echange_interne_impose

Inherits from: *condlim_base*

Internal heat exchange boundary condition with exchange coefficient.

Parameters are:

- **h_imp** (*type:* string) Exchange coefficient value expressed in W.m-2.K-1.
- **ch** (*type: front_field_base*) Boundary field type.

---

echange_interne_parfait

Synonyms: paroi_echange_interne_parfait

Inherits from: *condlim_base*

Internal heat exchange boundary condition with perfect (infinite) exchange coefficient.

---

entree_temperature_imposee_h

Inherits from: *condlim_base*

Particular case of class frontiere_ouverte_temperature_imposee for enthalpy equation.

Parameters are:

- **ch** (*type: front_field_base*) Boundary field type.

---

frontiere_ouverte

Inherits from: *condlim_base*

Boundary outlet condition on the boundary called bord (edge) (diffusion flux zero). This condition must be associated with a boundary outlet hydraulic condition.

Parameters are:

- **var_name** (*type:* string into ["t_ext", "c_ext", "y_ext", "k_eps_ext", "fluctu_temperature_ext", "flux_chaleur_turb_ext", "v2_ext", "a_ext", "tau_ext", "k_ext", "omega_ext"]) Field name.
- **ch** (*type: front_field_base*) Boundary field type.

---

frontiere_ouverte_concentration_imposee

Inherits from: *condlim_base*

Imposed concentration condition at an open boundary called bord (edge) (situation corresponding to a fluid inlet). This condition must be associated with an imposed inlet velocity condition.

Parameters are:

- **ch** (*type: front_field_base*) Boundary field type.

---

frontiere_ouverte_fraction_massique_imposee

Inherits from: *condlim_base*

not_set

Parameters are:

- **ch** (*type: front_field_base*) Boundary field type.

---

frontiere_ouverte_gradient_pression_impose

Inherits from: *condlim_base*

Normal imposed pressure gradient condition on the open boundary called bord (edge). This boundary condition may be only used in VDF discretization. The imposed $partial P/partial n$ value is expressed in Pa.m-1.

Parameters are:

- **ch** (*type: front_field_base*) Boundary field type.

---

frontiere_ouverte_gradient_pression_impose_vefprep1b

Inherits from: *condlim_base*

Keyword for an outlet boundary condition in VEF P1B/P1NC on the gradient of the pressure.

Parameters are:

- **ch** (*type: front_field_base*) Boundary field type.

---

frontiere_ouverte_gradient_pression_libre_vef

Inherits from: *condlim_base*

Class for outlet boundary condition in VEF like Orlansky. There is no reference for pressure for theses boundary conditions so it is better to add pressure condition (with Frontiere_ouverte_pression_imposee) on one or two cells (for symmetry in a channel) of the boundary where Orlansky conditions are imposed.

---

frontiere_ouverte_gradient_pression_libre_vefprep1b

Inherits from: *condlim_base*

Class for outlet boundary condition in VEF P1B/P1NC like Orlansky.

---

frontiere_ouverte_pression_imposee

Inherits from: *condlim_base*

Imposed pressure condition at the open boundary called bord (edge). The imposed pressure field is expressed in Pa.

Parameters are:

- **ch** (*type: front_field_base*) Boundary field type.

---

frontiere_ouverte_pression_imposee_orlansky

Inherits from: *condlim_base*

This boundary condition may only be used with VDF discretization. There is no reference for pressure for this boundary condition so it is better to add pressure condition (with Frontiere_ouverte_pression_imposee) on one or two cells (for symetry in a channel) of the boundary where Orlansky conditions are imposed.

---

frontiere_ouverte_pression_moyenne_imposee

Inherits from: *condlim_base*

Class for open boundary with pressure mean level imposed.

Parameters are:

- **pext** (*type:* double) Mean pressure.

---

frontiere_ouverte_rho_u_impose

Inherits from: *condlim_base*

This keyword is used to designate a condition of imposed mass rate at an open boundary called bord (edge). The imposed mass rate field at the inlet is vectorial and the imposed velocity values are expressed in kg.s-1. This boundary condition can be used only with the Quasi compressible model.

Parameters are:

- **ch** (*type: front_field_base*) Boundary field type.

---

frontiere_ouverte_temperature_imposee

Inherits from: *condlim_base*

Imposed temperature condition at the open boundary called bord (edge) (in the case of fluid inlet). This condition must be associated with an imposed inlet velocity condition. The imposed temperature value is expressed in oC or K.

Parameters are:

- **ch** (*type: front_field_base*) Boundary field type.

---

frontiere_ouverte_vitesse_imposee

Inherits from: *condlim_base*

Class for velocity-inlet boundary condition. The imposed velocity field at the inlet is vectorial and the imposed velocity values are expressed in m.s-1.

Parameters are:

- **ch** (*type: front_field_base*) Boundary field type.

---

frontiere_ouverte_vitesse_imposee_sortie

Inherits from: *condlim_base*

Sub-class for velocity boundary condition. The imposed velocity field at the open boundary is vectorial and the imposed velocity values are expressed in m.s-1.

Parameters are:

- **ch** (*type: front_field_base*) Boundary field type.

---

neumann

Inherits from: *condlim_base*

Neumann condition at the boundary called bord (edge) : 1). For Navier-Stokes equations, constraint imposed at the boundary; 2). For scalar transport equation, flux imposed at the boundary.

---

neumann_homogene

Inherits from: *condlim_base*

Homogeneous neumann boundary condition

---

neumann_paroi

Inherits from: *condlim_base*

Neumann boundary condition for mass equation (multiphase problem)

Parameters are:

- **ch** (*type: front_field_base*) Boundary field type.

---

neumann_paroi_adiabatique

Inherits from: *condlim_base*

Adiabatic wall neumann boundary condition

---

paroi

Inherits from: *condlim_base*

Impermeability condition at a wall called bord (edge) (standard flux zero). This condition must be associated with a wall type hydraulic condition.

---

paroi_adiabatique

Inherits from: *condlim_base*

Normal zero flux condition at the wall called bord (edge).

---

paroi_contact

Inherits from: *condlim_base*

Thermal condition between two domains. Important: the name of the boundaries in the two domains should be the same. (Warning: there is also an old limitation not yet fixed on the sequential algorithm in VDF to detect the matching faces on the two boundaries: faces should be ordered in the same way). The kind of condition depends on the discretization. In VDF, it is a heat exchange condition, and in VEF, a temperature condition.

Such a coupling requires coincident meshes for the moment. In case of non-coincident meshes, run is stopped and two external files are automatically generated in VEF (connectivity_failed_boundary_name and connectivity_failed_pb_name.med). In 2D, the keyword Decouper_bord_coincident associated to the connectivity_failed_boundary_name file allows to generate a new coincident mesh.

In 3D, for a first preliminary cut domain with HOMARD (fluid for instance), the second problem associated to pb_name (solide in a fluid/solid coupling problem) has to be submitted to HOMARD cutting procedure with connectivity_failed_pb_name.med.

Such a procedure works as while the primary refined mesh (fluid in our example) impacts the fluid/solid interface with a compact shape as described below (values 2 or 4 indicates the number of division from primary faces obtained in fluid domain at the interface after HOMARD cutting):

2-2-2-2-2-2

2-4-4-4-4-4-2 \; 2-2-2

2-4-4-4-4-2 \; 2-4-2

2-2-2-2-2 \; 2-2

OK

NL2 2-2 \; \; 2-2-2

2-4-2 \; 2-2

2-2 \; 2-2

NOT OK

Parameters are:

- **autrepb** (*type:* *pb_base*) Name of other problem.
- **nameb** (*type:* string) boundary name of the remote problem which should be the same than the local name

---

paroi_contact_fictif

Inherits from: *condlim_base*

This keyword is derivated from paroi_contact and is especially dedicated to compute coupled fluid/solid/fluid problem in case of thin material. Thanks to this option, solid is considered as a fictitious media (no mesh, no domain associated), and coupling is performed by considering instantaneous thermal equilibrium in it (for the moment).

Parameters are:

- **autrepb** (*type:* *pb_base*) Name of other problem.
- **nameb** (*type:* string) Name of bord.
- **conduct_fictif** (*type:* double) thermal conductivity
- **ep_fictive** (*type:* double) thickness of the fictitious media

---

paroi_decalee_robin

Inherits from: *condlim_base*

This keyword is used to designate a Robin boundary condition (a.u+b.du/dn=c) associated with the Pironneau methodology for the wall laws. The value of given by the delta option is the distance between the mesh (where symmetry boundary condition is applied) and the fictious wall. This boundary condition needs the definition of the dedicated source terms (Source_Robin or Source_Robin_Scalaire) according the equations used.

Parameters are:

- **delta** (*type:* double) not_set

---

paroi_defilante

Inherits from: *condlim_base*

Keyword to designate a condition where tangential velocity is imposed on the wall called bord (edge). If the velocity components set by the user is not tangential, projection is used.

Parameters are:

- **ch** (*type: front_field_base*) Boundary field type.

---

paroi_echange_contact_correlation_vdf

Inherits from: *condlim_base*

Class to define a thermohydraulic 1D model which will apply to a boundary of 2D or 3D domain.

Warning : For parallel calculation, the only possible partition will be according the axis of the model with the keyword Tranche.

Parameters are:

- **[dir]** (*type:* int) Direction (0 : axis X, 1 : axis Y, 2 : axis Z) of the 1D model.
- **[tinf]** (*type:* double) Inlet fluid temperature of the 1D model (oC or K).
- **[tsup]** (*type:* double) Outlet fluid temperature of the 1D model (oC or K).
- **[Lambda]** (*type:* string) Thermal conductivity of the fluid (W.m-1.K-1).
- **[rho]** (*type:* string) Mass density of the fluid (kg.m-3) which may be a function of the temperature T.
- **[dt_impr]** (*type:* double) Printing period in name_of_data_file_time.dat files of the 1D model results.
- **[cp]** (*type:* double) Calorific capacity value at a constant pressure of the fluid (J.kg-1.K-1).
- **[mu]** (*type:* string) Dynamic viscosity of the fluid (kg.m-1.s-1) which may be a function of thetemperature T.
- **[debit]** (*type:* double) Surface flow rate (kg.s-1.m-2) of the fluid into the channel.
- **[dh]** (*type:* double) Hydraulic diameter may be a function f(x) with x position along the 1D axis (xinf <= x <= xsup)
- **[volume]** (*type:* string) Exact volume of the 1D domain (m3) which may be a function of the hydraulic diameter (Dh) and the lateral surface (S) of the meshed boundary.
- **[nu]** (*type:* string) Nusselt number which may be a function of the Reynolds number (Re) and the Prandtl number (Pr).
- **[reprise_correlation]** (*type:* flag) Keyword in the case of a resuming calculation with this correlation.

---

paroi_echange_contact_correlation_vef

Inherits from: *condlim_base*

Class to define a thermohydraulic 1D model which will apply to a boundary of 2D or 3D domain.

Warning : For parallel calculation, the only possible partition will be according the axis of the model with the keyword Tranche_geom.

Parameters are:

- **[dir]** (*type:* int) Direction (0 : axis X, 1 : axis Y, 2 : axis Z) of the 1D model.
- **[tinf]** (*type:* double) Inlet fluid temperature of the 1D model (oC or K).
- **[tsup]** (*type:* double) Outlet fluid temperature of the 1D model (oC or K).
- **[Lambda]** (*type:* string) Thermal conductivity of the fluid (W.m-1.K-1).
- **[rho]** (*type:* string) Mass density of the fluid (kg.m-3) which may be a function of the temperature T.
- **[dt_impr]** (*type:* double) Printing period in name_of_data_file_time.dat files of the 1D model results.
- **[cp]** (*type:* double) Calorific capacity value at a constant pressure of the fluid (J.kg-1.K-1).
- **[mu]** (*type:* string) Dynamic viscosity of the fluid (kg.m-1.s-1) which may be a function of thetemperature T.
- **[debit]** (*type:* double) Surface flow rate (kg.s-1.m-2) of the fluid into the channel.
- **[n]** (*type:* int) Number of 1D cells of the 1D mesh.
- **[dh]** (*type:* string) Hydraulic diameter may be a function f(x) with x position along the 1D axis (xinf <= x <= xsup)
- **[surface]** (*type:* string) Section surface of the channel which may be function f(Dh,x) of the hydraulic diameter (Dh) and x position along the 1D axis (xinf <= x <= xsup)
- **[xinf]** (*type:* double) Position of the inlet of the 1D mesh on the axis direction.
- **[xsup]** (*type:* double) Position of the outlet of the 1D mesh on the axis direction.
- **[nu]** (*type:* string) Nusselt number which may be a function of the Reynolds number (Re) and the Prandtl number (Pr).
- **[emissivite_pour_rayonnement_entre_deux_plaques_quasi_infinies]** (*type:* double) Coefficient of emissivity for radiation between two quasi infinite plates.
- **[reprise_correlation]** (*type:* flag) Keyword in the case of a resuming calculation with this correlation.

---

paroi_echange_contact_vdf

Inherits from: *condlim_base*

Boundary condition type to model the heat flux between two problems. Important: the name of the boundaries in the two problems should be the same.

Parameters are:

- **autrepb** (*type: pb_base*) Name of other problem.
- **nameb** (*type:* string) Name of bord.
- **temp** (*type:* string) Name of field.

- **h** (*type:* double) Value assigned to a coefficient (expressed in W.K-1m-2) that characterises the contact between the two mediums. In order to model perfect contact, h must be taken to be infinite. This value must obviously be the same in both the two problems blocks. The surface thermal flux exchanged between the two mediums is represented by : fi = h (T1-T2) where 1/h = d1/lambda1 + 1/val_h_contact + d2/lambda2 where di : distance between the node where Ti and the wall is found.

---

paroi_echange_externe_impose

Inherits from: *condlim_base*

External type exchange condition with a heat exchange coefficient and an imposed external temperature.

Parameters are:

- **h_imp** (*type:* string) Heat exchange coefficient value (expressed in W.m-2.K-1).
- **himpc** (*type: front_field_base*) Boundary field type.
- **text** (*type:* string) External temperature value (expressed in oC or K).
- **ch** (*type: front_field_base*) Boundary field type.

---

paroi_echange_externe_impose_h

Inherits from: *condlim_base*

Particular case of class paroi_echange_externe_impose for enthalpy equation.

Parameters are:

- **h_imp** (*type:* string) Heat exchange coefficient value (expressed in W.m-2.K-1).
- **himpc** (*type: front_field_base*) Boundary field type.
- **text** (*type:* string) External temperature value (expressed in oC or K).
- **ch** (*type: front_field_base*) Boundary field type.

---

paroi_echange_global_impose

Inherits from: *condlim_base*

Global type exchange condition (internal) that is to say that diffusion on the first fluid mesh is not taken into consideration.

Parameters are:

- **h_imp** (*type:* string) Global exchange coefficient value. The global exchange coefficient value is expressed in W.m-2.K-1.
- **himpc** (*type: front_field_base*) Boundary field type.
- **text** (*type:* string) External temperature value. The external temperature value is expressed in oC or K.
- **ch** (*type: front_field_base*) Boundary field type.

---

paroi_fixe

Inherits from: *condlim_base*

Keyword to designate a situation of adherence to the wall called bord (edge) (normal and tangential velocity at the edge is zero).

---

paroi_fixe_iso_genepi2_sans_contribution_aux_vitesses_sommets

Inherits from: *condlim_base*

Boundary condition to obtain iso Geneppi2, without interest

---

paroi_flux_impose

Inherits from: *condlim_base*

Normal flux condition at the wall called bord (edge). The surface area of the flux (W.m-1 in 2D or W.m-2 in 3D) is imposed at the boundary according to the following convention: a positive flux is a flux that enters into the domain according to convention.

Parameters are:

- **ch** (*type: front_field_base*) Boundary field type.

---

paroi_knudsen_non_negligeable

Inherits from: *condlim_base*

Boundary condition for number of Knudsen (Kn) above 0.001 where slip-flow condition appears: the velocity near the wall depends on the shear stress : Kn=l/L with l is the mean-free-path of the molecules and L a characteristic length scale.

U(y=0)-Uwall=k(dU/dY)

Where k is a coefficient given by several laws:

Mawxell : k=(2-s)*l/s

Bestok&Karniadakis :k=(2-s)/s*L*Kn/(1+Kn)

Xue&Fan :k=(2-s)/s*L*tanh(Kn)

s is a value between 0 and 2 named accomodation coefficient. s=1 seems a good value.

Warning : The keyword is available for VDF calculation only for the moment.

Parameters are:

- **name_champ_1** (*type:* string into ["vitesse_paroi", "k"]) Field name.
- **champ_1** (*type: front_field_base*) Boundary field type.
- **name_champ_2** (*type:* string into ["vitesse_paroi", "k"]) Field name.
- **champ_2** (*type: front_field_base*) Boundary field type.

---

paroi_temperature_imposee

Inherits from: *condlim_base*

Imposed temperature condition at the wall called bord (edge).

Parameters are:

- **ch** (*type: front_field_base*) Boundary field type.

---

periodic

Synonyms: periodique

Inherits from: *condlim_base*

1). For Navier-Stokes equations, this keyword is used to indicate that the horizontal inlet velocity values are the same as the outlet velocity values, at every moment. As regards meshing, the inlet and outlet edges bear the same name.; 2). For scalar transport equation, this keyword is used to set a periodic condition on scalar. The two edges dealing with this periodic condition bear the same name.

---

scalaire_impose_paroi

Inherits from: *condlim_base*

Imposed temperature condition at the wall called bord (edge).

Parameters are:

- **ch** (*type: front_field_base*) Boundary field type.

---

sortie_libre_temperature_imposee_h

Inherits from: *condlim_base*

Open boundary for heat equation with enthalpy as unknown.

Parameters are:

- **ch** (*type: front_field_base*) Boundary field type.

---

symetrie

Inherits from: *condlim_base*

1). For Navier-Stokes equations, this keyword is used to designate a symmetry condition concerning the velocity at the boundary called bord (edge) (normal velocity at the edge equal to zero and tangential velocity gradient at the edge equal to zero); 2). For scalar transport equation, this keyword is used to set a symmetry condition on scalar on the boundary named bord (edge).

---

temperature_imposee_paroi

Inherits from: *condlim_base*

Imposed temperature condition at the wall called bord (edge).

Parameters are:

- **ch** (*type: front_field_base*) Boundary field type.

2.1.10 Keywords derived from condlims

condlims

Boundary conditions.

2.1.11 Keywords derived from definition_champs

definition_champs

List of definition champ

2.1.12 Keywords derived from discretisation_base

discretisation_base

Basic class for space discretization of thermohydraulic turbulent problems.

---

ef

Inherits from: *discretisation_base*

Element Finite discretization.

---

polymac

Inherits from: *discretisation_base*

polymac discretization (polymac discretization that is not compatible with pb_multi).

---

polymac_p0

Inherits from: *discretisation_base*

polymac_p0 discretization (previously covimac discretization compatible with pb_multi).

---

polymac_p0p1nc

Inherits from: *discretisation_base*

polymac_P0P1NC discretization (previously polymac discretization compatible with pb_multi).

---

vdf

Inherits from: *discretisation_base*

Finite difference volume discretization.

---

vef

Synonyms: vefprep1b

Inherits from: *discretisation_base*

Finite element volume discretization (P1NC/P1-bubble element). Since the 1.5.5 version, several new discretizations are available thanks to the optional keyword Read. By default, the VEFPreP1B keyword is equivalent to the former VEFPreP1B formulation (v1.5.4 and sooner). P0P1 (if used with the strong formulation for imposed pressure boundary) is equivalent to VEFPreP1B but the convergence is slower. VEFPreP1B dis is equivalent to VEFPreP1B dis Read dis { P0 P1 Changement_de_base_P1Bulle 1 Cl_pression_sommet_faible 0 }

Parameters are:

- **[changement_de_base_p1bulle]** (*type:* int into [0,1]) changement_de_base_p1bulle 1 This option may be used to have the P1NC/P0P1 formulation (value set to 0) or the P1NC/P1Bulle formulation (value set to 1, the default).
- **[p0]** (*type:* flag) Pressure nodes are added on element centres
- **[p1]** (*type:* flag) Pressure nodes are added on vertices
- **[pa]** (*type:* flag) Only available in 3D, pressure nodes are added on bones
- **[rt]** (*type:* flag) For P1NCP1B (in TrioCFD)
- **[modif_div_face_dirichlet]** (*type:* int into [0,1]) This option (by default 0) is used to extend control volumes for the momentum equation.
- **[cl_pression_sommet_faible]** (*type:* int into [0,1]) This option is used to specify a strong formulation (value set to 0, the default) or a weak formulation (value set to 1) for an imposed pressure boundary condition. The first formulation converges quicker and is stable in general cases. The second formulation should be used if there are several outlet boundaries with Neumann condition (see Ecoulement_Neumann test case for example).

2.1.13 Keywords derived from domaine

domaine

Keyword to create a domain.

---

domaineaxi1d

Inherits from: *domaine*

1D domain

---

ijk_grid_geometry

Inherits from: *domaine*

Object to define the grid that will represent the domain of the simulation in IJK discretization

Parameters are:

- **[perio_i]** (*type:* flag) flag to specify the border along the I direction is periodic
- **[perio_j]** (*type:* flag) flag to specify the border along the J direction is periodic
- **[perio_k]** (*type:* flag) flag to specify the border along the K direction is periodic
- **[nbelem_i]** (*type:* int) the number of elements of the grid in the I direction
- **[nbelem_j]** (*type:* int) the number of elements of the grid in the J direction
- **[nbelem_k]** (*type:* int) the number of elements of the grid in the K direction
- **[uniform_domain_size_i]** (*type:* double) the size of the elements along the I direction
- **[uniform_domain_size_j]** (*type:* double) the size of the elements along the J direction
- **[uniform_domain_size_k]** (*type:* double) the size of the elements along the K direction
- **[origin_i]** (*type:* double) I-coordinate of the origin of the grid
- **[origin_j]** (*type:* double) J-coordinate of the origin of the grid
- **[origin_k]** (*type:* double) K-coordinate of the origin of the grid

---

2.1.14 Keywords derived from field_base

champ_composite

Inherits from: *field_base*

Composite field. Used in multiphase problems to associate data to each phase.

Parameters are:

- **dim** (*type:* int) Number of field components.

- **bloc** (*type: bloc_lecture*) Values Various pieces of the field, defined per phase. Part 1 goes to phase 1, etc...

---

champ_don_base

Inherits from: *field_base*

Basic class for data fields (not calculated), p.e. physics properties.

---

champ_don_lu

Inherits from: *field_base*

Field to read a data field (values located at the center of the cells) in a file.

Parameters are:

- **dom** (*type: domaine*) Name of the domain.
- **nb_comp** (*type:* int) Number of field components.
- **file** (*type:* string) Name of the file. This file has the following format: nb_val_lues -> Number of values readen in th file Xi Yi Zi -> Coordinates readen in the file Ui Vi Wi -> Value of the field

---

champ_fonc_fonction

Inherits from: *field_base*

Field that is a function of another field.

Parameters are:

- **problem_name** (*type: pb_base*) Name of problem.
- **inco** (*type:* string) Name of the field (for example: temperature).
- **expression** (*type:* string list) Number of field components followed by the analytical expression for each field component.

---

champ_fonc_fonction_txyz

Inherits from: *field_base*

this refers to a field that is a function of another field and time and/or space coordinates

Parameters are:

- **problem_name** (*type: pb_base*) Name of problem.
- **inco** (*type:* string) Name of the field (for example: temperature).
- **expression** (*type:* string list) Number of field components followed by the analytical expression for each field component.

---

champ_fonc_fonction_txyz_morceaux

Inherits from: *field_base*

Field defined by analytical functions in each sub-domaine. On each zone, the value is defined as a function of x,y,z,t and of scalar value taken from a parameter field. This values is associated to the variable 'val' in the expression.

Parameters are:

- **problem_name** (*type: pb_base*) Name of the problem.
- **inco** (*type:* string) Name of the field (for example: temperature).
- **nb_comp** (*type:* int) Number of field components.
- **data** (*type: bloc_lecture*) { Defaut val_def sous_domaine_1 val_1 ... sous_domaine_i val_i } By default, the value val_def is assigned to the field. It takes the sous_domaine_i identifier Sous_Domaine (sub_area) type object function, val_i. Sous_Domaine (sub_area) type objects must have been previously defined if the operator wishes to use a champ_fonc_fonction_txyz_morceaux type object.

---

champ_fonc_interp

Inherits from: *field_base*

Field that is interpolated from a distant domain via MEDCoupling (remapper).

Parameters are:

- **nom_champ** (*type:* string) Name of the field (for example: temperature).
- **pb_loc** (*type:* string) Name of the local problem.
- **pb_dist** (*type:* string) Name of the distant problem.
- **[dom_loc]** (*type:* string) Name of the local domain.
- **[dom_dist]** (*type:* string) Name of the distant domain.
- **[default_value]** (*type:* string) Name of the distant domain.
- **nature** (*type:* string) Nature of the field (knowledge from MEDCoupling is required; IntensiveMaximum, IntensiveConservation, ...).
- **[use_overlapdec]** (*type:* string) Nature of the field (knowledge from MEDCoupling is required; IntensiveMaximum, IntensiveConservation, ...).

---

champ_fonc_med

Inherits from: *field_base*

Field to read a data field in a MED-format file .med at a specified time. It is very useful, for example, to resume a calculation with a new or refined geometry. The field post-processed on the new geometry at med format is used as initial condition for the resume.

Parameters are:

- **[use_existing_domain]** (*type:* flag) whether to optimize the field loading by indicating that the field is supported by the same mesh that was initially loaded as the domain

- **[last_time]** (*type:* flag) to use the last time of the MED file instead of the specified time. Mutually exclusive with 'time' parameter.
- **[decoup]** (*type:* string) specify a partition file.
- **[mesh]** (*type:* string) Name of the mesh supporting the field. This is the name of the mesh in the MED file, and if this mesh was also used to create the TRUST domain, loading can be optimized with option 'use_existing_domain'.
- **domain** (*type:* string) Name of the domain supporting the field. This is the name of the mesh in the MED file, and if this mesh was also used to create the TRUST domain, loading can be optimized with option 'use_existing_domain'.
- **file** (*type:* string) Name of the .med file.
- **field** (*type:* string) Name of field to load.
- **[loc]** (*type:* string into ["som", "elem"]) To indicate where the field is localised. Default to 'elem'.
- **[time]** (*type:* double) Timestep to load from the MED file. Mutually exclusive with 'last_time' flag.

---

champ_fonc_med_table_temps

Inherits from: *field_base*

Field defined as a fixed spatial shape scaled by a temporal coefficient

Parameters are:

- **[table_temps]** (*type:* string) Table containing the temporal coefficient used to scale the field
- **[table_temps_lue]** (*type:* string) Name of the file containing the values of the temporal coefficient used to scale the field
- **[use_existing_domain]** (*type:* flag) whether to optimize the field loading by indicating that the field is supported by the same mesh that was initially loaded as the domain
- **[last_time]** (*type:* flag) to use the last time of the MED file instead of the specified time. Mutually exclusive with 'time' parameter.
- **[decoup]** (*type:* string) specify a partition file.
- **[mesh]** (*type:* string) Name of the mesh supporting the field. This is the name of the mesh in the MED file, and if this mesh was also used to create the TRUST domain, loading can be optimized with option 'use_existing_domain'.
- **domain** (*type:* string) Name of the domain supporting the field. This is the name of the mesh in the MED file, and if this mesh was also used to create the TRUST domain, loading can be optimized with option 'use_existing_domain'.
- **file** (*type:* string) Name of the .med file.
- **field** (*type:* string) Name of field to load.
- **[loc]** (*type:* string into ["som", "elem"]) To indicate where the field is localised. Default to 'elem'.
- **[time]** (*type:* double) Timestep to load from the MED file. Mutually exclusive with 'last_time' flag.

---

champ_fonc_med_tabule

Inherits from: *field_base*

not_set

Parameters are:

- **[use_existing_domain]** (*type:* flag) whether to optimize the field loading by indicating that the field is supported by the same mesh that was initially loaded as the domain
- **[last_time]** (*type:* flag) to use the last time of the MED file instead of the specified time. Mutually exclusive with 'time' parameter.
- **[decoup]** (*type:* string) specify a partition file.
- **[mesh]** (*type:* string) Name of the mesh supporting the field. This is the name of the mesh in the MED file, and if this mesh was also used to create the TRUST domain, loading can be optimized with option 'use_existing_domain'.
- **domain** (*type:* string) Name of the domain supporting the field. This is the name of the mesh in the MED file, and if this mesh was also used to create the TRUST domain, loading can be optimized with option 'use_existing_domain'.
- **file** (*type:* string) Name of the .med file.
- **field** (*type:* string) Name of field to load.
- **[loc]** (*type:* string into ["som", "elem"]) To indicate where the field is localised. Default to 'elem'.
- **[time]** (*type:* double) Timestep to load from the MED file. Mutually exclusive with 'last_time' flag.

---

champ_fonc_reprise

Inherits from: *field_base*

This field is used to read a data field in a save file (.xyz or .sauv) at a specified time. It is very useful, for example, to run a thermohydraulic calculation with velocity initial condition read into a save file from a previous hydraulic calculation.

Parameters are:

- **[format]** (*type:* string into ["binaire", "formatte", "xyz", "single_hdf"]) Type of file (the file format). If xyz format is activated, the .xyz file from the previous calculation will be given for filename, and if formatte or binaire is choosen, the .sauv file of the previous calculation will be specified for filename. In the case of a parallel calculation, if the mesh partition does not changed between the previous calculation and the next one, the binaire format should be preferred, because is faster than the xyz format. If single_hdf is used, the same constraints/advantages as binaire apply, but a single (HDF5) file is produced on the filesystem instead of having one file per processor.
- **filename** (*type:* string) Name of the save file.
- **pb_name** (*type: pb_base*) Name of the problem.
- **champ** (*type:* string) Name of the problem unknown. It may also be the temporal average of a problem unknown (like moyenne_vitesse, moyenne_temperature,...)
- **[fonction]** (*type: fonction_champ_reprise*) Optional keyword to apply a function on the field being read in the save file (e.g. to read a temperature field in Celsius units and convert it for the calculation on Kelvin units, you will use: fonction 1 273.+val)

- **temps | time** (*type:* string) Time of the saved field in the save file or last_time. If you give the keyword last_time instead, the last time saved in the save file will be used.

---

champ_fonc_t

Inherits from: *field_base*

Field that is constant in space and is a function of time.

Parameters are:

- **val** (*type:* string list) Values of field components (time dependant functions).

---

champ_fonc_tabule

Inherits from: *field_base*

Field that is tabulated as a function of another field.

Parameters are:

- **pb_field** (*type: bloc_lecture*) block similar to { pb1 field1 } or { pb1 field1 ... pbN fieldN }
- **dim** (*type:* int) Number of field components.
- **bloc** (*type: bloc_lecture*) Values (the table (the value of the field at any time is calculated by linear interpolation from this table) or the analytical expression (with keyword expression to use an analytical expression)).

---

champ_fonc_tabule_morceaux

Synonyms: champ_tabule_morceaux

Inherits from: *field_base*

Field defined by tabulated data in each sub-domaine. It makes possible the definition of a field which is a function of other fields.

Parameters are:

- **domain_name** (*type: domaine*) Name of the domain.
- **nb_comp** (*type:* int) Number of field components.
- **data** (*type: bloc_lecture*) { Defaut val_def sous_domaine_1 val_1 ... sous_domaine_i val_i } By default, the value val_def is assigned to the field. It takes the sous_domaine_i identifier Sous_Domaine (sub_area) type object function, val_i. Sous_Domaine (sub_area) type objects must have been previously defined if the operator wishes to use a champ_fonc_tabule_morceaux type object.

---

champ_fonc_tabule_morceaux_interp

Inherits from: *field_base*

Field defined by tabulated data in each sub-domaine. It makes possible the definition of a field which is a function of other fields. Here we use MEDCoupling to interpolate fields between the two domains.

Parameters are:

- **problem_name** (*type:* *pb_base*) Name of the problem.
- **nb_comp** (*type:* int) Number of field components.
- **data** (*type:* *bloc_lecture*) { Defaut val_def sous_domaine_1 val_1 ... sous_domaine_i val_i } By default, the value val_def is assigned to the field. It takes the sous_domaine_i identifier Sous_Domaine (sub_area) type object function, val_i. Sous_Domaine (sub_area) type objects must have been previously defined if the operator wishes to use a champ_fonc_tabule_morceaux type object.

---

champ_init_canal_sinal

Inherits from: *field_base*

For a parabolic profile on U velocity with an unpredictable disturbance on V and W and a sinusoidal disturbance on V velocity.

Parameters are:

- **dim** (*type:* int) Number of field components.
- **bloc** (*type:* *bloc_lec_champ_init_canal_sinal*) Parameters for the class champ_init_canal_sinal.

---

champ_input_base

Inherits from: *field_base*

not_set

Parameters are:

- **nb_comp** (*type:* int) not_set
- **nom** (*type:* string) not_set
- **[initial_value]** (*type:* list) not_set
- **probleme** (*type:* string) not_set
- **[sous_zone]** (*type:* *sous_zone*) not_set

---

champ_input_p0

Inherits from: *field_base*

not_set

Parameters are:

- **nb_comp** (*type:* int) not_set
- **nom** (*type:* string) not_set
- **[initial_value]** (*type:* list) not_set
- **probleme** (*type:* string) not_set
- **[sous_zone]** (*type:* *sous_zone*) not_set

---

champ_input_p0_composite

Inherits from: *field_base*

Field used to define a classical champ input p0 field (for ICoCo), but with a predefined field for the initial state.

Parameters are:

- **[initial_field]** (*type:* *field_base*) The field used for initialization
- **[input_field]** (*type:* *champ_input_p0*) The input field for ICoCo
- **nb_comp** (*type:* int) not_set
- **nom** (*type:* string) not_set
- **[initial_value]** (*type:* list) not_set
- **probleme** (*type:* string) not_set
- **[sous_zone]** (*type:* *sous_zone*) not_set

---

champ_musig

Inherits from: *field_base*

MUSIG field. Used in multiphase problems to associate data to each phase.

Parameters are:

- **bloc** (*type:* *bloc_lecture*) Not set

---

champ_ostwald

Inherits from: *field_base*

This keyword is used to define the viscosity variation law:

Mu(T)= K(T)*(D:D/2)**((n-1)/2)

---

champ_parametrique

Inherits from: *field_base*

Parametric field

Parameters are:

- **fichier** (*type:* string) Filename where fields are read

---

champ_som_lu_vdf

Inherits from: *field_base*

Keyword to read in a file values located at the nodes of a mesh in VDF discretization.

Parameters are:

- **domain_name** (*type:* *domaine*) Name of the domain.
- **dim** (*type:* int) Value of the dimension of the field.
- **tolerance** (*type:* double) Value of the tolerance to check the coordinates of the nodes.
- **file** (*type:* string) name of the file This file has the following format: Xi Yi Zi -> Coordinates of the node Ui Vi Wi -> Value of the field on this node Xi+1 Yi+1 Zi+1 -> Next point Ui+1 Vi+1 Zi+1 -> Next value ...

---

champ_som_lu_vef

Inherits from: *field_base*

Keyword to read in a file values located at the nodes of a mesh in VEF discretization.

Parameters are:

- **domain_name** (*type:* *domaine*) Name of the domain.
- **dim** (*type:* int) Value of the dimension of the field.
- **tolerance** (*type:* double) Value of the tolerance to check the coordinates of the nodes.
- **file** (*type:* string) Name of the file. This file has the following format: Xi Yi Zi -> Coordinates of the node Ui Vi Wi -> Value of the field on this node Xi+1 Yi+1 Zi+1 -> Next point Ui+1 Vi+1 Zi+1 -> Next value ...

---

champ_tabule_temps

Inherits from: *field_base*

Field that is constant in space and tabulated as a function of time.

Parameters are:

- **dim** (*type:* int) Number of field components.
- **bloc** (*type: bloc_lecture*) Values as a table. The value of the field at any time is calculated by linear interpolation from this table.

---

champ_uniforme_morceaux

Inherits from: *field_base*

Field which is partly constant in space and stationary.

Parameters are:

- **nom_dom** (*type: domaine*) Name of the domain to which the sub-areas belong.
- **nb_comp** (*type:* int) Number of field components.
- **data** (*type: bloc_lecture*) { Defaut val_def sous_zone_1 val_1 ... sous_zone_i val_i } By default, the value val_def is assigned to the field. It takes the sous_zone_i identifier Sous_Zone (sub_area) type object value, val_i. Sous_Zone (sub_area) type objects must have been previously defined if the operator wishes to use a Champ_Uniforme_Morceaux(partly_uniform_field) type object.

---

champ_uniforme_morceaux_tabule_temps

Inherits from: *field_base*

this type of field is constant in space on one or several sub_zones and tabulated as a function of time.

Parameters are:

- **nom_dom** (*type: domaine*) Name of the domain to which the sub-areas belong.
- **nb_comp** (*type:* int) Number of field components.
- **data** (*type: bloc_lecture*) { Defaut val_def sous_zone_1 val_1 ... sous_zone_i val_i } By default, the value val_def is assigned to the field. It takes the sous_zone_i identifier Sous_Zone (sub_area) type object value, val_i. Sous_Zone (sub_area) type objects must have been previously defined if the operator wishes to use a Champ_Uniforme_Morceaux(partly_uniform_field) type object.

---

field_base

Synonyms: champ_base

Basic class of fields.

---

field_func_txyz

Synonyms: champ_fonc_txyz

Inherits from: *field_base*

Field defined by analytical functions. It makes it possible the definition of a field that depends on the time and the space.

Parameters are:

- **dom** (*type:* *domaine*) Name of domain of calculation.
- **val** (*type:* string list) List of functions on (t,x,y,z).

---

field_func_xyz

Synonyms: champ_fonc_xyz

Inherits from: *field_base*

Field defined by analytical functions. It makes it possible the definition of a field that depends on (x,y,z).

Parameters are:

- **dom** (*type:* *domaine*) Name of domain of calculation.
- **val** (*type:* string list) List of functions on (x,y,z).

---

init_par_partie

Inherits from: *field_base*

ne marche que pour n_comp=1

Parameters are:

- **n_comp** (*type:* int into [1]) not_set
- **val1** (*type:* double) not_set
- **val2** (*type:* double) not_set
- **val3** (*type:* double) not_set

---

tayl_green

Inherits from: *field_base*

Class Tayl_green.

Parameters are:

- **dim** (*type:* int) Dimension.

---

uniform_field

Synonyms: champ_uniforme

Inherits from: *field_base*

Field that is constant in space and stationary.

Parameters are:

- **val** (*type:* list) Values of field components.

---

valeur_totale_sur_volume

Inherits from: *field_base*

Similar as Champ_Uniforme_Morceaux with the same syntax. Used for source terms when we want to specify a source term with a value given for the volume (eg: heat in Watts) and not a value per volume unit (eg: heat in Watts/m3).

Parameters are:

- **nom_dom** (*type:* *domaine*) Name of the domain to which the sub-areas belong.
- **nb_comp** (*type:* int) Number of field components.
- **data** (*type:* *bloc_lecture*) { Defaut val_def sous_zone_1 val_1 ... sous_zone_i val_i } By default, the value val_def is assigned to the field. It takes the sous_zone_i identifier Sous_Zone (sub_area) type object value, val_i. Sous_Zone (sub_area) type objects must have been previously defined if the operator wishes to use a Champ_Uniforme_Morceaux(partly_uniform_field) type object.

2.1.15 Keywords derived from front_field_base

boundary_field_inward

Inherits from: *front_field_base*

this field is used to define the normal vector field standard at the boundary in VDF or VEF discretization.

Parameters are:

- **normal_value** (*type:* string) normal vector value (positive value for a vector oriented outside to inside) which can depend of the time.

---

ch_front_input

Inherits from: *front_field_base*

not_set

Parameters are:

- **nb_comp** (*type:* int) not_set
- **nom** (*type:* string) not_set
- **[initial_value]** (*type:* list) not_set
- **probleme** (*type:* string) not_set
- **[sous_zone]** (*type: sous_zone*) not_set

---

ch_front_input_uniforme

Inherits from: *front_field_base*

for coupling, you can use ch_front_input_uniforme which is a champ_front_uniforme, which use an external value. It must be used with Problem.setInputField.

Parameters are:

- **nb_comp** (*type:* int) not_set
- **nom** (*type:* string) not_set
- **[initial_value]** (*type:* list) not_set
- **probleme** (*type:* string) not_set
- **[sous_zone]** (*type: sous_zone*) not_set

---

champ_front_bruite

Inherits from: *front_field_base*

Field which is variable in time and space in a random manner.

Parameters are:

- **nb_comp** (*type:* int) Number of field components.
- **bloc** (*type: bloc_lecture*) { [N val L val] Moyenne m_1.....[m_i] Amplitude A_1.....[A_ i]}: Random nois: If N and L are not defined, the ith component of the field varies randomly around an average value m_i with a maximum amplitude A_i. White noise: If N and L are defined, these two additional parameters correspond to L, the domain length and N, the number of nodes in the domain. Noise frequency will be between 2*Pi/L and 2*Pi*N/(4*L). For example, formula for velocity: u=U0(t) v=U1(t)Uj(t)=Mj+2*Aj*bruit_blanc where bruit_blanc (white_noise) is the formula given in the mettre_a_jour (update) method of the Champ_front_bruite (noise_boundary_field) (Refer to the Champ_front_bruite.cpp file).

---

champ_front_calc

Inherits from: *front_field_base*

This keyword is used on a boundary to get a field from another boundary. The local and remote boundaries should have the same mesh. If not, the Champ_front_recyclage keyword could be used instead. It is used in the condition block at the limits of equation which itself refers to a problem called pb1. We are working under the supposition that pb1 is coupled to another problem.

Parameters are:

- **problem_name** (*type: pb_base*) Name of the other problem to which pb1 is coupled.
- **bord** (*type:* string) Name of the side which is the boundary between the 2 domains in the domain object description associated with the problem_name object.
- **field_name** (*type:* string) Name of the field containing the value that the user wishes to use at the boundary. The field_name object must be recognized by the problem_name object.

---

champ_front_composite

Inherits from: *front_field_base*

Composite front field. Used in multiphase problems to associate data to each phase.

Parameters are:

- **dim** (*type:* int) Number of field components.
- **bloc** (*type: bloc_lecture*) Values Various pieces of the field, defined per phase. Part 1 goes to phase 1, etc...

---

champ_front_contact_vef

Inherits from: *front_field_base*

This field is used on a boundary between a solid and fluid domain to exchange a calculated temperature at the contact face of the two domains according to the flux of the two problems.

Parameters are:

- **local_pb** (*type: pb_base*) Name of the problem.
- **local_boundary** (*type:* string) Name of the boundary.
- **remote_pb** (*type: pb_base*) Name of the second problem.
- **remote_boundary** (*type:* string) Name of the boundary in the second problem.

---

champ_front_debit

Inherits from: *front_field_base*

This field is used to define a flow rate field instead of a velocity field for a Dirichlet boundary condition on Navier-Stokes equations.

Parameters are:

- **ch** (*type:* *front_field_base*) uniform field in space to define the flow rate. It could be, for example, champ_front_uniforme, ch_front_input_uniform or champ_front_fonc_txyz that depends only on time.

---

champ_front_debit_massique

Inherits from: *front_field_base*

This field is used to define a flow rate field using the density

Parameters are:

- **ch** (*type:* *front_field_base*) uniform field in space to define the flow rate. It could be, for example, champ_front_uniforme, ch_front_input_uniform or champ_front_fonc_txyz that depends only on time.

---

champ_front_debit_qc_vdf

Inherits from: *front_field_base*

This keyword is used to define a flow rate field for quasi-compressible fluids in VDF discretization. The flow rate is kept constant during a transient.

Parameters are:

- **dimension | dim** (*type:* int) Problem dimension
- **liste** (*type:* *bloc_lecture*) List of the mass flow rate values [kg/s/m2] with the following syntaxe: { val1 ... valdim }
- **[moyen]** (*type:* string) Option to use rho mean value
- **pb_name** (*type:* string) Problem name

---

champ_front_debit_qc_vdf_fonc_t

Inherits from: *front_field_base*

This keyword is used to define a flow rate field for quasi-compressible fluids in VDF discretization. The flow rate could be constant or time-dependent.

Parameters are:

- **dimension | dim** (*type:* int) Problem dimension
- **liste** (*type:* *bloc_lecture*) List of the mass flow rate values [kg/s/m2] with the following syntaxe: { val1 ... valdim } where val1 ... valdim are constant or function of time.

- **[moyen]** (*type:* string) Option to use rho mean value
- **pb_name** (*type:* string) Problem name

---

champ_front_fonc_pois_ipsn

Inherits from: *front_field_base*

Boundary field champ_front_fonc_pois_ipsn.

Parameters are:

- **r_tube** (*type:* double) not_set
- **umoy** (*type:* list) not_set
- **r_loc** (*type:* double list) not_set

---

champ_front_fonc_pois_tube

Inherits from: *front_field_base*

Boundary field champ_front_fonc_pois_tube.

Parameters are:

- **r_tube** (*type:* double) not_set
- **umoy** (*type:* list) not_set
- **r_loc** (*type:* double list) not_set
- **r_loc_mult** (*type:* int list - size is dimension) not_set

---

champ_front_fonc_t

Inherits from: *front_field_base*

Boundary field that depends only on time.

Parameters are:

- **val** (*type:* string list) Values of field components (mathematical expressions).

---

champ_front_fonc_txyz

Inherits from: *front_field_base*

Boundary field which is not constant in space and in time.

Parameters are:

- **val** (*type:* string list) Values of field components (mathematical expressions).

---

champ_front_fonc_xyz

Inherits from: *front_field_base*

Boundary field which is not constant in space.

Parameters are:

- **val** (*type:* string list) Values of field components (mathematical expressions).

---

champ_front_fonction

Inherits from: *front_field_base*

boundary field that is function of another field

Parameters are:

- **dim** (*type:* int) Number of field components.
- **inco** (*type:* string) Name of the field (for example: temperature).
- **expression** (*type:* string) keyword to use a analytical expression like 10.*EXP(-0.1*val) where val be the keyword for the field.

---

champ_front_lu

Inherits from: *front_field_base*

boundary field which is given from data issued from a read file. The format of this file has to be the same that the one generated by Ecrire_fichier_xyz_valeur

Example for K and epsilon quantities to be defined for inlet condition in a boundary named 'entree': NL2entree frontiere_ouverte_K_Eps_impose Champ_Front_lu dom 2pb_K_EPS_PERIO_1006.306198.dat

Parameters are:

- **domaine | domain** (*type:* *domaine*) Name of domain
- **dim** (*type:* int) number of components
- **file** (*type:* string) path for the read file

---

champ_front_med

Inherits from: *front_field_base*

Field allowing the loading of a boundary condition from a MED file using Champ_fonc_med

Parameters are:

- **champ_fonc_med** (*type:* *field_base*) a champ_fonc_med loading the values of the unknown on a domain boundary

---

champ_front_musig

Inherits from: *front_field_base*

MUSIG front field. Used in multiphase problems to associate data to each phase.

Parameters are:

- **bloc** (*type:* *bloc_lecture*) Not set

---

champ_front_normal_vef

Inherits from: *front_field_base*

Field to define the normal vector field standard at the boundary in VEF discretization.

Parameters are:

- **mot** (*type:* string into ["valeur_normale"]) Name of vector field.
- **vit_tan** (*type:* double) normal vector value (positive value for a vector oriented outside to inside).

---

champ_front_parametrique

Inherits from: *front_field_base*

Parametric boundary field

Parameters are:

- **fichier** (*type:* string) Filename where boundary fields are read

---

champ_front_pression_from_u

Inherits from: *front_field_base*

this field is used to define a pressure field depending of a velocity field.

Parameters are:

- **expression** (*type:* string) value depending of a velocity (like $2*u_moy^2$).

---

champ_front_recyclage

Inherits from: *front_field_base*

Parameters are:

- **bloc** (*type:* string) not_set

---

champ_front_tabule

Inherits from: *front_field_base*

Constant field on the boundary, tabulated as a function of time.

Parameters are:

- **nb_comp** (*type:* int) Number of field components.
- **bloc** (*type:* *bloc_lecture*) {nt1 t2 t3 ....tn u1 [v1 w1 ...] u2 [v2 w2 ...] u3 [v3 w3 ...] ... un [vn wn ...] } Values are entered into a table based on n couples (ti, ui) if nb_comp value is 1. The value of a field at a given time is calculated by linear interpolation from this table.

---

champ_front_tabule_lu

Inherits from: *front_field_base*

Constant field on the boundary, tabulated from a specified column file. Lines starting with # are ignored.

Parameters are:

- **nb_comp** (*type:* int) Number of field components.
- **column_file** (*type:* string) Name of the column file.

---

champ_front_tangentiel_vef

Inherits from: *front_field_base*

Field to define the tangential velocity vector field standard at the boundary in VEF discretization.

Parameters are:

- **mot** (*type:* string into ["vitesse_tangentielle"]) Name of vector field.
- **vit_tan** (*type:* double) Vector field standard [m/s].

---

champ_front_uniforme

Inherits from: *front_field_base*

Boundary field which is constant in space and stationary.

Parameters are:

- **val** (*type:* list) Values of field components.

---

champ_front_xyz_debit

Inherits from: *front_field_base*

This field is used to define a flow rate field with a velocity profil which will be normalized to match the flow rate chosen.

Parameters are:

- **[velocity_profil]** (*type: front_field_base*) velocity_profil 0 velocity field to define the profil of velocity.
- **flow_rate** (*type: front_field_base*) flow_rate 1 uniform field in space to define the flow rate. It could be, for example, champ_front_uniforme, ch_front_input_uniform or champ_front_fonc_t

---

champ_front_xyz_tabule

Inherits from: *front_field_base*

Space dependent field on the boundary, tabulated as a function of time.

Parameters are:

- **val** (*type:* string list) Values of field components (mathematical expressions).
- **bloc** (*type: bloc_lecture*) {nt1 t2 t3 ....tn u1 [v1 w1 ...] u2 [v2 w2 ...] u3 [v3 w3 ...] ... un [vn wn ...] } Values are entered into a table based on n couples (ti, ui) if nb_comp value is 1. The value of a field at a given time is calculated by linear interpolation from this table.

---

front_field_base

Synonyms: champ_front_base

Basic class for fields at domain boundaries.

2.1.16 Keywords derived from ijk_splitting

ijk_splitting

Object to specify how the domain will be divided between processors in IJK discretization

Parameters are:

- **ijk_grid_geometry** (*type:* *ijk_grid_geometry*) the grid that will be splitted
- **nproc_i** (*type:* int) the number of processors into which we will divide the grid following the I direction
- **nproc_j** (*type:* int) the number of processors into which we will divide the grid following the J direction
- **nproc_k** (*type:* int) the number of processors into which we will divide the grid following the K direction

---

2.1.17 Keywords derived from interface_base

interface_base

Basic class for a liquid-gas interface (used in pb_multiphase)

Parameters are:

- **[tension_superficielle | surface_tension]** (*type:* double) surface tension

---

interface_sigma_constant

Inherits from: *interface_base*

Liquid-gas interface with a constant surface tension sigma

Parameters are:

- **[tension_superficielle | surface_tension]** (*type:* double) surface tension

---

saturation_base

Inherits from: *interface_base*

fluide-gas interface with phase change (used in pb_multiphase)

Parameters are:

- **[p_ref]** (*type:* double) not_set
- **[t_ref]** (*type:* double) not_set
- **[tension_superficielle | surface_tension]** (*type:* double) surface tension

---

saturation_constant

Inherits from: *interface_base*

Class for saturation constant

Parameters are:

- **[p_sat]** (*type:* double) Define the saturation pressure value (this is a required parameter)
- **[t_sat]** (*type:* double) Define the saturation temperature value (this is a required parameter)
- **[lvap]** (*type:* double) Latent heat of vaporization
- **[hlsat]** (*type:* double) Liquid saturation enthalpy
- **[hvsat]** (*type:* double) Vapor saturation enthalpy
- **[p_ref]** (*type:* double) not_set
- **[t_ref]** (*type:* double) not_set
- **[tension_superficielle | surface_tension]** (*type:* double) surface tension

---

saturation_sodium

Inherits from: *interface_base*

Class for saturation sodium

Parameters are:

- **[p_ref]** (*type:* double) Use to fix the pressure value in the closure law. If not specified, the value of the pressure unknown will be used
- **[t_ref]** (*type:* double) Use to fix the temperature value in the closure law. If not specified, the value of the temperature unknown will be used
- **[tension_superficielle | surface_tension]** (*type:* double) surface tension

---

2.1.18 Keywords derived from interpolation_ibm_base

interpolation_ibm_aucune

Synonyms: ibm_aucune

Inherits from: *interpolation_ibm_base*

Immersed Boundary Method (IBM): no interpolation.

Parameters are:

- **[impr]** (*type:* flag) To print IBM-related data
- **[nb_histo_boxes_impr]** (*type:* int) number of histogram boxes for printed data

---

interpolation_ibm_base

Base class for all the interpolation methods available in the Immersed Boundary Method (IBM).

Parameters are:

- **[impr]** (*type:* flag) To print IBM-related data
- **[nb_histo_boxes_impr]** (*type:* int) number of histogram boxes for printed data

---

interpolation_ibm_elem_fluid

Synonyms: ibm_element_fluide, interpolation_ibm_element_fluide

Inherits from: *interpolation_ibm_base*

Immersed Boundary Method (IBM): fluid element interpolation.

Parameters are:

- **points_fluides** (*type: field_base*) Node field giving the projection of the point below (points_solides) falling into the pure cell fluid
- **points_solides** (*type: field_base*) Node field giving the projection of the node on the immersed boundary
- **elements_fluides** (*type: field_base*) Node field giving the number of the element (cell) containing the pure fluid point
- **correspondance_elements** (*type: field_base*) Cell field giving the SALOME cell number
- **[impr]** (*type:* flag) To print IBM-related data
- **[nb_histo_boxes_impr]** (*type:* int) number of histogram boxes for printed data

---

interpolation_ibm_hybride

Synonyms: ibm_hybride

Inherits from: *interpolation_ibm_base*

Immersed Boundary Method (IBM): hybrid (fluid/mean gradient) interpolation.

Parameters are:

- **est_dirichlet** (*type: field_base*) Node field of booleans indicating whether the node belong to an element where the interface is
- **elements_solides** (*type: field_base*) Node field giving the element number containing the solid point
- **points_fluides** (*type: field_base*) Node field giving the projection of the point below (points_solides) falling into the pure cell fluid
- **points_solides** (*type: field_base*) Node field giving the projection of the node on the immersed boundary
- **elements_fluides** (*type: field_base*) Node field giving the number of the element (cell) containing the pure fluid point
- **correspondance_elements** (*type: field_base*) Cell field giving the SALOME cell number
- **[impr]** (*type:* flag) To print IBM-related data
- **[nb_histo_boxes_impr]** (*type:* int) number of histogram boxes for printed data

---

interpolation_ibm_mean_gradient

Synonyms: interpolation_ibm_gradient_moyen, ibm_gradient_moyen

Inherits from: *interpolation_ibm_base*

Immersed Boundary Method (IBM): mean gradient interpolation.

Parameters are:

- **points_solides** (*type: field_base*) Node field giving the projection of the node on the immersed boundary
- **est_dirichlet** (*type: field_base*) Node field of booleans indicating whether the node belong to an element where the interface is
- **correspondance_elements** (*type: field_base*) Cell field giving the SALOME cell number
- **elements_solides** (*type: field_base*) Node field giving the element number containing the solid point
- **[impr]** (*type:* flag) To print IBM-related data
- **[nb_histo_boxes_impr]** (*type:* int) number of histogram boxes for printed data

---

interpolation_ibm_power_law_tbl

Synonyms: ibm_power_law_tbl

Inherits from: *interpolation_ibm_base*

Immersed Boundary Method (IBM): power law interpolation.

Parameters are:

- **[formulation_linear_pwl]** (*type:* int) Choix formulation lineaire ou non
- **points_fluides** (*type: field_base*) Node field giving the projection of the point below (points_solides) falling into the pure cell fluid
- **points_solides** (*type: field_base*) Node field giving the projection of the node on the immersed boundary
- **elements_fluides** (*type: field_base*) Node field giving the number of the element (cell) containing the pure fluid point
- **correspondance_elements** (*type: field_base*) Cell field giving the SALOME cell number
- **[impr]** (*type:* flag) To print IBM-related data
- **[nb_histo_boxes_impr]** (*type:* int) number of histogram boxes for printed data

---

interpolation_ibm_power_law_tbl_u_star

Synonyms: ibm_power_law_tbl_u_star

Inherits from: *interpolation_ibm_base*

Immersed Boundary Method (IBM): law u star.

Parameters are:

- **points_solides** (*type: field_base*) Node field giving the projection of the node on the immersed boundary
- **est_dirichlet** (*type: field_base*) Node field of booleans indicating whether the node belong to an element where the interface is
- **correspondance_elements** (*type: field_base*) Cell field giving the SALOME cell number
- **elements_solides** (*type: field_base*) Node field giving the element number containing the solid point
- **[impr]** (*type:* flag) To print IBM-related data
- **[nb_histo_boxes_impr]** (*type:* int) number of histogram boxes for printed data

---

2.1.19 Keywords derived from interprete

interprete

Class ‘interprete_Tru’ has nothing special in itself, except that it needs to be known early, so that the Dataset_Tru class can test whether a given class is a child of ‘interprete_Tru’ So force its definition here, and hence avoid its automatic generation from the TRAD2 file.

---

analyse_angle

Inherits from: *interprete*

Keyword Analyse_angle prints the histogram of the largest angle of each mesh elements of the domain named name_domain. nb_histo is the histogram number of bins. It is called by default during the domain discretization with nb_histo set to 18. Useful to check the number of elements with angles above 90 degrees.

Parameters are:

- **domain_name** (*type: domaine*) Name of domain to resequence.
- **nb_histo** (*type:* int) not_set

---

associate

Synonyms: associer

Inherits from: *interprete*

This interpretor allows one object to be associated with another. The order of the two objects in this instruction is not important. The object objet_2 is associated to objet_1 if this makes sense; if not either objet_1 is associated to objet_2 or the program exits with error because it cannot execute the Associate (Associer) instruction. For example, to calculate water flow in a pipe, a Pb_Hydraulique type object needs to be defined. But also a Domaine type object to represent the pipe, a Scheme_euler_explicit type object for time discretization, a discretization type object (VDF or VEF) and a Fluide_Incompressible type object which will contain the water properties. These objects must then all be associated with the problem.

Parameters are:

- **objet_1** (*type:* string) Objet_1
- **objet_2** (*type:* string) Objet_2

---

axi

Inherits from: *interprete*

This keyword allows a 3D calculation to be executed using cylindrical coordinates (R,$jolitheta$,Z). If this instruction is not included, calculations are carried out using Cartesian coordinates.

---

bidim_axi

Inherits from: *interprete*

Keyword allowing a 2D calculation to be executed using axisymetric coordinates (R, Z). If this instruction is not included, calculations are carried out using Cartesian coordinates.

---

calculer_moments

Inherits from: *interprete*

Calculates and prints the torque (moment of force) exerted by the fluid on each boundary in output files (.out) of the domain nom_dom.

Parameters are:

- **nom_dom** (*type:* *domaine*) Name of domain.
- **mot** (*type:* *lecture_bloc_moment_base*) Keyword.

---

corriger_frontiere_periodique

Inherits from: *interprete*

The Corriger_frontiere_periodique keyword is mandatory to first define the periodic boundaries, to reorder the faces and eventually fix unaligned nodes of these boundaries. Faces on one side of the periodic domain are put first, then the faces on the opposite side, in the same order. It must be run in sequential before mesh splitting.

Parameters are:

- **domaine** (*type:* string) Name of domain.
- **bord** (*type:* string) the name of the boundary (which must contain two opposite sides of the domain)
- **[direction]** (*type:* list) defines the periodicity direction vector (a vector that points from one node on one side to the opposite node on the other side). This vector must be given if the automatic algorithm fails, that is:NL2 - when the node coordinates are not perfectly periodic - when the periodic direction is not aligned with the normal vector of the boundary faces
- **[fichier_post]** (*type:* string) .

---

create_domain_from_sous_zone

Synonyms: create_domain_from_sub_domain, create_domain_from_sub_domains

Inherits from: *interprete*

kept for backward compatibility. please use Create_domain_from_sub_domain

Parameters are:

- **[domaine_final]** (*type:* *domaine*) new domain in which faces are stored
- **[par_sous_zone]** (*type:* string) a sub-area allowing to choose the elements
- **domaine_init** (*type:* *domaine*) initial domain

---

create_domain_from_sub_domain

Synonyms: create_domain_from_sub_domains, create_domain_from_sous_zone

Inherits from: *interprete*

This keyword fills the domain domaine_final with the subdomaine par_sous_zone from the domain domaine_init. It is very useful when meshing several mediums with Gmsh. Each medium will be defined as a subdomaine into Gmsh. A MED mesh file will be saved from Gmsh and read with Lire_Med keyword by the TRUST data file. And with this keyword, a domain will be created for each medium in the TRUST data file.

Parameters are:

- **[domaine_final]** (*type:* *domaine*) new domain in which faces are stored
- **[par_sous_zone]** (*type:* string) a sub-area allowing to choose the elements
- **domaine_init** (*type:* *domaine*) initial domain

---

criteres_convergence

Inherits from: *interprete*

convergence criteria

Parameters are:

- **aco** (*type:* string into ["{"]) Opening curly bracket.
- **[inco]** (*type:* string) Unknown (i.e: alpha, temperature, velocity and pressure)
- **[val]** (*type:* double) Convergence threshold
- **acof** (*type:* string into ["}"]) Closing curly bracket.

---

debog

Inherits from: *interprete*

Class to debug some differences between two TRUST versions on a same data file.

If you want to compare the results of the same code in sequential and parallel calculation, first run (mode=0) in sequential mode (the files fichier1 and fichier2 will be written first) then the second run in parallel calculation (mode=1).

During the first run (mode=0), it prints into the file DEBOG, values at different points of the code thanks to the C++ instruction call. see for example in Kernel/Framework/Resoudre.cpp file the instruction: Debog::verifier(msg,value); Where msg is a string and value may be a double, an integer or an array.

During the second run (mode=1), it prints into a file Err_Debog.dbg the same messages than in the DEBOG file and checks if the differences between results from both codes are less than a given value (error). If not, it prints Ok else show the differences and the lines where it occured.

Parameters are:

- **pb** (*type:* *pb_gen_base*) Name of the problem to debug.
- **fichier1 | file1** (*type:* string) Name of the file where domain will be written in sequential calculation.
- **fichier2 | file2** (*type:* string) Name of the file where faces will be written in sequential calculation.

- **seuil** (*type:* double) Minimal value (by default 1.e-20) for the differences between the two codes.
- **mode** (*type:* int) By default -1 (nothing is written in the different files), you will set 0 for the sequential run, and 1 for the parallel run.

---

debut_bloc

Synonyms: {

Inherits from: *interprete*

Block's beginning.

---

decoupebord_pour_rayonnement

Synonyms: decoupebord

Inherits from: *interprete*

To subdivide the external boundary of a domain into several parts (may be useful for better accuracy when using radiation model in transparent medium). To specify the boundaries of the fine_domain_name domain to be splitted. These boundaries will be cut according the coarse mesh defined by either the keyword domaine_grossier (each boundary face of the coarse mesh coarse_domain_name will be used to group boundary faces of the fine mesh to define a new boundary), either by the keyword nb_parts_naif (each boundary of the fine mesh is splitted into a partition with nx*ny*nz elements), either by a geometric condition given by a formulae with the keyword condition_geometrique. If used, the coarse_domain_name domain should have the same boundaries name of the fine_domain_name domain.

A mesh file (ASCII format, except if binaire option is specified) named by default newgeom (or specified by the nom_fichier_sortie keyword) will be created and will contain the fine_domain_name domain with the splitted boundaries named boundary_name%I (where I is between from 0 and n-1). Furthermore, several files named boundary_name%I and boundary_name_xv will be created, containing the definition of the subdived boundaries. newgeom will be used to calculate view factors with geom2ansys script whereas only the boundary_name_xv files will be necessary for the radiation calculation. The file listb will contain the list of the boundaries boundary_name%I.

Parameters are:

- **domaine** (*type: domaine*) not_set
- **[domaine_grossier]** (*type: domaine*) not_set
- **[nb_parts_naif]** (*type:* int list) not_set
- **[nb_parts_geom]** (*type:* int list) not_set
- **[condition_geometrique]** (*type:* string list) not_set
- **bords_a_decouper** (*type:* string list) not_set
- **[nom_fichier_sortie]** (*type:* string) not_set
- **[binaire]** (*type:* int) not_set

---

decouper_bord_coincident

Inherits from: *interprete*

In case of non-coincident meshes and a paroi_contact condition, run is stopped and two external files are automatically generated in VEF (connectivity_failed_boundary_name and connectivity_failed_pb_name.med). In 2D, the keyword Decouper_bord_coincident associated to the connectivity_failed_boundary_name file allows to generate a new coincident mesh.

Parameters are:

- **domain_name** (*type:* *domaine*) Name of domain.
- **bord** (*type:* string) connectivity_failed_boundary_name

---

dilate

Inherits from: *interprete*

Keyword to multiply the whole coordinates of the geometry.

Parameters are:

- **domain_name** (*type:* *domaine*) Name of domain.
- **alpha** (*type:* double) Value of dilatation coefficient.

---

dimension

Inherits from: *interprete*

Keyword allowing calculation dimensions to be set (2D or 3D), where dim is an integer set to 2 or 3. This instruction is mandatory.

Parameters are:

- **dim** (*type:* int into [2,3]) Number of dimensions.

---

disable_tu

Inherits from: *interprete*

Flag to disable the writing of the .TU files

---

discretiser_domaine

Inherits from: *interprete*

Useful to discretize the domain domain_name (faces will be created) without defining a problem.

Parameters are:

- **domain_name** (*type: domaine*) Name of the domain.

---

discretize

Synonyms: discretiser

Inherits from: *interprete*

Keyword to discretise a problem problem_name according to the discretization dis.

IMPORTANT: A number of objects must be already associated (a domain, time scheme, central object) prior to invoking the Discretize (Discretiser) keyword. The physical properties of this central object must also have been read.

Parameters are:

- **problem_name** (*type: pb_gen_base*) Name of problem.
- **dis** (*type: discretisation_base*) Name of the discretization object.

---

distance_paroi

Inherits from: *interprete*

Class to generate external file Wall_length.xyz devoted for instance, for mixing length modelling. In this file, are saved the coordinates of each element (center of gravity) of dom domain and minimum distance between this point and boundaries (specified bords) that user specifies in data file (typically, those associated to walls). A field Distance_paroi is available to post process the distance to the wall.

Parameters are:

- **dom** (*type: domaine*) Name of domain.
- **bords** (*type:* string list) Boundaries.
- **format** (*type:* string into ["binaire", "formatte"]) Value for format may be binaire (a binary file Wall_length.xyz is written) or formatte (moreover, a formatted file Wall_length_formatted.xyz is written).

---

ecrire_champ_med

Inherits from: *interprete*

Keyword to write a field to MED format into a file.

Parameters are:

- **nom_dom** (*type: domaine*) domain name

- **nom_chp** (*type: field_base*) field name
- **file** (*type:* string) file name

---

ecrire_fichier_formatte

Inherits from: *interprete*

Keyword to write the object of name name_obj to a file filename in ASCII format.

Parameters are:

- **name_obj** (*type:* string) Name of the object to be written.
- **filename** (*type:* string) Name of the file.

---

ecrire_fichier_xyz_valeur

Inherits from: *interprete*

This keyword is used to write the values of a field only for some boundaries in a text file with the following format: n_valeur

x_1 y_1 [z_1] val_1

...

x_n y_n [z_n] val_n

The created files are named : pbname_fieldname_[boundaryname]_time.dat

Parameters are:

- **[binary_file]** (*type:* flag) To write file in binary format
- **[dt]** (*type:* double) File writing frequency
- **[fields]** (*type:* string list) Names of the fields we want to write
- **[boundaries]** (*type:* string list) Names of the boundaries on which to write fields

---

ecrire_med

Synonyms: write_med

Inherits from: *interprete*

Write a domain to MED format into a file.

Parameters are:

- **nom_dom** (*type: domaine*) Name of domain.
- **file** (*type:* string) Name of file.

---

ecriturelecturespecial

Inherits from: *interprete*

Class to write or not to write a .xyz file on the disk at the end of the calculation.

Parameters are:

- **type** (*type:* string) If set to 0, no xyz file is created. If set to EFichierBin, it uses prior 1.7.0 way of reading xyz files (now LecFicDiffuseBin). If set to EcrFicPartageBin, it uses prior 1.7.0 way of writing xyz files (now EcrFicPartageMPIIO).

---

espece

Inherits from: *interprete*

not_set

Parameters are:

- **mu** (*type:* *field_base*) Species dynamic viscosity value (kg.m-1.s-1).
- **cp** (*type:* *field_base*) Species specific heat value (J.kg-1.K-1).
- **masse_molaire** (*type:* double) Species molar mass.

---

execute_parallel

Inherits from: *interprete*

This keyword allows to run several computations in parallel on processors allocated to TRUST. The set of processors is split in N subsets and each subset will read and execute a different data file. Error messages usualy written to stderr and stdout are redirected to .log files (journaling must be activated).

Parameters are:

- **liste_cas** (*type:* string list) N datafile1 ... datafileN. datafileX the name of a TRUST data file without the .data extension.
- **[nb_procs]** (*type:* int list) nb_procs is the number of processors needed to run each data file. If not given, TRUST assumes that computations are sequential.

---

export

Inherits from: *interprete*

Class to make the object have a global range, if not its range will apply to the block only (the associated object will be destroyed on exiting the block).

---

extract_2d_from_3d

Inherits from: *interprete*

Keyword to extract a 2D mesh by selecting a boundary of the 3D mesh. To generate a 2D axisymmetric mesh prefer Extract_2Daxi_from_3D keyword.

Parameters are:

- **dom3d** (*type:* *domaine*) Domain name of the 3D mesh
- **bord** (*type:* string) Boundary name. This boundary becomes the new 2D mesh and all the boundaries, in 3D, attached to the selected boundary, give their name to the new boundaries, in 2D.
- **dom2d** (*type:* string) Domain name of the new 2D mesh

---

extract_2daxi_from_3d

Inherits from: *interprete*

Keyword to extract a 2D axisymetric mesh by selecting a boundary of the 3D mesh.

Parameters are:

- **dom3d** (*type:* *domaine*) Domain name of the 3D mesh
- **bord** (*type:* string) Boundary name. This boundary becomes the new 2D mesh and all the boundaries, in 3D, attached to the selected boundary, give their name to the new boundaries, in 2D.
- **dom2d** (*type:* string) Domain name of the new 2D mesh

---

extraire_domaine

Inherits from: *interprete*

Keyword to create a new domain built with the domain elements of the pb_name problem verifying the two conditions given by Condition_elements. The problem pb_name should have been discretized.

Parameters are:

- **domaine** (*type:* *domaine*) Domain in which faces are saved
- **probleme** (*type:* *pb_base*) Problem from which faces should be extracted
- **[condition_elements]** (*type:* string) not_set
- **[sous_domaine | sous_zone]** (*type:* *sous_zone*) not_set

---

extraire_plan

Inherits from: *interprete*

This keyword extracts a plane mesh named domain_name (this domain should have been declared before) from the mesh of the pb_name problem. The plane can be either a triangle (defined by the keywords Origine, Point1, Point2 and Triangle), either a regular quadrangle (with keywords Origine, Point1 and Point2), or either a generalized quadrangle (with keywords Origine, Point1, Point2, Point3). The keyword Epaisseur specifies the thickness of volume around the plane which contains the faces of the extracted mesh. The keyword via_extraire_surface will create a plan and use Extraire_surface algorithm. Inverse_condition_element keyword then will be used in the case where the plane is a boundary not well oriented, and avec_certains_bords_pour_extraire_surface is the option related to the Extraire_surface option named avec_certains_bords.

Parameters are:

- **domaine** (*type: domaine*) domain name
- **probleme** (*type: pb_base*) pb_name
- **origine** (*type:* list) not_set
- **point1** (*type:* list) not_set
- **point2** (*type:* list) not_set
- **[point3]** (*type:* list) not_set
- **[triangle]** (*type:* flag) not_set
- **epaisseur** (*type:* double) thickness
- **[via_extraire_surface]** (*type:* flag) not_set
- **[inverse_condition_element]** (*type:* flag) not_set
- **[avec_certains_bords_pour_extraire_surface]** (*type:* string list) name of boundaries to include when extracting plan

---

extraire_surface

Inherits from: *interprete*

This keyword extracts a surface mesh named domain_name (this domain should have been declared before) from the mesh of the pb_name problem. The surface mesh is defined by one or two conditions. The first condition is about elements with Condition_elements. For example: Condition_elements x*x+y*y+z*z<1

Will define a surface mesh with external faces of the mesh elements inside the sphere of radius 1 located at (0,0,0). The second condition Condition_faces is useful to give a restriction.NL2 By default, the faces from the boundaries are not added to the surface mesh excepted if option avec_les_bords is given (all the boundaries are added), or if the option avec_certains_bords is used to add only some boundaries.

Parameters are:

- **domaine** (*type: domaine*) Domain in which faces are saved
- **probleme** (*type: pb_base*) Problem from which faces should be extracted
- **[condition_elements]** (*type:* string) condition on center of elements
- **[condition_faces]** (*type:* string) not_set
- **[avec_les_bords]** (*type:* flag) not_set

- **[avec_certains_bords]** (*type:* string list) not_set

---

extrudebord

Inherits from: *interprete*

Class to generate an extruded mesh from a boundary of a tetrahedral or an hexahedral mesh.

Warning: If the initial domain is a tetrahedral mesh, the boundary will be moved in the XY plane then extrusion will be applied (you should maybe use the Transformer keyword on the final domain to have the domain you really want). You can use the keyword Postraiter_domaine to generate a lata|med|... file to visualize your initial and final meshes.

This keyword can be used for example to create a periodic box extracted from a boundary of a tetrahedral or a hexaedral mesh. This periodic box may be used then to engender turbulent inlet flow condition for the main domain.NL2 Note that ExtrudeBord in VEF generates 3 or 14 tetrahedra from extruded prisms.

Parameters are:

- **domaine_init** (*type:* *domaine*) Initial domain with hexaedras or tetrahedras.
- **direction** (*type:* double list) Directions for the extrusion.
- **nb_tranches** (*type:* int) Number of elements in the extrusion direction.
- **domaine_final** (*type:* string) Extruded domain.
- **nom_bord** (*type:* string) Name of the boundary of the initial domain where extrusion will be applied.
- **[hexa_old]** (*type:* flag) Old algorithm for boundary extrusion from a hexahedral mesh.
- **[trois_tetra]** (*type:* flag) To extrude in 3 tetrahedras instead of 14 tetrahedras.
- **[vingt_tetra]** (*type:* flag) To extrude in 20 tetrahedras instead of 14 tetrahedras.
- **[sans_passer_par_le2d]** (*type:* int) Only for non-regression

---

extrudeparoi

Inherits from: *interprete*

Keyword dedicated in 3D (VEF) to create prismatic layer at wall. Each prism is cut into 3 tetraedra.

Parameters are:

- **domaine** (*type:* *domaine*) Name of the domain.
- **nom_bord** (*type:* string) Name of the (no-slip) boundary for creation of prismatic layers.
- **[epaisseur]** (*type:* list) n r1 r2 .... rn : (relative or absolute) width for each layer.
- **[critere_absolu]** (*type:* int) relative (0, the default) or absolute (1) width for each layer.
- **[projection_normale_bord]** (*type:* flag) keyword to project layers on the same plane that contiguous boundaries. defaut values are : epaisseur_relative 1 0.5 projection_normale_bord 1

---

extruder

Inherits from: *interprete*

Class to create a 3D tetrahedral/hexahedral mesh (a prism is cut in 14) from a 2D triangular/quadrangular mesh.

Parameters are:

- **domaine | domain_name** (*type:* *domaine*) Name of the domain.
- **direction** (*type:* *troisf*) Direction of the extrude operation.
- **nb_tranches** (*type:* int) Number of elements in the extrusion direction.

---

extruder_en20

Inherits from: *interprete*

It does the same task as Extruder except that a prism is cut into 20 tetraedra instead of 3. The name of the boundaries will be devant (front) and derriere (back). But you can change these names with the keyword RegroupeBord.

Parameters are:

- **domaine | domain_name** (*type:* *domaine*) Name of the domain.
- **[direction]** (*type:* *troisf*) 0 Direction of the extrude operation.
- **nb_tranches** (*type:* int) Number of elements in the extrusion direction.

---

extruder_en3

Inherits from: *interprete*

Class to create a 3D tetrahedral/hexahedral mesh (a prism is cut in 3) from a 2D triangular/quadrangular mesh. The names of the boundaries (by default, devant (front) and derriere (back)) may be edited by the keyword nom_cl_devant and nom_cl_derriere. If NULL is written for nom_cl, then no boundary condition is generated at this place.

Recommendation : to ensure conformity between meshes (in case of fluid/solid coupling) it is recommended to extrude all the domains at the same time.

Parameters are:

- **domaine | domain_name** (*type:* string list) List of the domains
- **[nom_cl_devant]** (*type:* string) New name of the first boundary.
- **[nom_cl_derriere]** (*type:* string) New name of the second boundary.
- **direction** (*type:* *troisf*) Direction of the extrude operation.
- **nb_tranches** (*type:* int) Number of elements in the extrusion direction.

---

facsec

Inherits from: *interprete*

To parameter the safety factor for the time step during the simulation.

Parameters are:

- **[facsec_ini]** (*type:* double) Initial facsec taken into account at the beginning of the simulation.
- **[facsec_max]** (*type:* double) Maximum ratio allowed between time step and stability time returned by CFL condition. The initial ratio given by facsec keyword is changed during the calculation with the implicit scheme but it couldn't be higher than facsec_max value.NL2 Warning: Some implicit schemes do not permit high facsec_max, example Schema_Adams_Moulton_order_3 needs facsec=facsec_max=1. Advice:NL2 The calculation may start with a facsec specified by the user and increased by the algorithm up to the facsec_max limit. But the user can also choose to specify a constant facsec (facsec_max will be set to facsec value then). Faster convergence has been seen and depends on the kind of calculation: NL2-Hydraulic only or thermal hydraulic with forced convection and low coupling between velocity and temperature (Boussinesq value beta low), facsec between 20-30NL2-Thermal hydraulic with forced convection and strong coupling between velocity and temperature (Boussinesq value beta high), facsec between 90-100 NL2-Thermohydralic with natural convection, facsec around 300NL2 -Conduction only, facsec can be set to a very high value (1e8) as if the scheme was unconditionally stableNL2These values can also be used as rule of thumb for initial facsec with a facsec_max limit higher.
- **[rapport_residus]** (*type:* double) Ratio between the residual at time n and the residual at time n+1 above which the facsec is increased by multiplying by sqrt(rapport_residus) (1.2 by default).
- **[nb_ite_sans_accel_max]** (*type:* int) Maximum number of iterations without facsec increases (20000 by default): if facsec does not increase with the previous condition (ration between 2 consecutive residuals too high), we increase it by force after nb_ite_sans_accel_max iterations.

---

fin_bloc

Synonyms: }

Inherits from: *interprete*

Block's end.

---

imprimer_flux

Inherits from: *interprete*

This keyword prints the flux per face at the specified domain boundaries in the data set. The fluxes are written to the .face files at a frequency defined by dt_impr, the evaluation printing frequency (refer to time scheme keywords). By default, fluxes are incorporated onto the edges before being displayed.

Parameters are:

- **domain_name** (*type: domaine*) Name of the domain.
- **noms_bord** (*type: bloc_lecture*) List of boundaries, for ex: { Bord1 Bord2 }

---

imprimer_flux_sum

Inherits from: *interprete*

This keyword prints the sum of the flux per face at the domain boundaries defined by the user in the data set. The fluxes are written into the .out files at a frequency defined by dt_impr, the evaluation printing frequency (refer to time scheme keywords).

Parameters are:

- **domain_name** (*type:* *domaine*) Name of the domain.
- **noms_bord** (*type:* *bloc_lecture*) List of boundaries, for ex: { Bord1 Bord2 }

---

integrer_champ_med

Inherits from: *interprete*

his keyword is used to calculate a flow rate from a velocity MED field read before. The method is either debit_total to calculate the flow rate on the whole surface, either integrale_en_z to calculate flow rates between z=zmin and z=zmax on nb_tranche surfaces. The output file indicates first the flow rate for the whole surface and then lists for each tranche : the height z, the surface average value, the surface area and the flow rate. For the debit_total method, only one tranche is considered.NL2 file :z Sum(u.dS)/Sum(dS) Sum(dS) Sum(u.dS)

Parameters are:

- **champ_med** (*type:* *champ_fonc_med*) not_set
- **methode** (*type:* string into ["integrale_en_z", "debit_total"]) to choose between the integral following z or over the entire height (debit_total corresponds to zmin=-DMAXFLOAT, ZMax=DMAXFLOAT, nb_tranche=1)
- **[zmin]** (*type:* double) not_set
- **[zmax]** (*type:* double) not_set
- **[nb_tranche]** (*type:* int) not_set
- **[fichier_sortie]** (*type:* string) name of the output file, by default: integrale.

---

interprete_geometrique_base

Inherits from: *interprete*

Class for interpreting a data file

---

lata_to_med

Synonyms: lata_2_med

Inherits from: *interprete*

To convert results file written with LATA format to MED file. Warning: Fields located on faces are not supported yet.

Parameters are:

- **[format]** (*type: format_lata_to_med*) generated file post_med.data use format (MED or LATA or LML keyword).
- **file** (*type:* string) LATA file to convert to the new format.
- **file_med** (*type:* string) Name of the MED file.

---

lata_to_other

Synonyms: lata_2_other

Inherits from: *interprete*

To convert results file written with LATA format to MED or LML format. Warning: Fields located at faces are not supported yet.

Parameters are:

- **[format]** (*type:* string into ["lml", "lata", "lata_v2", "med"]) Results format (MED or LATA or LML keyword).
- **file** (*type:* string) LATA file to convert to the new format.
- **file_post** (*type:* string) Name of file post.

---

link_cgns_files

Inherits from: *interprete*

Creates a single CGNS xxxx.cgns file that links to a xxxx.grid.cgns and xxxx.solution.*.cgns files

Parameters are:

- **base_name** (*type:* string) Base name of the gid/solution cgns files.
- **output_name** (*type:* string) Name of the output cgns file.

---

lire_ideas

Inherits from: *interprete*

Read a geom in a unv file. 3D tetra mesh elements only may be read by TRUST.

Parameters are:

- **nom_dom** (*type: domaine*) Name of domain.
- **file** (*type:* string) Name of file.

---

lml_to_lata

Synonyms: lml_2_lata

Inherits from: *interprete*

To convert results file written with LML format to a single LATA file.

Parameters are:

- **file_lml** (*type:* string) LML file to convert to the new format.
- **file_lata** (*type:* string) Name of the single LATA file.

---

mailler

Inherits from: *interprete*

The Mailler (Mesh) interpretor allows a Domain type object domaine to be meshed with objects objet_1, objet_2, etc...

Parameters are:

- **domaine** (*type: domaine*) Name of domain.
- **bloc** (*type: list_bloc_mailler*) Instructions to mesh.

---

maillerparallel

Inherits from: *interprete*

creates a parallel distributed hexaedral mesh of a parallelipipedic box. It is equivalent to creating a mesh with a single Pave, splitting it with Decouper and reloading it in parallel with Scatter. It only works in 3D at this time. It can also be used for a sequential computation (with all NPARTS=1)}

Parameters are:

- **domain** (*type: domaine*) the name of the domain to mesh (it must be an empty domain object).
- **nb_nodes** (*type:* int list) dimension defines the spatial dimension (currently only dimension=3 is supported), and nX, nY and nZ defines the total number of nodes in the mesh in each direction.
- **splitting** (*type:* int list) dimension is the spatial dimension and npartsX, npartsY and npartsZ are the number of parts created. The product of the number of parts must be equal to the number of processors used for the computation.
- **ghost_thickness** (*type:* int) the number of ghost cells (equivalent to the epaisseur_joint parameter of Decouper.
- **[perio_x]** (*type:* flag) change the splitting method to provide a valid mesh for periodic boundary conditions.
- **[perio_y]** (*type:* flag) change the splitting method to provide a valid mesh for periodic boundary conditions.
- **[perio_z]** (*type:* flag) change the splitting method to provide a valid mesh for periodic boundary conditions.
- **[function_coord_x]** (*type:* string) By default, the meshing algorithm creates nX nY nZ coordinates ranging between 0 and 1 (eg a unity size box). If function_coord_x} is specified, it is used to transform the [0,1] segment to the coordinates of the nodes. funcX must be a function of the x variable only.
- **[function_coord_y]** (*type:* string) like function_coord_x for y

- **[function_coord_z]** (*type:* string) like function_coord_x for z
- **[file_coord_x]** (*type:* string) Keyword to read the Nx floating point values used as nodes coordinates in the file.
- **[file_coord_y]** (*type:* string) idem file_coord_x for y
- **[file_coord_z]** (*type:* string) idem file_coord_x for z
- **[boundary_xmin]** (*type:* string) the name of the boundary at the minimum X direction. If it not provided, the default boundary names are xmin, xmax, ymin, ymax, zmin and zmax. If the mesh is periodic in a given direction, only the MIN boundary name is used, for both sides of the box.
- **[boundary_xmax]** (*type:* string) not_set
- **[boundary_ymin]** (*type:* string) not_set
- **[boundary_ymax]** (*type:* string) not_set
- **[boundary_zmin]** (*type:* string) not_set
- **[boundary_zmax]** (*type:* string) not_set

---

merge_med

Inherits from: *interprete*

This keyword allows to merge multiple MED files produced during a parallel computation into a single MED file.

Parameters are:

- **med_files_base_name** (*type:* string) Base name of multiple med files that should appear as base_name_xxxxx.med, where xxxxx denotes the MPI rank number. If you specify NOM_DU_CAS, it will automatically take the basename from your datafile's name.
- **time_iterations** (*type:* string into ["all_times", "last_time"]) Identifies whether to merge all time iterations present in the MED files or only the last one.

---

modif_bord_to_raccord

Inherits from: *interprete*

Keyword to convert a boundary of domain_name domain of kind Bord to a boundary of kind Raccord (named boundary_name). It is useful when using meshes with boundaries of kind Bord defined and to run a coupled calculation.

Parameters are:

- **domaine | domain** (*type:* *domaine*) Name of domain
- **nom_bord** (*type:* string) Name of the boundary to transform.

---

modifydomaineaxi1d

Synonyms: convert_1d_to_1daxi

Inherits from: *interprete*

Convert a 1D mesh to 1D axisymmetric mesh

Parameters are:

- **dom** (*type:* string) not_set
- **bloc** (*type: bloc_lecture*) not_set

---

moyenne_volumique

Inherits from: *interprete*

This keyword should be used after Resoudre keyword. It computes the convolution product of one or more fields with a given filtering function.

Parameters are:

- **nom_pb** (*type: pb_base*) name of the problem where the source fields will be searched.
- **nom_domaine** (*type: domaine*) name of the destination domain (for example, it can be a coarser mesh, but for optimal performance in parallel, the domain should be split with the same algorithm as the computation mesh, eg, same tranche parameters for example)
- **noms_champs** (*type:* string list) name of the source fields (these fields must be accessible from the postraitement) N source_field1 source_field2 ... source_fieldN
- **[format_post]** (*type:* string) gives the fileformat for the result (by default : lata)
- **[nom_fichier_post]** (*type:* string) indicates the filename where the result is written
- **fonction_filtre** (*type: bloc_lecture*) to specify the given filter Fonction_filtre {NL2 type filter_typeNL2 demie-largeur lNL2 [omega w] [expression string]NL2 } NL2 type filter_type : This parameter specifies the filtering function. Valid filter_type are:NL2 Boite is a box filter, $f(x,y,z)=(abs(x)<l)*(abs(y) <l)*(abs(z) <l) / (8 l^3)$NL2 Chapeau is a hat filter (product of hat filters in each direction) centered on the origin, the half-width of the filter being l and its integral being 1.NL2 Quadra is a 2nd order filter.NL2 Gaussienne is a normalized gaussian filter of standard deviation sigma in each direction (all field elements outside a cubic box defined by clipping_half_width are ignored, hence, taking clipping_half_width=2.5*sigma yields an integral of 0.99 for a uniform unity field).NL2 Parser allows a user defined function of the x,y,z variables. All elements outside a cubic box defined by clipping_half_width are ignored. The parser is much slower than the equivalent c++ coded function...NL2 demie-largeur l : This parameter specifies the half width of the filterNL2 [omega w] : This parameter must be given for the gaussienne filter. It defines the standard deviation of the gaussian filter.NL2 [expression string] : This parameter must be given for the parser filter type. This expression will be interpreted by the math parser with the predefined variables x, y and z.
- **[localisation]** (*type:* string into ["elem", "som"]) indicates where the convolution product should be computed: either on the elements or on the nodes of the destination domain.

---

multigrid_solver

Inherits from: *interprete*

Object defining a multigrid solver in IJK discretization

Parameters are:

- **[coarsen_operators]** (*type: coarsen_operators*) Definition of the number of grids that will be used, in addition to the finest (original) grid, followed by the list of the coarsen operators that will be applied to get those grids
- **[ghost_size]** (*type:* int) Number of ghost cells known by each processor in each of the three directions
- **[relax_jacobi]** (*type:* list) Parameter between 0 and 1 that will be used in the Jacobi method to solve equation on each grid. Should be around 0.7
- **[pre_smooth_steps]** (*type:* int list) First integer of the list indicates the numbers of integers that has to be read next. Following integers define the numbers of iterations done before solving the equation on each grid. For example, 2 7 8 means that we have a list of 2 integers, the first one tells us to perform 7 pre-smooth steps on the first grid, the second one tells us to perform 8 pre-smooth steps on the second grid. If there are more than 2 grids in the solver, then the remaining ones will have as many pre-smooth steps as the last mentionned number (here, 8)
- **[smooth_steps]** (*type:* int list) First integer of the list indicates the numbers of integers that has to be read next. Following integers define the numbers of iterations done after solving the equation on each grid. Same behavior as pre_smooth_steps
- **[nb_full_mg_steps]** (*type:* int list) Number of multigrid iterations at each level
- **[solveur_grossier]** (*type: solveur_sys_base*) Name of the iterative solver that will be used to solve the system on the coarsest grid. This resolution must be more precise than the ones occurring on the fine grids. The threshold of this solver must therefore be lower than seuil defined above.
- **[seuil]** (*type:* double) Define an upper bound on the norm of the final residue (i.e. the one obtained after applying the multigrid solver). With hybrid precision, as long as we have not obtained a residue whose norm is lower than the imposed threshold, we keep applying the solver
- **[impr]** (*type:* flag) Flag to display some info on the resolution on eahc grid
- **[solver_precision]** (*type:* string into ["mixed", "double"]) Precision with which the variables at stake during the resolution of the system will be stored. We can have a simple or double precision or both. In the case of a hybrid precision, the multigrid solver is launched in simple precision, but the residual is calculated in double precision.
- **[iterations_mixed_solver]** (*type:* int) Define the maximum number of iterations in mixed precision solver

---

multiplefiles

Inherits from: *interprete*

Change MPI rank limit for multiple files during I/O

Parameters are:

- **type** (*type:* int) New MPI rank limit

---

nettoiepasnoeuds

Inherits from: *interprete*

Keyword NettoiePasNoeuds does not delete useless nodes (nodes without elements) from a domain.

Parameters are:

- **domain_name** (*type: domaine*) Name of domain.

---

op_conv_ef_stab_polymac_face

Inherits from: *interprete*

Class Op_Conv_EF_Stab_PolyMAC_Face_PolyMAC

Parameters are:

- **[alpha]** (*type:* double) parametre ajustant la stabilisation de 0 (schema centre) a 1 (schema amont)

---

op_conv_ef_stab_polymac_p0_face

Inherits from: *interprete*

Class Op_Conv_EF_Stab_PolyMAC_P0_Face

---

op_conv_ef_stab_polymac_p0p1nc_elem

Synonyms: op_conv_ef_stab_polymac_p0_elem

Inherits from: *interprete*

Class Op_Conv_EF_Stab_PolyMAC_P0P1NC_Elem

Parameters are:

- **[alpha]** (*type:* double) parametre ajustant la stabilisation de 0 (schema centre) a 1 (schema amont)

---

op_conv_ef_stab_polymac_p0p1nc_face

Inherits from: *interprete*

Class Op_Conv_EF_Stab_PolyMAC_P0P1NC_Face

---

option_cgns

Inherits from: *interprete*

Class for CGNS options.

Parameters are:

- **[single_precision]** (*type:* flag) If used, data will be written with a single_precision format inside the CGNS file (it concerns both mesh coordinates and field values).
- **[multiple_files]** (*type:* flag) If used, data will be written in separate files (ie: one file per processor).
- **[parallel_over_zone]** (*type:* flag) If used, data will be written in separate zones (ie: one zone per processor). This is not so performant but easier to read later ...
- **[use_links]** (*type:* flag) If used, data will be written in separate files; one file for mesh, and then one file for solution time. Links will be used.

---

option_interpolation

Inherits from: *interprete*

Class for interpolation fields using MEDCoupling.

Parameters are:

- **[sans_dec | without_dec]** (*type:* flag) Use remapper even for a parallel calculation
- **[sharing_algo]** (*type:* int) Setting the DEC sharing algo : 0,1,2

---

option_polymac

Inherits from: *interprete*

Class of PolyMAC options.

Parameters are:

- **[use_osqp]** (*type:* flag) Flag to use the old formulation of the M2 matrix provided by the OSQP library

---

option_polymac_p0

Inherits from: *interprete*

Class of PolyMAC_P0 options.

Parameters are:

- **[interp_ve1]** (*type:* flag) Flag to enable a first order velocity face-to-element interpolation (the default value is 0 which means a second order interpolation)
- **[traitement_axi]** (*type:* flag) Flag used to relax the time-step stability criterion in case of a thin slice geometry while modelling an axi-symetrical case

---

option_vdf

Inherits from: *interprete*

Class of VDF options.

Parameters are:

- **[traitement_coins]** (*type:* string into ["oui", "non"]) Treatment of corners (yes or no). This option modifies slightly the calculations at the outlet of the plane channel. It supposes that the boundary continues after channel outlet (i.e. velocity vector remains parallel to the boundary).
- **[traitement_gradients]** (*type:* string into ["oui", "non"]) Treatment of gradient calculations (yes or no). This option modifies slightly the gradient calculation at the corners and activates also the corner treatment option.
- **[p_imposee_aux_faces]** (*type:* string into ["oui", "non"]) Pressure imposed at the faces (yes or no).
- **[all_options | toutes_les_options]** (*type:* flag) Activates all Option_VDF options. If used, must be used alone without specifying the other options, nor combinations.

---

orientefacesbord

Inherits from: *interprete*

Keyword to modify the order of the boundary vertices included in a domain, such that the surface normals are outer pointing.

Parameters are:

- **domain_name** (*type: domaine*) Name of domain.

---

parallel_io_parameters

Inherits from: *interprete*

Object to handle parallel files in IJK discretization

Parameters are:

- **[block_size_bytes]** (*type:* int) File writes will be performed by chunks of this size (in bytes). This parameter will not be taken into account if block_size_megabytes has been defined
- **[block_size_megabytes]** (*type:* int) File writes will be performed by chunks of this size (in megabytes). The size should be a multiple of the GPFS block size or lustre stripping size (typically several megabytes)
- **[writing_processes]** (*type:* int) This is the number of processes that will write concurrently to the file system (this must be set according to the capacity of the filesystem, set to 1 on small computers, can be up to 64 or 128 on very large systems).
- **[bench_ijk_splitting_write]** (*type:* string) Name of the splitting object we want to use to run a parallel write bench (optional parameter)
- **[bench_ijk_splitting_read]** (*type:* string) Name of the splitting object we want to use to run a parallel read bench (optional parameter)

---

partition

Synonyms: decouper

Inherits from: *interprete*

Class for parallel calculation to cut a domain for each processor. By default, this keyword is commented in the reference test cases.

Parameters are:

- **domaine** (*type: domaine*) Name of the domain to be cut.
- **bloc_decouper** (*type: bloc_decouper*) Description how to cut a domain.

---

partition_multi

Synonyms: decouper_multi

Inherits from: *interprete*

allows to partition multiple domains in contact with each other in parallel: necessary for resolution monolithique in implicit schemes and for all coupled problems using PolyMAC_P0P1NC. By default, this keyword is commented in the reference test cases.

Parameters are:

- **aco** (*type:* string into ["{"]) Opening curly bracket.
- **domaine1** (*type:* string into ["domaine"]) not set.
- **dom** (*type: domaine*) Name of the first domain to be cut.
- **blocdecoupdom1** (*type: bloc_decouper*) Partition bloc for the first domain.
- **domaine2** (*type:* string into ["domaine"]) not set.
- **dom2** (*type: domaine*) Name of the second domain to be cut.
- **blocdecoupdom2** (*type: bloc_decouper*) Partition bloc for the second domain.
- **acof** (*type:* string into ["}"]) Closing curly bracket.

---

pilote_icoco

Inherits from: *interprete*

not_set

Parameters are:

- **pb_name** (*type:* string) not_set
- **main** (*type:* string) not_set

---

polyedriser

Inherits from: *interprete*

cast hexahedra into polyhedra so that the indexing of the mesh vertices is compatible with PolyMAC_P0P1NC discretization. Must be used in PolyMAC_P0P1NC discretization if a hexahedral mesh has been produced with TRUST's internal mesh generator.

Parameters are:

- **domain_name** (*type:* *domaine*) Name of domain.

---

postraiter_domaine

Inherits from: *interprete*

To write one or more domains in a file with a specified format (MED,LML,LATA,SINGLE_LATA,CGNS).

Parameters are:

- **format** (*type:* string into ["lml", "lata", "single_lata", "lata_v2", "med", "cgns"]) File format.
- **[binaire]** (*type:* int into [0,1]) Binary (binaire 1) or ASCII (binaire 0) may be used. By default, it is 0 for LATA and only ASCII is available for LML and only binary is available for MED.
- **[ecrire_frontiere]** (*type:* int into [0,1]) This option will write (if set to 1, the default) or not (if set to 0) the boundaries as fields into the file (it is useful to not add the boundaries when writing a domain extracted from another domain)
- **[fichier | file]** (*type:* string) The file name can be changed with the fichier option.
- **[joints_non_postraites]** (*type:* int into [0,1]) The joints_non_postraites (1 by default) will not write the boundaries between the partitioned mesh.
- **[domaine | domain]** (*type:* *domaine*) Name of domain
- **[domaines]** (*type:* *bloc_lecture*) Names of domains : { name1 name2 }

---

precisiongeom

Inherits from: *interprete*

Class to change the way floating-point number comparison is done. By default, two numbers are equal if their absolute difference is smaller than 1e-10. The keyword is useful to modify this value. Moreover, nodes coordinates will be written in .geom files with this same precision.

Parameters are:

- **precision** (*type:* double) New value of precision.

---

raffiner_anisotrope

Inherits from: *interprete*

Only for VEF discretizations, allows to cut triangle elements in 3, or tetrahedra in 4 parts, by defining a new summit located at the center of the element: includepng{{raffineranisotri.pdf}}{{4}} includepng{{raffineranisotetra.jpeg}}{{6}}

Note that such a cut creates flat elements (anisotropic).

Parameters are:

- **domain_name** (*type:* *domaine*) Name of domain.

---

raffiner_isotrope

Synonyms: raffiner_simplexes

Inherits from: *interprete*

For VDF and VEF discretizations, allows to cut triangles/quadrangles or tetrahedral/hexaedras elements respectively in 4 or 8 new ones by defining new summits located at the middle of edges (and center of faces and elements for quadrangles and hexaedra). Such a cut preserves the shape of original elements (isotropic). For 2D elements: includetabfig{{raffinerisotri.pdf}}{{4}}{{raffinerisorect.pdf}}{{4}} For 3D elements: includetabfig{{raffinerisotetra.jpeg}}{{6}}{{raffinerisohexa.jpeg}}{{5}}.

Parameters are:

- **domain_name** (*type:* *domaine*) Name of domain.

---

raffiner_isotrope_parallele

Inherits from: *interprete*

Refine parallel mesh in parallel

Parameters are:

- **name_of_initial_domaines | name_of_initial_zones** (*type:* string) name of initial Domaines
- **name_of_new_domaines | name_of_new_zones** (*type:* string) name of new Domaines
- **[ascii]** (*type:* flag) writing Domaines in ascii format
- **[single_hdf]** (*type:* flag) writing Domaines in hdf format

---

read

Synonyms: lire

Inherits from: *interprete*

Interpretor to read the a_object objet defined between the braces.

Parameters are:

- **a_object** (*type:* string) Object to be read.
- **bloc** (*type:* string) Definition of the object.

---

read_file

Synonyms: lire_fichier

Inherits from: *interprete*

Keyword to read the object name_obj contained in the file filename.

This is notably used when the calculation domain has already been meshed and the mesh contains the file filename, simply write read_file dom filename (where dom is the name of the meshed domain).

If the filename is ;, is to execute a data set given in the file of name name_obj (a space must be entered between the semi-colon and the file name).

Parameters are:

- **name_obj** (*type:* string) Name of the object to be read.
- **filename** (*type:* string) Name of the file.

---

read_file_bin

Synonyms: lire_fichier_bin, read_file_binary

Inherits from: *interprete*

Keyword to read an object name_obj in the unformatted type file filename.

Parameters are:

- **name_obj** (*type:* string) Name of the object to be read.
- **filename** (*type:* string) Name of the file.

---

read_med

Synonyms: lire_med

Inherits from: *interprete*

Keyword to read MED mesh files where 'domain' corresponds to the domain name, 'file' corresponds to the file (written in the MED format) containing the mesh named mesh_name.

Note about naming boundaries: When reading 'file', TRUST will detect boundaries between domains (Raccord) when the name of the boundary begins by **type_raccord_**. For example, a boundary named type_raccord_wall in 'file' will be considered by TRUST as a boundary named 'wall' between two domains.

NB: To read several domains from a mesh issued from a MED file, use Read_Med to read the mesh then use Create_domain_from_sub_domain keyword.

NB: If the MED file contains one or several subdomaine defined as a group of volumes, then Read_MED will read it and will create two files domain_name_ssz.geo and domain_name_ssz_par.geo defining the subdomaines for sequential and/or parallel calculations. These subdomaines will be read in sequential in the datafile by including (after Read_Med keyword) something like:

Read_Med ....

Read_file domain_name_ssz.geo ;

During the parallel calculation, you will include something:

Scatter { ... }

Read_file domain_name_ssz_par.geo ;

Parameters are:

- **[convertalltopoly]** (*type:* flag) Option to convert mesh with mixed cells into polyhedral/polygonal cells
- **domain | domaine** (*type:* *domaine*) Corresponds to the domain name.
- **file | fichier** (*type:* string) File (written in the MED format, with extension '.med') containing the mesh
- **[mesh | maillage]** (*type:* string) Name of the mesh in med file. If not specified, the first mesh will be read.
- **[exclude_groups | exclure_groupes]** (*type:* string list) List of face groups to skip in the MED file.
- **[include_additional_face_groups | inclure_groupes_faces_additionnels]** (*type:* string list) List of face groups to read and register in the MED file.

---

read_tgrid

Synonyms: lire_tgrid

Inherits from: *interprete*

Keyword to reaf Tgrid/Gambit mesh files. 2D (triangles or quadrangles) and 3D (tetra or hexa elements) meshes, may be read by TRUST.

Parameters are:

- **dom** (*type:* *domaine*) Name of domaine.
- **filename** (*type:* string) Name of file containing the mesh.

---

read_unsupported_ascii_file_from_icem

Inherits from: *interprete*

not_set

Parameters are:

- **name_obj** (*type:* string) Name of the object to be read.
- **filename** (*type:* string) Name of the file.

---

rectify_mesh

Synonyms: orienter_simplexes

Inherits from: *interprete*

Keyword to raffine a mesh

Parameters are:

- **domain_name** (*type:* *domaine*) Name of domain.

---

redresser_hexaedres_vdf

Inherits from: *interprete*

Keyword to convert a domain (named domain_name) with quadrilaterals/VEF hexaedras which looks like rectangles/VDF hexaedras into a domain with real rectangles/VDF hexaedras.

Parameters are:

- **domain_name** (*type:* *domaine*) Name of domain to resequence.

---

refine_mesh

Inherits from: *interprete*

not_set

Parameters are:

- **domaine** (*type:* *domaine*) not_set

---

regroupebord

Inherits from: *interprete*

Keyword to build one boundary new_bord with several boundaries of the domain named domaine.

Parameters are:

- **domaine | domain** (*type:* *domaine*) Name of domain
- **new_bord** (*type:* string) Name of the new boundary
- **bords** (*type:* *bloc_lecture*) { Bound1 Bound2 }

---

remove_elem

Inherits from: *interprete*

Keyword to remove element from a VDF mesh (named domaine_name), either from an explicit list of elements or from a geometric condition defined by a condition f(x,y)>0 in 2D and f(x,y,z)>0 in 3D. All the new borders generated are gathered in one boundary called : newBord (to rename it, use RegroupeBord keyword. To split it to different boundaries, use DecoupeBord_Pour_Rayonnement keyword). Example of a removed zone of radius 0.2 centered at (x,y)=(0.5,0.5):

Remove_elem dom { fonction $0.2*0.2-(x-0.5)^2-(y-0.5)^2>0$ }

Warning : the thickness of removed zone has to be large enough to avoid singular nodes as decribed below : includepng{{removeelem.png}}{{11.234}}

Parameters are:

- **domaine | domain** (*type:* *domaine*) Name of domain
- **bloc** (*type:* *remove_elem_bloc*) not_set

---

remove_invalid_internal_boundaries

Inherits from: *interprete*

Keyword to suppress an internal boundary of the domain_name domain. Indeed, some mesh tools may define internal boundaries (eg: for post processing task after the calculation) but TRUST does not support it yet.

Parameters are:

- **domain_name** (*type:* *domaine*) Name of domain.

---

reorienter_tetraedres

Inherits from: *interprete*

This keyword is mandatory for front-tracking computations with the VEF discretization. For each tetrahedral element of the domain, it checks if it has a positive volume. If the volume (determinant of the three vectors) is negative, it swaps two nodes to reverse the orientation of this tetrahedron.

Parameters are:

- **domain_name** (*type:* *domaine*) Name of domain.

---

reorienter_triangles

Inherits from: *interprete*

not_set

Parameters are:

- **domain_name** (*type:* *domaine*) Name of domain.

---

resequencing

Synonyms: reordonner

Inherits from: *interprete*

The Reordonner interpretor is required sometimes for a VDF mesh which is not produced by the internal mesher. Example where this is used:

Read_file dom fichier.geom

Reordonner dom

Observations: This keyword is redundant when the mesh that is read is correctly sequenced in the TRUST sense. This significant mesh operation may take some time... The message returned by TRUST is not explicit when the Reordonner (Resequencing) keyword is required but not included in the data set...

Parameters are:

- **domain_name** (*type:* *domaine*) Name of domain to resequence.

---

residuals

Inherits from: *interprete*

To specify how the residuals will be computed.

Parameters are:

- **[norm]** (*type:* string into ["l2", "max"]) allows to choose the norm we want to use (max norm by default). Possible to specify L2-norm.

- **[relative]** (*type:* string into ["0", "1", "2"]) This is the old keyword seuil_statio_relatif_deconseille. If it is set to 1, it will normalize the residuals with the residuals of the first 5 timesteps (default is 0). if set to 2, residual will be computed as R/(max-min).

---

rotation

Inherits from: *interprete*

Keyword to rotate the geometry of an arbitrary angle around an axis aligned with Ox, Oy or Oz axis.

Parameters are:

- **domain_name** (*type:* *domaine*) Name of domain to wich the transformation is applied.
- **dir** (*type:* string into ["x", "y", "z"]) X, Y or Z to indicate the direction of the rotation axis
- **coord1** (*type:* double) coordinates of the center of rotation in the plane orthogonal to the rotation axis. These coordinates must be specified in the direct triad sense.
- **coord2** (*type:* double) not_set
- **angle** (*type:* double) angle of rotation (in degrees)

---

scatter

Inherits from: *interprete*

Class to read a partionned mesh from the files during a parallel calculation. The files are in binary format.

Parameters are:

- **file** (*type:* string) Name of file.
- **domaine** (*type:* *domaine*) Name of domain.

---

scattermed

Inherits from: *interprete*

This keyword will read the partition of the domain_name domain into a the MED format files file.med created by Medsplitter.

Parameters are:

- **file** (*type:* string) Name of file.
- **domaine** (*type:* *domaine*) Name of domain.

---

solve

Synonyms: resoudre

Inherits from: *interprete*

Interpretor to start calculation with TRUST.

Parameters are:

- **pb** (*type: pb_gen_base*) Name of problem to be solved.

---

stat_per_proc_perf_log

Inherits from: *interprete*

Keyword allowing to activate the detailed statistics per processor (by default this is false, and only the master proc will produce stats).

Parameters are:

- **flg** (*type:* int) A flag that can be either 0 or 1 to turn off (default) or on the detailed stats.

---

supprime_bord

Inherits from: *interprete*

Keyword to remove boundaries (named Boundary_name1 Boundary_name2) of the domain named domain_name.

Parameters are:

- **domaine | domain** (*type: domaine*) Name of domain
- **bords** (*type: list_nom*) { Boundary_name1 Boundaray_name2 }

---

system

Inherits from: *interprete*

To run Unix commands from the data file. Example: System 'echo The End | mail trust@cea.fr'

Parameters are:

- **cmd** (*type:* string) command to execute.

---

test_solveur

Inherits from: *interprete*

To test several solvers

Parameters are:

- **[fichier_secmem]** (*type:* string) Filename containing the second member B
- **[fichier_matrice]** (*type:* string) Filename containing the matrix A
- **[fichier_solution]** (*type:* string) Filename containing the solution x
- **[nb_test]** (*type:* int) Number of tests to measure the time resolution (one preconditionnement)
- **[impr]** (*type:* flag) To print the convergence solver
- **[solveur]** (*type: solveur_sys_base*) To specify a solver
- **[fichier_solveur]** (*type:* string) To specify a file containing a list of solvers
- **[genere_fichier_solveur]** (*type:* double) To create a file of the solver with a threshold convergence
- **[seuil_verification]** (*type:* double) Check if the solution satisfy ||Ax-B||<precision
- **[pas_de_solution_initiale]** (*type:* flag) Resolution isn't initialized with the solution x
- **[ascii]** (*type:* flag) Ascii files

---

test_sse_kernels

Inherits from: *interprete*

Object to test the different kernel methods used in the multigrid solver in IJK discretization

Parameters are:

- **[nmax]** (*type:* int) Number of tests we want to perform

---

testeur

Inherits from: *interprete*

not_set

Parameters are:

- **data** (*type: bloc_lecture*) not_set

---

testeur_medcoupling

Inherits from: *interprete*

not_set

Parameters are:

- **pb_name** (*type:* string) Name of domain.
- **field_name | filed_name** (*type:* string) Name of domain.

---

tetraedriser

Inherits from: *interprete*

To achieve a tetrahedral mesh based on a mesh comprising blocks, the Tetraedriser (Tetrahedralise) interpretor is used in VEF discretization. Initial block is divided in 6 tetrahedra: includepng{{tetraedriser.jpeg}}{{5}}

Parameters are:

- **domain_name** (*type:* *domaine*) Name of domain.

---

tetraedriser_homogene

Inherits from: *interprete*

Use the Tetraedriser_homogene (Homogeneous_Tetrahedralisation) interpretor in VEF discretization to mesh a block in tetrahedrals. Each block hexahedral is no longer divided into 6 tetrahedrals (keyword Tetraedriser (Tetrahedralise)), it is now broken down into 40 tetrahedrals. Thus a block defined with 11 nodes in each X, Y, Z direction will contain 10*10*10*40=40,000 tetrahedrals. This also allows problems in the mesh corners with the P1NC/P1iso/P1bulle or P1/P1 discretization items to be avoided. Initial block is divided in 40 tetrahedra: includepng{{tetraedriserhomogene.jpeg}}{{5}}

Parameters are:

- **domain_name** (*type:* *domaine*) Name of domain.

---

tetraedriser_homogene_compact

Inherits from: *interprete*

This new discretization generates tetrahedral elements from cartesian or non-cartesian hexahedral elements. The process cut each hexahedral in 6 pyramids, each of them being cut then in 4 tetrahedral. So, in comparison with tetra_homogene, less elements (*24 instead of*40) with more homogeneous volumes are generated. Moreover, this process is done in a faster way. Initial block is divided in 24 tetrahedra: includepng{{tetraedriserhomogenecompact.jpeg}}{{5}}

Parameters are:

- **domain_name** (*type:* *domaine*) Name of domain.

---

tetraedriser_homogene_fin

Inherits from: *interprete*

Tetraedriser_homogene_fin is the recommended option to tetrahedralise blocks. As an extension (subdivision) of Tetraedriser_homogene_compact, this last one cut each initial block in 48 tetrahedra (against 24, previously). This cutting ensures :

- a correct cutting in the corners (in respect to pressure discretization PreP1B),
- a better isotropy of elements than with Tetraedriser_homogene_compact,
- a better alignment of summits (this could have a benefit effect on calculation

near walls since first elements in contact with it are all contained in the same constant thickness and ii/ by the way, a 3D cartesian grid based on summits can be engendered and used to realise spectral analysis in HIT for instance). Initial block is divided in 48 tetrahedra: includepng{{tetraedriserhomogenefin.jpeg}}{{5}}

Parameters are:

- **domain_name** (*type: domaine*) Name of domain.

---

tetraedriser_par_prisme

Inherits from: *interprete*

Tetraedriser_par_prisme generates 6 iso-volume tetrahedral element from primary hexahedral one (contrarily to the 5 elements ordinarily generated by tetraedriser). This element is suitable for calculation of gradients at the summit (coincident with the gravity centre of the jointed elements related with) and spectra (due to a better alignment of the points). includetabfig{{tetraedriserparprisme.jpeg}}{{5}}{{tetraedriserparprisme2.jpeg}}{{5}} Initial block is divided in 6 prismes.

Parameters are:

- **domain_name** (*type: domaine*) Name of domain.

---

transformer

Inherits from: *interprete*

Keyword to transform the coordinates of the geometry.

Exemple to rotate your mesh by a 90o rotation and to scale the z coordinates by a factor 2: Transformer domain_name -y -x 2*z

Parameters are:

- **domain_name** (*type: domaine*) Name of domain.
- **formule** (*type:* string list) Function_for_x Function_for_y [Function_for z]

---

triangulate

Synonyms: trianguler

Inherits from: *interprete*

To achieve a triangular mesh from a mesh comprising rectangles (2 triangles per rectangle). Should be used in VEF discretization. Principle:

includepng{{trianguler.pdf}}{{10}}

Parameters are:

- **domain_name** (*type: domaine*) Name of domain.

---

trianguler_fin

Inherits from: *interprete*

Trianguler_fin is the recommended option to triangulate rectangles.

As an extension (subdivision) of Triangulate_h option, this one cut each initial rectangle in 8 triangles (against 4, previously). This cutting ensures :

- a correct cutting in the corners (in respect to pressure discretization PreP1B).
- a better isotropy of elements than with Trianguler_h option.
- a better alignment of summits (this could have a benefit effect on calculation

near walls since first elements in contact with it are all contained in the same constant thickness, and, by this way, a 2D cartesian grid based on summits can be engendered and used to realize statistical analysis in plane channel configuration for instance). Principle:

includepng{{triangulerfin.pdf}}{{10}}

Parameters are:

- **domain_name** (*type: domaine*) Name of domain.

---

trianguler_h

Inherits from: *interprete*

To achieve a triangular mesh from a mesh comprising rectangles (4 triangles per rectangle). Should be used in VEF discretization. Principle:

includepng{{triangulerh.pdf}}{{10}}

Parameters are:

- **domain_name** (*type: domaine*) Name of domain.

---

verifier_qualite_raffinements

Inherits from: *interprete*

not_set

Parameters are:

- **domain_names** (*type: vect_nom*) not_set

---

verifier_simplexes

Inherits from: *interprete*

Keyword to raffine a simplexes

Parameters are:

- **domain_name** (*type: domaine*) Name of domain.

---

verifiercoin

Inherits from: *interprete*

This keyword subdivides inconsistent 2D/3D cells used with VEFPreP1B discretization. Must be used before the mesh is discretized. NL1 The Read_file option can be used only if the file.decoupage_som was previously created by TRUST. This option, only in 2D, reverses the common face at two cells (at least one is inconsistent), through the nodes opposed. In 3D, the option has no effect.

The expert_only option deactivates, into the VEFPreP1B divergence operator, the test of inconsistent cells.

Parameters are:

- **domain_name | dom** (*type: domaine*) Name of the domaine
- **bloc** (*type: verifiercoin_bloc*) not_set

---

write

Synonyms: ecrire

Inherits from: *interprete*

Keyword to write the object of name name_obj to a standard outlet.

Parameters are:

- **name_obj** (*type:* string) Name of the object to be written.

---

write_file

Synonyms: ecrire_fichier, ecrire_fichier_bin

Inherits from: *interprete*

Keyword to write the object of name name_obj to a file filename. Since the v1.6.3, the default format is now binary format file.

Parameters are:

- **name_obj** (*type:* string) Name of the object to be written.
- **filename** (*type:* string) Name of the file.

2.1.20 Keywords derived from list_bloc_mailler

list_bloc_mailler

List of block mesh.

2.1.21 Keywords derived from list_bord

list_bord

The block sides.

2.1.22 Keywords derived from list_info_med

list_info_med

not_set

2.1.23 Keywords derived from list_list_nom

list_list_nom

pour les groupes

2.1.24 Keywords derived from list_nom

list_nom

List of name.

2.1.25 Keywords derived from list_nom_virgule

list_nom_virgule

List of name.

2.1.26 Keywords derived from list_stat_post

list_stat_post

Post-processing for statistics

2.1.27 Keywords derived from list_un_pb

list_un_pb

pour les groupes

2.1.28 Keywords derived from listchamp_generique

listchamp_generique

XXX

2.1.29 Keywords derived from listdeuxmots_sacc

listdeuxmots_sacc

List of groups of two words (without curly brackets).

2.1.30 Keywords derived from liste_post

liste_post

Keyword to use several results files. List of objects of post-processing (with name)

2.1.31 Keywords derived from liste_post_ok

liste_post_ok

Keyword to use several results files. List of objects of post-processing (with name)

2.1.32 Keywords derived from liste_sonde_tble

liste_sonde_tble

not_set

2.1.33 Keywords derived from listeqn

listeqn

List of equations.

2.1.34 Keywords derived from listobj

listobj

List of objects.

2.1.35 Keywords derived from listpoints

listpoints

Points.

2.1.36 Keywords derived from listsous_zone_valeur

listsous_zone_valeur

List of groups of two words.

2.1.37 Keywords derived from loi_etat_base

binaire_gaz_parfait_qc

Inherits from: *loi_etat_base*

Class for perfect gas binary mixtures state law used with a quasi-compressible fluid under the iso-thermal and iso-bar assumptions.

Parameters are:

- **molar_mass1** (*type:* double) Molar mass of species 1 (in kg/mol).
- **molar_mass2** (*type:* double) Molar mass of species 2 (in kg/mol).
- **mu1** (*type:* double) Dynamic viscosity of species 1 (in kg/m.s).
- **mu2** (*type:* double) Dynamic viscosity of species 2 (in kg/m.s).
- **temperature** (*type:* double) Temperature (in Kelvin) which will be constant during the simulation since this state law only works for iso-thermal conditions.
- **diffusion_coeff** (*type:* double) Diffusion coefficient assumed the same for both species (in m2/s).

---

binaire_gaz_parfait_wc

Inherits from: *loi_etat_base*

Class for perfect gas binary mixtures state law used with a weakly-compressible fluid under the iso-thermal and iso-bar assumptions.

Parameters are:

- **molar_mass1** (*type:* double) Molar mass of species 1 (in kg/mol).
- **molar_mass2** (*type:* double) Molar mass of species 2 (in kg/mol).
- **mu1** (*type:* double) Dynamic viscosity of species 1 (in kg/m.s).
- **mu2** (*type:* double) Dynamic viscosity of species 2 (in kg/m.s).
- **temperature** (*type:* double) Temperature (in Kelvin) which will be constant during the simulation since this state law only works for iso-thermal conditions.
- **diffusion_coeff** (*type:* double) Diffusion coefficient assumed the same for both species (in m2/s).

---

coolprop_qc

Inherits from: *loi_etat_base*

Class for using CoolProp with QC problem

Parameters are:

- **cp** (*type:* double) Specific heat at constant pressure (J/kg/K).
- **fluid** (*type:* string) Fluid name in the CoolProp model
- **model** (*type:* string) CoolProp model name

---

coolprop_wc

Inherits from: *loi_etat_base*

Class for using CoolProp with WC problem

Parameters are:

- **cp** (*type:* double) Specific heat at constant pressure (J/kg/K).
- **fluid** (*type:* string) Fluid name in the CoolProp model
- **model** (*type:* string) CoolProp model name

---

eos_qc

Inherits from: *loi_etat_base*

Class for using EOS with QC problem

Parameters are:

- **cp** (*type:* double) Specific heat at constant pressure (J/kg/K).
- **fluid** (*type:* string) Fluid name in the EOS model
- **model** (*type:* string) EOS model name

---

eos_wc

Inherits from: *loi_etat_base*

Class for using EOS with WC problem

Parameters are:

- **cp** (*type:* double) Specific heat at constant pressure (J/kg/K).
- **fluid** (*type:* string) Fluid name in the EOS model
- **model** (*type:* string) EOS model name

---

loi_etat_base

Basic class for state laws used with a dilatable fluid.

---

loi_etat_gaz_parfait_base

Inherits from: *loi_etat_base*

Basic class for perfect gases state laws used with a dilatable fluid.

---

loi_etat_gaz_reel_base

Inherits from: *loi_etat_base*

Basic class for real gases state laws used with a dilatable fluid.

---

loi_etat_tppi_base

Inherits from: *loi_etat_base*

Basic class for thermo-physical properties interface (TPPI) used for dilatable problems

---

multi_gaz_parfait_qc

Inherits from: *loi_etat_base*

Class for perfect gas multi-species mixtures state law used with a quasi-compressible fluid.

Parameters are:

- **sc** (*type:* double) Schmidt number of the gas Sc=nu/D (D: diffusion coefficient of the mixing).
- **prandtl** (*type:* double) Prandtl number of the gas Pr=mu*Cp/lambda
- **[cp]** (*type:* double) Specific heat at constant pressure of the gas Cp.
- **[dtol_fraction]** (*type:* double) Delta tolerance on mass fractions for check testing (default value 1.e-6).
- **[correction_fraction]** (*type:* flag) To force mass fractions between 0. and 1.
- **[ignore_check_fraction]** (*type:* flag) Not to check if mass fractions between 0. and 1.

---

multi_gaz_parfait_wc

Inherits from: *loi_etat_base*

Class for perfect gas multi-species mixtures state law used with a weakly-compressible fluid.

Parameters are:

- **species_number** (*type:* int) Number of species you are considering in your problem.
- **diffusion_coeff** (*type: field_base*) Diffusion coefficient of each species, defined with a Champ_uniforme of dimension equals to the species_number.
- **molar_mass** (*type: field_base*) Molar mass of each species, defined with a Champ_uniforme of dimension equals to the species_number.
- **mu** (*type: field_base*) Dynamic viscosity of each species, defined with a Champ_uniforme of dimension equals to the species_number.
- **cp** (*type: field_base*) Specific heat at constant pressure of the gas Cp, defined with a Champ_uniforme of dimension equals to the species_number..
- **prandtl** (*type:* double) Prandtl number of the gas Pr=mu*Cp/lambda.

---

perfect_gaz_qc

Synonyms: gaz_parfait_qc

Inherits from: *loi_etat_base*

Class for perfect gas state law used with a quasi-compressible fluid.

Parameters are:

- **cp** (*type:* double) Specific heat at constant pressure (J/kg/K).
- **[cv]** (*type:* double) Specific heat at constant volume (J/kg/K).
- **[gamma]** (*type:* double) Cp/Cv
- **prandtl** (*type:* double) Prandtl number of the gas Pr=mu*Cp/lambda
- **[rho_constant_pour_debug]** (*type: field_base*) For developers to debug the code with a constant rho.

---

perfect_gaz_wc

Synonyms: gaz_parfait_wc

Inherits from: *loi_etat_base*

Class for perfect gas state law used with a weakly-compressible fluid.

Parameters are:

- **cp** (*type:* double) Specific heat at constant pressure (J/kg/K).
- **[cv]** (*type:* double) Specific heat at constant volume (J/kg/K).
- **[gamma]** (*type:* double) Cp/Cv

- **prandtl** (*type:* double) Prandtl number of the gas Pr=mu*Cp/lambda

---

rhot_gaz_parfait_qc

Inherits from: *loi_etat_base*

Class for perfect gas used with a quasi-compressible fluid where the state equation is defined as rho = f(T).

Parameters are:

- **cp** (*type:* double) Specific heat at constant pressure of the gas Cp.
- **[prandtl]** (*type:* double) Prandtl number of the gas Pr=mu*Cp/lambda
- **[rho_xyz]** (*type: field_base*) Defined with a Champ_Fonc_xyz to define a constant rho with time (space dependent)
- **[rho_t]** (*type:* string) Expression of T used to calculate rho. This can lead to a variable rho, both in space and in time.
- **[t_min]** (*type:* double) Temperature may, in some cases, locally and temporarily be very small (and negative) even though computation converges. T_min keyword allows to set a lower limit of temperature (in Kelvin, -1000 by default). WARNING: DO NOT USE THIS KEYWORD WITHOUT CHECKING CAREFULY YOUR RESULTS!

---

rhot_gaz_reel_qc

Inherits from: *loi_etat_base*

Class for real gas state law used with a quasi-compressible fluid.

Parameters are:

- **bloc** (*type: bloc_lecture*) Description.

2.1.38 Keywords derived from loi_fermeture_base

loi_fermeture_base

Class for appends fermeture to problem

---

loi_fermeture_test

Inherits from: *loi_fermeture_base*

Loi for test only

Parameters are:

- **[coef]** (*type:* double) coefficient

2.1.39 Keywords derived from loi_horaire

loi_horaire

to define the movement with a time-dependant law for the solid interface.

Parameters are:

- **position** (*type:* string list) not_set
- **vitesse** (*type:* string list) not_set
- **[rotation]** (*type:* string list) not_set
- **[derivee_rotation]** (*type:* string list) not_set

2.1.40 Keywords derived from milieu_base

constituant

Inherits from: *milieu_base*

Constituent.

Parameters are:

- **[rho]** (*type: field_base*) Density (kg.m-3).
- **[cp]** (*type: field_base*) Specific heat (J.kg-1.K-1).
- **[Lambda | lambda_u]** (*type: field_base*) Conductivity (W.m-1.K-1).
- **[coefficient_diffusion]** (*type: field_base*) Constituent diffusion coefficient value (m2.s-1). If a multi-constituent problem is being processed, the diffusivite will be a vectorial and each components will be the diffusion of the constituent.
- **[gravite]** (*type: field_base*) Gravity field (optional).
- **[porosites_champ]** (*type: field_base*) The porosity is given at each element and the porosity at each face, Psi(face), is calculated by the average of the porosities of the two neighbour elements Psi(elem1), Psi(elem2) : Psi(face)=2/(1/Psi(elem1)+1/Psi(elem2)). This keyword is optional.
- **[diametre_hyd_champ]** (*type: field_base*) Hydraulic diameter field (optional).
- **[porosites]** (*type: porosites*) Porosities.

---

fluide_base

Inherits from: *milieu_base*

Basic class for fluids.

Parameters are:

- **[indice]** (*type: field_base*) Refractivity of fluid.
- **[kappa]** (*type: field_base*) Absorptivity of fluid (m-1).

- **[gravite]** (*type: field_base*) Gravity field (optional).
- **[porosites_champ]** (*type: field_base*) The porosity is given at each element and the porosity at each face, Psi(face), is calculated by the average of the porosities of the two neighbour elements Psi(elem1), Psi(elem2) : Psi(face)=2/(1/Psi(elem1)+1/Psi(elem2)). This keyword is optional.
- **[diametre_hyd_champ]** (*type: field_base*) Hydraulic diameter field (optional).
- **[porosites]** (*type: porosites*) Porosities.

---

fluide_dilatable_base

Inherits from: *milieu_base*

Basic class for dilatable fluids.

Parameters are:

- **[indice]** (*type: field_base*) Refractivity of fluid.
- **[kappa]** (*type: field_base*) Absorptivity of fluid (m-1).
- **[gravite]** (*type: field_base*) Gravity field (optional).
- **[porosites_champ]** (*type: field_base*) The porosity is given at each element and the porosity at each face, Psi(face), is calculated by the average of the porosities of the two neighbour elements Psi(elem1), Psi(elem2) : Psi(face)=2/(1/Psi(elem1)+1/Psi(elem2)). This keyword is optional.
- **[diametre_hyd_champ]** (*type: field_base*) Hydraulic diameter field (optional).
- **[porosites]** (*type: porosites*) Porosities.

---

fluide_incompressible

Inherits from: *milieu_base*

Class for non-compressible fluids.

Parameters are:

- **[beta_th]** (*type: field_base*) Thermal expansion (K-1).
- **[mu]** (*type: field_base*) Dynamic viscosity (kg.m-1.s-1).
- **[beta_co]** (*type: field_base*) Volume expansion coefficient values in concentration.
- **[rho]** (*type: field_base*) Density (kg.m-3).
- **[cp]** (*type: field_base*) Specific heat (J.kg-1.K-1).
- **[Lambda | lambda_u]** (*type: field_base*) Conductivity (W.m-1.K-1).
- **[porosites]** (*type: bloc_lecture*) Porosity (optional)
- **[indice]** (*type: field_base*) Refractivity of fluid.
- **[kappa]** (*type: field_base*) Absorptivity of fluid (m-1).
- **[gravite]** (*type: field_base*) Gravity field (optional).

- **[porosites_champ]** (*type: field_base*) The porosity is given at each element and the porosity at each face, Psi(face), is calculated by the average of the porosities of the two neighbour elements Psi(elem1), Psi(elem2) : Psi(face)=2/(1/Psi(elem1)+1/Psi(elem2)). This keyword is optional.
- **[diametre_hyd_champ]** (*type: field_base*) Hydraulic diameter field (optional).

---

fluide_ostwald

Inherits from: *milieu_base*

Non-Newtonian fluids governed by Ostwald's law. The law applicable to stress tensor is:

tau=K(T)*(D:D/2)**((n-1)/2)*D Where:

D refers to the deformation tensor

K refers to fluid consistency (may be a function of the temperature T)

n refers to the fluid structure index n=1 for a Newtonian fluid, n<1 for a rheofluidifier fluid, n>1 for a rheothickening fluid.

Parameters are:

- **[k]** (*type: field_base*) Fluid consistency.
- **[n]** (*type: field_base*) Fluid structure index.
- **[beta_th]** (*type: field_base*) Thermal expansion (K-1).
- **[mu]** (*type: field_base*) Dynamic viscosity (kg.m-1.s-1).
- **[beta_co]** (*type: field_base*) Volume expansion coefficient values in concentration.
- **[rho]** (*type: field_base*) Density (kg.m-3).
- **[cp]** (*type: field_base*) Specific heat (J.kg-1.K-1).
- **[Lambda | lambda_u]** (*type: field_base*) Conductivity (W.m-1.K-1).
- **[porosites]** (*type: bloc_lecture*) Porosity (optional)
- **[indice]** (*type: field_base*) Refractivity of fluid.
- **[kappa]** (*type: field_base*) Absorptivity of fluid (m-1).
- **[gravite]** (*type: field_base*) Gravity field (optional).
- **[porosites_champ]** (*type: field_base*) The porosity is given at each element and the porosity at each face, Psi(face), is calculated by the average of the porosities of the two neighbour elements Psi(elem1), Psi(elem2) : Psi(face)=2/(1/Psi(elem1)+1/Psi(elem2)). This keyword is optional.
- **[diametre_hyd_champ]** (*type: field_base*) Hydraulic diameter field (optional).

---

fluide_quasi_compressible

Inherits from: *milieu_base*

Quasi-compressible flow with a low mach number assumption; this means that the thermo-dynamic pressure (used in state law) is uniform in space.

Parameters are:

- **[sutherland]** (*type:* *bloc_sutherland*) Sutherland law for viscosity and for conductivity.
- **[pression]** (*type:* double) Initial thermo-dynamic pressure used in the assosciated state law.
- **[loi_etat]** (*type:* *loi_etat_base*) The state law that will be associated to the Quasi-compressible fluid.
- **[traitement_pth]** (*type:* string into ["edo", "constant", "conservation_masse"]) Particular treatment for the thermodynamic pressure Pth ; there are three possibilities: 1) with the keyword 'edo' the code computes Pth solving an O.D.E. ; in this case, the mass is not strictly conserved (it is the default case for quasi compressible computation): 2) the keyword 'conservation_masse' forces the conservation of the mass (closed geometry or with periodic boundaries condition) 3) the keyword 'constant' makes it possible to have a constant Pth ; it's the good choice when the flow is open (e.g. with pressure boundary conditions). It is possible to monitor the volume averaged value for temperature and density, plus Pth evolution in the .evol_glob file.
- **[traitement_rho_gravite]** (*type:* string into ["standard", "moins_rho_moyen"]) It may be :1) `standard` : the gravity term is evaluted with rho*g (It is the default). 2) `moins_rho_moyen` : the gravity term is evaluated with (rho-rhomoy) *g. Unknown pressure is then P*=P+rhomoy*g*z. It is useful when you apply uniforme pressure boundary condition like P*=0.
- **[temps_debut_prise_en_compte_drho_dt]** (*type:* double) While time<value, dRho/dt is set to zero (Rho, volumic mass). Useful for some calculation during the first time steps with big variation of temperature and volumic mass.
- **[omega_relaxation_drho_dt]** (*type:* double) Optional option to have a relaxed algorithm to solve the mass equation. value is used (1 per default) to specify omega.
- **[Lambda | lambda_u]** (*type:* *field_base*) Conductivity (W.m-1.K-1).
- **[mu]** (*type:* *field_base*) Dynamic viscosity (kg.m-1.s-1).
- **[indice]** (*type:* *field_base*) Refractivity of fluid.
- **[kappa]** (*type:* *field_base*) Absorptivity of fluid (m-1).
- **[gravite]** (*type:* *field_base*) Gravity field (optional).
- **[porosites_champ]** (*type:* *field_base*) The porosity is given at each element and the porosity at each face, Psi(face), is calculated by the average of the porosities of the two neighbour elements Psi(elem1), Psi(elem2) : Psi(face)=2/(1/Psi(elem1)+1/Psi(elem2)). This keyword is optional.
- **[diametre_hyd_champ]** (*type:* *field_base*) Hydraulic diameter field (optional).
- **[porosites]** (*type:* *porosites*) Porosities.

---

fluide_reel_base

Inherits from: *milieu_base*

Class for real fluids.

Parameters are:

- **[indice]** (*type: field_base*) Refractivity of fluid.
- **[kappa]** (*type: field_base*) Absorptivity of fluid (m-1).
- **[gravite]** (*type: field_base*) Gravity field (optional).
- **[porosites_champ]** (*type: field_base*) The porosity is given at each element and the porosity at each face, Psi(face), is calculated by the average of the porosities of the two neighbour elements Psi(elem1), Psi(elem2) : Psi(face)=2/(1/Psi(elem1)+1/Psi(elem2)). This keyword is optional.
- **[diametre_hyd_champ]** (*type: field_base*) Hydraulic diameter field (optional).
- **[porosites]** (*type: porosites*) Porosities.

---

fluide_sodium_gaz

Inherits from: *milieu_base*

Class for Fluide_sodium_liquide

Parameters are:

- **[p_ref]** (*type:* double) Use to set the pressure value in the closure law. If not specified, the value of the pressure unknown will be used
- **[t_ref]** (*type:* double) Use to set the temperature value in the closure law. If not specified, the value of the temperature unknown will be used
- **[indice]** (*type: field_base*) Refractivity of fluid.
- **[kappa]** (*type: field_base*) Absorptivity of fluid (m-1).
- **[gravite]** (*type: field_base*) Gravity field (optional).
- **[porosites_champ]** (*type: field_base*) The porosity is given at each element and the porosity at each face, Psi(face), is calculated by the average of the porosities of the two neighbour elements Psi(elem1), Psi(elem2) : Psi(face)=2/(1/Psi(elem1)+1/Psi(elem2)). This keyword is optional.
- **[diametre_hyd_champ]** (*type: field_base*) Hydraulic diameter field (optional).
- **[porosites]** (*type: porosites*) Porosities.

---

fluide_sodium_liquide

Inherits from: *milieu_base*

Class for Fluide_sodium_liquide

Parameters are:

- **[p_ref]** (*type:* double) Use to set the pressure value in the closure law. If not specified, the value of the pressure unknown will be used
- **[t_ref]** (*type:* double) Use to set the temperature value in the closure law. If not specified, the value of the temperature unknown will be used
- **[indice]** (*type: field_base*) Refractivity of fluid.
- **[kappa]** (*type: field_base*) Absorptivity of fluid (m-1).
- **[gravite]** (*type: field_base*) Gravity field (optional).
- **[porosites_champ]** (*type: field_base*) The porosity is given at each element and the porosity at each face, Psi(face), is calculated by the average of the porosities of the two neighbour elements Psi(elem1), Psi(elem2) : Psi(face)=2/(1/Psi(elem1)+1/Psi(elem2)). This keyword is optional.
- **[diametre_hyd_champ]** (*type: field_base*) Hydraulic diameter field (optional).
- **[porosites]** (*type: porosites*) Porosities.

---

fluide_stiffened_gas

Inherits from: *milieu_base*

Class for Stiffened Gas

Parameters are:

- **[gamma]** (*type:* double) Heat capacity ratio (Cp/Cv)
- **[pinf]** (*type:* double) Stiffened gas pressure constant (if set to zero, the state law becomes identical to that of perfect gases)
- **[mu]** (*type:* double) Dynamic viscosity
- **[Lambda]** (*type:* double) Thermal conductivity
- **[cv]** (*type:* double) Thermal capacity at constant volume
- **[q]** (*type:* double) Reference energy
- **[q_prim]** (*type:* double) Model constant
- **[indice]** (*type: field_base*) Refractivity of fluid.
- **[kappa]** (*type: field_base*) Absorptivity of fluid (m-1).
- **[gravite]** (*type: field_base*) Gravity field (optional).
- **[porosites_champ]** (*type: field_base*) The porosity is given at each element and the porosity at each face, Psi(face), is calculated by the average of the porosities of the two neighbour elements Psi(elem1), Psi(elem2) : Psi(face)=2/(1/Psi(elem1)+1/Psi(elem2)). This keyword is optional.
- **[diametre_hyd_champ]** (*type: field_base*) Hydraulic diameter field (optional).
- **[porosites]** (*type: porosites*) Porosities.

fluide_weakly_compressible

Inherits from: *milieu_base*

Weakly-compressible flow with a low mach number assumption; this means that the thermo-dynamic pressure (used in state law) can vary in space.

Parameters are:

- **[loi_etat]** (*type: loi_etat_base*) The state law that will be associated to the Weakly-compressible fluid.
- **[sutherland]** (*type: bloc_sutherland*) Sutherland law for viscosity and for conductivity.
- **[traitement_pth]** (*type:* string into ["constant"]) Particular treatment for the thermodynamic pressure Pth ; there is currently one possibility: 1) the keyword 'constant' makes it possible to have a constant Pth but not uniform in space ; it's the good choice when the flow is open (e.g. with pressure boundary conditions).
- **[Lambda | lambda_u]** (*type: field_base*) Conductivity (W.m-1.K-1).
- **[mu]** (*type: field_base*) Dynamic viscosity (kg.m-1.s-1).
- **[pression_thermo]** (*type:* double) Initial thermo-dynamic pressure used in the assosciated state law.
- **[pression_xyz]** (*type: field_base*) Initial thermo-dynamic pressure used in the assosciated state law. It should be defined with as a Champ_Fonc_xyz.
- **[use_total_pressure]** (*type:* int) Flag (0 or 1) used to activate and use the total pressure in the assosciated state law. The default value of this Flag is 0.
- **[use_hydrostatic_pressure]** (*type:* int) Flag (0 or 1) used to activate and use the hydro-static pressure in the assosciated state law. The default value of this Flag is 0.
- **[use_grad_pression_eos]** (*type:* int) Flag (0 or 1) used to specify whether or not the gradient of the thermo-dynamic pressure will be taken into account in the source term of the temperature equation (case of a non-uniform pressure). The default value of this Flag is 1 which means that the gradient is used in the source.
- **[time_activate_ptot]** (*type:* double) Time (in seconds) at which the total pressure will be used in the assosciated state law.
- **[indice]** (*type: field_base*) Refractivity of fluid.
- **[kappa]** (*type: field_base*) Absorptivity of fluid (m-1).
- **[gravite]** (*type: field_base*) Gravity field (optional).
- **[porosites_champ]** (*type: field_base*) The porosity is given at each element and the porosity at each face, Psi(face), is calculated by the average of the porosities of the two neighbour elements Psi(elem1), Psi(elem2) : Psi(face)=2/(1/Psi(elem1)+1/Psi(elem2)). This keyword is optional.
- **[diametre_hyd_champ]** (*type: field_base*) Hydraulic diameter field (optional).
- **[porosites]** (*type: porosites*) Porosities.

milieu_base

Basic class for medium (physics properties of medium).

Parameters are:

- **[gravite]** (*type: field_base*) Gravity field (optional).
- **[porosites_champ]** (*type: field_base*) The porosity is given at each element and the porosity at each face, Psi(face), is calculated by the average of the porosities of the two neighbour elements Psi(elem1), Psi(elem2) : Psi(face)=2/(1/Psi(elem1)+1/Psi(elem2)). This keyword is optional.
- **[diametre_hyd_champ]** (*type: field_base*) Hydraulic diameter field (optional).
- **[porosites]** (*type: porosites*) Porosities.

---

solide

Inherits from: *milieu_base*

Solid with cp and/or rho non-uniform.

Parameters are:

- **[rho]** (*type: field_base*) Density (kg.m-3).
- **[cp]** (*type: field_base*) Specific heat (J.kg-1.K-1).
- **[Lambda | lambda_u]** (*type: field_base*) Conductivity (W.m-1.K-1).
- **[user_field]** (*type: field_base*) user defined field.
- **[gravite]** (*type: field_base*) Gravity field (optional).
- **[porosites_champ]** (*type: field_base*) The porosity is given at each element and the porosity at each face, Psi(face), is calculated by the average of the porosities of the two neighbour elements Psi(elem1), Psi(elem2) : Psi(face)=2/(1/Psi(elem1)+1/Psi(elem2)). This keyword is optional.
- **[diametre_hyd_champ]** (*type: field_base*) Hydraulic diameter field (optional).
- **[porosites]** (*type: porosites*) Porosities.

2.1.41 Keywords derived from milieu_composite

milieu_composite

Composite medium made of several sub mediums.

2.1.42 Keywords derived from milieu_musig

milieu_musig

MUSIG medium made of several sub mediums.

2.1.43 Keywords derived from modele_turbulence_scal_base

modele_turbulence_scal_base

Basic class for turbulence model for energy equation.

Parameters are:

- **[dt_impr_nusselt]** (*type:* double) Keyword to print local values of Nusselt number and temperature near a wall during a turbulent calculation. The values will be printed in the _Nusselt.face file each dt_impr_nusselt time period. The local Nusselt expression is as follows : Nu = ((lambda+lambda_t)/lambda)*d_wall/d_eq where d_wall is the distance from the first mesh to the wall and d_eq is given by the wall law. This option also gives the value of d_eq and h = (lambda+lambda_t)/d_eq and the fluid temperature of the first mesh near the wall. For the Neumann boundary conditions (flux_impose), the <<equivalent>> wall temperature given by the wall law is also printed (Tparoi equiv.) preceded for VEF calculation by the edge temperature <<T face de bord>>.
- **[turbulence_paroi]** (*type: turbulence_paroi_scalaire_base*) Keyword to set the wall law.

---

modele_turbulence_scal_null

Synonyms: null

Inherits from: *modele_turbulence_scal_base*

Null scalar turbulence model (turbulent diffusivity = 0) which can be used with a turbulent problem.

Parameters are:

- **[dt_impr_nusselt]** (*type:* double) Keyword to print local values of Nusselt number and temperature near a wall during a turbulent calculation. The values will be printed in the _Nusselt.face file each dt_impr_nusselt time period. The local Nusselt expression is as follows : Nu = ((lambda+lambda_t)/lambda)*d_wall/d_eq where d_wall is the distance from the first mesh to the wall and d_eq is given by the wall law. This option also gives the value of d_eq and h = (lambda+lambda_t)/d_eq and the fluid temperature of the first mesh near the wall. For the Neumann boundary conditions (flux_impose), the <<equivalent>> wall temperature given by the wall law is also printed (Tparoi equiv.) preceded for VEF calculation by the edge temperature <<T face de bord>>.

---

prandtl

Inherits from: *modele_turbulence_scal_base*

The Prandtl model. For the scalar equations, only the model based on Reynolds analogy is available. If K_Epsilon was selected in the hydraulic equation, Prandtl must be selected for the convection-diffusion temperature equation coupled to the hydraulic equation and Schmidt for the concentration equations.

Parameters are:

- **[prdt]** (*type:* string) Keyword to modify the constant (Prdt) of Prandtl model : Alphat=Nut/Prdt Default value is 0.9
- **[prandt_turbulent_fonction_nu_t_alpha]** (*type:* string) Optional keyword to specify turbulent diffusivity (by default, alpha_t=nu_t/Prt) with another formulae, for example: alpha_t=nu_t2/(0,7*alpha+0,85*nu_tt) with the string nu_t*nu_t/(0,7*alpha+0,85*nu_t) where alpha is the thermal diffusivity.
- **[dt_impr_nusselt]** (*type:* double) Keyword to print local values of Nusselt number and temperature near a wall during a turbulent calculation. The values will be printed in the _Nusselt.face file each dt_impr_nusselt time period. The local Nusselt expression is as follows : Nu = ((lambda+lambda_t)/lambda)*d_wall/d_eq where d_wall is the distance from the first mesh to the wall and d_eq is given by the wall law. This option also gives the value of d_eq and h = (lambda+lambda_t)/d_eq and the fluid temperature of the first mesh near the wall. For the Neumann boundary conditions (flux_impose), the <<equivalent>> wall temperature given by the wall law is also printed (Tparoi equiv.) preceded for VEF calculation by the edge temperature <<T face de bord>>.
- **[turbulence_paroi]** (*type: turbulence_paroi_scalaire_base*) Keyword to set the wall law.

---

schmidt

Inherits from: *modele_turbulence_scal_base*

The Schmidt model. For the scalar equations, only the model based on Reynolds analogy is available. If K_Epsilon was selected in the hydraulic equation, Schmidt must be selected for the convection-diffusion temperature equation coupled to the hydraulic equation and Schmidt for the concentration equations.

Parameters are:

- **[scturb]** (*type:* double) Keyword to modify the constant (Sct) of Schmlidt model : Dt=Nut/Sct Default value is 0.7.
- **[dt_impr_nusselt]** (*type:* double) Keyword to print local values of Nusselt number and temperature near a wall during a turbulent calculation. The values will be printed in the _Nusselt.face file each dt_impr_nusselt time period. The local Nusselt expression is as follows : Nu = ((lambda+lambda_t)/lambda)*d_wall/d_eq where d_wall is the distance from the first mesh to the wall and d_eq is given by the wall law. This option also gives the value of d_eq and h = (lambda+lambda_t)/d_eq and the fluid temperature of the first mesh near the wall. For the Neumann boundary conditions (flux_impose), the <<equivalent>> wall temperature given by the wall law is also printed (Tparoi equiv.) preceded for VEF calculation by the edge temperature <<T face de bord>>.
- **[turbulence_paroi]** (*type: turbulence_paroi_scalaire_base*) Keyword to set the wall law.

2.1.44 Keywords derived from mor_eqn

conduction

Inherits from: *mor_eqn*

Heat equation.

Parameters are:

- **[disable_equation_residual]** (*type:* string) The equation residual will not be used for the problem residual used when checking time convergence or computing dynamic time-step
- **[convection]** (*type: bloc_convection*) Keyword to alter the convection scheme.
- **[diffusion]** (*type: bloc_diffusion*) Keyword to specify the diffusion operator.
- **[conditions_limites | boundary_conditions]** (*type: condlims*) Boundary conditions.
- **[conditions_initiales | initial_conditions]** (*type: condinits*) Initial conditions.
- **[sources]** (*type: sources*) To introduce a source term into an equation (in case of several source terms into the same equation, the blocks corresponding to the various terms need to be separated by a comma)
- **[ecrire_fichier_xyz_valeur]** (*type: ecrire_fichier_xyz_valeur*) This keyword is used to write the values of a field only for some boundaries in a text file
- **[parametre_equation]** (*type: parametre_equation_base*) Keyword used to specify additional parameters for the equation
- **[equation_non_resolue]** (*type:* string) The equation will not be solved while condition(t) is verified if equation_non_resolue keyword is used. Exemple: The Navier-Stokes equations are not solved between time t0 and t1. Navier_Sokes_Standard { equation_non_resolue (t>t0)*(t<t1) }

---

convection_diffusion_chaleur_qc

Inherits from: *mor_eqn*

Temperature equation for a quasi-compressible fluid.

Parameters are:

- **[mode_calcul_convection]** (*type:* string into ["ancien", "divut_moins_tdivu", "divrhout_moins_tdivrhou"]) Option to set the form of the convective operatorNL2 divrhouT_moins_Tdivrhou (the default since 1.6.8): rho.u.gradT = div(rho.u.T)- Tdiv(rho.u.1) NL2ancien: u.gradT = div(u.T) - T.div(u) divuT_moins_Tdivu : u.gradT = div(u.T) - Tdiv(u.1)
- **[disable_equation_residual]** (*type:* string) The equation residual will not be used for the problem residual used when checking time convergence or computing dynamic time-step
- **[convection]** (*type: bloc_convection*) Keyword to alter the convection scheme.
- **[diffusion]** (*type: bloc_diffusion*) Keyword to specify the diffusion operator.
- **[conditions_limites | boundary_conditions]** (*type: condlims*) Boundary conditions.
- **[conditions_initiales | initial_conditions]** (*type: condinits*) Initial conditions.
- **[sources]** (*type: sources*) To introduce a source term into an equation (in case of several source terms into the same equation, the blocks corresponding to the various terms need to be separated by a comma)

- **[ecrire_fichier_xyz_valeur]** (*type: ecrire_fichier_xyz_valeur*) This keyword is used to write the values of a field only for some boundaries in a text file
- **[parametre_equation]** (*type: parametre_equation_base*) Keyword used to specify additional parameters for the equation
- **[equation_non_resolue]** (*type:* string) The equation will not be solved while condition(t) is verified if equation_non_resolue keyword is used. Exemple: The Navier-Stokes equations are not solved between time t0 and t1. Navier_Sokes_Standard { equation_non_resolue (t>t0)*(t<t1) }

---

convection_diffusion_chaleur_turbulent_qc

Inherits from: *mor_eqn*

Temperature equation for a quasi-compressible fluid as well as the associated turbulence model equations.

Parameters are:

- **[modele_turbulence]** (*type: modele_turbulence_scal_base*) Turbulence model for the temperature (energy) conservation equation.
- **[mode_calcul_convection]** (*type:* string into ["ancien", "divut_moins_tdivu", "divrhout_moins_tdivrhou"]) Option to set the form of the convective operatorNL2 divrhouT_moins_Tdivrhou (the default since 1.6.8): rho.u.gradT = div(rho.u.T)- Tdiv(rho.u.1) NL2ancien: u.gradT = div(u.T) - T.div(u) divuT_moins_Tdivu : u.gradT = div(u.T) - Tdiv(u.1)
- **[disable_equation_residual]** (*type:* string) The equation residual will not be used for the problem residual used when checking time convergence or computing dynamic time-step
- **[convection]** (*type: bloc_convection*) Keyword to alter the convection scheme.
- **[diffusion]** (*type: bloc_diffusion*) Keyword to specify the diffusion operator.
- **[conditions_limites | boundary_conditions]** (*type: condlims*) Boundary conditions.
- **[conditions_initiales | initial_conditions]** (*type: condinits*) Initial conditions.
- **[sources]** (*type: sources*) To introduce a source term into an equation (in case of several source terms into the same equation, the blocks corresponding to the various terms need to be separated by a comma)
- **[ecrire_fichier_xyz_valeur]** (*type: ecrire_fichier_xyz_valeur*) This keyword is used to write the values of a field only for some boundaries in a text file
- **[parametre_equation]** (*type: parametre_equation_base*) Keyword used to specify additional parameters for the equation
- **[equation_non_resolue]** (*type:* string) The equation will not be solved while condition(t) is verified if equation_non_resolue keyword is used. Exemple: The Navier-Stokes equations are not solved between time t0 and t1. Navier_Sokes_Standard { equation_non_resolue (t>t0)*(t<t1) }

---

convection_diffusion_chaleur_wc

Inherits from: *mor_eqn*

Temperature equation for a weakly-compressible fluid.

Parameters are:

- **[disable_equation_residual]** (*type:* string) The equation residual will not be used for the problem residual used when checking time convergence or computing dynamic time-step
- **[convection]** (*type: bloc_convection*) Keyword to alter the convection scheme.
- **[diffusion]** (*type: bloc_diffusion*) Keyword to specify the diffusion operator.
- **[conditions_limites | boundary_conditions]** (*type: condlims*) Boundary conditions.
- **[conditions_initiales | initial_conditions]** (*type: condinits*) Initial conditions.
- **[sources]** (*type: sources*) To introduce a source term into an equation (in case of several source terms into the same equation, the blocks corresponding to the various terms need to be separated by a comma)
- **[ecrire_fichier_xyz_valeur]** (*type: ecrire_fichier_xyz_valeur*) This keyword is used to write the values of a field only for some boundaries in a text file
- **[parametre_equation]** (*type: parametre_equation_base*) Keyword used to specify additional parameters for the equation
- **[equation_non_resolue]** (*type:* string) The equation will not be solved while condition(t) is verified if equation_non_resolue keyword is used. Exemple: The Navier-Stokes equations are not solved between time t0 and t1. Navier_Sokes_Standard { equation_non_resolue (t>t0)*(t<t1) }

---

convection_diffusion_concentration

Inherits from: *mor_eqn*

Constituent transport vectorial equation (concentration diffusion convection).

Parameters are:

- **[nom_inconnue]** (*type:* string) Keyword Nom_inconnue will rename the unknown of this equation with the given name. In the postprocessing part, the concentration field will be accessible with this name. This is usefull if you want to track more than one concentration (otherwise, only the concentration field in the first concentration equation can be accessed).
- **[alias]** (*type:* string) not_set
- **[masse_molaire]** (*type:* double) not_set
- **[disable_equation_residual]** (*type:* string) The equation residual will not be used for the problem residual used when checking time convergence or computing dynamic time-step
- **[convection]** (*type: bloc_convection*) Keyword to alter the convection scheme.
- **[diffusion]** (*type: bloc_diffusion*) Keyword to specify the diffusion operator.
- **[conditions_limites | boundary_conditions]** (*type: condlims*) Boundary conditions.
- **[conditions_initiales | initial_conditions]** (*type: condinits*) Initial conditions.
- **[sources]** (*type: sources*) To introduce a source term into an equation (in case of several source terms into the same equation, the blocks corresponding to the various terms need to be separated by a comma)

- **[ecrire_fichier_xyz_valeur]** (*type: ecrire_fichier_xyz_valeur*) This keyword is used to write the values of a field only for some boundaries in a text file
- **[parametre_equation]** (*type: parametre_equation_base*) Keyword used to specify additional parameters for the equation
- **[equation_non_resolue]** (*type:* string) The equation will not be solved while condition(t) is verified if equation_non_resolue keyword is used. Exemple: The Navier-Stokes equations are not solved between time t0 and t1. Navier_Sokes_Standard { equation_non_resolue (t>t0)*(t<t1) }

---

convection_diffusion_concentration_turbulent

Inherits from: *mor_eqn*

Constituent transport equations (concentration diffusion convection) as well as the associated turbulence model equations.

Parameters are:

- **[modele_turbulence]** (*type: modele_turbulence_scal_base*) Turbulence model to be used in the constituent transport equations. The only model currently available is Schmidt.
- **[nom_inconnue]** (*type:* string) Keyword Nom_inconnue will rename the unknown of this equation with the given name. In the postprocessing part, the concentration field will be accessible with this name. This is usefull if you want to track more than one concentration (otherwise, only the concentration field in the first concentration equation can be accessed).
- **[alias]** (*type:* string) not_set
- **[masse_molaire]** (*type:* double) not_set
- **[disable_equation_residual]** (*type:* string) The equation residual will not be used for the problem residual used when checking time convergence or computing dynamic time-step
- **[convection]** (*type: bloc_convection*) Keyword to alter the convection scheme.
- **[diffusion]** (*type: bloc_diffusion*) Keyword to specify the diffusion operator.
- **[conditions_limites | boundary_conditions]** (*type: condlims*) Boundary conditions.
- **[conditions_initiales | initial_conditions]** (*type: condinits*) Initial conditions.
- **[sources]** (*type: sources*) To introduce a source term into an equation (in case of several source terms into the same equation, the blocks corresponding to the various terms need to be separated by a comma)
- **[ecrire_fichier_xyz_valeur]** (*type: ecrire_fichier_xyz_valeur*) This keyword is used to write the values of a field only for some boundaries in a text file
- **[parametre_equation]** (*type: parametre_equation_base*) Keyword used to specify additional parameters for the equation
- **[equation_non_resolue]** (*type:* string) The equation will not be solved while condition(t) is verified if equation_non_resolue keyword is used. Exemple: The Navier-Stokes equations are not solved between time t0 and t1. Navier_Sokes_Standard { equation_non_resolue (t>t0)*(t<t1) }

---

convection_diffusion_espece_binaire_qc

Inherits from: *mor_eqn*

Species conservation equation for a binary quasi-compressible fluid.

Parameters are:

- **[disable_equation_residual]** (*type:* string) The equation residual will not be used for the problem residual used when checking time convergence or computing dynamic time-step
- **[convection]** (*type: bloc_convection*) Keyword to alter the convection scheme.
- **[diffusion]** (*type: bloc_diffusion*) Keyword to specify the diffusion operator.
- **[conditions_limites | boundary_conditions]** (*type: condlims*) Boundary conditions.
- **[conditions_initiales | initial_conditions]** (*type: condinits*) Initial conditions.
- **[sources]** (*type: sources*) To introduce a source term into an equation (in case of several source terms into the same equation, the blocks corresponding to the various terms need to be separated by a comma)
- **[ecrire_fichier_xyz_valeur]** (*type: ecrire_fichier_xyz_valeur*) This keyword is used to write the values of a field only for some boundaries in a text file
- **[parametre_equation]** (*type: parametre_equation_base*) Keyword used to specify additional parameters for the equation
- **[equation_non_resolue]** (*type:* string) The equation will not be solved while condition(t) is verified if equation_non_resolue keyword is used. Exemple: The Navier-Stokes equations are not solved between time t0 and t1. Navier_Sokes_Standard { equation_non_resolue (t>t0)*(t<t1) }

---

convection_diffusion_espece_binaire_turbulent_qc

Inherits from: *mor_eqn*

Species conservation equation for a binary quasi-compressible fluid as well as the associated turbulence model equations.

Parameters are:

- **[modele_turbulence]** (*type: modele_turbulence_scal_base*) Turbulence model for the species conservation equation.
- **[disable_equation_residual]** (*type:* string) The equation residual will not be used for the problem residual used when checking time convergence or computing dynamic time-step
- **[convection]** (*type: bloc_convection*) Keyword to alter the convection scheme.
- **[diffusion]** (*type: bloc_diffusion*) Keyword to specify the diffusion operator.
- **[conditions_limites | boundary_conditions]** (*type: condlims*) Boundary conditions.
- **[conditions_initiales | initial_conditions]** (*type: condinits*) Initial conditions.
- **[sources]** (*type: sources*) To introduce a source term into an equation (in case of several source terms into the same equation, the blocks corresponding to the various terms need to be separated by a comma)
- **[ecrire_fichier_xyz_valeur]** (*type: ecrire_fichier_xyz_valeur*) This keyword is used to write the values of a field only for some boundaries in a text file
- **[parametre_equation]** (*type: parametre_equation_base*) Keyword used to specify additional parameters for the equation

- **[equation_non_resolue]** (*type:* string) The equation will not be solved while condition(t) is verified if equation_non_resolue keyword is used. Exemple: The Navier-Stokes equations are not solved between time t0 and t1. Navier_Sokes_Standard { equation_non_resolue (t>t0)*(t<t1) }

---

convection_diffusion_espece_binaire_wc

Inherits from: *mor_eqn*

Species conservation equation for a binary weakly-compressible fluid.

Parameters are:

- **[disable_equation_residual]** (*type:* string) The equation residual will not be used for the problem residual used when checking time convergence or computing dynamic time-step
- **[convection]** (*type: bloc_convection*) Keyword to alter the convection scheme.
- **[diffusion]** (*type: bloc_diffusion*) Keyword to specify the diffusion operator.
- **[conditions_limites | boundary_conditions]** (*type: condlims*) Boundary conditions.
- **[conditions_initiales | initial_conditions]** (*type: condinits*) Initial conditions.
- **[sources]** (*type: sources*) To introduce a source term into an equation (in case of several source terms into the same equation, the blocks corresponding to the various terms need to be separated by a comma)
- **[ecrire_fichier_xyz_valeur]** (*type: ecrire_fichier_xyz_valeur*) This keyword is used to write the values of a field only for some boundaries in a text file
- **[parametre_equation]** (*type: parametre_equation_base*) Keyword used to specify additional parameters for the equation
- **[equation_non_resolue]** (*type:* string) The equation will not be solved while condition(t) is verified if equation_non_resolue keyword is used. Exemple: The Navier-Stokes equations are not solved between time t0 and t1. Navier_Sokes_Standard { equation_non_resolue (t>t0)*(t<t1) }

---

convection_diffusion_espece_multi_qc

Inherits from: *mor_eqn*

Species conservation equation for a multi-species quasi-compressible fluid.

Parameters are:

- **[espece]** (*type: espece*) Assosciate a species (with its properties) to the equation
- **[disable_equation_residual]** (*type:* string) The equation residual will not be used for the problem residual used when checking time convergence or computing dynamic time-step
- **[convection]** (*type: bloc_convection*) Keyword to alter the convection scheme.
- **[diffusion]** (*type: bloc_diffusion*) Keyword to specify the diffusion operator.
- **[conditions_limites | boundary_conditions]** (*type: condlims*) Boundary conditions.
- **[conditions_initiales | initial_conditions]** (*type: condinits*) Initial conditions.
- **[sources]** (*type: sources*) To introduce a source term into an equation (in case of several source terms into the same equation, the blocks corresponding to the various terms need to be separated by a comma)

- **[ecrire_fichier_xyz_valeur]** (*type: ecrire_fichier_xyz_valeur*) This keyword is used to write the values of a field only for some boundaries in a text file
- **[parametre_equation]** (*type: parametre_equation_base*) Keyword used to specify additional parameters for the equation
- **[equation_non_resolue]** (*type:* string) The equation will not be solved while condition(t) is verified if equation_non_resolue keyword is used. Exemple: The Navier-Stokes equations are not solved between time t0 and t1. Navier_Sokes_Standard { equation_non_resolue (t>t0)*(t<t1) }

---

convection_diffusion_espece_multi_turbulent_qc

Inherits from: *mor_eqn*

not_set

Parameters are:

- **[modele_turbulence]** (*type: modele_turbulence_scal_base*) Turbulence model to be used.
- **espece** (*type: espece*) not_set
- **[disable_equation_residual]** (*type:* string) The equation residual will not be used for the problem residual used when checking time convergence or computing dynamic time-step
- **[convection]** (*type: bloc_convection*) Keyword to alter the convection scheme.
- **[diffusion]** (*type: bloc_diffusion*) Keyword to specify the diffusion operator.
- **[conditions_limites | boundary_conditions]** (*type: condlims*) Boundary conditions.
- **[conditions_initiales | initial_conditions]** (*type: condinits*) Initial conditions.
- **[sources]** (*type: sources*) To introduce a source term into an equation (in case of several source terms into the same equation, the blocks corresponding to the various terms need to be separated by a comma)
- **[ecrire_fichier_xyz_valeur]** (*type: ecrire_fichier_xyz_valeur*) This keyword is used to write the values of a field only for some boundaries in a text file
- **[parametre_equation]** (*type: parametre_equation_base*) Keyword used to specify additional parameters for the equation
- **[equation_non_resolue]** (*type:* string) The equation will not be solved while condition(t) is verified if equation_non_resolue keyword is used. Exemple: The Navier-Stokes equations are not solved between time t0 and t1. Navier_Sokes_Standard { equation_non_resolue (t>t0)*(t<t1) }

---

convection_diffusion_espece_multi_wc

Inherits from: *mor_eqn*

Species conservation equation for a multi-species weakly-compressible fluid.

Parameters are:

- **[disable_equation_residual]** (*type:* string) The equation residual will not be used for the problem residual used when checking time convergence or computing dynamic time-step
- **[convection]** (*type: bloc_convection*) Keyword to alter the convection scheme.

- **[diffusion]** (*type: bloc_diffusion*) Keyword to specify the diffusion operator.
- **[conditions_limites | boundary_conditions]** (*type: condlims*) Boundary conditions.
- **[conditions_initiales | initial_conditions]** (*type: condinits*) Initial conditions.
- **[sources]** (*type: sources*) To introduce a source term into an equation (in case of several source terms into the same equation, the blocks corresponding to the various terms need to be separated by a comma)
- **[ecrire_fichier_xyz_valeur]** (*type: ecrire_fichier_xyz_valeur*) This keyword is used to write the values of a field only for some boundaries in a text file
- **[parametre_equation]** (*type: parametre_equation_base*) Keyword used to specify additional parameters for the equation
- **[equation_non_resolue]** (*type:* string) The equation will not be solved while condition(t) is verified if equation_non_resolue keyword is used. Exemple: The Navier-Stokes equations are not solved between time t0 and t1. Navier_Sokes_Standard { equation_non_resolue (t>t0)*(t<t1) }

---

convection_diffusion_temperature

Inherits from: *mor_eqn*

Energy equation (temperature diffusion convection).

Parameters are:

- **[penalisation_l2_ftd]** (*type: penalisation_l2_ftd*) to activate or not (the default is Direct Forcing method) the Penalized Direct Forcing method to impose the specified temperature on the solid-fluid interface.
- **[disable_equation_residual]** (*type:* string) The equation residual will not be used for the problem residual used when checking time convergence or computing dynamic time-step
- **[convection]** (*type: bloc_convection*) Keyword to alter the convection scheme.
- **[diffusion]** (*type: bloc_diffusion*) Keyword to specify the diffusion operator.
- **[conditions_limites | boundary_conditions]** (*type: condlims*) Boundary conditions.
- **[conditions_initiales | initial_conditions]** (*type: condinits*) Initial conditions.
- **[sources]** (*type: sources*) To introduce a source term into an equation (in case of several source terms into the same equation, the blocks corresponding to the various terms need to be separated by a comma)
- **[ecrire_fichier_xyz_valeur]** (*type: ecrire_fichier_xyz_valeur*) This keyword is used to write the values of a field only for some boundaries in a text file
- **[parametre_equation]** (*type: parametre_equation_base*) Keyword used to specify additional parameters for the equation
- **[equation_non_resolue]** (*type:* string) The equation will not be solved while condition(t) is verified if equation_non_resolue keyword is used. Exemple: The Navier-Stokes equations are not solved between time t0 and t1. Navier_Sokes_Standard { equation_non_resolue (t>t0)*(t<t1) }

---

convection_diffusion_temperature_turbulent

Inherits from: *mor_eqn*

Energy equation (temperature diffusion convection) as well as the associated turbulence model equations.

Parameters are:

- **[modele_turbulence]** (*type: modele_turbulence_scal_base*) Turbulence model for the energy equation.
- **[disable_equation_residual]** (*type:* string) The equation residual will not be used for the problem residual used when checking time convergence or computing dynamic time-step
- **[convection]** (*type: bloc_convection*) Keyword to alter the convection scheme.
- **[diffusion]** (*type: bloc_diffusion*) Keyword to specify the diffusion operator.
- **[conditions_limites | boundary_conditions]** (*type: condlims*) Boundary conditions.
- **[conditions_initiales | initial_conditions]** (*type: condinits*) Initial conditions.
- **[sources]** (*type: sources*) To introduce a source term into an equation (in case of several source terms into the same equation, the blocks corresponding to the various terms need to be separated by a comma)
- **[ecrire_fichier_xyz_valeur]** (*type: ecrire_fichier_xyz_valeur*) This keyword is used to write the values of a field only for some boundaries in a text file
- **[parametre_equation]** (*type: parametre_equation_base*) Keyword used to specify additional parameters for the equation
- **[equation_non_resolue]** (*type:* string) The equation will not be solved while condition(t) is verified if equation_non_resolue keyword is used. Exemple: The Navier-Stokes equations are not solved between time t0 and t1. Navier_Sokes_Standard { equation_non_resolue (t>t0)*(t<t1) }

---

echelle_temporelle_turbulente

Inherits from: *mor_eqn*

Turbulent Dissipation time scale equation for a turbulent mono/multi-phase problem (available in TrioCFD)

Parameters are:

- **[disable_equation_residual]** (*type:* string) The equation residual will not be used for the problem residual used when checking time convergence or computing dynamic time-step
- **[convection]** (*type: bloc_convection*) Keyword to alter the convection scheme.
- **[diffusion]** (*type: bloc_diffusion*) Keyword to specify the diffusion operator.
- **[conditions_limites | boundary_conditions]** (*type: condlims*) Boundary conditions.
- **[conditions_initiales | initial_conditions]** (*type: condinits*) Initial conditions.
- **[sources]** (*type: sources*) To introduce a source term into an equation (in case of several source terms into the same equation, the blocks corresponding to the various terms need to be separated by a comma)
- **[ecrire_fichier_xyz_valeur]** (*type: ecrire_fichier_xyz_valeur*) This keyword is used to write the values of a field only for some boundaries in a text file
- **[parametre_equation]** (*type: parametre_equation_base*) Keyword used to specify additional parameters for the equation

- **[equation_non_resolue]** (*type:* string) The equation will not be solved while condition(t) is verified if equation_non_resolue keyword is used. Exemple: The Navier-Stokes equations are not solved between time t0 and t1. Navier_Sokes_Standard { equation_non_resolue (t>t0)*(t<t1) }

---

energie_cinetique_turbulente

Inherits from: *mor_eqn*

Turbulent kinetic Energy conservation equation for a turbulent mono/multi-phase problem (available in TrioCFD)

Parameters are:

- **[disable_equation_residual]** (*type:* string) The equation residual will not be used for the problem residual used when checking time convergence or computing dynamic time-step
- **[convection]** (*type: bloc_convection*) Keyword to alter the convection scheme.
- **[diffusion]** (*type: bloc_diffusion*) Keyword to specify the diffusion operator.
- **[conditions_limites | boundary_conditions]** (*type: condlims*) Boundary conditions.
- **[conditions_initiales | initial_conditions]** (*type: condinits*) Initial conditions.
- **[sources]** (*type: sources*) To introduce a source term into an equation (in case of several source terms into the same equation, the blocks corresponding to the various terms need to be separated by a comma)
- **[ecrire_fichier_xyz_valeur]** (*type: ecrire_fichier_xyz_valeur*) This keyword is used to write the values of a field only for some boundaries in a text file
- **[parametre_equation]** (*type: parametre_equation_base*) Keyword used to specify additional parameters for the equation
- **[equation_non_resolue]** (*type:* string) The equation will not be solved while condition(t) is verified if equation_non_resolue keyword is used. Exemple: The Navier-Stokes equations are not solved between time t0 and t1. Navier_Sokes_Standard { equation_non_resolue (t>t0)*(t<t1) }

---

energie_cinetique_turbulente_wit

Inherits from: *mor_eqn*

Bubble Induced Turbulent kinetic Energy equation for a turbulent multi-phase problem (available in TrioCFD)

Parameters are:

- **[disable_equation_residual]** (*type:* string) The equation residual will not be used for the problem residual used when checking time convergence or computing dynamic time-step
- **[convection]** (*type: bloc_convection*) Keyword to alter the convection scheme.
- **[diffusion]** (*type: bloc_diffusion*) Keyword to specify the diffusion operator.
- **[conditions_limites | boundary_conditions]** (*type: condlims*) Boundary conditions.
- **[conditions_initiales | initial_conditions]** (*type: condinits*) Initial conditions.
- **[sources]** (*type: sources*) To introduce a source term into an equation (in case of several source terms into the same equation, the blocks corresponding to the various terms need to be separated by a comma)

- **[ecrire_fichier_xyz_valeur]** (*type: ecrire_fichier_xyz_valeur*) This keyword is used to write the values of a field only for some boundaries in a text file
- **[parametre_equation]** (*type: parametre_equation_base*) Keyword used to specify additional parameters for the equation
- **[equation_non_resolue]** (*type:* string) The equation will not be solved while condition(t) is verified if equation_non_resolue keyword is used. Exemple: The Navier-Stokes equations are not solved between time t0 and t1. Navier_Sokes_Standard { equation_non_resolue (t>t0)*(t<t1) }

---

energie_multiphase

Inherits from: *mor_eqn*

Internal energy conservation equation for a multi-phase problem where the unknown is the temperature

Parameters are:

- **[disable_equation_residual]** (*type:* string) The equation residual will not be used for the problem residual used when checking time convergence or computing dynamic time-step
- **[convection]** (*type: bloc_convection*) Keyword to alter the convection scheme.
- **[diffusion]** (*type: bloc_diffusion*) Keyword to specify the diffusion operator.
- **[conditions_limites | boundary_conditions]** (*type: condlims*) Boundary conditions.
- **[conditions_initiales | initial_conditions]** (*type: condinits*) Initial conditions.
- **[sources]** (*type: sources*) To introduce a source term into an equation (in case of several source terms into the same equation, the blocks corresponding to the various terms need to be separated by a comma)
- **[ecrire_fichier_xyz_valeur]** (*type: ecrire_fichier_xyz_valeur*) This keyword is used to write the values of a field only for some boundaries in a text file
- **[parametre_equation]** (*type: parametre_equation_base*) Keyword used to specify additional parameters for the equation
- **[equation_non_resolue]** (*type:* string) The equation will not be solved while condition(t) is verified if equation_non_resolue keyword is used. Exemple: The Navier-Stokes equations are not solved between time t0 and t1. Navier_Sokes_Standard { equation_non_resolue (t>t0)*(t<t1) }

---

eqn_base

Inherits from: *mor_eqn*

Basic class for equations.

Parameters are:

- **[disable_equation_residual]** (*type:* string) The equation residual will not be used for the problem residual used when checking time convergence or computing dynamic time-step
- **[convection]** (*type: bloc_convection*) Keyword to alter the convection scheme.
- **[diffusion]** (*type: bloc_diffusion*) Keyword to specify the diffusion operator.
- **[conditions_limites | boundary_conditions]** (*type: condlims*) Boundary conditions.

- **[conditions_initiales | initial_conditions]** (*type: condinits*) Initial conditions.
- **[sources]** (*type: sources*) To introduce a source term into an equation (in case of several source terms into the same equation, the blocks corresponding to the various terms need to be separated by a comma)
- **[ecrire_fichier_xyz_valeur]** (*type: ecrire_fichier_xyz_valeur*) This keyword is used to write the values of a field only for some boundaries in a text file
- **[parametre_equation]** (*type: parametre_equation_base*) Keyword used to specify additional parameters for the equation
- **[equation_non_resolue]** (*type:* string) The equation will not be solved while condition(t) is verified if equation_non_resolue keyword is used. Exemple: The Navier-Stokes equations are not solved between time t0 and t1. Navier_Sokes_Standard { equation_non_resolue (t>t0)*(t<t1) }

---

masse_multiphase

Inherits from: *mor_eqn*

Mass consevation equation for a multi-phase problem where the unknown is the alpha (void fraction)

Parameters are:

- **[disable_equation_residual]** (*type:* string) The equation residual will not be used for the problem residual used when checking time convergence or computing dynamic time-step
- **[convection]** (*type: bloc_convection*) Keyword to alter the convection scheme.
- **[diffusion]** (*type: bloc_diffusion*) Keyword to specify the diffusion operator.
- **[conditions_limites | boundary_conditions]** (*type: condlims*) Boundary conditions.
- **[conditions_initiales | initial_conditions]** (*type: condinits*) Initial conditions.
- **[sources]** (*type: sources*) To introduce a source term into an equation (in case of several source terms into the same equation, the blocks corresponding to the various terms need to be separated by a comma)
- **[ecrire_fichier_xyz_valeur]** (*type: ecrire_fichier_xyz_valeur*) This keyword is used to write the values of a field only for some boundaries in a text file
- **[parametre_equation]** (*type: parametre_equation_base*) Keyword used to specify additional parameters for the equation
- **[equation_non_resolue]** (*type:* string) The equation will not be solved while condition(t) is verified if equation_non_resolue keyword is used. Exemple: The Navier-Stokes equations are not solved between time t0 and t1. Navier_Sokes_Standard { equation_non_resolue (t>t0)*(t<t1) }

---

mor_eqn

Class of equation pieces (morceaux d'equation).

---

navier_stokes_qc

Inherits from: *mor_eqn*

Navier-Stokes equation for a quasi-compressible fluid.

Parameters are:

- **[methode_calcul_pression_initiale]** (*type:* string into ["avec_les_cl", "avec_sources", "avec_sources_et_operateurs", "sans_rien"]) Keyword to select an option for the pressure calculation before the fist time step. Options are : avec_les_cl (default option lapP=0 is solved with Neuman boundary conditions on pressure if any), avec_sources (lapP=f is solved with Neuman boundaries conditions and f integrating the source terms of the Navier-Stokes equations) and avec_sources_et_operateurs (lapP=f is solved as with the previous option avec_sources but f integrating also some operators of the Navier-Stokes equations). The two last options are useful and sometime necessary when source terms are implicited when using an implicit time scheme to solve the Navier-Stokes equations.
- **[projection_initiale]** (*type:* int) Keyword to suppress, if boolean equals 0, the initial projection which checks DivU=0. By default, boolean equals 1.
- **[solveur_pression]** (*type: solveur_sys_base*) Linear pressure system resolution method.
- **[solveur_bar]** (*type: solveur_sys_base*) This keyword is used to define when filtering operation is called (typically for EF convective scheme, standard diffusion operator and Source_Qdm_lambdaup). A file (solveur.bar) is then created and used for inversion procedure. Syntax is the same then for pressure solver (GCP is required for multi-processor calculations and, in a general way, for big meshes).
- **[dt_projection]** (*type: deuxmots*) nb value : This keyword checks every nb time-steps the equality of velocity divergence to zero. value is the criteria convergency for the solver used.
- **[seuil_divu]** (*type: floatfloat*) value factor : this keyword is intended to minimise the number of iterations during the pressure system resolution. The convergence criteria during this step ('seuil' in solveur_pression) is dynamically adapted according to the mass conservation. At tn , the linear system Ax=B is considered as solved if the residual ||Ax-B||<seuil(tn). For tn+1, the threshold value seuil(tn+1) will be evualated as: If (|max(DivU)*dt|<value) Seuil(tn+1)= Seuil(tn)*factor Else Seuil(tn+1)= Seuil(tn)*factor Endif The first parameter (value) is the mass evolution the user is ready to accept per timestep, and the second one (factor) is the factor of evolution for 'seuil' (for example 1.1, so 10% per timestep). Investigations has to be lead to know more about the effects of these two last parameters on the behaviour of the simulations.
- **[traitement_particulier]** (*type: traitement_particulier*) Keyword to post-process particular values.
- **[correction_matrice_projection_initiale]** (*type:* int) (IBM advanced) fix matrix of initial projection for PDF
- **[correction_calcul_pression_initiale]** (*type:* int) (IBM advanced) fix initial pressure computation for PDF
- **[correction_vitesse_projection_initiale]** (*type:* int) (IBM advanced) fix initial velocity computation for PDF
- **[correction_matrice_pression]** (*type:* int) (IBM advanced) fix pressure matrix for PDF
- **[correction_vitesse_modifie]** (*type:* int) (IBM advanced) fix velocity for PDF
- **[gradient_pression_qdm_modifie]** (*type:* int) (IBM advanced) fix pressure gradient
- **[correction_pression_modifie]** (*type:* int) (IBM advanced) fix pressure for PDF
- **[postraiter_gradient_pression_sans_masse]** (*type:* flag) (IBM advanced) avoid mass matrix multiplication for the gradient postprocessing
- **[disable_equation_residual]** (*type:* string) The equation residual will not be used for the problem residual used when checking time convergence or computing dynamic time-step
- **[convection]** (*type: bloc_convection*) Keyword to alter the convection scheme.
- **[diffusion]** (*type: bloc_diffusion*) Keyword to specify the diffusion operator.

- **[conditions_limites | boundary_conditions]** (*type: condlims*) Boundary conditions.
- **[conditions_initiales | initial_conditions]** (*type: condinits*) Initial conditions.
- **[sources]** (*type: sources*) To introduce a source term into an equation (in case of several source terms into the same equation, the blocks corresponding to the various terms need to be separated by a comma)
- **[ecrire_fichier_xyz_valeur]** (*type: ecrire_fichier_xyz_valeur*) This keyword is used to write the values of a field only for some boundaries in a text file
- **[parametre_equation]** (*type: parametre_equation_base*) Keyword used to specify additional parameters for the equation
- **[equation_non_resolue]** (*type:* string) The equation will not be solved while condition(t) is verified if equation_non_resolue keyword is used. Exemple: The Navier-Stokes equations are not solved between time t0 and t1. Navier_Sokes_Standard { equation_non_resolue (t>t0)*(t<t1) }

---

navier_stokes_standard

Inherits from: *mor_eqn*

Navier-Stokes equations.

Parameters are:

- **[methode_calcul_pression_initiale]** (*type:* string into ["avec_les_cl", "avec_sources", "avec_sources_et_operateurs", "sans_rien"]) Keyword to select an option for the pressure calculation before the fist time step. Options are : avec_les_cl (default option lapP=0 is solved with Neuman boundary conditions on pressure if any), avec_sources (lapP=f is solved with Neuman boundaries conditions and f integrating the source terms of the Navier-Stokes equations) and avec_sources_et_operateurs (lapP=f is solved as with the previous option avec_sources but f integrating also some operators of the Navier-Stokes equations). The two last options are useful and sometime necessary when source terms are implicited when using an implicit time scheme to solve the Navier-Stokes equations.
- **[projection_initiale]** (*type:* int) Keyword to suppress, if boolean equals 0, the initial projection which checks DivU=0. By default, boolean equals 1.
- **[solveur_pression]** (*type: solveur_sys_base*) Linear pressure system resolution method.
- **[solveur_bar]** (*type: solveur_sys_base*) This keyword is used to define when filtering operation is called (typically for EF convective scheme, standard diffusion operator and Source_Qdm_lambdaup). A file (solveur.bar) is then created and used for inversion procedure. Syntax is the same then for pressure solver (GCP is required for multi-processor calculations and, in a general way, for big meshes).
- **[dt_projection]** (*type: deuxmots*) nb value : This keyword checks every nb time-steps the equality of velocity divergence to zero. value is the criteria convergency for the solver used.
- **[seuil_divu]** (*type: floatfloat*) value factor : this keyword is intended to minimise the number of iterations during the pressure system resolution. The convergence criteria during this step ('seuil' in solveur_pression) is dynamically adapted according to the mass conservation. At tn , the linear system Ax=B is considered as solved if the residual ||Ax-B||<seuil(tn). For tn+1, the threshold value seuil(tn+1) will be evualated as: If (|max(DivU)*dt|<value) Seuil(tn+1)= Seuil(tn)*factor Else Seuil(tn+1)= Seuil(tn)*factor Endif The first parameter (value) is the mass evolution the user is ready to accept per timestep, and the second one (factor) is the factor of evolution for 'seuil' (for example 1.1, so 10% per timestep). Investigations has to be lead to know more about the effects of these two last parameters on the behaviour of the simulations.
- **[traitement_particulier]** (*type: traitement_particulier*) Keyword to post-process particular values.
- **[correction_matrice_projection_initiale]** (*type:* int) (IBM advanced) fix matrix of initial projection for PDF

- **[correction_calcul_pression_initiale]** (*type:* int) (IBM advanced) fix initial pressure computation for PDF
- **[correction_vitesse_projection_initiale]** (*type:* int) (IBM advanced) fix initial velocity computation for PDF
- **[correction_matrice_pression]** (*type:* int) (IBM advanced) fix pressure matrix for PDF
- **[correction_vitesse_modifie]** (*type:* int) (IBM advanced) fix velocity for PDF
- **[gradient_pression_qdm_modifie]** (*type:* int) (IBM advanced) fix pressure gradient
- **[correction_pression_modifie]** (*type:* int) (IBM advanced) fix pressure for PDF
- **[postraiter_gradient_pression_sans_masse]** (*type:* flag) (IBM advanced) avoid mass matrix multiplication for the gradient postprocessing
- **[disable_equation_residual]** (*type:* string) The equation residual will not be used for the problem residual used when checking time convergence or computing dynamic time-step
- **[convection]** (*type: bloc_convection*) Keyword to alter the convection scheme.
- **[diffusion]** (*type: bloc_diffusion*) Keyword to specify the diffusion operator.
- **[conditions_limites | boundary_conditions]** (*type: condlims*) Boundary conditions.
- **[conditions_initiales | initial_conditions]** (*type: condinits*) Initial conditions.
- **[sources]** (*type: sources*) To introduce a source term into an equation (in case of several source terms into the same equation, the blocks corresponding to the various terms need to be separated by a comma)
- **[ecrire_fichier_xyz_valeur]** (*type: ecrire_fichier_xyz_valeur*) This keyword is used to write the values of a field only for some boundaries in a text file
- **[parametre_equation]** (*type: parametre_equation_base*) Keyword used to specify additional parameters for the equation
- **[equation_non_resolue]** (*type:* string) The equation will not be solved while condition(t) is verified if equation_non_resolue keyword is used. Exemple: The Navier-Stokes equations are not solved between time t0 and t1. Navier_Sokes_Standard { equation_non_resolue (t>t0)*(t<t1) }

---

navier_stokes_turbulent

Inherits from: *mor_eqn*

Navier-Stokes equations as well as the associated turbulence model equations.

Parameters are:

- **[modele_turbulence]** (*type: modele_turbulence_hyd_deriv*) Turbulence model for Navier-Stokes equations.
- **[methode_calcul_pression_initiale]** (*type:* string into ["avec_les_cl", "avec_sources", "avec_sources_et_operateurs", "sans_rien"]) Keyword to select an option for the pressure calculation before the fist time step. Options are : avec_les_cl (default option lapP=0 is solved with Neuman boundary conditions on pressure if any), avec_sources (lapP=f is solved with Neuman boundaries conditions and f integrating the source terms of the Navier-Stokes equations) and avec_sources_et_operateurs (lapP=f is solved as with the previous option avec_sources but f integrating also some operators of the Navier-Stokes equations). The two last options are useful and sometime necessary when source terms are implicited when using an implicit time scheme to solve the Navier-Stokes equations.
- **[projection_initiale]** (*type:* int) Keyword to suppress, if boolean equals 0, the initial projection which checks DivU=0. By default, boolean equals 1.
- **[solveur_pression]** (*type: solveur_sys_base*) Linear pressure system resolution method.

- **[solveur_bar]** (*type: solveur_sys_base*) This keyword is used to define when filtering operation is called (typically for EF convective scheme, standard diffusion operator and Source_Qdm_lambdaup). A file (solveur.bar) is then created and used for inversion procedure. Syntax is the same then for pressure solver (GCP is required for multi-processor calculations and, in a general way, for big meshes).
- **[dt_projection]** (*type: deuxmots*) nb value : This keyword checks every nb time-steps the equality of velocity divergence to zero. value is the criteria convergency for the solver used.
- **[seuil_divu]** (*type: floatfloat*) value factor : this keyword is intended to minimise the number of iterations during the pressure system resolution. The convergence criteria during this step ('seuil' in solveur_pression) is dynamically adapted according to the mass conservation. At tn , the linear system Ax=B is considered as solved if the residual ||Ax-B||<seuil(tn). For tn+1, the threshold value seuil(tn+1) will be evualated as: If (|max(DivU)*dt|<value) Seuil(tn+1)= Seuil(tn)*factor Else Seuil(tn+1)= Seuil(tn)*factor Endif The first parameter (value) is the mass evolution the user is ready to accept per timestep, and the second one (factor) is the factor of evolution for 'seuil' (for example 1.1, so 10% per timestep). Investigations has to be lead to know more about the effects of these two last parameters on the behaviour of the simulations.
- **[traitement_particulier]** (*type: traitement_particulier*) Keyword to post-process particular values.
- **[correction_matrice_projection_initiale]** (*type:* int) (IBM advanced) fix matrix of initial projection for PDF
- **[correction_calcul_pression_initiale]** (*type:* int) (IBM advanced) fix initial pressure computation for PDF
- **[correction_vitesse_projection_initiale]** (*type:* int) (IBM advanced) fix initial velocity computation for PDF
- **[correction_matrice_pression]** (*type:* int) (IBM advanced) fix pressure matrix for PDF
- **[correction_vitesse_modifie]** (*type:* int) (IBM advanced) fix velocity for PDF
- **[gradient_pression_qdm_modifie]** (*type:* int) (IBM advanced) fix pressure gradient
- **[correction_pression_modifie]** (*type:* int) (IBM advanced) fix pressure for PDF
- **[postraiter_gradient_pression_sans_masse]** (*type:* flag) (IBM advanced) avoid mass matrix multiplication for the gradient postprocessing
- **[disable_equation_residual]** (*type:* string) The equation residual will not be used for the problem residual used when checking time convergence or computing dynamic time-step
- **[convection]** (*type: bloc_convection*) Keyword to alter the convection scheme.
- **[diffusion]** (*type: bloc_diffusion*) Keyword to specify the diffusion operator.
- **[conditions_limites | boundary_conditions]** (*type: condlims*) Boundary conditions.
- **[conditions_initiales | initial_conditions]** (*type: condinits*) Initial conditions.
- **[sources]** (*type: sources*) To introduce a source term into an equation (in case of several source terms into the same equation, the blocks corresponding to the various terms need to be separated by a comma)
- **[ecrire_fichier_xyz_valeur]** (*type: ecrire_fichier_xyz_valeur*) This keyword is used to write the values of a field only for some boundaries in a text file
- **[parametre_equation]** (*type: parametre_equation_base*) Keyword used to specify additional parameters for the equation
- **[equation_non_resolue]** (*type:* string) The equation will not be solved while condition(t) is verified if equation_non_resolue keyword is used. Exemple: The Navier-Stokes equations are not solved between time t0 and t1. Navier_Sokes_Standard { equation_non_resolue (t>t0)*(t<t1) }

navier_stokes_turbulent_qc

Inherits from: *mor_eqn*

Navier-Stokes equations under low Mach number as well as the associated turbulence model equations.

Parameters are:

- **[modele_turbulence]** (*type: modele_turbulence_hyd_deriv*) Turbulence model for Navier-Stokes equations.
- **[methode_calcul_pression_initiale]** (*type:* string into ["avec_les_cl", "avec_sources", "avec_sources_et_operateurs", "sans_rien"]) Keyword to select an option for the pressure calculation before the fist time step. Options are : avec_les_cl (default option lapP=0 is solved with Neuman boundary conditions on pressure if any), avec_sources (lapP=f is solved with Neuman boundaries conditions and f integrating the source terms of the Navier-Stokes equations) and avec_sources_et_operateurs (lapP=f is solved as with the previous option avec_sources but f integrating also some operators of the Navier-Stokes equations). The two last options are useful and sometime necessary when source terms are implicited when using an implicit time scheme to solve the Navier-Stokes equations.
- **[projection_initiale]** (*type:* int) Keyword to suppress, if boolean equals 0, the initial projection which checks DivU=0. By default, boolean equals 1.
- **[solveur_pression]** (*type: solveur_sys_base*) Linear pressure system resolution method.
- **[solveur_bar]** (*type: solveur_sys_base*) This keyword is used to define when filtering operation is called (typically for EF convective scheme, standard diffusion operator and Source_Qdm_lambdaup). A file (solveur.bar) is then created and used for inversion procedure. Syntax is the same then for pressure solver (GCP is required for multi-processor calculations and, in a general way, for big meshes).
- **[dt_projection]** (*type: deuxmots*) nb value : This keyword checks every nb time-steps the equality of velocity divergence to zero. value is the criteria convergency for the solver used.
- **[seuil_divu]** (*type: floatfloat*) value factor : this keyword is intended to minimise the number of iterations during the pressure system resolution. The convergence criteria during this step ('seuil' in solveur_pression) is dynamically adapted according to the mass conservation. At tn , the linear system Ax=B is considered as solved if the residual ||Ax-B||<seuil(tn). For tn+1, the threshold value seuil(tn+1) will be evualated as: If (|max(DivU)*dt|<value) Seuil(tn+1)= Seuil(tn)*factor Else Seuil(tn+1)= Seuil(tn)*factor Endif The first parameter (value) is the mass evolution the user is ready to accept per timestep, and the second one (factor) is the factor of evolution for 'seuil' (for example 1.1, so 10% per timestep). Investigations has to be lead to know more about the effects of these two last parameters on the behaviour of the simulations.
- **[traitement_particulier]** (*type: traitement_particulier*) Keyword to post-process particular values.
- **[correction_matrice_projection_initiale]** (*type:* int) (IBM advanced) fix matrix of initial projection for PDF
- **[correction_calcul_pression_initiale]** (*type:* int) (IBM advanced) fix initial pressure computation for PDF
- **[correction_vitesse_projection_initiale]** (*type:* int) (IBM advanced) fix initial velocity computation for PDF
- **[correction_matrice_pression]** (*type:* int) (IBM advanced) fix pressure matrix for PDF
- **[correction_vitesse_modifie]** (*type:* int) (IBM advanced) fix velocity for PDF
- **[gradient_pression_qdm_modifie]** (*type:* int) (IBM advanced) fix pressure gradient
- **[correction_pression_modifie]** (*type:* int) (IBM advanced) fix pressure for PDF
- **[postraiter_gradient_pression_sans_masse]** (*type:* flag) (IBM advanced) avoid mass matrix multiplication for the gradient postprocessing
- **[disable_equation_residual]** (*type:* string) The equation residual will not be used for the problem residual used when checking time convergence or computing dynamic time-step
- **[convection]** (*type: bloc_convection*) Keyword to alter the convection scheme.

- **[diffusion]** (*type: bloc_diffusion*) Keyword to specify the diffusion operator.
- **[conditions_limites | boundary_conditions]** (*type: condlims*) Boundary conditions.
- **[conditions_initiales | initial_conditions]** (*type: condinits*) Initial conditions.
- **[sources]** (*type: sources*) To introduce a source term into an equation (in case of several source terms into the same equation, the blocks corresponding to the various terms need to be separated by a comma)
- **[ecrire_fichier_xyz_valeur]** (*type: ecrire_fichier_xyz_valeur*) This keyword is used to write the values of a field only for some boundaries in a text file
- **[parametre_equation]** (*type: parametre_equation_base*) Keyword used to specify additional parameters for the equation
- **[equation_non_resolue]** (*type:* string) The equation will not be solved while condition(t) is verified if equation_non_resolue keyword is used. Exemple: The Navier-Stokes equations are not solved between time t0 and t1. Navier_Sokes_Standard { equation_non_resolue (t>t0)*(t<t1) }

---

navier_stokes_wc

Inherits from: *mor_eqn*

Navier-Stokes equation for a weakly-compressible fluid.

Parameters are:

- **[methode_calcul_pression_initiale]** (*type:* string into ["avec_les_cl", "avec_sources", "avec_sources_et_operateurs", "sans_rien"]) Keyword to select an option for the pressure calculation before the fist time step. Options are : avec_les_cl (default option lapP=0 is solved with Neuman boundary conditions on pressure if any), avec_sources (lapP=f is solved with Neuman boundaries conditions and f integrating the source terms of the Navier-Stokes equations) and avec_sources_et_operateurs (lapP=f is solved as with the previous option avec_sources but f integrating also some operators of the Navier-Stokes equations). The two last options are useful and sometime necessary when source terms are implicited when using an implicit time scheme to solve the Navier-Stokes equations.
- **[projection_initiale]** (*type:* int) Keyword to suppress, if boolean equals 0, the initial projection which checks DivU=0. By default, boolean equals 1.
- **[solveur_pression]** (*type: solveur_sys_base*) Linear pressure system resolution method.
- **[solveur_bar]** (*type: solveur_sys_base*) This keyword is used to define when filtering operation is called (typically for EF convective scheme, standard diffusion operator and Source_Qdm_lambdaup). A file (solveur.bar) is then created and used for inversion procedure. Syntax is the same then for pressure solver (GCP is required for multi-processor calculations and, in a general way, for big meshes).
- **[dt_projection]** (*type: deuxmots*) nb value : This keyword checks every nb time-steps the equality of velocity divergence to zero. value is the criteria convergency for the solver used.
- **[seuil_divu]** (*type: floatfloat*) value factor : this keyword is intended to minimise the number of iterations during the pressure system resolution. The convergence criteria during this step ('seuil' in solveur_pression) is dynamically adapted according to the mass conservation. At tn , the linear system Ax=B is considered as solved if the residual ||Ax-B||<seuil(tn). For tn+1, the threshold value seuil(tn+1) will be evualated as: If (|max(DivU)*dt|<value) Seuil(tn+1)= Seuil(tn)*factor Else Seuil(tn+1)= Seuil(tn)*factor Endif The first parameter (value) is the mass evolution the user is ready to accept per timestep, and the second one (factor) is the factor of evolution for 'seuil' (for example 1.1, so 10% per timestep). Investigations has to be lead to know more about the effects of these two last parameters on the behaviour of the simulations.
- **[traitement_particulier]** (*type: traitement_particulier*) Keyword to post-process particular values.

- **[correction_matrice_projection_initiale]** (*type:* int) (IBM advanced) fix matrix of initial projection for PDF
- **[correction_calcul_pression_initiale]** (*type:* int) (IBM advanced) fix initial pressure computation for PDF
- **[correction_vitesse_projection_initiale]** (*type:* int) (IBM advanced) fix initial velocity computation for PDF
- **[correction_matrice_pression]** (*type:* int) (IBM advanced) fix pressure matrix for PDF
- **[correction_vitesse_modifie]** (*type:* int) (IBM advanced) fix velocity for PDF
- **[gradient_pression_qdm_modifie]** (*type:* int) (IBM advanced) fix pressure gradient
- **[correction_pression_modifie]** (*type:* int) (IBM advanced) fix pressure for PDF
- **[postraiter_gradient_pression_sans_masse]** (*type:* flag) (IBM advanced) avoid mass matrix multiplication for the gradient postprocessing
- **[disable_equation_residual]** (*type:* string) The equation residual will not be used for the problem residual used when checking time convergence or computing dynamic time-step
- **[convection]** (*type: bloc_convection*) Keyword to alter the convection scheme.
- **[diffusion]** (*type: bloc_diffusion*) Keyword to specify the diffusion operator.
- **[conditions_limites | boundary_conditions]** (*type: condlims*) Boundary conditions.
- **[conditions_initiales | initial_conditions]** (*type: condinits*) Initial conditions.
- **[sources]** (*type: sources*) To introduce a source term into an equation (in case of several source terms into the same equation, the blocks corresponding to the various terms need to be separated by a comma)
- **[ecrire_fichier_xyz_valeur]** (*type: ecrire_fichier_xyz_valeur*) This keyword is used to write the values of a field only for some boundaries in a text file
- **[parametre_equation]** (*type: parametre_equation_base*) Keyword used to specify additional parameters for the equation
- **[equation_non_resolue]** (*type:* string) The equation will not be solved while condition(t) is verified if equation_non_resolue keyword is used. Exemple: The Navier-Stokes equations are not solved between time t0 and t1. Navier_Sokes_Standard { equation_non_resolue (t>t0)*(t<t1) }

---

qdm_multiphase

Inherits from: *mor_eqn*

Momentum conservation equation for a multi-phase problem where the unknown is the velocity

Parameters are:

- **[solveur_pression]** (*type: solveur_sys_base*) Linear pressure system resolution method.
- **[evanescence]** (*type: bloc_lecture*) Management of the vanishing phase (when alpha tends to 0 or 1)
- **[disable_equation_residual]** (*type:* string) The equation residual will not be used for the problem residual used when checking time convergence or computing dynamic time-step
- **[convection]** (*type: bloc_convection*) Keyword to alter the convection scheme.
- **[diffusion]** (*type: bloc_diffusion*) Keyword to specify the diffusion operator.
- **[conditions_limites | boundary_conditions]** (*type: condlims*) Boundary conditions.
- **[conditions_initiales | initial_conditions]** (*type: condinits*) Initial conditions.

- **[sources]** (*type: sources*) To introduce a source term into an equation (in case of several source terms into the same equation, the blocks corresponding to the various terms need to be separated by a comma)
- **[ecrire_fichier_xyz_valeur]** (*type: ecrire_fichier_xyz_valeur*) This keyword is used to write the values of a field only for some boundaries in a text file
- **[parametre_equation]** (*type: parametre_equation_base*) Keyword used to specify additional parameters for the equation
- **[equation_non_resolue]** (*type:* string) The equation will not be solved while condition(t) is verified if equation_non_resolue keyword is used. Exemple: The Navier-Stokes equations are not solved between time t0 and t1. Navier_Sokes_Standard { equation_non_resolue (t>t0)*(t<t1) }

---

taux_dissipation_turbulent

Inherits from: *mor_eqn*

Turbulent Dissipation frequency equation for a turbulent mono/multi-phase problem (available in TrioCFD)

Parameters are:

- **[disable_equation_residual]** (*type:* string) The equation residual will not be used for the problem residual used when checking time convergence or computing dynamic time-step
- **[convection]** (*type: bloc_convection*) Keyword to alter the convection scheme.
- **[diffusion]** (*type: bloc_diffusion*) Keyword to specify the diffusion operator.
- **[conditions_limites | boundary_conditions]** (*type: condlims*) Boundary conditions.
- **[conditions_initiales | initial_conditions]** (*type: condinits*) Initial conditions.
- **[sources]** (*type: sources*) To introduce a source term into an equation (in case of several source terms into the same equation, the blocks corresponding to the various terms need to be separated by a comma)
- **[ecrire_fichier_xyz_valeur]** (*type: ecrire_fichier_xyz_valeur*) This keyword is used to write the values of a field only for some boundaries in a text file
- **[parametre_equation]** (*type: parametre_equation_base*) Keyword used to specify additional parameters for the equation
- **[equation_non_resolue]** (*type:* string) The equation will not be solved while condition(t) is verified if equation_non_resolue keyword is used. Exemple: The Navier-Stokes equations are not solved between time t0 and t1. Navier_Sokes_Standard { equation_non_resolue (t>t0)*(t<t1) }

2.1.45 Keywords derived from nom

nom

Class to name the TRUST objects.

Parameters are:

- **[mot]** (*type:* string) Chain of characters.

---

nom_anonyme

Inherits from: *nom*

not_set

Parameters are:

- **[mot]** (*type:* string) Chain of characters.

2.1.46 Keywords derived from objet_lecture

bloc_convection

Inherits from: *objet_lecture*

not_set

Parameters are:

- **aco** (*type:* string into ["{"]) Opening curly bracket.
- **operateur** (*type:* *convection_deriv*) not_set
- **acof** (*type:* string into ["}"]) Closing curly bracket.

---

bloc_couronne

Inherits from: *objet_lecture*

Class to create a couronne (2D).

Parameters are:

- **name** (*type:* string into ["origine"]) Keyword to define the center of the circle.
- **origin | origine** (*type:* double list) Center of the circle.
- **name3** (*type:* string into ["ri"]) Keyword to define the interior radius.
- **ri** (*type:* double) Interior radius.
- **name4** (*type:* string into ["re"]) Keyword to define the exterior radius.
- **re** (*type:* double) Exterior radius.

---

bloc_criteres_convergence

Inherits from: *objet_lecture*

Not set

Parameters are:

- **bloc_lecture** (*type:* string) not_set

---

bloc_decouper

Inherits from: *objet_lecture*

Auxiliary class to cut a domain.

Parameters are:

- **[partitionneur | partition_tool]** (*type: partitionneur_deriv*) Defines the partitionning algorithm (the effective C++ object used is 'Partitionneur_ALGORITHM_NAME').
- **[larg_joint]** (*type:* int) This keyword specifies the thickness of the virtual ghost domaine (data known by one processor though not owned by it). The default value is 1 and is generally correct for all algorithms except the QUICK convection scheme that require a thickness of 2. Since the 1.5.5 version, the VEF discretization imply also a thickness of 2 (except VEF P0). Any non-zero positive value can be used, but the amount of data to store and exchange between processors grows quickly with the thickness.
- **[nom_zones | zones_name]** (*type:* string) Name of the files containing the different partition of the domain. The files will be : name_0001.Zones name_0002.Zones ... name_000n.Zones. If this keyword is not specified, the geometry is not written on disk (you might just want to generate a 'ecrire_decoupage' or 'ecrire_lata').
- **[ecrire_decoupage]** (*type:* string) After having called the partitionning algorithm, the resulting partition is written on disk in the specified filename. See also partitionneur Fichier_Decoupage. This keyword is useful to change the partition numbers: first, you write the partition into a file with the option ecrire_decoupage. This file contains the domaine number for each element's mesh. Then you can easily permute domaine numbers in this file. Then read the new partition to create the .Zones files with the Fichier_Decoupage keyword.
- **[ecrire_lata]** (*type:* string) Save the partition field in a LATA format file for visualization
- **[ecrire_med]** (*type:* string) Save the partition field in a MED format file for visualization
- **[nb_parts_tot]** (*type:* int) Keyword to generates N .Domaine files, instead of the default number M obtained after the partitionning algorithm. N must be greater or equal to M. This option might be used to perform coupled parallel computations. Supplemental empty domaines from M to N-1 are created. This keyword is used when you want to run a parallel calculation on several domains with for example, 2 processors on a first domain and 10 on the second domain because the first domain is very small compare to second one. You will write Nb_parts 2 and Nb_parts_tot 10 for the first domain and Nb_parts 10 for the second domain.
- **[periodique]** (*type:* string list) N BOUNDARY_NAME_1 BOUNDARY_NAME_2 ... : N is the number of boundary names given. Periodic boundaries must be declared by this method. The partitionning algorithm will ensure that facing nodes and faces in the periodic boundaries are located on the same processor.
- **[reorder]** (*type:* int) If this option is set to 1 (0 by default), the partition is renumbered in order that the processes which communicate the most are nearer on the network. This may slighlty improves parallel performance.
- **[single_hdf]** (*type:* flag) Optional keyword to enable you to write the partitioned domaines in a single file in hdf5 format.
- **[print_more_infos]** (*type:* int) If this option is set to 1 (0 by default), print infos about number of remote elements (ghosts) and additional infos about the quality of partitionning. Warning, it slows down the cutting operations.

bloc_diffusion

Inherits from: *objet_lecture*

not_set

Parameters are:

- **aco** (*type:* string into [“{“]) Opening curly bracket.
- **[operateur]** (*type: diffusion_deriv*) if none is specified, the diffusive scheme used is a 2nd-order scheme.
- **[op_implicite]** (*type: op_implicite*) To have diffusive implicitation, it use Uzawa algorithm. Very useful when viscosity has large variations.
- **acof** (*type:* string into [“}”]) Closing curly bracket.

---

bloc_diffusion_standard

Inherits from: *objet_lecture*

grad_Ubar 1 makes the gradient calculated through the filtered values of velocity (P1-conform).NL2 nu 1 (respectively nut 1) takes the molecular viscosity (eddy viscosity) into account in the velocity gradient part of the diffusion expression.

nu_transp 1 (respectively nut_transp 1) takes the molecular viscosity (eddy viscosity) into account according in the TRANSPOSED velocity gradient part of the diffusion expression.NL2 filtrer_resu 1 allows to filter the resulting diffusive fluxes contribution.

Parameters are:

- **mot1** (*type:* string into [“grad_ubar”, “nu”, “nut”, “nu_transp”, “nut_transp”, “filtrer_resu”]) not_set
- **val1** (*type:* int into [0,1]) not_set
- **mot2** (*type:* string into [“grad_ubar”, “nu”, “nut”, “nu_transp”, “nut_transp”, “filtrer_resu”]) not_set
- **val2** (*type:* int into [0,1]) not_set
- **mot3** (*type:* string into [“grad_ubar”, “nu”, “nut”, “nu_transp”, “nut_transp”, “filtrer_resu”]) not_set
- **val3** (*type:* int into [0,1]) not_set
- **mot4** (*type:* string into [“grad_ubar”, “nu”, “nut”, “nu_transp”, “nut_transp”, “filtrer_resu”]) not_set
- **val4** (*type:* int into [0,1]) not_set
- **mot5** (*type:* string into [“grad_ubar”, “nu”, “nut”, “nu_transp”, “nut_transp”, “filtrer_resu”]) not_set
- **val5** (*type:* int into [0,1]) not_set
- **mot6** (*type:* string into [“grad_ubar”, “nu”, “nut”, “nu_transp”, “nut_transp”, “filtrer_resu”]) not_set
- **val6** (*type:* int into [0,1]) not_set

---

bloc_ef

Inherits from: *objet_lecture*

not_set

Parameters are:

- **mot1** (*type:* string into ["transportant_bar", "transporte_bar", "filtrer_resu", "antisym"]) not_set
- **val1** (*type:* int into [0,1]) not_set
- **mot2** (*type:* string into ["transportant_bar", "transporte_bar", "filtrer_resu", "antisym"]) not_set
- **val2** (*type:* int into [0,1]) not_set
- **mot3** (*type:* string into ["transportant_bar", "transporte_bar", "filtrer_resu", "antisym"]) not_set
- **val3** (*type:* int into [0,1]) not_set
- **mot4** (*type:* string into ["transportant_bar", "transporte_bar", "filtrer_resu", "antisym"]) not_set
- **val4** (*type:* int into [0,1]) not_set

---

bloc_lec_champ_init_canal_sinal

Inherits from: *objet_lecture*

Parameters for the class champ_init_canal_sinal.

in 2D:

U=ucent*y(2h-y)/h/h

V=ampli_bruit*rand+ampli_sin*sin(omega*x)

rand: unpredictable value between -1 and 1.

in 3D:

U=ucent*y(2h-y)/h/h

V=ampli_bruit*rand1+ampli_sin*sin(omega*x)

W=ampli_bruit*rand2

rand1 and rand2: unpredictables values between -1 and 1.

Parameters are:

- **ucent** (*type:* double) Velocity value at the center of the channel.
- **h** (*type:* double) Half hength of the channel.
- **ampli_bruit** (*type:* double) Amplitude for the disturbance.
- **[ampli_sin]** (*type:* double) Amplitude for the sinusoidal disturbance (by default equals to ucent/10).
- **omega** (*type:* double) Value of pulsation for the of the sinusoidal disturbance.
- **[dir_flow]** (*type:* int into [0,1,2]) Flow direction for the initialization of the flow in a channel. - if dir_flow=0, the flow direction is X - if dir_flow=1, the flow direction is Y - if dir_flow=2, the flow direction is Z Default value for dir_flow is 0

- **[dir_wall]** (*type:* int into [0,1,2]) Wall direction for the initialization of the flow in a channel. - if dir_wall=0, the normal to the wall is in X direction - if dir_wall=1, the normal to the wall is in Y direction - if dir_wall=2, the normal to the wall is in Z direction Default value for dir_flow is 1
- **[min_dir_flow]** (*type:* double) Value of the minimum coordinate in the flow direction for the initialization of the flow in a channel. Default value for dir_flow is 0.
- **[min_dir_wall]** (*type:* double) Value of the minimum coordinate in the wall direction for the initialization of the flow in a channel. Default value for dir_flow is 0.

---

bloc_lecture

Inherits from: *objet_lecture*

to read between two braces

Parameters are:

- **bloc_lecture** (*type:* string) not_set

---

bloc_lecture_poro

Inherits from: *objet_lecture*

Surface and volume porosity values.

Parameters are:

- **volumique** (*type:* double) Volume porosity value.
- **surfacique** (*type:* list) Surface porosity values (in X, Y, Z directions).

---

bloc_origine_cotes

Inherits from: *objet_lecture*

Class to create a rectangle (or a box).

Parameters are:

- **name** (*type:* string into ["origine"]) Keyword to define the origin of the rectangle (or the box).
- **origin | origine** (*type:* double list) Coordinates of the origin of the rectangle (or the box).
- **name2** (*type:* string into ["cotes"]) Keyword to define the length along the axes.
- **cotes** (*type:* double list) Length along the axes.

---

bloc_pave

Inherits from: *objet_lecture*

Class to create a pave.

Parameters are:

- **[origine]** (*type:* double list) Keyword to define the pave (block) origin, that is to say one of the 8 block points (or 4 in a 2D coordinate system).
- **[longueurs]** (*type:* double list) Keyword to define the block dimensions, that is to say knowing the origin, length along the axes.
- **[nombre_de_noeuds]** (*type:* int list - size is dimension) Keyword to define the discretization (nodenumber) in each direction.
- **[facteurs]** (*type:* double list) Keyword to define stretching factors for mesh discretization in each direction. This is a real number which must be positive (by default 1.0). A stretching factor other than 1 allows refinement on one edge in one direction.
- **[symx]** (*type:* flag) Keyword to define a block mesh that is symmetrical with respect to the YZ plane (respectively Y-axis in 2D) passing through the block centre.
- **[symy]** (*type:* flag) Keyword to define a block mesh that is symmetrical with respect to the XZ plane (respectively X-axis in 2D) passing through the block centre.
- **[symz]** (*type:* flag) Keyword defining a block mesh that is symmetrical with respect to the XY plane passing through the block centre.
- **[xtanh]** (*type:* double) Keyword to generate mesh with tanh (hyperbolic tangent) variation in the X-direction.
- **[xtanh_dilatation]** (*type:* int into [-1,0,1]) Keyword to generate mesh with tanh (hyperbolic tangent) variation in the X-direction. xtanh_dilatation: The value may be -1,0,1 (0 by default): 0: coarse mesh at the middle of the channel and smaller near the walls -1: coarse mesh at the left side of the channel and smaller at the right side 1: coarse mesh at the right side of the channel and smaller near the left side of the channel.
- **[xtanh_taille_premiere_maille]** (*type:* double) Size of the first cell of the mesh with tanh (hyperbolic tangent) variation in the X-direction.
- **[ytanh]** (*type:* double) Keyword to generate mesh with tanh (hyperbolic tangent) variation in the Y-direction.
- **[ytanh_dilatation]** (*type:* int into [-1,0,1]) Keyword to generate mesh with tanh (hyperbolic tangent) variation in the Y-direction. ytanh_dilatation: The value may be -1,0,1 (0 by default): 0: coarse mesh at the middle of the channel and smaller near the walls -1: coarse mesh at the bottom of the channel and smaller near the top 1: coarse mesh at the top of the channel and smaller near the bottom.
- **[ytanh_taille_premiere_maille]** (*type:* double) Size of the first cell of the mesh with tanh (hyperbolic tangent) variation in the Y-direction.
- **[ztanh]** (*type:* double) Keyword to generate mesh with tanh (hyperbolic tangent) variation in the Z-direction.
- **[ztanh_dilatation]** (*type:* int into [-1,0,1]) Keyword to generate mesh with tanh (hyperbolic tangent) variation in the Z-direction. tanh_dilatation: The value may be -1,0,1 (0 by default): 0: coarse mesh at the middle of the channel and smaller near the walls -1: coarse mesh at the back of the channel and smaller near the front 1: coarse mesh at the front of the channel and smaller near the back.
- **[ztanh_taille_premiere_maille]** (*type:* double) Size of the first cell of the mesh with tanh (hyperbolic tangent) variation in the Z-direction.

bloc_pdf_model

Inherits from: *objet_lecture*

not_set

Parameters are:

- **eta** (*type:* double) penalization coefficient
- **[temps_relaxation_coefficient_pdf]** (*type:* double) time relaxation on the forcing term to help
- **[echelle_relaxation_coefficient_pdf]** (*type:* double) time relaxation on the forcing term to help convergence
- **[local]** (*type:* flag) whether the prescribed velocity is expressed in the global or local basis
- **[vitesse_imposee_data]** (*type: field_base*) Prescribed velocity as a field
- **[vitesse_imposee_fonction]** (*type: troismots*) Prescribed velocity as a set of ananlytical component

---

bloc_sutherland

Inherits from: *objet_lecture*

Sutherland law for viscosity mu(T)=mu0*((T0+C)/(T+C))*(T/T0)**1.5 and (optional) for conductivity lambda(T)=mu0*Cp/Prandtl*((T0+Slambda)/(T+Slambda))*(T/T0)**1.5

Parameters are:

- **problem_name** (*type: pb_base*) Name of problem.
- **mu0** (*type:* string into ["mu0"]) not_set
- **mu0_val** (*type:* double) not_set
- **t0** (*type:* string into ["t0"]) not_set
- **t0_val** (*type:* double) not_set
- **[slambda]** (*type:* string into ["slambda"]) not_set
- **[s]** (*type:* double) not_set
- **c** (*type:* string into ["c"]) not_set
- **c_val** (*type:* double) not_set

---

bloc_tube

Inherits from: *objet_lecture*

Class to create a tube (3D).

Parameters are:

- **name** (*type:* string into ["origine"]) Keyword to define the center of the tube.
- **origin | origine** (*type:* double list) Center of the tube.
- **name2** (*type:* string into ["dir"]) Keyword to define the direction of the main axis.

- **direction** (*type:* string into ["x", "y", "z"]) direction of the main axis X, Y or Z
- **name3** (*type:* string into ["ri"]) Keyword to define the interior radius.
- **ri** (*type:* double) Interior radius.
- **name4** (*type:* string into ["re"]) Keyword to define the exterior radius.
- **re** (*type:* double) Exterior radius.
- **name5** (*type:* string into ["hauteur"]) Keyword to define the heigth of the tube.
- **h** (*type:* double) Heigth of the tube.

---

bord

Inherits from: *objet_lecture*

The block side is not in contact with another block and boundary conditions are applied to it.

Parameters are:

- **nom** (*type:* string) Name of block side.
- **defbord** (*type: defbord*) Definition of block side.

---

bord_base

Inherits from: *objet_lecture*

Basic class for block sides. Block sides that are neither edges nor connectors are not specified. The duplicate nodes of two blocks in contact are automatically recognized and deleted.

---

bords_ecrire

Inherits from: *objet_lecture*

not_set

Parameters are:

- **chaine** (*type:* string into ["bords"]) not_set
- **bords** (*type:* string list) Keyword to post-process only on some boundaries : bords nb_bords boundary1 ... boundaryn where nb_bords : number of boundaries boundary1 ... boundaryn : name of the boundaries.

---

calcul

Inherits from: *objet_lecture*

The centre of gravity will be calculated.

---

canal

Inherits from: *objet_lecture*

Keyword for statistics on a periodic plane channel.

Parameters are:

- **[dt_impr_moy_spat]** (*type:* double) Period to print the spatial average (default value is 1e6).
- **[dt_impr_moy_temp]** (*type:* double) Period to print the temporal average (default value is 1e6).
- **[debut_stat]** (*type:* double) Time to start the temporal averaging (default value is 1e6).
- **[fin_stat]** (*type:* double) Time to end the temporal averaging (default value is 1e6).
- **[pulsation_w]** (*type:* double) Pulsation for phase averaging (in case of pulsating forcing term) (no default value).
- **[nb_points_par_phase]** (*type:* int) Number of samples to represent phase average all along a period (no default value).
- **[reprise]** (*type:* string) val_moy_temp_xxxxxx.sauv : Keyword to resume a calculation with previous averaged quantities. Note that for thermal and turbulent problems, averages on temperature and turbulent viscosity are automatically calculated. To resume a calculation with phase averaging, val_moy_temp_xxxxxx.sauv_phase file is required on the directory where the job is submitted (this last file will be then automatically loaded by TRUST).

---

centre_de_gravite

Inherits from: *objet_lecture*

To specify the centre of gravity.

Parameters are:

- **point** (*type:* *un_point*) A centre of gravity.

---

champ_a_post

Inherits from: *objet_lecture*

Field to be post-processed.

Parameters are:

- **champ** (*type:* string) Name of the post-processed field.
- **[localisation]** (*type:* string into ["elem", "som", "faces"]) Localisation of post-processed field values: The two available values are elem, som, or faces (LATA format only) used respectively to select field values at mesh centres (CHAMPMAILLE type field in the lml file) or at mesh nodes (CHAMPPOINT type field in the lml file). If no selection is made, localisation is set to som by default.

champs_posts

Inherits from: *objet_lecture*

Field's write mode.

Parameters are:

- **[format]** (*type:* string into ["binaire", "formatte"]) Type of file.
- **mot** (*type:* string into ["dt_post", "nb_pas_dt_post"]) Keyword to set the kind of the field's write frequency. Either a time period or a time step period.
- **period** (*type:* string) Value of the period which can be like (2.*t).
- **champs | fields** (*type: champs_a_post*) Post-processed fields.

---

champs_posts_fichier

Inherits from: *objet_lecture*

Field's write mode.

Parameters are:

- **[format]** (*type:* string into ["binaire", "formatte"]) Type of file.
- **mot** (*type:* string into ["dt_post", "nb_pas_dt_post"]) Keyword to set the kind of the field's write frequency. Either a time period or a time step period.
- **period** (*type:* string) Value of the period which can be like (2.*t).
- **fichier | file** (*type:* string) name of file

---

chmoy_faceperio

Inherits from: *objet_lecture*

non documente

Parameters are:

- **bloc** (*type: bloc_lecture*) not_set

---

circle_3

Inherits from: *objet_lecture*

Keyword to define several probes located on a circle (in 3-D space).

Parameters are:

- **nbr** (*type:* int) Number of probes between teta1 and teta2 (angles given in degrees).
- **point_deb** (*type:* *un_point*) Center of the circle.
- **direction** (*type:* int into [0,1,2]) Axis normal to the circle plane (0:x axis, 1:y axis, 2:z axis).
- **radius** (*type:* double) Radius of the circle.
- **theta1** (*type:* double) First angle.
- **theta2** (*type:* double) Second angle.

---

circle

Inherits from: *objet_lecture*

Keyword to define several probes located on a circle.

Parameters are:

- **nbr** (*type:* int) Number of probes between teta1 and teta2 (angles given in degrees).
- **point_deb** (*type:* *un_point*) Center of the circle.
- **[direction]** (*type:* int into [0,1,2]) Axis normal to the circle plane (0:x axis, 1:y axis, 2:z axis).
- **radius** (*type:* double) Radius of the circle.
- **theta1** (*type:* double) First angle.
- **theta2** (*type:* double) Second angle.

---

coarsen_operator_uniform

Inherits from: *objet_lecture*

Object defining the uniform coarsening process of the given grid in IJK discretization

Parameters are:

- **[coarsen_operator_uniform]** (*type:* string) not_set
- **aco** (*type:* string into [“{“]) opening curly brace
- **[coarsen_i]** (*type:* string into [“coarsen_i”]) not_set
- **[coarsen_i_val]** (*type:* int) Integer indicating the number by which we will divide the number of elements in the I direction (in order to obtain a coarser grid)
- **[coarsen_j]** (*type:* string into [“coarsen_j”]) not_set
- **[coarsen_j_val]** (*type:* int) Integer indicating the number by which we will divide the number of elements in the J direction (in order to obtain a coarser grid)

- **[coarsen_k]** (*type:* string into [“coarsen_k”]) not_set
- **[coarsen_k_val]** (*type:* int) Integer indicating the number by which we will divide the number of elements in the K direction (in order to obtain a coarser grid)
- **acof** (*type:* string into [“}”]) closing curly brace

---

condinit

Inherits from: *objet_lecture*

Initial condition.

Parameters are:

- **nom** (*type:* string) Name of initial condition field.
- **ch** (*type: field_base*) Type field and the initial values.

---

condlimlu

Inherits from: *objet_lecture*

Boundary condition specified.

Parameters are:

- **bord** (*type:* string) Name of the edge where the boundary condition applies.
- **cl** (*type: condlim_base*) Boundary condition at the boundary called bord (edge).

---

convection_ale

Synonyms: ale

Inherits from: *objet_lecture*

A convective scheme for ALE (Arbitrary Lagrangian-Eulerian) framework.

Parameters are:

- **opconv** (*type: bloc_convection*) Choice between: amont and muscl Example: convection { ALE { amont } }

---

convection_amont

Synonyms: amont

Inherits from: *objet_lecture*

Keyword for upwind scheme for VDF or VEF discretizations. In VEF discretization equivalent to generic amont for TRUST version 1.5 or later. The previous upwind scheme can be used with the obsolete in future amont_old keyword.

---

convection_amont_old

Synonyms: amont_old

Inherits from: *objet_lecture*

Only for VEF discretization, obsolete keyword, see amont.

---

convection_btd

Synonyms: btd

Inherits from: *objet_lecture*

Only for EF discretization.

Parameters are:

- **btd** (*type:* double) not_set
- **facteur** (*type:* double) not_set

---

convection_centre4

Synonyms: centre4

Inherits from: *objet_lecture*

For VDF and VEF discretizations.

---

convection_centre

Synonyms: centre

Inherits from: *objet_lecture*

For VDF and VEF discretizations.

---

convection_centre_old

Synonyms: centre_old

Inherits from: *objet_lecture*

Only for VEF discretization.

---

convection_deriv

Inherits from: *objet_lecture*

not_set

---

convection_di_l2

Synonyms: di_l2

Inherits from: *objet_lecture*

Only for VEF discretization.

---

convection_ef

Synonyms: ef

Inherits from: *objet_lecture*

For VEF calculations, a centred convective scheme based on Finite Elements formulation can be called through the following data:NL2

Convection { EF transportant_bar val transporte_bar val antisym val filtrer_resu val }NL2

This scheme is 2nd order accuracy (and get better the property of kinetic energy conservation). Due to possible problems of instabilities phenomena, this scheme has to be coupled with stabilisation process (see Source_Qdm_lambdaup).These two last data are equivalent from a theoretical point of view in variationnal writing to : div((u. grad ub , vb) - (u. grad vb, ub)), where vb corresponds to the filtered reference test functions.NL2

Remark:NL2 This class requires to define a filtering operator : see solveur_bar

Parameters are:

- **[mot1]** (*type:* string into ["defaut_bar"]) equivalent to transportant_bar 0 transporte_bar 1 filtrer_resu 1 antisym 1
- **[bloc_ef]** (*type: bloc_ef*) not_set

---

convection_ef_stab

Synonyms: ef_stab

Inherits from: *objet_lecture*

Keyword for a VEF convective scheme.

Parameters are:

- **[alpha]** (*type:* double) To weight the scheme centering with the factor double (between 0 (full centered) and 1 (mix between upwind and centered), by default 1). For scalar equation, it is adviced to use alpha=1 and for the momentum equation, alpha=0.2 is adviced.
- **[test]** (*type:* int) Developer option to compare old and new version of EF_stab

- **[tdivu]** (*type:* flag) To have the convective operator calculated as div(TU)-TdivU(=UgradT).
- **[old]** (*type:* flag) To use old version of EF_stab scheme (default no).
- **[volumes_etendus]** (*type:* flag) Option for the scheme to use the extended volumes (default, yes).
- **[volumes_non_etendus]** (*type:* flag) Option for the scheme to not use the extended volumes (default, no).
- **[amont_sous_zone]** (*type: sous_zone*) Option to degenerate EF_stab scheme into Amont (upwind) scheme in the sub zone of name sz_name. The sub zone may be located arbitrarily in the domain but the more often this option will be activated in a zone where EF_stab scheme generates instabilities as for free outlet for example.
- **[alpha_sous_zone]** (*type: listsous_zone_valeur*) Option to change locally the alpha value on N sub-zones named sub_zone_name_I. Generally, it is used to prevent from a local divergence by increasing locally the alpha parameter.

---

convection_generic

Synonyms: generic

Inherits from: *objet_lecture*

Keyword for generic calling of upwind and muscl convective scheme in VEF discretization. For muscl scheme, limiters and order for fluxes calculations have to be specified. The available limiters are : minmod - vanleer -vanalbada - chakravarthy - superbee, and the order of accuracy is 1 or 2. Note that chakravarthy is a non-symmetric limiter and superbee may engender results out of physical limits. By consequence, these two limiters are not recommended.

Examples:

convection { generic amont }NL2 convection { generic muscl minmod 1 }NL2 convection { generic muscl vanleer 2 }NL2

In case of results out of physical limits with muscl scheme (due for instance to strong non-conformal velocity flow field), user can redefine in data file a lower order and a smoother limiter, as : convection { generic muscl minmod 1 }

Parameters are:

- **type** (*type:* string into ["amont", "muscl", "centre"]) type of scheme
- **[limiteur]** (*type:* string into ["minmod", "vanleer", "vanalbada", "chakravarthy", "superbee"]) type of limiter
- **[ordre]** (*type:* int into [1,2,3]) order of accuracy
- **[alpha]** (*type:* double) alpha

---

convection_kquick

Synonyms: kquick

Inherits from: *objet_lecture*

Only for VEF discretization.

---

convection_muscl3

Synonyms: muscl3

Inherits from: *objet_lecture*

Keyword for a scheme using a ponderation between muscl and center schemes in VEF.

Parameters are:

- **[alpha]** (*type:* double) To weight the scheme centering with the factor double (between 0 (full centered) and 1 (muscl), by default 1).

---

convection_muscl

Synonyms: muscl

Inherits from: *objet_lecture*

Keyword for muscl scheme in VEF discretization equivalent to generic muscl vanleer 2 for the 1.5 version or later. The previous muscl scheme can be used with the obsolete in future muscl_old keyword.

---

convection_muscl_new

Synonyms: muscl_new

Inherits from: *objet_lecture*

Only for VEF discretization.

---

convection_muscl_old

Synonyms: muscl_old

Inherits from: *objet_lecture*

Only for VEF discretization.

---

convection_negligeable

Synonyms: negligeable

Inherits from: *objet_lecture*

For VDF and VEF discretizations. Suppresses the convection operator.

---

convection_quick

Synonyms: quick

Inherits from: *objet_lecture*

Only for VDF discretization.

---

convection_supg

Synonyms: supg

Inherits from: *objet_lecture*

Only for EF discretization.

Parameters are:

- **facteur** (*type:* double) not_set

---

corps_postraitement

Inherits from: *objet_lecture*

not_set

Parameters are:

- **[fichier]** (*type:* string) Name of file.
- **[format]** (*type:* string into ["lml", "lata", "single_lata", "lata_v2", "med", "med_major", "cgns"]) This optional parameter specifies the format of the output file. The basename used for the output file is the basename of the data file. For the fmt parameter, choices are lml or lata. A short description of each format can be found below. The default value is lml.
- **[domaine]** (*type:* string) This optional parameter specifies the domain on which the data should be interpolated before it is written in the output file. The default is to write the data on the domain of the current problem (no interpolation).
- **[sous_domaine | sous_zone]** (*type:* string) This optional parameter specifies the sub_domaine on which the data should be interpolated before it is written in the output file. It is only available for sequential computation.
- **[parallele]** (*type:* string into ["simple", "multiple", "mpi-io"]) Select simple (single file, sequential write), multiple (several files, parallel write), or mpi-io (single file, parallel write) for LATA format
- **[definition_champs]** (*type:* *definition_champs*) Keyword to create new or more complex field for advanced postprocessing.
- **[definition_champs_fichier | definition_champs_file]** (*type:* *definition_champs_fichier*) Definition_champs read from file.
- **[sondes | probes]** (*type:* *sondes*) Probe.
- **[sondes_fichier | probes_file]** (*type:* *sondes_fichier*) Probe read from a file.
- **[sondes_mobiles | mobile_probes]** (*type:* *sondes*) Mobile probes useful for ALE, their positions will be updated in the mesh.

- **[sondes_mobiles_fichier | mobile_probes_file]** (*type:* *sondes_fichier*) Mobile probes read in a file
- **[deprecatedkeepduplicatedprobes]** (*type:* int) Flag to not remove duplicated probes in .son files (1: keep duplicate probes, 0: remove duplicate probes)
- **[champs | fields]** (*type:* *champs_posts*) Field's write mode.
- **[champs_fichier | fields_file]** (*type:* *champs_posts_fichier*) Fields read from file.
- **[statistiques | statistics]** (*type:* *stats_posts*) Statistics between two points fixed : start of integration time and end of integration time.
- **[statistiques_fichier | statistics_file]** (*type:* *stats_posts_fichier*) Statistics read from file.
- **[statistiques_en_serie | serial_statistics]** (*type:* *stats_serie_posts*) Statistics between two points not fixed : on period of integration.
- **[statistiques_en_serie_fichier | serial_statistics_file]** (*type:* *stats_serie_posts_fichier*) Serial_statistics read from a file
- **[suffix_for_reset]** (*type:* string) Suffix used to modify the postprocessing file name if the ICoCo resetTime() method is invoked.

---

defbord_3

Inherits from: *objet_lecture*

2-D edge (plane) in the 3-D space.

Parameters are:

- **dir** (*type:* string into ["x", "y", "z"]) Edge is perpendicular to this direction.
- **eq** (*type:* string into [" "]) Equality sign.
- **pos** (*type:* double) Position value.
- **pos2_min** (*type:* double) Minimal value.
- **inf1** (*type:* string into ["< "]) Less than or equal to sign.
- **dir2** (*type:* string into ["x", "y"]) Edge is parallel to this direction.
- **inf2** (*type:* string into ["< "]) Less than or equal to sign.
- **pos2_max** (*type:* double) Maximal value.
- **pos3_min** (*type:* double) Minimal value.
- **inf3** (*type:* string into ["< "]) Less than or equal to sign.
- **dir3** (*type:* string into ["y", "z"]) Edge is parallel to this direction.
- **inf4** (*type:* string into ["< "]) Less than or equal to sign.
- **pos3_max** (*type:* double) Maximal value.

---

defbord

Inherits from: *objet_lecture*

Class to define an edge.

---

definition_champ

Inherits from: *objet_lecture*

Keyword to create new complex field for advanced postprocessing.

Parameters are:

- **name** (*type:* string) The name of the new created field.
- **champ_generique** (*type:* *champ_generique_base*) not_set

---

definition_champs_fichier

Inherits from: *objet_lecture*

Keyword to read definition_champs from a file

Parameters are:

- **fichier | file** (*type:* string) name of file

---

deuxentiers

Inherits from: *objet_lecture*

Two integers.

Parameters are:

- **int1** (*type:* int) First integer.
- **int2** (*type:* int) Second integer.

---

deuxmots

Inherits from: *objet_lecture*

Two words.

Parameters are:

- **mot_1** (*type:* string) First word.
- **mot_2** (*type:* string) Second word.

---

diffusion_deriv

Inherits from: *objet_lecture*

not_set

---

diffusion_negligeable

Synonyms: negligeable

Inherits from: *objet_lecture*

the diffusivity will not taken in count

---

diffusion_option

Synonyms: option

Inherits from: *objet_lecture*

not_set

Parameters are:

- **bloc_lecture** (*type: bloc_lecture*) not_set

---

diffusion_p1ncp1b

Synonyms: p1ncp1b

Inherits from: *objet_lecture*

not_set

---

diffusion_stab

Synonyms: stab

Inherits from: *objet_lecture*

keyword allowing consistent and stable calculations even in case of obtuse angle meshes.

Parameters are:

- **[standard]** (*type:* int) to recover the same results as calculations made by standard laminar diffusion operator. However, no stabilization technique is used and calculations may be unstable when working with obtuse angle meshes (by default 0)
- **[info]** (*type:* int) developer option to get the stabilizing ratio (by default 0)
- **[new_jacobian]** (*type:* int) when implicit time schemes are used, this option defines a new jacobian that may be more suitable to get stationary solutions (by default 0)

- **[nu]** (*type:* int) (respectively nut 1) takes the molecular viscosity (resp. eddy viscosity) into account in the velocity gradient part of the diffusion expression (by default nu=1 and nut=1)
- **[nut]** (*type:* int) not_set
- **[nu_transp]** (*type:* int) (respectively nut_transp 1) takes the molecular viscosity (resp. eddy viscosity) into account in the transposed velocity gradient part of the diffusion expression (by default nu_transp=0 and nut_transp=1)
- **[nut_transp]** (*type:* int) not_set

---

diffusion_standard

Synonyms: standard

Inherits from: *objet_lecture*

A new keyword, intended for LES calculations, has been developed to optimise and parameterise each term of the diffusion operator. Remark:NL2

1. This class requires to define a filtering operator : see solveur_barNL2 2. The former (original) version: diffusion { } -which omitted some of the term of the diffusion operator- can be recovered by using the following parameters in the new class :NL2 diffusion { standard grad_Ubar 0 nu 1 nut 1 nu_transp 0 nut_transp 1 filtrer_resu 0}.

Parameters are:

- **[mot1]** (*type:* string into ["defaut_bar"]) equivalent to grad_Ubar 1 nu 1 nut 1 nu_transp 1 nut_transp 1 filtrer_resu 1
- **[bloc_diffusion_standard]** (*type: bloc_diffusion_standard*) not_set

---

diffusion_turbulente_multiphase

Synonyms: turbulente

Inherits from: *objet_lecture*

Turbulent diffusion operator for multiphase problem

Parameters are:

- **[type]** (*type: type_diffusion_turbulente_multiphase_deriv*) Turbulence model for multiphase problem

---

difusion_p1b

Synonyms: p1b

Inherits from: *objet_lecture*

not_set

---

domain

Inherits from: *objet_lecture*

Class to reuse a domain.

Parameters are:

- **domain_name** (*type: domaine*) Name of domain.

---

dt_impr_ustar_mean_only

Inherits from: *objet_lecture*

not_set

Parameters are:

- **dt_impr** (*type:* double) not_set
- **[boundaries]** (*type:* string list) not_set

---

ec

Inherits from: *objet_lecture*

Keyword to print total kinetic energy into the referential linked to the domain (keyword Ec). In the case where the domain is moving into a Galilean referential, the keyword Ec_dans_repere_fixe will print total kinetic energy in the Galilean referential whereas Ec will print the value calculated into the moving referential linked to the domain

Parameters are:

- **[ec]** (*type:* flag) not_set
- **[ec_dans_repere_fixe]** (*type:* flag) not_set
- **[periode]** (*type:* double) periode is the keyword to set the period of printing into the file datafile_Ec.son or datafile_Ec_dans_repere_fixe.son.

---

entierfloat

Inherits from: *objet_lecture*

An integer and a real.

Parameters are:

- **the_int** (*type:* int) Integer.
- **the_float** (*type:* double) Real.

---

epsilon

Inherits from: *objet_lecture*

Two points will be confused if the distance between them is less than eps. By default, eps is set to 1e-12. The keyword Epsilon allows an alternative value to be assigned to eps.

Parameters are:

- **eps** (*type:* double) New value of precision.

---

floatfloat

Inherits from: *objet_lecture*

Two reals.

Parameters are:

- **a** (*type:* double) First real.
- **b** (*type:* double) Second real.

---

fonction_champ_reprise

Inherits from: *objet_lecture*

not_set

Parameters are:

- **mot** (*type:* string into ["fonction"]) not_set
- **fonction** (*type:* string list) n f1(val) f2(val) ... fn(val)] time

---

form_a_nb_points

Inherits from: *objet_lecture*

The structure fonction is calculated on nb points and we should add the 2 directions (0:OX, 1:OY, 2:OZ) constituting the homegeneity planes. Example for channel flows, planes parallel to the walls.

Parameters are:

- **nb** (*type:* int into [4]) Number of points.
- **dir1** (*type:* entier(max=2) First direction.
- **dir2** (*type:* entier(max=2) Second direction.

---

format_file

Inherits from: *objet_lecture*

File formatted.

Parameters are:

- **[format]** (*type:* string into ["binaire", "formatte", "xyz", "single_hdf"]) Type of file (the file format).
- **name_file** (*type:* string) Name of file.

---

format_lata_to_med

Inherits from: *objet_lecture*

not_set

Parameters are:

- **mot** (*type:* string into ["format_post_sup"]) not_set
- **[format]** (*type:* string into ["lml", "lata", "lata_v2", "med"]) generated file post_med.data use format (MED or LATA or LML keyword).

---

fourfloat

Inherits from: *objet_lecture*

Four reals.

Parameters are:

- **a** (*type:* double) First real.
- **b** (*type:* double) Second real.
- **c** (*type:* double) Third real.
- **d** (*type:* double) Fourth real.

---

info_med

Inherits from: *objet_lecture*

not_set

Parameters are:

- **file_med** (*type:* string) Name of the MED file.
- **domaine** (*type:* string) Name of domain.
- **pb_post** (*type:* *pb_post*) not_set

---

internes

Inherits from: *objet_lecture*

To indicate that the block has a set of internal faces (these faces will be duplicated automatically by the program and will be processed in a manner similar to edge faces).

Two boundaries with the same boundary conditions may have the same name (whether or not they belong to the same block).

The keyword Internes (Internal) must be used to execute a calculation with plates, followed by the equation of the surface area covered by the plates.

Parameters are:

- **nom** (*type:* string) Name of block side.
- **defbord** (*type: defbord*) Definition of block side.

---

lecture_bloc_moment_base

Inherits from: *objet_lecture*

Auxiliary class to compute and print the moments.

---

longitudinale

Inherits from: *objet_lecture*

Class to define the pressure loss in the direction of the tube bundle.

Parameters are:

- **dir** (*type:* string into ["x", "y", "z"]) Direction.
- **dd** (*type:* double) Tube bundle hydraulic diameter value. This value is expressed in m.
- **ch_a** (*type:* string into ["a", "cf"]) Keyword to be used to set law coefficient values for the coefficient of regular pressure losses.
- **a** (*type:* double) Value of a law coefficient for regular pressure losses.
- **[ch_b]** (*type:* string into ["b"]) Keyword to be used to set law coefficient values for regular pressure losses.
- **[b]** (*type:* double) Value of a law coefficient for regular pressure losses.

---

longueur_melange

Inherits from: *objet_lecture*

This model is based on mixing length modelling. For a non academic configuration, formulation used in the code can be expressed basically as :

$nu_t=(Kappa.y)^2$.dU/dy

Till a maximum distance (dmax) set by the user in the data file, y is set equal to the distance from the wall (dist_w) calculated previously and saved in file Wall_length.xyz. [see Distance_paroi keyword]

Then (from y=dmax), y decreases as an exponential function : y=dmax*exp[-2.*(dist_w-dmax)/dmax]

Parameters are:

- **[canalx]** (*type:* double) [height] : plane channel according to Ox direction (for the moment, formulation in the code relies on fixed heigh : H=2).
- **[tuyauz]** (*type:* double) [diameter] : pipe according to Oz direction (for the moment, formulation in the code relies on fixed diameter : D=2).
- **[verif_dparoi]** (*type:* string) not_set
- **[dmax]** (*type:* double) Maximum distance.
- **[fichier]** (*type:* string) not_set
- **[fichier_ecriture_k_eps]** (*type:* string) When a resume with k-epsilon model is envisaged, this keyword allows to generate external MED-format file with evaluation of k and epsilon quantities (based on eddy turbulent viscosity and turbulent characteristic length returned by mixing length model). The frequency of the MED file print is set equal to dt_impr_ustar. Moreover, k-eps MED field is automatically saved at the last time step. MED file is then used for resuming a K-Epsilon calculation with the Champ_Fonc_Med keyword.
- **[formulation_a_nb_points]** (*type: form_a_nb_points*) The structure fonction is calculated on nb points and we should add the 2 directions (0:OX, 1:OY, 2:OZ) constituting the homegeneity planes. Example for channel flows, planes parallel to the walls.
- **[longueur_maille]** (*type:* string into ["volume", "volume_sans_lissage", "scotti", "arrete"]) Different ways to calculate the characteristic length may be specified : volume : It is the default option. Characteristic length is based on the cubic root of the volume cells. A smoothing procedure is applied to avoid discontinuities of this quantity in VEF from a cell to another. volume_sans_lissage : For VEF only. Characteristic length is based on the cubic root of the volume cells (without smoothing procedure).NL2 scotti : Characteristic length is based on the cubic root of the volume cells and the Scotti correction is applied to take into account the stretching of the cell in the case of anisotropic meshes. arete : For VEF only. Characteristic length relies on the max edge (+ smoothing procedure) is taken into account.
- **[correction_visco_turb_pour_controle_pas_de_temps]** (*type:* flag) Keyword to set a limitation to low time steps due to high values of turbulent viscosity. The limit for turbulent viscosity is calculated so that diffusive time-step is equal or higher than convective time-step. For a stationary flow, the correction for turbulent viscosity should apply only during the first time steps and not when permanent state is reached. To check that, we could post process the corr_visco_turb field which is the correction of turbulent viscosity: it should be 1. on the whole domain.
- **[correction_visco_turb_pour_controle_pas_de_temps_parametre]** (*type:* double) Keyword to set a limitation to low time steps due to high values of turbulent viscosity. The limit for turbulent viscosity is the ratio between diffusive time-step and convective time-step is higher or equal to the given value [0-1]
- **[turbulence_paroi]** (*type: turbulence_paroi_base*) Keyword to set the wall law.

- **[dt_impr_ustar]** (*type:* double) This keyword is used to print the values (U +, d+, u$star$) obtained with the wall laws into a file named datafile_ProblemName_Ustar.face and periode refers to the printing period, this value is expressed in seconds.
- **[dt_impr_ustar_mean_only]** (*type:* *dt_impr_ustar_mean_only*) This keyword is used to print the mean values of u* (obtained with the wall laws) on each boundary, into a file named datafile_ProblemName_Ustar_mean_only.out. periode refers to the printing period, this value is expressed in seconds. If you don't use the optional keyword boundaries, all the boundaries will be considered. If you use it, you must specify nb_boundaries which is the number of boundaries on which you want to calculate the mean values of u*, then you have to specify their names.
- **[nut_max]** (*type:* double) Upper limitation of turbulent viscosity (default value 1.e8).

---

mailler_base

Inherits from: *objet_lecture*

Basic class to mesh.

---

methode_loi_horaire

Synonyms: loi_horaire

Inherits from: *objet_lecture*

not_set

Parameters are:

- **nom_loi** (*type:* *loi_horaire*) not_set

---

methode_transport_deriv

Inherits from: *objet_lecture*

Basic class for method of transport of interface.

---

mod_turb_hyd_rans

Inherits from: *objet_lecture*

Class for RANS turbulence model for Navier-Stokes equations.

Parameters are:

- **[k_min]** (*type:* double) Lower limitation of k (default value 1.e-10).
- **[quiet]** (*type:* flag) To disable printing of information about K and Epsilon/Omega.

- **[correction_visco_turb_pour_controle_pas_de_temps]** (*type:* flag) Keyword to set a limitation to low time steps due to high values of turbulent viscosity. The limit for turbulent viscosity is calculated so that diffusive time-step is equal or higher than convective time-step. For a stationary flow, the correction for turbulent viscosity should apply only during the first time steps and not when permanent state is reached. To check that, we could post process the corr_visco_turb field which is the correction of turbulent viscosity: it should be 1. on the whole domain.
- **[correction_visco_turb_pour_controle_pas_de_temps_parametre]** (*type:* double) Keyword to set a limitation to low time steps due to high values of turbulent viscosity. The limit for turbulent viscosity is the ratio between diffusive time-step and convective time-step is higher or equal to the given value [0-1]
- **[turbulence_paroi]** (*type: turbulence_paroi_base*) Keyword to set the wall law.
- **[dt_impr_ustar]** (*type:* double) This keyword is used to print the values (U +, d+, u$star$) obtained with the wall laws into a file named datafile_ProblemName_Ustar.face and periode refers to the printing period, this value is expressed in seconds.
- **[dt_impr_ustar_mean_only]** (*type: dt_impr_ustar_mean_only*) This keyword is used to print the mean values of u* (obtained with the wall laws) on each boundary, into a file named datafile_ProblemName_Ustar_mean_only.out. periode refers to the printing period, this value is expressed in seconds. If you don't use the optional keyword boundaries, all the boundaries will be considered. If you use it, you must specify nb_boundaries which is the number of boundaries on which you want to calculate the mean values of u*, then you have to specify their names.
- **[nut_max]** (*type:* double) Upper limitation of turbulent viscosity (default value 1.e8).

---

mod_turb_hyd_ss_maille

Inherits from: *objet_lecture*

Class for sub-grid turbulence model for Navier-Stokes equations.

Parameters are:

- **[formulation_a_nb_points]** (*type: form_a_nb_points*) The structure fonction is calculated on nb points and we should add the 2 directions (0:OX, 1:OY, 2:OZ) constituting the homegeneity planes. Example for channel flows, planes parallel to the walls.
- **[longueur_maille]** (*type:* string into ["volume", "volume_sans_lissage", "scotti", "arrete"]) Different ways to calculate the characteristic length may be specified : volume : It is the default option. Characteristic length is based on the cubic root of the volume cells. A smoothing procedure is applied to avoid discontinuities of this quantity in VEF from a cell to another. volume_sans_lissage : For VEF only. Characteristic length is based on the cubic root of the volume cells (without smoothing procedure).NL2 scotti : Characteristic length is based on the cubic root of the volume cells and the Scotti correction is applied to take into account the stretching of the cell in the case of anisotropic meshes. arete : For VEF only. Characteristic length relies on the max edge (+ smoothing procedure) is taken into account.
- **[correction_visco_turb_pour_controle_pas_de_temps]** (*type:* flag) Keyword to set a limitation to low time steps due to high values of turbulent viscosity. The limit for turbulent viscosity is calculated so that diffusive time-step is equal or higher than convective time-step. For a stationary flow, the correction for turbulent viscosity should apply only during the first time steps and not when permanent state is reached. To check that, we could post process the corr_visco_turb field which is the correction of turbulent viscosity: it should be 1. on the whole domain.
- **[correction_visco_turb_pour_controle_pas_de_temps_parametre]** (*type:* double) Keyword to set a limitation to low time steps due to high values of turbulent viscosity. The limit for turbulent viscosity is the ratio between diffusive time-step and convective time-step is higher or equal to the given value [0-1]

- **[turbulence_paroi]** (*type:* *turbulence_paroi_base*) Keyword to set the wall law.
- **[dt_impr_ustar]** (*type:* double) This keyword is used to print the values (U +, d+, u$star$) obtained with the wall laws into a file named datafile_ProblemName_Ustar.face and periode refers to the printing period, this value is expressed in seconds.
- **[dt_impr_ustar_mean_only]** (*type:* *dt_impr_ustar_mean_only*) This keyword is used to print the mean values of u* (obtained with the wall laws) on each boundary, into a file named datafile_ProblemName_Ustar_mean_only.out. periode refers to the printing period, this value is expressed in seconds. If you don't use the optional keyword boundaries, all the boundaries will be considered. If you use it, you must specify nb_boundaries which is the number of boundaries on which you want to calculate the mean values of u*, then you have to specify their names.
- **[nut_max]** (*type:* double) Upper limitation of turbulent viscosity (default value 1.e8).

---

modele_turbulence_hyd_deriv

Inherits from: *objet_lecture*

Basic class for turbulence model for Navier-Stokes equations.

Parameters are:

- **[correction_visco_turb_pour_controle_pas_de_temps]** (*type:* flag) Keyword to set a limitation to low time steps due to high values of turbulent viscosity. The limit for turbulent viscosity is calculated so that diffusive time-step is equal or higher than convective time-step. For a stationary flow, the correction for turbulent viscosity should apply only during the first time steps and not when permanent state is reached. To check that, we could post process the corr_visco_turb field which is the correction of turbulent viscosity: it should be 1. on the whole domain.
- **[correction_visco_turb_pour_controle_pas_de_temps_parametre]** (*type:* double) Keyword to set a limitation to low time steps due to high values of turbulent viscosity. The limit for turbulent viscosity is the ratio between diffusive time-step and convective time-step is higher or equal to the given value [0-1]
- **[turbulence_paroi]** (*type:* *turbulence_paroi_base*) Keyword to set the wall law.
- **[dt_impr_ustar]** (*type:* double) This keyword is used to print the values (U +, d+, u$star$) obtained with the wall laws into a file named datafile_ProblemName_Ustar.face and periode refers to the printing period, this value is expressed in seconds.
- **[dt_impr_ustar_mean_only]** (*type:* *dt_impr_ustar_mean_only*) This keyword is used to print the mean values of u* (obtained with the wall laws) on each boundary, into a file named datafile_ProblemName_Ustar_mean_only.out. periode refers to the printing period, this value is expressed in seconds. If you don't use the optional keyword boundaries, all the boundaries will be considered. If you use it, you must specify nb_boundaries which is the number of boundaries on which you want to calculate the mean values of u*, then you have to specify their names.
- **[nut_max]** (*type:* double) Upper limitation of turbulent viscosity (default value 1.e8).

---

modele_turbulence_hyd_null

Synonyms: null

Inherits from: *objet_lecture*

Null turbulence model (turbulent viscosity = 0) which can be used with a turbulent problem.

Parameters are:

- **[correction_visco_turb_pour_controle_pas_de_temps]** (*type:* flag) Keyword to set a limitation to low time steps due to high values of turbulent viscosity. The limit for turbulent viscosity is calculated so that diffusive time-step is equal or higher than convective time-step. For a stationary flow, the correction for turbulent viscosity should apply only during the first time steps and not when permanent state is reached. To check that, we could post process the corr_visco_turb field which is the correction of turbulent viscosity: it should be 1. on the whole domain.
- **[correction_visco_turb_pour_controle_pas_de_temps_parametre]** (*type:* double) Keyword to set a limitation to low time steps due to high values of turbulent viscosity. The limit for turbulent viscosity is the ratio between diffusive time-step and convective time-step is higher or equal to the given value [0-1]
- **[turbulence_paroi]** (*type: turbulence_paroi_base*) Keyword to set the wall law.
- **[dt_impr_ustar]** (*type:* double) This keyword is used to print the values (U +, d+, u$star$) obtained with the wall laws into a file named datafile_ProblemName_Ustar.face and periode refers to the printing period, this value is expressed in seconds.
- **[dt_impr_ustar_mean_only]** (*type: dt_impr_ustar_mean_only*) This keyword is used to print the mean values of u* (obtained with the wall laws) on each boundary, into a file named datafile_ProblemName_Ustar_mean_only.out. periode refers to the printing period, this value is expressed in seconds. If you don't use the optional keyword boundaries, all the boundaries will be considered. If you use it, you must specify nb_boundaries which is the number of boundaries on which you want to calculate the mean values of u*, then you have to specify their names.
- **[nut_max]** (*type:* double) Upper limitation of turbulent viscosity (default value 1.e8).

---

nom_postraitement

Inherits from: *objet_lecture*

not_set

Parameters are:

- **nom** (*type:* string) Name of the post-processing.
- **post** (*type: postraitement_base*) the post

---

numero_elem_sur_maitre

Inherits from: *objet_lecture*

Keyword to define a probe at the special element. Useful for min/max sonde.

Parameters are:

- **numero** (*type:* int) element number

---

objet_lecture

Auxiliary class for reading.

---

op_implicite

Inherits from: *objet_lecture*

not_set

Parameters are:

- **implicite** (*type:* string into ["implicite"]) not_set
- **mot** (*type:* string into ["solveur"]) not_set
- **solveur** (*type: solveur_sys_base*) not_set

---

parametre_diffusion_implicite

Inherits from: *objet_lecture*

To specify additional parameters for the equation when using impliciting diffusion

Parameters are:

- **[crank]** (*type:* int into [0,1]) Use (1) or not (0, default) a Crank Nicholson method for the diffusion implicitation algorithm. Setting crank to 1 increases the order of the algorithm from 1 to 2.
- **[preconditionnement_diag]** (*type:* int into [0,1]) The CG used to solve the implicitation of the equation diffusion operator is not preconditioned by default. If this option is set to 1, a diagonal preconditionning is used. Warning: this option is not necessarily more efficient, depending on the treated case.
- **[niter_max_diffusion_implicite]** (*type:* int) Change the maximum number of iterations for the CG (Conjugate Gradient) algorithm when solving the diffusion implicitation of the equation.
- **[seuil_diffusion_implicite]** (*type:* double) Change the threshold convergence value used by default for the CG resolution for the diffusion implicitation of this equation.
- **[solveur]** (*type: solveur_sys_base*) Method (different from the default one, Conjugate Gradient) to solve the linear system.

---

parametre_equation_base

Inherits from: *objet_lecture*

Basic class for parametre_equation

---

parametre_implicite

Inherits from: *objet_lecture*

Keyword to change for this equation only the parameter of the implicit scheme used to solve the problem.

Parameters are:

- **[seuil_convergence_implicite]** (*type:* double) Keyword to change for this equation only the value of seuil_convergence_implicite used in the implicit scheme.
- **[seuil_convergence_solveur]** (*type:* double) Keyword to change for this equation only the value of seuil_convergence_solveur used in the implicit scheme
- **[solveur]** (*type:* *solveur_sys_base*) Keyword to change for this equation only the solver used in the implicit scheme
- **[resolution_explicite]** (*type:* flag) To solve explicitly the equation whereas the scheme is an implicit scheme.
- **[equation_non_resolue]** (*type:* flag) Keyword to specify that the equation is not solved.
- **[equation_frequence_resolue]** (*type:* string) Keyword to specify that the equation is solved only every n time steps (n is an integer or given by a time-dependent function f(t)).

---

pave

Inherits from: *objet_lecture*

Class to create a pave (block) with boundaries.

Parameters are:

- **name** (*type:* string) Name of the pave (block).
- **bloc** (*type:* *bloc_pave*) Definition of the pave (block).
- **list_bord** (*type:* *list_bord*) Domain boundaries definition.

---

penalisation_l2_ftd_lec

Inherits from: *objet_lecture*

not_set

Parameters are:

- **[postraiter_gradient_pression_sans_masse]** (*type:* int) (IBM advanced) avoid mass matrix multiplication for the gradient postprocessing

- **[correction_matrice_projection_initiale]** (*type:* int) (IBM advanced) fix matrix of initial projection for PDF
- **[correction_calcul_pression_initiale]** (*type:* int) (IBM advanced) fix initial pressure computation for PDF
- **[correction_vitesse_projection_initiale]** (*type:* int) (IBM advanced) fix initial velocity computation for PDF
- **[correction_matrice_pression]** (*type:* int) (IBM advanced) fix pressure matrix for PDF
- **[matrice_pression_penalisee_h1]** (*type:* int) (IBM advanced) fix pressure matrix for PDF
- **[correction_vitesse_modifie]** (*type:* int) (IBM advanced) fix velocity for PDF
- **[correction_pression_modifie]** (*type:* int) (IBM advanced) fix pressure for PDF
- **[gradient_pression_qdm_modifie]** (*type:* int) (IBM advanced) fix pressure gradient
- **bord** (*type:* string) not_set
- **val** (*type:* list) not_set

---

plan

Inherits from: *objet_lecture*

Keyword to set the number of probe layout points. The file format is type .lml

Parameters are:

- **nbr** (*type:* int) Number of probes in the first direction.
- **nbr2** (*type:* int) Number of probes in the second direction.
- **point_deb** (*type:* *un_point*) First point defining the angle. This angle should be positive.
- **point_fin** (*type:* *un_point*) Second point defining the angle. This angle should be positive.
- **point_fin_2** (*type:* *un_point*) Third point defining the angle. This angle should be positive.

---

point

Inherits from: *objet_lecture*

Point as class-daughter of Points.

Parameters are:

- **points** (*type:* *listpoints*) Probe points.

---

points

Inherits from: *objet_lecture*

Keyword to define the number of probe points. The file is arranged in columns.

Parameters are:

- **points** (*type:* *listpoints*) Probe points.

---

position_like

Inherits from: *objet_lecture*

Keyword to define a probe at the same position of another probe named autre_sonde.

Parameters are:

- **autre_sonde** (*type:* string) Name of the other probe.

---

postraitement

Synonyms: post_processing

Inherits from: *objet_lecture*

An object of post-processing (without name).

Parameters are:

- **[fichier]** (*type:* string) Name of file.
- **[format]** (*type:* string into ["lml", "lata", "single_lata", "lata_v2", "med", "med_major", "cgns"]) This optional parameter specifies the format of the output file. The basename used for the output file is the basename of the data file. For the fmt parameter, choices are lml or lata. A short description of each format can be found below. The default value is lml.
- **[domaine]** (*type:* string) This optional parameter specifies the domain on which the data should be interpolated before it is written in the output file. The default is to write the data on the domain of the current problem (no interpolation).
- **[sous_domaine | sous_zone]** (*type:* string) This optional parameter specifies the sub_domaine on which the data should be interpolated before it is written in the output file. It is only available for sequential computation.
- **[parallele]** (*type:* string into ["simple", "multiple", "mpi-io"]) Select simple (single file, sequential write), multiple (several files, parallel write), or mpi-io (single file, parallel write) for LATA format
- **[definition_champs]** (*type:* *definition_champs*) Keyword to create new or more complex field for advanced postprocessing.
- **[definition_champs_fichier | definition_champs_file]** (*type:* *definition_champs_fichier*) Definition_champs read from file.
- **[sondes | probes]** (*type:* *sondes*) Probe.
- **[sondes_fichier | probes_file]** (*type:* *sondes_fichier*) Probe read from a file.
- **[sondes_mobiles | mobile_probes]** (*type:* *sondes*) Mobile probes useful for ALE, their positions will be updated in the mesh.

- **[sondes_mobiles_fichier | mobile_probes_file]** (*type:* *sondes_fichier*) Mobile probes read in a file
- **[deprecatedkeepduplicatedprobes]** (*type:* int) Flag to not remove duplicated probes in .son files (1: keep duplicate probes, 0: remove duplicate probes)
- **[champs | fields]** (*type:* *champs_posts*) Field's write mode.
- **[champs_fichier | fields_file]** (*type:* *champs_posts_fichier*) Fields read from file.
- **[statistiques | statistics]** (*type:* *stats_posts*) Statistics between two points fixed : start of integration time and end of integration time.
- **[statistiques_fichier | statistics_file]** (*type:* *stats_posts_fichier*) Statistics read from file.
- **[statistiques_en_serie | serial_statistics]** (*type:* *stats_serie_posts*) Statistics between two points not fixed : on period of integration.
- **[statistiques_en_serie_fichier | serial_statistics_file]** (*type:* *stats_serie_posts_fichier*) Serial_statistics read from a file
- **[suffix_for_reset]** (*type:* string) Suffix used to modify the postprocessing file name if the ICoCo resetTime() method is invoked.

---

postraitement_base

Inherits from: *objet_lecture*

not_set

---

profils_thermo

Inherits from: *objet_lecture*

non documente

Parameters are:

- **bloc** (*type:* *bloc_lecture*) not_set

---

raccord

Inherits from: *objet_lecture*

The block side is in contact with the block of another domain (case of two coupled problems).

Parameters are:

- **type1** (*type:* string into ["local", "distant"]) Contact type.
- **type2** (*type:* string into ["homogene"]) Contact type.
- **nom** (*type:* string) Name of block side.
- **defbord** (*type:* *defbord*) Definition of block side.

---

radius

Inherits from: *objet_lecture*

not_set

Parameters are:

- **nbr** (*type:* int) Number of probe points of the segment, evenly distributed.
- **point_deb** (*type: un_point*) First outer probe segment point.
- **radius** (*type:* double) not_set
- **teta1** (*type:* double) not_set
- **teta2** (*type:* double) not_set

---

reaction

Inherits from: *objet_lecture*

Keyword to describe reaction:

w =K pow(T,beta) exp(-Ea/(R T)) Pi pow(Reactif_i,activitivity_i).

If K_inv >0,

w= K pow(T,beta) exp(-Ea/(R T)) (Pi pow(Reactif_i,activitivity_i) - Kinv/exp(-c_r_Ea/(R T)) Pi pow(Produit_i,activitivity_i))

Parameters are:

- **reactifs** (*type:* string) LHS of equation (ex CH4+2*O2)
- **produits** (*type:* string) RHS of equation (ex CO2+2*H20)
- **[constante_taux_reaction]** (*type:* double) constante of cinetic K
- **[coefficients_activites]** (*type: bloc_lecture*) coefficients od ativity (exemple { CH4 1 O2 2 })
- **enthalpie_reaction** (*type:* double) DH
- **energie_activation** (*type:* double) Ea
- **exposant_beta** (*type:* double) Beta
- **[contre_reaction]** (*type:* double) K_inv
- **[contre_energie_activation]** (*type:* double) c_r_Ea

---

remove_elem_bloc

Inherits from: *objet_lecture*

not_set

Parameters are:

- **[liste]** (*type:* int list) not_set
- **[fonction]** (*type:* string) not_set

---

segment

Inherits from: *objet_lecture*

Keyword to define the number of probe segment points. The file is arranged in columns.

Parameters are:

- **nbr** (*type:* int) Number of probe points of the segment, evenly distributed.
- **point_deb** (*type:* *un_point*) First outer probe segment point.
- **point_fin** (*type:* *un_point*) Second outer probe segment point.

---

segmentfacesx

Inherits from: *objet_lecture*

Segment probe where points are moved to the nearest x faces

Parameters are:

- **nbr** (*type:* int) Number of probe points of the segment, evenly distributed.
- **point_deb** (*type:* *un_point*) First outer probe segment point.
- **point_fin** (*type:* *un_point*) Second outer probe segment point.

---

segmentfacesy

Inherits from: *objet_lecture*

Segment probe where points are moved to the nearest y faces

Parameters are:

- **nbr** (*type:* int) Number of probe points of the segment, evenly distributed.
- **point_deb** (*type:* *un_point*) First outer probe segment point.
- **point_fin** (*type:* *un_point*) Second outer probe segment point.

---

segmentfacesz

Inherits from: *objet_lecture*

Segment probe where points are moved to the nearest z faces

Parameters are:

- **nbr** (*type:* int) Number of probe points of the segment, evenly distributed.
- **point_deb** (*type:* *un_point*) First outer probe segment point.
- **point_fin** (*type:* *un_point*) Second outer probe segment point.

---

segmentpoints

Inherits from: *objet_lecture*

This keyword is used to define a probe segment from specifics points. The nom_champ field is sampled at ns specifics points.

Parameters are:

- **points** (*type:* *listpoints*) Probe points.

---

sonde

Inherits from: *objet_lecture*

Keyword is used to define the probes. Observations: the probe coordinates should be given in Cartesian coordinates (X, Y, Z), including axisymmetric.

Parameters are:

- **nom_sonde** (*type:* string) Name of the file in which the values taken over time will be saved. The complete file name is nom_sonde.son.
- **[special]** (*type:* string into [“grav”, “som”, “nodes”, “chsom”, “gravcl”]) Option to change the positions of the probes. Several options are available: grav : each probe is moved to the nearest cell center of the mesh; som : each probe is moved to the nearest vertex of the mesh nodes : each probe is moved to the nearest face center of the mesh; chsom : only available for P1NC sampled field. The values of the probes are calculated according to P1-Conform corresponding field. gravcl : Extend to the domain face boundary a cell-located segment probe in order to have the boundary condition for the field. For this type the extreme probe point has to be on the face center of gravity.
- **nom_inco** (*type:* string) Name of the sampled field.
- **mperiode** (*type:* string into [“periode”]) Keyword to set the sampled field measurement frequency.
- **prd** (*type:* double) Period value. Every prd seconds, the field value calculated at the previous time step is written to the nom_sonde.son file.
- **type** (*type:* *sonde_base*) Type of probe.

---

sonde_base

Inherits from: *objet_lecture*

Basic probe. Probes refer to sensors that allow a value or several points of the domain to be monitored over time. The probes may be a set of points defined one by one (keyword Points) or a set of points evenly distributed over a straight segment (keyword Segment) or arranged according to a layout (keyword Plan) or according to a parallelepiped (keyword Volume). The fields allow all the values of a physical value on the domain to be known at several moments in time.

---

sonde_tble

Inherits from: *objet_lecture*

not_set

Parameters are:

- **name** (*type:* string) not_set
- **point** (*type: un_point*) not_set

---

sondes_fichier

Inherits from: *objet_lecture*

Keyword to read probes from a file

Parameters are:

- **fichier | file** (*type:* string) name of file

---

sous_maille_smago

Inherits from: *objet_lecture*

Smagorinsky sub-grid turbulence model.

Nut=Cs1*Cs1*l*l*sqrt(2*S*S)

K=Cs2*Cs2*l*l*2*S

Parameters are:

- **[cs]** (*type:* double) This is an optional keyword and the value is used to set the constant used in the Smagorinsky model (This is currently only valid for Smagorinsky models and it is set to 0.18 by default) .
- **[formulation_a_nb_points]** (*type: form_a_nb_points*) The structure fonction is calculated on nb points and we should add the 2 directions (0:OX, 1:OY, 2:OZ) constituting the homegeneity planes. Example for channel flows, planes parallel to the walls.

- **[longueur_maille]** (*type:* string into ["volume", "volume_sans_lissage", "scotti", "arrete"]) Different ways to calculate the characteristic length may be specified : volume : It is the default option. Characteristic length is based on the cubic root of the volume cells. A smoothing procedure is applied to avoid discontinuities of this quantity in VEF from a cell to another. volume_sans_lissage : For VEF only. Characteristic length is based on the cubic root of the volume cells (without smoothing procedure).NL2 scotti : Characteristic length is based on the cubic root of the volume cells and the Scotti correction is applied to take into account the stretching of the cell in the case of anisotropic meshes. arete : For VEF only. Characteristic length relies on the max edge (+ smoothing procedure) is taken into account.
- **[correction_visco_turb_pour_controle_pas_de_temps]** (*type:* flag) Keyword to set a limitation to low time steps due to high values of turbulent viscosity. The limit for turbulent viscosity is calculated so that diffusive time-step is equal or higher than convective time-step. For a stationary flow, the correction for turbulent viscosity should apply only during the first time steps and not when permanent state is reached. To check that, we could post process the corr_visco_turb field which is the correction of turbulent viscosity: it should be 1. on the whole domain.
- **[correction_visco_turb_pour_controle_pas_de_temps_parametre]** (*type:* double) Keyword to set a limitation to low time steps due to high values of turbulent viscosity. The limit for turbulent viscosity is the ratio between diffusive time-step and convective time-step is higher or equal to the given value [0-1]
- **[turbulence_paroi]** (*type: turbulence_paroi_base*) Keyword to set the wall law.
- **[dt_impr_ustar]** (*type:* double) This keyword is used to print the values (U +, d+, u$star$) obtained with the wall laws into a file named datafile_ProblemName_Ustar.face and periode refers to the printing period, this value is expressed in seconds.
- **[dt_impr_ustar_mean_only]** (*type: dt_impr_ustar_mean_only*) This keyword is used to print the mean values of u* (obtained with the wall laws) on each boundary, into a file named datafile_ProblemName_Ustar_mean_only.out. periode refers to the printing period, this value is expressed in seconds. If you don't use the optional keyword boundaries, all the boundaries will be considered. If you use it, you must specify nb_boundaries which is the number of boundaries on which you want to calculate the mean values of u*, then you have to specify their names.
- **[nut_max]** (*type:* double) Upper limitation of turbulent viscosity (default value 1.e8).

---

sous_maille_wale

Inherits from: *objet_lecture*

This is the WALE-model. It is a new sub-grid scale model for eddy-viscosity in LES that has the following properties :

- it goes naturally to 0 at the wall (it doesn't need any information on the wall

position or geometry)

- it has the proper wall scaling in o(y3) in the vicinity of the wall
- it reproduces correctly the laminar to turbulent transition.

Parameters are:

- **[cw]** (*type:* double) The unique parameter (constant) of the WALE-model (by default value 0.5).
- **[formulation_a_nb_points]** (*type: form_a_nb_points*) The structure fonction is calculated on nb points and we should add the 2 directions (0:OX, 1:OY, 2:OZ) constituting the homegeneity planes. Example for channel flows, planes parallel to the walls.

- **[longueur_maille]** (*type:* string into ["volume", "volume_sans_lissage", "scotti", "arrete"]) Different ways to calculate the characteristic length may be specified : volume : It is the default option. Characteristic length is based on the cubic root of the volume cells. A smoothing procedure is applied to avoid discontinuities of this quantity in VEF from a cell to another. volume_sans_lissage : For VEF only. Characteristic length is based on the cubic root of the volume cells (without smoothing procedure).NL2 scotti : Characteristic length is based on the cubic root of the volume cells and the Scotti correction is applied to take into account the stretching of the cell in the case of anisotropic meshes. arete : For VEF only. Characteristic length relies on the max edge (+ smoothing procedure) is taken into account.
- **[correction_visco_turb_pour_controle_pas_de_temps]** (*type:* flag) Keyword to set a limitation to low time steps due to high values of turbulent viscosity. The limit for turbulent viscosity is calculated so that diffusive time-step is equal or higher than convective time-step. For a stationary flow, the correction for turbulent viscosity should apply only during the first time steps and not when permanent state is reached. To check that, we could post process the corr_visco_turb field which is the correction of turbulent viscosity: it should be 1. on the whole domain.
- **[correction_visco_turb_pour_controle_pas_de_temps_parametre]** (*type:* double) Keyword to set a limitation to low time steps due to high values of turbulent viscosity. The limit for turbulent viscosity is the ratio between diffusive time-step and convective time-step is higher or equal to the given value [0-1]
- **[turbulence_paroi]** (*type: turbulence_paroi_base*) Keyword to set the wall law.
- **[dt_impr_ustar]** (*type:* double) This keyword is used to print the values (U +, d+, u$star$) obtained with the wall laws into a file named datafile_ProblemName_Ustar.face and periode refers to the printing period, this value is expressed in seconds.
- **[dt_impr_ustar_mean_only]** (*type: dt_impr_ustar_mean_only*) This keyword is used to print the mean values of u* (obtained with the wall laws) on each boundary, into a file named datafile_ProblemName_Ustar_mean_only.out. periode refers to the printing period, this value is expressed in seconds. If you don't use the optional keyword boundaries, all the boundaries will be considered. If you use it, you must specify nb_boundaries which is the number of boundaries on which you want to calculate the mean values of u*, then you have to specify their names.
- **[nut_max]** (*type:* double) Upper limitation of turbulent viscosity (default value 1.e8).

---

sous_zone_valeur

Inherits from: *objet_lecture*

Two words.

Parameters are:

- **sous_zone** (*type: sous_zone*) sous zone
- **valeur** (*type:* double) value

---

spec_pdcr_base

Inherits from: *objet_lecture*

Class to read the source term modelling the presence of a bundle of tubes in a flow. Cf=A Re-B.

Parameters are:

- **ch_a** (*type:* string into ["a", "cf"]) Keyword to be used to set law coefficient values for the coefficient of regular pressure losses.
- **a** (*type:* double) Value of a law coefficient for regular pressure losses.
- **[ch_b]** (*type:* string into ["b"]) Keyword to be used to set law coefficient values for regular pressure losses.
- **[b]** (*type:* double) Value of a law coefficient for regular pressure losses.

---

stat_post_correlation

Synonyms: champ_post_statistiques_correlation, correlation

Inherits from: *objet_lecture*

not_set

Parameters are:

- **first_field** (*type:* string) not_set
- **second_field** (*type:* string) not_set
- **[localisation]** (*type:* string into ["elem", "som", "faces"]) Localisation of post-processed field value

---

stat_post_deriv

Inherits from: *objet_lecture*

not_set

---

stat_post_ecart_type

Synonyms: champ_post_statistiques_ecart_type, ecart_type

Inherits from: *objet_lecture*

not_set

Parameters are:

- **field** (*type:* string) not_set
- **[localisation]** (*type:* string into ["elem", "som", "faces"]) Localisation of post-processed field value

---

stat_post_moyenne

Synonyms: champ_post_statistiques_moyenne, moyenne

Inherits from: *objet_lecture*

not_set

Parameters are:

- **field** (*type:* string) not_set
- **[localisation]** (*type:* string into ["elem", "som", "faces"]) Localisation of post-processed field value

---

stat_post_t_deb

Synonyms: t_deb

Inherits from: *objet_lecture*

not_set

Parameters are:

- **val** (*type:* double) not_set

---

stat_post_t_fin

Synonyms: t_fin

Inherits from: *objet_lecture*

not_set

Parameters are:

- **val** (*type:* double) not_set

---

stats_posts

Inherits from: *objet_lecture*

Field's write mode.

Parameters are:

- **mot** (*type:* string into ["dt_post", "nb_pas_dt_post"]) Keyword to set the kind of the field's write frequency. Either a time period or a time step period.
- **period** (*type:* string) Value of the period which can be like (2.*t).
- **champs | fields** (*type:* *list_stat_post*) Post-processed fields.

---

stats_posts_fichier

Inherits from: *objet_lecture*

Field's write mode.

Parameters are:

- **mot** (*type:* string into ["dt_post", "nb_pas_dt_post"]) Keyword to set the kind of the field's write frequency. Either a time period or a time step period.
- **period** (*type:* string) Value of the period which can be like (2.*t).
- **fichier | file** (*type:* string) name of file

---

stats_serie_posts

Inherits from: *objet_lecture*

Post-processing for statistics.

Parameters are:

- **mot** (*type:* string into ["dt_integr"]) Keyword is used to set the statistics period of integration and write period.
- **dt_integr** (*type:* double) Average on dt_integr time interval is post-processed every dt_integr seconds.
- **stat** (*type: list_stat_post*) not_set

---

stats_serie_posts_fichier

Inherits from: *objet_lecture*

Post-processing for statistics.

Parameters are:

- **mot** (*type:* string into ["dt_integr"]) Keyword is used to set the statistics period of integration and write period.
- **dt_integr** (*type:* double) Average on dt_integr time interval is post-processed every dt_integr seconds.
- **fichier | file** (*type:* string) name of file

---

temperature

Inherits from: *objet_lecture*

not_set

Parameters are:

- **bord** (*type:* string) not_set
- **direction** (*type:* int) not_set

---

thi

Inherits from: *objet_lecture*

Keyword for a THI (Homogeneous Isotropic Turbulence) calculation.

Parameters are:

- **init_ec** (*type:* int) Keyword to renormalize initial velocity so that kinetic energy equals to the value given by keyword val_Ec.
- **[val_ec]** (*type:* double) Keyword to impose a value for kinetic energy by velocity renormalizated if init_Ec value is 1.
- **[facon_init]** (*type:* int into ["0", "1"]) Keyword to specify how kinetic energy is computed (0 or 1).
- **[calc_spectre]** (*type:* int into ["0", "1"]) Calculate or not the spectrum of kinetic energy. Files called Sorties_THI are written with inside four columns : time:t global_kinetic_energy:Ec enstrophy:D skewness:S If calc_spectre is set to 1, a file Sorties_THI2_2 is written with three columns : time:t kinetic_energy_at_kc=32 enstrophy_at_kc=32 If calc_spectre is set to 1, a file spectre_xxxxx is written with two columns at each time xxxxx : frequency:k energy:E(k).
- **[periode_calc_spectre]** (*type:* double) Period for calculating spectrum of kinetic energy
- **[spectre_3d]** (*type:* int into ["0", "1"]) Calculate or not the 3D spectrum
- **[spectre_1d]** (*type:* int into ["0", "1"]) Calculate or not the 1D spectrum
- **[conservation_ec]** (*type:* flag) If set to 1, velocity field will be changed as to have a constant kinetic energy (default 0)
- **[longueur_boite]** (*type:* double) Length of the calculation domain

---

traitement_particulier

Inherits from: *objet_lecture*

Auxiliary class to post-process particular values.

Parameters are:

- **aco** (*type:* string into ["{"]) Opening curly bracket.
- **trait_part** (*type:* *traitement_particulier_base*) Type of traitement_particulier.
- **acof** (*type:* string into ["}"]) Closing curly bracket.

---

traitement_particulier_base

Inherits from: *objet_lecture*

Basic class to post-process particular values.

---

transversale

Inherits from: *objet_lecture*

Class to define the pressure loss in the direction perpendicular to the tube bundle.

Parameters are:

- **dir** (*type:* string into ["x", "y", "z"]) Direction.
- **dd** (*type:* double) Value of the tube bundle step.
- **chaine_d** (*type:* string into ["d"]) Keyword to be used to set the value of the tube external diameter.
- **d** (*type:* double) Value of the tube external diameter.
- **ch_a** (*type:* string into ["a", "cf"]) Keyword to be used to set law coefficient values for the coefficient of regular pressure losses.
- **a** (*type:* double) Value of a law coefficient for regular pressure losses.
- **[ch_b]** (*type:* string into ["b"]) Keyword to be used to set law coefficient values for regular pressure losses.
- **[b]** (*type:* double) Value of a law coefficient for regular pressure losses.

---

troisf

Inherits from: *objet_lecture*

Auxiliary class to extrude.

Parameters are:

- **lx** (*type:* double) X direction of the extrude operation.
- **ly** (*type:* double) Y direction of the extrude operation.
- **lz** (*type:* double) Z direction of the extrude operation.

---

troismots

Inherits from: *objet_lecture*

Three words.

Parameters are:

- **mot_1** (*type:* string) First word.
- **mot_2** (*type:* string) Snd word.
- **mot_3** (*type:* string) Third word.

---

twofloat

Inherits from: *objet_lecture*

two reals.

Parameters are:

- **a** (*type:* double) First real.
- **b** (*type:* double) Second real.

---

type_diffusion_turbulente_multiphase_deriv

Inherits from: *objet_lecture*

not_set

---

type_diffusion_turbulente_multiphase_l_melange

Synonyms: l_melange

Inherits from: *objet_lecture*

not_set

Parameters are:

- **l_melange** (*type:* double) not_set

---

type_diffusion_turbulente_multiphase_prandtl

Synonyms: prandtl

Inherits from: *objet_lecture*

Scalar Prandtl model.

Parameters are:

- **[pr_t | prandtl_turbulent]** (*type:* double) Prandtl's model constant. By default it is se to 0.9.

---

type_diffusion_turbulente_multiphase_sgdh

Synonyms: sgdh

Inherits from: *objet_lecture*

not_set

Parameters are:

- **[pr_t | prandtl_turbulent]** (*type:* double) not_set
- **[sigma | sigma_turbulent]** (*type:* double) not_set
- **[no_alpha]** (*type:* flag) not_set
- **[gas_turb]** (*type:* flag) not_set

---

type_diffusion_turbulente_multiphase_smago

Synonyms: smago

Inherits from: *objet_lecture*

LES Smagorinsky type.

Parameters are:

- **[cs]** (*type:* double) Smagorinsky's model constant. By default it is se to 0.18.

---

type_diffusion_turbulente_multiphase_wale

Synonyms: wale

Inherits from: *objet_lecture*

LES WALE type.

Parameters are:

- **[cw]** (*type:* double) WALE's model constant. By default it is se to 0.5.

---

type_perte_charge_deriv

Inherits from: *objet_lecture*

not_set

---

type_perte_charge_dp

Synonyms: dp

Inherits from: *objet_lecture*

DP field should have 3 components defining dp, dDP/dQ, Q0

Parameters are:

- **dp_field** (*type: field_base*) the parameters of the previous formula (DP = dp + dDP/dQ * (Q - Q0)): uniform_field 3 dp dDP/dQ Q0 where Q0 is a mass flow rate (kg/s).

---

type_perte_charge_dp_regul

Synonyms: dp_regul

Inherits from: *objet_lecture*

Keyword used to regulate the DP value in order to match a target flow rate. Syntax : dp_regul { DP0 d deb d eps e }

Parameters are:

- **dp0** (*type:* double) initial value of DP
- **deb** (*type:* string) target flow rate in kg/s
- **eps** (*type:* string) strength of the regulation (low values might be slow to find the target flow rate, high values might oscillate around the target value)

---

type_postraitement_ft_lata

Inherits from: *objet_lecture*

not_set

Parameters are:

- **type** (*type:* string into ["postraitement_ft_lata", "postraitement_lata"]) not_set
- **nom** (*type:* string) Name of the post-processing.
- **bloc** (*type:* string) not_set

---

type_un_post

Inherits from: *objet_lecture*

not_set

Parameters are:

- **type** (*type:* string into ["postraitement", "post_processing"]) not_set
- **post** (*type: un_postraitement*) not_set

---

un_pb

Inherits from: *objet_lecture*

pour les groupes

Parameters are:

- **mot** (*type:* *pb_base*) the string

---

un_point

Inherits from: *objet_lecture*

A point.

Parameters are:

- **pos** (*type:* double list) Point coordinates.

---

un_postraitement

Inherits from: *objet_lecture*

An object of post-processing (with name).

Parameters are:

- **nom** (*type:* string) Name of the post-processing.
- **post** (*type:* *corps_postraitement*) Definition of the post-processing.

---

un_postraitement_spec

Inherits from: *objet_lecture*

An object of post-processing (with type +name).

Parameters are:

- **[type_un_post]** (*type:* *type_un_post*) not_set
- **[type_postraitement_ft_lata]** (*type:* *type_postraitement_ft_lata*) not_set

---

verifiercoin_bloc

Inherits from: *objet_lecture*

not_set

Parameters are:

- **[read_file | filename | lire_fichier]** (*type:* string) name of the *.decoupage_som file
- **[expert_only]** (*type:* flag) to not check the mesh

---

volume

Inherits from: *objet_lecture*

Keyword to define the probe volume in a parallelepiped passing through 4 points and the number of probes in each direction.

Parameters are:

- **nbr** (*type:* int) Number of probes in the first direction.
- **nbr2** (*type:* int) Number of probes in the second direction.
- **nbr3** (*type:* int) Number of probes in the third direction.
- **point_deb** (*type:* *un_point*) Point of origin.
- **point_fin** (*type:* *un_point*) Point defining the first direction (from point of origin).
- **point_fin_2** (*type:* *un_point*) Point defining the second direction (from point of origin).
- **point_fin_3** (*type:* *un_point*) Point defining the third direction (from point of origin).

2.1.47 Keywords derived from partitionneur_deriv

partitionneur_deriv

not_set

Parameters are:

- **[nb_parts]** (*type:* int) The number of non empty parts that must be generated (generally equal to the number of processors in the parallel run).

---

partitionneur_fichier_decoupage

Synonyms: fichier_decoupage

Inherits from: *partitionneur_deriv*

This algorithm reads an array of integer values on the disc, one value for each mesh element. Each value is interpreted as the target part number n>=0 for this element. The number of parts created is the highest value in the array plus one. Empty parts can be created if some values are not present in the array.

The file format is ASCII, and contains space, tab or carriage-return separated integer values. The first value is the number nb_elem of elements in the domain, followed by nb_elem integer values (positive or zero).

This algorithm has been designed to work together with the 'ecrire_decoupage' option. You can generate a partition with any other algorithm, write it to disc, modify it, and read it again to generate the .Zone files.

Contrary to other partitioning algorithms, no correction is applied by default to the partition (eg. element 0 on processor 0 and corrections for periodic boundaries). If 'corriger_partition' is specified, these corrections are applied.

Parameters are:

- **fichier** (*type:* string) FILENAME
- **[corriger_partition]** (*type:* flag) not_set
- **[nb_parts]** (*type:* int) The number of non empty parts that must be generated (generally equal to the number of processors in the parallel run).

---

partitionneur_fichier_med

Synonyms: fichier_med

Inherits from: *partitionneur_deriv*

Partitioning a domain using a MED file containing an integer field providing for each element the processor number on which the element should be located.

Parameters are:

- **file** (*type:* string) file name of the MED file to load
- **[field]** (*type:* string) field name of the integer (or double) field to load
- **[nb_parts]** (*type:* int) The number of non empty parts that must be generated (generally equal to the number of processors in the parallel run).

---

partitionneur_metis

Synonyms: metis

Inherits from: *partitionneur_deriv*

Metis is an external partitionning library. It is a general algorithm that will generate a partition of the domain.

Parameters are:

- **[kmetis]** (*type:* flag) The default values are pmetis, default parameters are automatically chosen by Metis. 'kmetis' is faster than pmetis option but the last option produces better partitioning quality. In both cases, the partitioning quality may be slightly improved by increasing the nb_essais option (by default N=1). It will compute N partitions and will keep the best one (smallest edge cut number). But this option is CPU expensive, taking N=10 will multiply the CPU cost of partitioning by 10. Experiments show that only marginal improvements can be obtained with non default parameters.
- **[use_weights]** (*type:* flag) If use_weights is specified, weighting of the element-element links in the graph is used to force metis to keep opposite periodic elements on the same processor. This option can slightly improve the partitionning quality but it consumes more memory and takes more time. It is not mandatory since a correction algorithm is always applied afterwards to ensure a correct partitionning for periodic boundaries.
- **[nb_parts]** (*type:* int) The number of non empty parts that must be generated (generally equal to the number of processors in the parallel run).

---

partitionneur_partition

Synonyms: decouper, partition

Inherits from: *partitionneur_deriv*

This algorithm re-use the partition of the domain named DOMAINE_NAME. It is useful to partition for example a post processing domain. The partition should match with the calculation domain.

Parameters are:

- **domaine** (*type:* *domaine*) domain name
- **[nb_parts]** (*type:* int) The number of non empty parts that must be generated (generally equal to the number of processors in the parallel run).

---

partitionneur_sous_dom

Synonyms: sous_dom

Inherits from: *partitionneur_deriv*

Given a global partition of a global domain, 'sous-domaine' allows to produce a conform partition of a sub-domain generated from the bigger one using the keyword create_domain_from_sous_domaine. The sub-domain will be partitionned in a conform fashion with the global domain.

Parameters are:

- **fichier** (*type:* string) fichier
- **fichier_ssz** (*type:* string) fichier sous zonne
- **[nb_parts]** (*type:* int) The number of non empty parts that must be generated (generally equal to the number of processors in the parallel run).

---

partitionneur_sous_domaines

Synonyms: partitionneur_sous_zones

Inherits from: *partitionneur_deriv*

This algorithm will create one part for each specified subdomaine/domain. All elements contained in the first subdomaine/domain are put in the first part, all remaining elements contained in the second subdomaine/domain in the second part, etc…

If all elements of the current domain are contained in the specified subdomaines/domain, then N parts are created, otherwise, a supplemental part is created with the remaining elements.

If no subdomaine is specified, all subdomaines defined in the domain are used to split the mesh.

Parameters are:

- **[sous_zones]** (*type:* string list) N SUBZONE_NAME_1 SUBZONE_NAME_2 …
- **[domaines]** (*type:* string list) N DOMAIN_NAME_1 DOMAIN_NAME_2 …
- **[nb_parts]** (*type:* int) The number of non empty parts that must be generated (generally equal to the number of processors in the parallel run).

---

partitionneur_sous_zones

Synonyms: sous_zones, partitionneur_sous_domaines

Inherits from: *partitionneur_deriv*

This algorithm will create one part for each specified subzone. All elements contained in the first subzone are put in the first part, all remaining elements contained in the second subzone in the second part, etc…

If all elements of the domain are contained in the specified subzones, then N parts are created, otherwise, a supplemental part is created with the remaining elements.

If no subzone is specified, all subzones defined in the domain are used to split the mesh.

Parameters are:

- **sous_zones** (*type:* string list) N SUBZONE_NAME_1 SUBZONE_NAME_2 …
- **[nb_parts]** (*type:* int) The number of non empty parts that must be generated (generally equal to the number of processors in the parallel run).

---

partitionneur_tranche

Synonyms: tranche

Inherits from: *partitionneur_deriv*

This algorithm will create a geometrical partitionning by slicing the mesh in the two or three axis directions, based on the geometric center of each mesh element. nz must be given if dimension=3. Each slice contains the same number of elements (slices don't have the same geometrical width, and for VDF meshes, slice boundaries are generally not flat except if the number of mesh elements in each direction is an exact multiple of the number of slices). First, nx slices in the X direction are created, then each slice is split in ny slices in the Y direction, and finally, each part is split in nz slices in the Z direction. The resulting number of parts is nx*ny*nz. If one particular direction has been declared

periodic, the default slicing (0, 1, 2, . . . , n-1)is replaced by (0, 1, 2, . . . n-1, 0), each of the two '0' slices having twice less elements than the other slices.

Parameters are:

- **[tranches]** (*type:* int list - size is dimension) Partitioned by nx in the X direction, ny in the Y direction, nz in the Z direction. Works only for structured meshes. No warranty for unstructured meshes.
- **[nb_parts]** (*type:* int) The number of non empty parts that must be generated (generally equal to the number of processors in the parallel run).

---

partitionneur_union

Synonyms: union

Inherits from: *partitionneur_deriv*

Let several local domains be generated from a bigger one using the keyword create_domain_from_sous_domaine, and let their partitions be generated in the usual way. Provided the list of partition files for each small domain, the keyword 'union' will partition the global domain in a conform fashion with the smaller domains.

Parameters are:

- **liste** (*type: bloc_lecture*) List of the partition files with the following syntaxe: {sous_domaine1 decoupage1 . . . sous_domaineim decoupageim } where sous_domaine1 . . . sous_zomeim are small domains names and decoupage1 . . . decoupageim are partition files.
- **[nb_parts]** (*type:* int) The number of non empty parts that must be generated (generally equal to the number of processors in the parallel run).

2.1.48 Keywords derived from pb_gen_base

coupled_problem

Synonyms: probleme_couple

Inherits from: *pb_gen_base*

This instruction causes a probleme_couple type object to be created. This type of object has an associated problem list, that is, the coupling of n problems among them may be processed. Coupling between these problems is carried out explicitly via conditions at particular contact limits. Each problem may be associated either with the Associate keyword or with the Read/groupes keywords. The difference is that in the first case, the four problems exchange values then calculate their timestep, rather in the second case, the same strategy is used for all the problems listed inside one group, but the second group of problem exchange values with the first group of problems after the first group did its timestep. So, the first case may then also be written like this:

Probleme_Couple pbc

Read pbc { groupes { { pb1 , pb2 , pb3 , pb4 } } }

There is a physical environment per problem (however, the same physical environment could be common to several problems).

Each problem is resolved in a domain.

Warning : Presently, coupling requires coincident meshes. In case of non-coincident meshes, boundary condition 'paroi_contact' in VEF returns error message (see paroi_contact for correcting procedure).

Parameters are:

- **[groupes]** (*type: list_list_nom*) { groupes { { pb1 , pb2 } , { pb3 , pb4 } } }

---

pb_avec_liste_conc

Inherits from: *pb_gen_base*

Class to create a classical problem with a list of scalar concentration equations.

Parameters are:

- **list_equations** (*type: listeqn*) convection_diffusion_concentration equations. The unknown of the concentration equation number N is named concentrationN. This keyword is used to define initial conditions and the post processing fields. This kind of problem is very useful to test in only one data file (and then only one calculation) different schemes or different boundary conditions for the scalar transport equation.
- **[milieu]** (*type: milieu_base*) The medium associated with the problem.
- **[constituant]** (*type: constituant*) Constituent.
- **[postraitement | post_processing]** (*type: corps_postraitement*) One post-processing (without name).
- **[postraitements | post_processings]** (*type: postraitements*) List of Postraitement objects (with name).
- **[liste_de_postraitements]** (*type: liste_post_ok*) This
- **[liste_postraitements]** (*type: liste_post*) This block defines the output files to be written during the computation. The output format is lata in order to use OpenDX to draw the results. This block can be divided in one or several sub-blocks that can be written at different frequencies and in different directories. Attention. The directory lata used in this example should be created before running the computation or the lata files will be lost.
- **[sauvegarde]** (*type: format_file*) Keyword used when calculation results are to be backed up. When a coupling is performed, the backup-recovery file name must be well specified for each problem. In this case, you must save to different files and correctly specify these files when resuming the calculation.
- **[sauvegarde_simple]** (*type: format_file*) The same keyword than Sauvegarde except, the last time step only is saved.
- **[reprise]** (*type: format_file*) Keyword to resume a calculation based on the name_file file (see the class format_file). If format_reprise is xyz, the name_file file should be the .xyz file created by the previous calculation. With this file, it is possible to resume a parallel calculation on P processors, whereas the previous calculation has been run on N (N<>P) processors. Should the calculation be resumed, values for the tinit (see schema_temps_base) time fields are taken from the name_file file. If there is no backup corresponding to this time in the name_file, TRUST exits in error.
- **[resume_last_time]** (*type: format_file*) Keyword to resume a calculation based on the name_file file, resume the calculation at the last time found in the file (tinit is set to last time of saved files).

---

pb_avec_passif

Inherits from: *pb_gen_base*

Class to create a classical problem with a scalar transport equation (e.g: temperature or concentration) and an additional set of passive scalars (e.g: temperature or concentration) equations.

Parameters are:

- **equations_scalaires_passifs** (*type: listeqn*) Passive scalar equations. The unknowns of the passive scalar equation number N are named temperatureN or concentrationN or fraction_massiqueN. This keyword is used to define initial conditions and the post processing fields. This kind of problem is very useful to test in only one data file (and then only one calculation) different schemes or different boundary conditions for the scalar transport equation.
- **[milieu]** (*type: milieu_base*) The medium associated with the problem.
- **[constituant]** (*type: constituant*) Constituent.
- **[postraitement | post_processing]** (*type: corps_postraitement*) One post-processing (without name).
- **[postraitements | post_processings]** (*type: postraitements*) List of Postraitement objects (with name).
- **[liste_de_postraitements]** (*type: liste_post_ok*) This
- **[liste_postraitements]** (*type: liste_post*) This block defines the output files to be written during the computation. The output format is lata in order to use OpenDX to draw the results. This block can be divided in one or several sub-blocks that can be written at different frequencies and in different directories. Attention. The directory lata used in this example should be created before running the computation or the lata files will be lost.
- **[sauvegarde]** (*type: format_file*) Keyword used when calculation results are to be backed up. When a coupling is performed, the backup-recovery file name must be well specified for each problem. In this case, you must save to different files and correctly specify these files when resuming the calculation.
- **[sauvegarde_simple]** (*type: format_file*) The same keyword than Sauvegarde except, the last time step only is saved.
- **[reprise]** (*type: format_file*) Keyword to resume a calculation based on the name_file file (see the class format_file). If format_reprise is xyz, the name_file file should be the .xyz file created by the previous calculation. With this file, it is possible to resume a parallel calculation on P processors, whereas the previous calculation has been run on N (N<>P) processors. Should the calculation be resumed, values for the tinit (see schema_temps_base) time fields are taken from the name_file file. If there is no backup corresponding to this time in the name_file, TRUST exits in error.
- **[resume_last_time]** (*type: format_file*) Keyword to resume a calculation based on the name_file file, resume the calculation at the last time found in the file (tinit is set to last time of saved files).

---

pb_base

Inherits from: *pb_gen_base*

Resolution of equations on a domain. A problem is defined by creating an object and assigning the problem type that the user wishes to resolve. To enter values for the problem objects created, the Lire (Read) interpretor is used with a data block.

Parameters are:

- **[milieu]** (*type: milieu_base*) The medium associated with the problem.
- **[constituant]** (*type: constituant*) Constituent.

- **[postraitement | post_processing]** (*type: corps_postraitement*) One post-processing (without name).
- **[postraitements | post_processings]** (*type: postraitements*) List of Postraitement objects (with name).
- **[liste_de_postraitements]** (*type: liste_post_ok*) This
- **[liste_postraitements]** (*type: liste_post*) This block defines the output files to be written during the computation. The output format is lata in order to use OpenDX to draw the results. This block can be divided in one or several sub-blocks that can be written at different frequencies and in different directories. Attention. The directory lata used in this example should be created before running the computation or the lata files will be lost.
- **[sauvegarde]** (*type: format_file*) Keyword used when calculation results are to be backed up. When a coupling is performed, the backup-recovery file name must be well specified for each problem. In this case, you must save to different files and correctly specify these files when resuming the calculation.
- **[sauvegarde_simple]** (*type: format_file*) The same keyword than Sauvegarde except, the last time step only is saved.
- **[reprise]** (*type: format_file*) Keyword to resume a calculation based on the name_file file (see the class format_file). If format_reprise is xyz, the name_file file should be the .xyz file created by the previous calculation. With this file, it is possible to resume a parallel calculation on P processors, whereas the previous calculation has been run on N (N<>P) processors. Should the calculation be resumed, values for the tinit (see schema_temps_base) time fields are taken from the name_file file. If there is no backup corresponding to this time in the name_file, TRUST exits in error.
- **[resume_last_time]** (*type: format_file*) Keyword to resume a calculation based on the name_file file, resume the calculation at the last time found in the file (tinit is set to last time of saved files).

---

pb_conduction

Inherits from: *pb_gen_base*

Resolution of the heat equation.

Parameters are:

- **[solide]** (*type: solide*) The medium associated with the problem.
- **[conduction]** (*type: conduction*) Heat equation.
- **[milieu]** (*type: milieu_base*) The medium associated with the problem.
- **[constituant]** (*type: constituant*) Constituent.
- **[postraitement | post_processing]** (*type: corps_postraitement*) One post-processing (without name).
- **[postraitements | post_processings]** (*type: postraitements*) List of Postraitement objects (with name).
- **[liste_de_postraitements]** (*type: liste_post_ok*) This
- **[liste_postraitements]** (*type: liste_post*) This block defines the output files to be written during the computation. The output format is lata in order to use OpenDX to draw the results. This block can be divided in one or several sub-blocks that can be written at different frequencies and in different directories. Attention. The directory lata used in this example should be created before running the computation or the lata files will be lost.
- **[sauvegarde]** (*type: format_file*) Keyword used when calculation results are to be backed up. When a coupling is performed, the backup-recovery file name must be well specified for each problem. In this case, you must save to different files and correctly specify these files when resuming the calculation.
- **[sauvegarde_simple]** (*type: format_file*) The same keyword than Sauvegarde except, the last time step only is saved.

- **[reprise]** (*type: format_file*) Keyword to resume a calculation based on the name_file file (see the class format_file). If format_reprise is xyz, the name_file file should be the .xyz file created by the previous calculation. With this file, it is possible to resume a parallel calculation on P processors, whereas the previous calculation has been run on N (N<>P) processors. Should the calculation be resumed, values for the tinit (see schema_temps_base) time fields are taken from the name_file file. If there is no backup corresponding to this time in the name_file, TRUST exits in error.
- **[resume_last_time]** (*type: format_file*) Keyword to resume a calculation based on the name_file file, resume the calculation at the last time found in the file (tinit is set to last time of saved files).

---

pb_gen_base

Basic class for problems.

---

pb_hydraulique

Inherits from: *pb_gen_base*

Resolution of the Navier-Stokes equations.

Parameters are:

- **fluide_incompressible** (*type: fluide_incompressible*) The fluid medium associated with the problem.
- **navier_stokes_standard** (*type: navier_stokes_standard*) Navier-Stokes equations.
- **[milieu]** (*type: milieu_base*) The medium associated with the problem.
- **[constituant]** (*type: constituant*) Constituent.
- **[postraitement | post_processing]** (*type: corps_postraitement*) One post-processing (without name).
- **[postraitements | post_processings]** (*type: postraitements*) List of Postraitement objects (with name).
- **[liste_de_postraitements]** (*type: liste_post_ok*) This
- **[liste_postraitements]** (*type: liste_post*) This block defines the output files to be written during the computation. The output format is lata in order to use OpenDX to draw the results. This block can be divided in one or several sub-blocks that can be written at different frequencies and in different directories. Attention. The directory lata used in this example should be created before running the computation or the lata files will be lost.
- **[sauvegarde]** (*type: format_file*) Keyword used when calculation results are to be backed up. When a coupling is performed, the backup-recovery file name must be well specified for each problem. In this case, you must save to different files and correctly specify these files when resuming the calculation.
- **[sauvegarde_simple]** (*type: format_file*) The same keyword than Sauvegarde except, the last time step only is saved.
- **[reprise]** (*type: format_file*) Keyword to resume a calculation based on the name_file file (see the class format_file). If format_reprise is xyz, the name_file file should be the .xyz file created by the previous calculation. With this file, it is possible to resume a parallel calculation on P processors, whereas the previous calculation has been run on N (N<>P) processors. Should the calculation be resumed, values for the tinit (see schema_temps_base) time fields are taken from the name_file file. If there is no backup corresponding to this time in the name_file, TRUST exits in error.
- **[resume_last_time]** (*type: format_file*) Keyword to resume a calculation based on the name_file file, resume the calculation at the last time found in the file (tinit is set to last time of saved files).

pb_hydraulique_cloned_concentration

Inherits from: *pb_gen_base*

Resolution of Navier-Stokes/multiple constituent transport equations.

Parameters are:

- **fluide_incompressible** (*type: fluide_incompressible*) The fluid medium associated with the problem.
- **[constituant]** (*type: constituant*) Constituents.
- **[navier_stokes_standard]** (*type: navier_stokes_standard*) Navier-Stokes equations.
- **[convection_diffusion_concentration]** (*type: convection_diffusion_concentration*) Constituent transport vectorial equation (concentration diffusion convection).
- **[milieu]** (*type: milieu_base*) The medium associated with the problem.
- **[postraitement | post_processing]** (*type: corps_postraitement*) One post-processing (without name).
- **[postraitements | post_processings]** (*type: postraitements*) List of Postraitement objects (with name).
- **[liste_de_postraitements]** (*type: liste_post_ok*) This
- **[liste_postraitements]** (*type: liste_post*) This block defines the output files to be written during the computation. The output format is lata in order to use OpenDX to draw the results. This block can be divided in one or several sub-blocks that can be written at different frequencies and in different directories. Attention. The directory lata used in this example should be created before running the computation or the lata files will be lost.
- **[sauvegarde]** (*type: format_file*) Keyword used when calculation results are to be backed up. When a coupling is performed, the backup-recovery file name must be well specified for each problem. In this case, you must save to different files and correctly specify these files when resuming the calculation.
- **[sauvegarde_simple]** (*type: format_file*) The same keyword than Sauvegarde except, the last time step only is saved.
- **[reprise]** (*type: format_file*) Keyword to resume a calculation based on the name_file file (see the class format_file). If format_reprise is xyz, the name_file file should be the .xyz file created by the previous calculation. With this file, it is possible to resume a parallel calculation on P processors, whereas the previous calculation has been run on N (N<>P) processors. Should the calculation be resumed, values for the tinit (see schema_temps_base) time fields are taken from the name_file file. If there is no backup corresponding to this time in the name_file, TRUST exits in error.
- **[resume_last_time]** (*type: format_file*) Keyword to resume a calculation based on the name_file file, resume the calculation at the last time found in the file (tinit is set to last time of saved files).

pb_hydraulique_cloned_concentration_turbulent

Inherits from: *pb_gen_base*

Resolution of Navier-Stokes/multiple constituent transport equations, with turbulence modelling.

Parameters are:

- **fluide_incompressible** (*type: fluide_incompressible*) The fluid medium associated with the problem.
- **[constituant]** (*type: constituant*) Constituents.
- **[navier_stokes_turbulent]** (*type: navier_stokes_turbulent*) Navier-Stokes equations as well as the associated turbulence model equations.
- **[convection_diffusion_concentration_turbulent]** (*type: convection_diffusion_concentration_turbulent*) Constituent transport equations (concentration diffusion convection) as well as the associated turbulence model equations.
- **[milieu]** (*type: milieu_base*) The medium associated with the problem.
- **[postraitement | post_processing]** (*type: corps_postraitement*) One post-processing (without name).
- **[postraitements | post_processings]** (*type: postraitements*) List of Postraitement objects (with name).
- **[liste_de_postraitements]** (*type: liste_post_ok*) This
- **[liste_postraitements]** (*type: liste_post*) This block defines the output files to be written during the computation. The output format is lata in order to use OpenDX to draw the results. This block can be divided in one or several sub-blocks that can be written at different frequencies and in different directories. Attention. The directory lata used in this example should be created before running the computation or the lata files will be lost.
- **[sauvegarde]** (*type: format_file*) Keyword used when calculation results are to be backed up. When a coupling is performed, the backup-recovery file name must be well specified for each problem. In this case, you must save to different files and correctly specify these files when resuming the calculation.
- **[sauvegarde_simple]** (*type: format_file*) The same keyword than Sauvegarde except, the last time step only is saved.
- **[reprise]** (*type: format_file*) Keyword to resume a calculation based on the name_file file (see the class format_file). If format_reprise is xyz, the name_file file should be the .xyz file created by the previous calculation. With this file, it is possible to resume a parallel calculation on P processors, whereas the previous calculation has been run on N (N<>P) processors. Should the calculation be resumed, values for the tinit (see schema_temps_base) time fields are taken from the name_file file. If there is no backup corresponding to this time in the name_file, TRUST exits in error.
- **[resume_last_time]** (*type: format_file*) Keyword to resume a calculation based on the name_file file, resume the calculation at the last time found in the file (tinit is set to last time of saved files).

---

pb_hydraulique_concentration

Inherits from: *pb_gen_base*

Resolution of Navier-Stokes/multiple constituent transport equations.

Parameters are:

- **fluide_incompressible** (*type: fluide_incompressible*) The fluid medium associated with the problem.
- **[constituant]** (*type: constituant*) Constituents.

- **[navier_stokes_standard]** (*type: navier_stokes_standard*) Navier-Stokes equations.
- **[convection_diffusion_concentration]** (*type: convection_diffusion_concentration*) Constituent transport vectorial equation (concentration diffusion convection).
- **[milieu]** (*type: milieu_base*) The medium associated with the problem.
- **[postraitement | post_processing]** (*type: corps_postraitement*) One post-processing (without name).
- **[postraitements | post_processings]** (*type: postraitements*) List of Postraitement objects (with name).
- **[liste_de_postraitements]** (*type: liste_post_ok*) This
- **[liste_postraitements]** (*type: liste_post*) This block defines the output files to be written during the computation. The output format is lata in order to use OpenDX to draw the results. This block can be divided in one or several sub-blocks that can be written at different frequencies and in different directories. Attention. The directory lata used in this example should be created before running the computation or the lata files will be lost.
- **[sauvegarde]** (*type: format_file*) Keyword used when calculation results are to be backed up. When a coupling is performed, the backup-recovery file name must be well specified for each problem. In this case, you must save to different files and correctly specify these files when resuming the calculation.
- **[sauvegarde_simple]** (*type: format_file*) The same keyword than Sauvegarde except, the last time step only is saved.
- **[reprise]** (*type: format_file*) Keyword to resume a calculation based on the name_file file (see the class format_file). If format_reprise is xyz, the name_file file should be the .xyz file created by the previous calculation. With this file, it is possible to resume a parallel calculation on P processors, whereas the previous calculation has been run on N (N<>P) processors. Should the calculation be resumed, values for the tinit (see schema_temps_base) time fields are taken from the name_file file. If there is no backup corresponding to this time in the name_file, TRUST exits in error.
- **[resume_last_time]** (*type: format_file*) Keyword to resume a calculation based on the name_file file, resume the calculation at the last time found in the file (tinit is set to last time of saved files).

---

pb_hydraulique_concentration_scalaires_passifs

Inherits from: *pb_gen_base*

Resolution of Navier-Stokes/multiple constituent transport equations with the additional passive scalar equations.

Parameters are:

- **fluide_incompressible** (*type: fluide_incompressible*) The fluid medium associated with the problem.
- **[constituant]** (*type: constituant*) Constituents.
- **[navier_stokes_standard]** (*type: navier_stokes_standard*) Navier-Stokes equations.
- **[convection_diffusion_concentration]** (*type: convection_diffusion_concentration*) Constituent transport equations (concentration diffusion convection).
- **equations_scalaires_passifs** (*type: listeqn*) Passive scalar equations. The unknowns of the passive scalar equation number N are named temperatureN or concentrationN or fraction_massiqueN. This keyword is used to define initial conditions and the post processing fields. This kind of problem is very useful to test in only one data file (and then only one calculation) different schemes or different boundary conditions for the scalar transport equation.
- **[milieu]** (*type: milieu_base*) The medium associated with the problem.
- **[postraitement | post_processing]** (*type: corps_postraitement*) One post-processing (without name).

- **[postraitements | post_processings]** (*type: postraitements*) List of Postraitement objects (with name).
- **[liste_de_postraitements]** (*type: liste_post_ok*) This
- **[liste_postraitements]** (*type: liste_post*) This block defines the output files to be written during the computation. The output format is lata in order to use OpenDX to draw the results. This block can be divided in one or several sub-blocks that can be written at different frequencies and in different directories. Attention. The directory lata used in this example should be created before running the computation or the lata files will be lost.
- **[sauvegarde]** (*type: format_file*) Keyword used when calculation results are to be backed up. When a coupling is performed, the backup-recovery file name must be well specified for each problem. In this case, you must save to different files and correctly specify these files when resuming the calculation.
- **[sauvegarde_simple]** (*type: format_file*) The same keyword than Sauvegarde except, the last time step only is saved.
- **[reprise]** (*type: format_file*) Keyword to resume a calculation based on the name_file file (see the class format_file). If format_reprise is xyz, the name_file file should be the .xyz file created by the previous calculation. With this file, it is possible to resume a parallel calculation on P processors, whereas the previous calculation has been run on N (N<>P) processors. Should the calculation be resumed, values for the tinit (see schema_temps_base) time fields are taken from the name_file file. If there is no backup corresponding to this time in the name_file, TRUST exits in error.
- **[resume_last_time]** (*type: format_file*) Keyword to resume a calculation based on the name_file file, resume the calculation at the last time found in the file (tinit is set to last time of saved files).

---

pb_hydraulique_concentration_turbulent

Inherits from: *pb_gen_base*

Resolution of Navier-Stokes/multiple constituent transport equations, with turbulence modelling.

Parameters are:

- **fluide_incompressible** (*type: fluide_incompressible*) The fluid medium associated with the problem.
- **[constituant]** (*type: constituant*) Constituents.
- **[navier_stokes_turbulent]** (*type: navier_stokes_turbulent*) Navier-Stokes equations as well as the associated turbulence model equations.
- **[convection_diffusion_concentration_turbulent]** (*type: convection_diffusion_concentration_turbulent*) Constituent transport equations (concentration diffusion convection) as well as the associated turbulence model equations.
- **[milieu]** (*type: milieu_base*) The medium associated with the problem.
- **[postraitement | post_processing]** (*type: corps_postraitement*) One post-processing (without name).
- **[postraitements | post_processings]** (*type: postraitements*) List of Postraitement objects (with name).
- **[liste_de_postraitements]** (*type: liste_post_ok*) This
- **[liste_postraitements]** (*type: liste_post*) This block defines the output files to be written during the computation. The output format is lata in order to use OpenDX to draw the results. This block can be divided in one or several sub-blocks that can be written at different frequencies and in different directories. Attention. The directory lata used in this example should be created before running the computation or the lata files will be lost.
- **[sauvegarde]** (*type: format_file*) Keyword used when calculation results are to be backed up. When a coupling is performed, the backup-recovery file name must be well specified for each problem. In this case, you must save to different files and correctly specify these files when resuming the calculation.

- **[sauvegarde_simple]** (*type: format_file*) The same keyword than Sauvegarde except, the last time step only is saved.
- **[reprise]** (*type: format_file*) Keyword to resume a calculation based on the name_file file (see the class format_file). If format_reprise is xyz, the name_file file should be the .xyz file created by the previous calculation. With this file, it is possible to resume a parallel calculation on P processors, whereas the previous calculation has been run on N (N<>P) processors. Should the calculation be resumed, values for the tinit (see schema_temps_base) time fields are taken from the name_file file. If there is no backup corresponding to this time in the name_file, TRUST exits in error.
- **[resume_last_time]** (*type: format_file*) Keyword to resume a calculation based on the name_file file, resume the calculation at the last time found in the file (tinit is set to last time of saved files).

---

pb_hydraulique_concentration_turbulent_scalaires_passifs

Inherits from: *pb_gen_base*

Resolution of Navier-Stokes/multiple constituent transport equations, with turbulence modelling and with the additional passive scalar equations.

Parameters are:

- **fluide_incompressible** (*type: fluide_incompressible*) The fluid medium associated with the problem.
- **[constituant]** (*type: constituant*) Constituents.
- **[navier_stokes_turbulent]** (*type: navier_stokes_turbulent*) Navier-Stokes equations as well as the associated turbulence model equations.
- **[convection_diffusion_concentration_turbulent]** (*type: convection_diffusion_concentration_turbulent*) Constituent transport equations (concentration diffusion convection) as well as the associated turbulence model equations.
- **equations_scalaires_passifs** (*type: listeqn*) Passive scalar equations. The unknowns of the passive scalar equation number N are named temperatureN or concentrationN or fraction_massiqueN. This keyword is used to define initial conditions and the post processing fields. This kind of problem is very useful to test in only one data file (and then only one calculation) different schemes or different boundary conditions for the scalar transport equation.
- **[milieu]** (*type: milieu_base*) The medium associated with the problem.
- **[postraitement | post_processing]** (*type: corps_postraitement*) One post-processing (without name).
- **[postraitements | post_processings]** (*type: postraitements*) List of Postraitement objects (with name).
- **[liste_de_postraitements]** (*type: liste_post_ok*) This
- **[liste_postraitements]** (*type: liste_post*) This block defines the output files to be written during the computation. The output format is lata in order to use OpenDX to draw the results. This block can be divided in one or several sub-blocks that can be written at different frequencies and in different directories. Attention. The directory lata used in this example should be created before running the computation or the lata files will be lost.
- **[sauvegarde]** (*type: format_file*) Keyword used when calculation results are to be backed up. When a coupling is performed, the backup-recovery file name must be well specified for each problem. In this case, you must save to different files and correctly specify these files when resuming the calculation.
- **[sauvegarde_simple]** (*type: format_file*) The same keyword than Sauvegarde except, the last time step only is saved.

- **[reprise]** (*type: format_file*) Keyword to resume a calculation based on the name_file file (see the class format_file). If format_reprise is xyz, the name_file file should be the .xyz file created by the previous calculation. With this file, it is possible to resume a parallel calculation on P processors, whereas the previous calculation has been run on N (N<>P) processors. Should the calculation be resumed, values for the tinit (see schema_temps_base) time fields are taken from the name_file file. If there is no backup corresponding to this time in the name_file, TRUST exits in error.
- **[resume_last_time]** (*type: format_file*) Keyword to resume a calculation based on the name_file file, resume the calculation at the last time found in the file (tinit is set to last time of saved files).

---

pb_hydraulique_list_concentration

Inherits from: *pb_gen_base*

Resolution of Navier-Stokes/multiple constituent transport equations.

Parameters are:

- **fluide_incompressible** (*type: fluide_incompressible*) The fluid medium associated with the problem.
- **[constituant]** (*type: constituant*) Constituents.
- **[navier_stokes_standard]** (*type: navier_stokes_standard*) Navier-Stokes equations.
- **list_equations** (*type: listeqn*) convection_diffusion_concentration equations. The unknown of the concentration equation number N is named concentrationN. This keyword is used to define initial conditions and the post processing fields. This kind of problem is very useful to test in only one data file (and then only one calculation) different schemes or different boundary conditions for the scalar transport equation.
- **[milieu]** (*type: milieu_base*) The medium associated with the problem.
- **[postraitement | post_processing]** (*type: corps_postraitement*) One post-processing (without name).
- **[postraitements | post_processings]** (*type: postraitements*) List of Postraitement objects (with name).
- **[liste_de_postraitements]** (*type: liste_post_ok*) This
- **[liste_postraitements]** (*type: liste_post*) This block defines the output files to be written during the computation. The output format is lata in order to use OpenDX to draw the results. This block can be divided in one or several sub-blocks that can be written at different frequencies and in different directories. Attention. The directory lata used in this example should be created before running the computation or the lata files will be lost.
- **[sauvegarde]** (*type: format_file*) Keyword used when calculation results are to be backed up. When a coupling is performed, the backup-recovery file name must be well specified for each problem. In this case, you must save to different files and correctly specify these files when resuming the calculation.
- **[sauvegarde_simple]** (*type: format_file*) The same keyword than Sauvegarde except, the last time step only is saved.
- **[reprise]** (*type: format_file*) Keyword to resume a calculation based on the name_file file (see the class format_file). If format_reprise is xyz, the name_file file should be the .xyz file created by the previous calculation. With this file, it is possible to resume a parallel calculation on P processors, whereas the previous calculation has been run on N (N<>P) processors. Should the calculation be resumed, values for the tinit (see schema_temps_base) time fields are taken from the name_file file. If there is no backup corresponding to this time in the name_file, TRUST exits in error.
- **[resume_last_time]** (*type: format_file*) Keyword to resume a calculation based on the name_file file, resume the calculation at the last time found in the file (tinit is set to last time of saved files).

---

pb_hydraulique_list_concentration_turbulent

Inherits from: *pb_gen_base*

Resolution of Navier-Stokes/multiple constituent transport equations, with turbulence modelling.

Parameters are:

- **fluide_incompressible** (*type: fluide_incompressible*) The fluid medium associated with the problem.
- **[constituant]** (*type: constituant*) Constituents.
- **[navier_stokes_turbulent]** (*type: navier_stokes_turbulent*) Navier-Stokes equations as well as the associated turbulence model equations.
- **list_equations** (*type: listeqn*) convection_diffusion_concentration equations. The unknown of the concentration equation number N is named concentrationN. This keyword is used to define initial conditions and the post processing fields. This kind of problem is very useful to test in only one data file (and then only one calculation) different schemes or different boundary conditions for the scalar transport equation.
- **[milieu]** (*type: milieu_base*) The medium associated with the problem.
- **[postraitement | post_processing]** (*type: corps_postraitement*) One post-processing (without name).
- **[postraitements | post_processings]** (*type: postraitements*) List of Postraitement objects (with name).
- **[liste_de_postraitements]** (*type: liste_post_ok*) This
- **[liste_postraitements]** (*type: liste_post*) This block defines the output files to be written during the computation. The output format is lata in order to use OpenDX to draw the results. This block can be divided in one or several sub-blocks that can be written at different frequencies and in different directories. Attention. The directory lata used in this example should be created before running the computation or the lata files will be lost.
- **[sauvegarde]** (*type: format_file*) Keyword used when calculation results are to be backed up. When a coupling is performed, the backup-recovery file name must be well specified for each problem. In this case, you must save to different files and correctly specify these files when resuming the calculation.
- **[sauvegarde_simple]** (*type: format_file*) The same keyword than Sauvegarde except, the last time step only is saved.
- **[reprise]** (*type: format_file*) Keyword to resume a calculation based on the name_file file (see the class format_file). If format_reprise is xyz, the name_file file should be the .xyz file created by the previous calculation. With this file, it is possible to resume a parallel calculation on P processors, whereas the previous calculation has been run on N (N<>P) processors. Should the calculation be resumed, values for the tinit (see schema_temps_base) time fields are taken from the name_file file. If there is no backup corresponding to this time in the name_file, TRUST exits in error.
- **[resume_last_time]** (*type: format_file*) Keyword to resume a calculation based on the name_file file, resume the calculation at the last time found in the file (tinit is set to last time of saved files).

pb_hydraulique_melange_binaire_qc

Inherits from: *pb_gen_base*

Resolution of a binary mixture problem for a quasi-compressible fluid with an iso-thermal condition.

Keywords for the unknowns other than pressure, velocity, fraction_massique are :

masse_volumique : density

pression : reduced pressure

pression_tot : total pressure.

Parameters are:

- **fluide_quasi_compressible** (*type: fluide_quasi_compressible*) The fluid medium associated with the problem.
- **[constituant]** (*type: constituant*) The various constituants associated to the problem.
- **navier_stokes_qc** (*type: navier_stokes_qc*) Navier-Stokes equation for a quasi-compressible fluid.
- **convection_diffusion_espece_binaire_qc** (*type: convection_diffusion_espece_binaire_qc*) Species conservation equation for a binary quasi-compressible fluid.
- **[milieu]** (*type: milieu_base*) The medium associated with the problem.
- **[postraitement | post_processing]** (*type: corps_postraitement*) One post-processing (without name).
- **[postraitements | post_processings]** (*type: postraitements*) List of Postraitement objects (with name).
- **[liste_de_postraitements]** (*type: liste_post_ok*) This
- **[liste_postraitements]** (*type: liste_post*) This block defines the output files to be written during the computation. The output format is lata in order to use OpenDX to draw the results. This block can be divided in one or several sub-blocks that can be written at different frequencies and in different directories. Attention. The directory lata used in this example should be created before running the computation or the lata files will be lost.
- **[sauvegarde]** (*type: format_file*) Keyword used when calculation results are to be backed up. When a coupling is performed, the backup-recovery file name must be well specified for each problem. In this case, you must save to different files and correctly specify these files when resuming the calculation.
- **[sauvegarde_simple]** (*type: format_file*) The same keyword than Sauvegarde except, the last time step only is saved.
- **[reprise]** (*type: format_file*) Keyword to resume a calculation based on the name_file file (see the class format_file). If format_reprise is xyz, the name_file file should be the .xyz file created by the previous calculation. With this file, it is possible to resume a parallel calculation on P processors, whereas the previous calculation has been run on N (N<>P) processors. Should the calculation be resumed, values for the tinit (see schema_temps_base) time fields are taken from the name_file file. If there is no backup corresponding to this time in the name_file, TRUST exits in error.
- **[resume_last_time]** (*type: format_file*) Keyword to resume a calculation based on the name_file file, resume the calculation at the last time found in the file (tinit is set to last time of saved files).

pb_hydraulique_melange_binaire_turbulent_qc

Inherits from: *pb_gen_base*

Resolution of a turbulent binary mixture problem for a quasi-compressible fluid with an iso-thermal condition.

Parameters are:

- **fluide_quasi_compressible** (*type: fluide_quasi_compressible*) The fluid medium associated with the problem.
- **navier_stokes_turbulent_qc** (*type: navier_stokes_turbulent_qc*) Navier-Stokes equation for a quasi-compressible fluid as well as the associated turbulence model equations.
- **convection_diffusion_espece_binaire_turbulent_qc** (*type: convection_diffusion_espece_binaire_turbulent_qc*) Species conservation equation for a quasi-compressible fluid as well as the associated turbulence model equations.
- **[milieu]** (*type: milieu_base*) The medium associated with the problem.
- **[constituant]** (*type: constituant*) Constituent.
- **[postraitement | post_processing]** (*type: corps_postraitement*) One post-processing (without name).
- **[postraitements | post_processings]** (*type: postraitements*) List of Postraitement objects (with name).
- **[liste_de_postraitements]** (*type: liste_post_ok*) This
- **[liste_postraitements]** (*type: liste_post*) This block defines the output files to be written during the computation. The output format is lata in order to use OpenDX to draw the results. This block can be divided in one or several sub-blocks that can be written at different frequencies and in different directories. Attention. The directory lata used in this example should be created before running the computation or the lata files will be lost.
- **[sauvegarde]** (*type: format_file*) Keyword used when calculation results are to be backed up. When a coupling is performed, the backup-recovery file name must be well specified for each problem. In this case, you must save to different files and correctly specify these files when resuming the calculation.
- **[sauvegarde_simple]** (*type: format_file*) The same keyword than Sauvegarde except, the last time step only is saved.
- **[reprise]** (*type: format_file*) Keyword to resume a calculation based on the name_file file (see the class format_file). If format_reprise is xyz, the name_file file should be the .xyz file created by the previous calculation. With this file, it is possible to resume a parallel calculation on P processors, whereas the previous calculation has been run on N (N<>P) processors. Should the calculation be resumed, values for the tinit (see schema_temps_base) time fields are taken from the name_file file. If there is no backup corresponding to this time in the name_file, TRUST exits in error.
- **[resume_last_time]** (*type: format_file*) Keyword to resume a calculation based on the name_file file, resume the calculation at the last time found in the file (tinit is set to last time of saved files).

---

pb_hydraulique_melange_binaire_wc

Inherits from: *pb_gen_base*

Resolution of a binary mixture problem for a weakly-compressible fluid with an iso-thermal condition.

Keywords for the unknowns other than pressure, velocity, fraction_massique are :

masse_volumique : density

pression : reduced pressure

pression_tot : total pressure

pression_hydro : hydro-static pressure

pression_eos : pressure used in state equation.

Parameters are:

- **fluide_weakly_compressible** (*type: fluide_weakly_compressible*) The fluid medium associated with the problem.
- **navier_stokes_wc** (*type: navier_stokes_wc*) Navier-Stokes equation for a weakly-compressible fluid.
- **convection_diffusion_espece_binaire_wc** (*type: convection_diffusion_espece_binaire_wc*) Species conservation equation for a binary weakly-compressible fluid.
- **[milieu]** (*type: milieu_base*) The medium associated with the problem.
- **[constituant]** (*type: constituant*) Constituent.
- **[postraitement | post_processing]** (*type: corps_postraitement*) One post-processing (without name).
- **[postraitements | post_processings]** (*type: postraitements*) List of Postraitement objects (with name).
- **[liste_de_postraitements]** (*type: liste_post_ok*) This
- **[liste_postraitements]** (*type: liste_post*) This block defines the output files to be written during the computation. The output format is lata in order to use OpenDX to draw the results. This block can be divided in one or several sub-blocks that can be written at different frequencies and in different directories. Attention. The directory lata used in this example should be created before running the computation or the lata files will be lost.
- **[sauvegarde]** (*type: format_file*) Keyword used when calculation results are to be backed up. When a coupling is performed, the backup-recovery file name must be well specified for each problem. In this case, you must save to different files and correctly specify these files when resuming the calculation.
- **[sauvegarde_simple]** (*type: format_file*) The same keyword than Sauvegarde except, the last time step only is saved.
- **[reprise]** (*type: format_file*) Keyword to resume a calculation based on the name_file file (see the class format_file). If format_reprise is xyz, the name_file file should be the .xyz file created by the previous calculation. With this file, it is possible to resume a parallel calculation on P processors, whereas the previous calculation has been run on N (N<>P) processors. Should the calculation be resumed, values for the tinit (see schema_temps_base) time fields are taken from the name_file file. If there is no backup corresponding to this time in the name_file, TRUST exits in error.
- **[resume_last_time]** (*type: format_file*) Keyword to resume a calculation based on the name_file file, resume the calculation at the last time found in the file (tinit is set to last time of saved files).

---

pb_hydraulique_turbulent

Inherits from: *pb_gen_base*

Resolution of Navier-Stokes equations with turbulence modelling.

Parameters are:

- **fluide_incompressible** (*type: fluide_incompressible*) The fluid medium associated with the problem.
- **navier_stokes_turbulent** (*type: navier_stokes_turbulent*) Navier-Stokes equations as well as the associated turbulence model equations.
- **[milieu]** (*type: milieu_base*) The medium associated with the problem.

- **[constituant]** (*type: constituant*) Constituent.
- **[postraitement | post_processing]** (*type: corps_postraitement*) One post-processing (without name).
- **[postraitements | post_processings]** (*type: postraitements*) List of Postraitement objects (with name).
- **[liste_de_postraitements]** (*type: liste_post_ok*) This
- **[liste_postraitements]** (*type: liste_post*) This block defines the output files to be written during the computation. The output format is lata in order to use OpenDX to draw the results. This block can be divided in one or several sub-blocks that can be written at different frequencies and in different directories. Attention. The directory lata used in this example should be created before running the computation or the lata files will be lost.
- **[sauvegarde]** (*type: format_file*) Keyword used when calculation results are to be backed up. When a coupling is performed, the backup-recovery file name must be well specified for each problem. In this case, you must save to different files and correctly specify these files when resuming the calculation.
- **[sauvegarde_simple]** (*type: format_file*) The same keyword than Sauvegarde except, the last time step only is saved.
- **[reprise]** (*type: format_file*) Keyword to resume a calculation based on the name_file file (see the class format_file). If format_reprise is xyz, the name_file file should be the .xyz file created by the previous calculation. With this file, it is possible to resume a parallel calculation on P processors, whereas the previous calculation has been run on N (N<>P) processors. Should the calculation be resumed, values for the tinit (see schema_temps_base) time fields are taken from the name_file file. If there is no backup corresponding to this time in the name_file, TRUST exits in error.
- **[resume_last_time]** (*type: format_file*) Keyword to resume a calculation based on the name_file file, resume the calculation at the last time found in the file (tinit is set to last time of saved files).

---

pb_multiphase

Inherits from: *pb_gen_base*

A problem that allows the resolution of N-phases with 3*N equations

Parameters are:

- **[milieu_composite]** (*type: bloc_lecture*) The composite medium associated with the problem.
- **[milieu_musig]** (*type: bloc_lecture*) The composite medium associated with the problem.
- **[correlations]** (*type: bloc_lecture*) List of correlations used in specific source terms (i.e. interfacial flux, interfacial friction, ...)
- **qdm_multiphase** (*type: qdm_multiphase*) Momentum conservation equation for a multi-phase problem where the unknown is the velocity
- **masse_multiphase** (*type: masse_multiphase*) Mass consevation equation for a multi-phase problem where the unknown is the alpha (void fraction)
- **energie_multiphase** (*type: energie_multiphase*) Internal energy conservation equation for a multi-phase problem where the unknown is the temperature
- **[echelle_temporelle_turbulente]** (*type: echelle_temporelle_turbulente*) Turbulent Dissipation time scale equation for a turbulent mono/multi-phase problem (available in TrioCFD)
- **[energie_cinetique_turbulente]** (*type: energie_cinetique_turbulente*) Turbulent kinetic Energy conservation equation for a turbulent mono/multi-phase problem (available in TrioCFD)

- **[energie_cinetique_turbulente_wit]** (*type: energie_cinetique_turbulente_wit*) Bubble Induced Turbulent kinetic Energy equation for a turbulent multi-phase problem (available in TrioCFD)
- **[taux_dissipation_turbulent]** (*type: taux_dissipation_turbulent*) Turbulent Dissipation frequency equation for a turbulent mono/multi-phase problem (available in TrioCFD)
- **[milieu]** (*type: milieu_base*) The medium associated with the problem.
- **[constituant]** (*type: constituant*) Constituent.
- **[postraitement | post_processing]** (*type: corps_postraitement*) One post-processing (without name).
- **[postraitements | post_processings]** (*type: postraitements*) List of Postraitement objects (with name).
- **[liste_de_postraitements]** (*type: liste_post_ok*) This
- **[liste_postraitements]** (*type: liste_post*) This block defines the output files to be written during the computation. The output format is lata in order to use OpenDX to draw the results. This block can be divided in one or several sub-blocks that can be written at different frequencies and in different directories. Attention. The directory lata used in this example should be created before running the computation or the lata files will be lost.
- **[sauvegarde]** (*type: format_file*) Keyword used when calculation results are to be backed up. When a coupling is performed, the backup-recovery file name must be well specified for each problem. In this case, you must save to different files and correctly specify these files when resuming the calculation.
- **[sauvegarde_simple]** (*type: format_file*) The same keyword than Sauvegarde except, the last time step only is saved.
- **[reprise]** (*type: format_file*) Keyword to resume a calculation based on the name_file file (see the class format_file). If format_reprise is xyz, the name_file file should be the .xyz file created by the previous calculation. With this file, it is possible to resume a parallel calculation on P processors, whereas the previous calculation has been run on N (N<>P) processors. Should the calculation be resumed, values for the tinit (see schema_temps_base) time fields are taken from the name_file file. If there is no backup corresponding to this time in the name_file, TRUST exits in error.
- **[resume_last_time]** (*type: format_file*) Keyword to resume a calculation based on the name_file file, resume the calculation at the last time found in the file (tinit is set to last time of saved files).

---

pb_multiphase_hem

Synonyms: pb_hem

Inherits from: *pb_gen_base*

A problem that allows the resolution of 2-phases mechanicaly and thermally coupled with 3 equations

Parameters are:

- **[milieu_composite]** (*type: bloc_lecture*) The composite medium associated with the problem.
- **[milieu_musig]** (*type: bloc_lecture*) The composite medium associated with the problem.
- **[correlations]** (*type: bloc_lecture*) List of correlations used in specific source terms (i.e. interfacial flux, interfacial friction, ...)
- **qdm_multiphase** (*type: qdm_multiphase*) Momentum conservation equation for a multi-phase problem where the unknown is the velocity
- **masse_multiphase** (*type: masse_multiphase*) Mass consevation equation for a multi-phase problem where the unknown is the alpha (void fraction)

- **energie_multiphase** (*type: energie_multiphase*) Internal energy conservation equation for a multi-phase problem where the unknown is the temperature
- **[echelle_temporelle_turbulente]** (*type: echelle_temporelle_turbulente*) Turbulent Dissipation time scale equation for a turbulent mono/multi-phase problem (available in TrioCFD)
- **[energie_cinetique_turbulente]** (*type: energie_cinetique_turbulente*) Turbulent kinetic Energy conservation equation for a turbulent mono/multi-phase problem (available in TrioCFD)
- **[energie_cinetique_turbulente_wit]** (*type: energie_cinetique_turbulente_wit*) Bubble Induced Turbulent kinetic Energy equation for a turbulent multi-phase problem (available in TrioCFD)
- **[taux_dissipation_turbulent]** (*type: taux_dissipation_turbulent*) Turbulent Dissipation frequency equation for a turbulent mono/multi-phase problem (available in TrioCFD)
- **[milieu]** (*type: milieu_base*) The medium associated with the problem.
- **[constituant]** (*type: constituant*) Constituent.
- **[postraitement | post_processing]** (*type: corps_postraitement*) One post-processing (without name).
- **[postraitements | post_processings]** (*type: postraitements*) List of Postraitement objects (with name).
- **[liste_de_postraitements]** (*type: liste_post_ok*) This
- **[liste_postraitements]** (*type: liste_post*) This block defines the output files to be written during the computation. The output format is lata in order to use OpenDX to draw the results. This block can be divided in one or several sub-blocks that can be written at different frequencies and in different directories. Attention. The directory lata used in this example should be created before running the computation or the lata files will be lost.
- **[sauvegarde]** (*type: format_file*) Keyword used when calculation results are to be backed up. When a coupling is performed, the backup-recovery file name must be well specified for each problem. In this case, you must save to different files and correctly specify these files when resuming the calculation.
- **[sauvegarde_simple]** (*type: format_file*) The same keyword than Sauvegarde except, the last time step only is saved.
- **[reprise]** (*type: format_file*) Keyword to resume a calculation based on the name_file file (see the class format_file). If format_reprise is xyz, the name_file file should be the .xyz file created by the previous calculation. With this file, it is possible to resume a parallel calculation on P processors, whereas the previous calculation has been run on N (N<>P) processors. Should the calculation be resumed, values for the tinit (see schema_temps_base) time fields are taken from the name_file file. If there is no backup corresponding to this time in the name_file, TRUST exits in error.
- **[resume_last_time]** (*type: format_file*) Keyword to resume a calculation based on the name_file file, resume the calculation at the last time found in the file (tinit is set to last time of saved files).

---

pb_post

Inherits from: *pb_gen_base*

not_set

Parameters are:

- **[milieu]** (*type: milieu_base*) The medium associated with the problem.
- **[constituant]** (*type: constituant*) Constituent.
- **[postraitement | post_processing]** (*type: corps_postraitement*) One post-processing (without name).
- **[postraitements | post_processings]** (*type: postraitements*) List of Postraitement objects (with name).

- **[liste_de_postraitements]** (*type: liste_post_ok*) This
- **[liste_postraitements]** (*type: liste_post*) This block defines the output files to be written during the computation. The output format is lata in order to use OpenDX to draw the results. This block can be divided in one or several sub-blocks that can be written at different frequencies and in different directories. Attention. The directory lata used in this example should be created before running the computation or the lata files will be lost.
- **[sauvegarde]** (*type: format_file*) Keyword used when calculation results are to be backed up. When a coupling is performed, the backup-recovery file name must be well specified for each problem. In this case, you must save to different files and correctly specify these files when resuming the calculation.
- **[sauvegarde_simple]** (*type: format_file*) The same keyword than Sauvegarde except, the last time step only is saved.
- **[reprise]** (*type: format_file*) Keyword to resume a calculation based on the name_file file (see the class format_file). If format_reprise is xyz, the name_file file should be the .xyz file created by the previous calculation. With this file, it is possible to resume a parallel calculation on P processors, whereas the previous calculation has been run on N (N<>P) processors. Should the calculation be resumed, values for the tinit (see schema_temps_base) time fields are taken from the name_file file. If there is no backup corresponding to this time in the name_file, TRUST exits in error.
- **[resume_last_time]** (*type: format_file*) Keyword to resume a calculation based on the name_file file, resume the calculation at the last time found in the file (tinit is set to last time of saved files).

---

pb_thermohydraulique

Inherits from: *pb_gen_base*

Resolution of thermohydraulic problem.

Parameters are:

- **[fluide_incompressible]** (*type: fluide_incompressible*) The fluid medium associated with the problem (only one possibility).
- **[fluide_ostwald]** (*type: fluide_ostwald*) The fluid medium associated with the problem (only one possibility).
- **[fluide_sodium_liquide]** (*type: fluide_sodium_liquide*) The fluid medium associated with the problem (only one possibility).
- **[fluide_sodium_gaz]** (*type: fluide_sodium_gaz*) The fluid medium associated with the problem (only one possibility).
- **[navier_stokes_standard]** (*type: navier_stokes_standard*) Navier-Stokes equations.
- **[convection_diffusion_temperature]** (*type: convection_diffusion_temperature*) Energy equation (temperature diffusion convection).
- **[milieu]** (*type: milieu_base*) The medium associated with the problem.
- **[constituant]** (*type: constituant*) Constituent.
- **[postraitement | post_processing]** (*type: corps_postraitement*) One post-processing (without name).
- **[postraitements | post_processings]** (*type: postraitements*) List of Postraitement objects (with name).
- **[liste_de_postraitements]** (*type: liste_post_ok*) This
- **[liste_postraitements]** (*type: liste_post*) This block defines the output files to be written during the computation. The output format is lata in order to use OpenDX to draw the results. This block can be divided in one or several

sub-blocks that can be written at different frequencies and in different directories. Attention. The directory lata used in this example should be created before running the computation or the lata files will be lost.

- **[sauvegarde]** (*type: format_file*) Keyword used when calculation results are to be backed up. When a coupling is performed, the backup-recovery file name must be well specified for each problem. In this case, you must save to different files and correctly specify these files when resuming the calculation.
- **[sauvegarde_simple]** (*type: format_file*) The same keyword than Sauvegarde except, the last time step only is saved.
- **[reprise]** (*type: format_file*) Keyword to resume a calculation based on the name_file file (see the class format_file). If format_reprise is xyz, the name_file file should be the .xyz file created by the previous calculation. With this file, it is possible to resume a parallel calculation on P processors, whereas the previous calculation has been run on N (N<>P) processors. Should the calculation be resumed, values for the tinit (see schema_temps_base) time fields are taken from the name_file file. If there is no backup corresponding to this time in the name_file, TRUST exits in error.
- **[resume_last_time]** (*type: format_file*) Keyword to resume a calculation based on the name_file file, resume the calculation at the last time found in the file (tinit is set to last time of saved files).

---

pb_thermohydraulique_cloned_concentration

Inherits from: *pb_gen_base*

Resolution of Navier-Stokes/energy/multiple constituent transport equations.

Parameters are:

- **fluide_incompressible** (*type: fluide_incompressible*) The fluid medium associated with the problem.
- **[constituant]** (*type: constituant*) Constituents.
- **[navier_stokes_standard]** (*type: navier_stokes_standard*) Navier-Stokes equations.
- **[convection_diffusion_concentration]** (*type: convection_diffusion_concentration*) Constituent transport equations (concentration diffusion convection).
- **[convection_diffusion_temperature]** (*type: convection_diffusion_temperature*) Energy equation (temperature diffusion convection).
- **[milieu]** (*type: milieu_base*) The medium associated with the problem.
- **[postraitement | post_processing]** (*type: corps_postraitement*) One post-processing (without name).
- **[postraitements | post_processings]** (*type: postraitements*) List of Postraitement objects (with name).
- **[liste_de_postraitements]** (*type: liste_post_ok*) This
- **[liste_postraitements]** (*type: liste_post*) This block defines the output files to be written during the computation. The output format is lata in order to use OpenDX to draw the results. This block can be divided in one or several sub-blocks that can be written at different frequencies and in different directories. Attention. The directory lata used in this example should be created before running the computation or the lata files will be lost.
- **[sauvegarde]** (*type: format_file*) Keyword used when calculation results are to be backed up. When a coupling is performed, the backup-recovery file name must be well specified for each problem. In this case, you must save to different files and correctly specify these files when resuming the calculation.
- **[sauvegarde_simple]** (*type: format_file*) The same keyword than Sauvegarde except, the last time step only is saved.

- **[reprise]** (*type: format_file*) Keyword to resume a calculation based on the name_file file (see the class format_file). If format_reprise is xyz, the name_file file should be the .xyz file created by the previous calculation. With this file, it is possible to resume a parallel calculation on P processors, whereas the previous calculation has been run on N (N<>P) processors. Should the calculation be resumed, values for the tinit (see schema_temps_base) time fields are taken from the name_file file. If there is no backup corresponding to this time in the name_file, TRUST exits in error.
- **[resume_last_time]** (*type: format_file*) Keyword to resume a calculation based on the name_file file, resume the calculation at the last time found in the file (tinit is set to last time of saved files).

---

pb_thermohydraulique_cloned_concentration_turbulent

Inherits from: *pb_gen_base*

Resolution of Navier-Stokes/energy/multiple constituent transport equations, with turbulence modelling.

Parameters are:

- **fluide_incompressible** (*type: fluide_incompressible*) The fluid medium associated with the problem.
- **[constituant]** (*type: constituant*) Constituents.
- **[navier_stokes_turbulent]** (*type: navier_stokes_turbulent*) Navier-Stokes equations as well as the associated turbulence model equations.
- **[convection_diffusion_concentration_turbulent]** (*type: convection_diffusion_concentration_turbulent*) Constituent transport equations (concentration diffusion convection) as well as the associated turbulence model equations.
- **[convection_diffusion_temperature_turbulent]** (*type: convection_diffusion_temperature_turbulent*) Energy equation (temperature diffusion convection) as well as the associated turbulence model equations.
- **[milieu]** (*type: milieu_base*) The medium associated with the problem.
- **[postraitement | post_processing]** (*type: corps_postraitement*) One post-processing (without name).
- **[postraitements | post_processings]** (*type: postraitements*) List of Postraitement objects (with name).
- **[liste_de_postraitements]** (*type: liste_post_ok*) This
- **[liste_postraitements]** (*type: liste_post*) This block defines the output files to be written during the computation. The output format is lata in order to use OpenDX to draw the results. This block can be divided in one or several sub-blocks that can be written at different frequencies and in different directories. Attention. The directory lata used in this example should be created before running the computation or the lata files will be lost.
- **[sauvegarde]** (*type: format_file*) Keyword used when calculation results are to be backed up. When a coupling is performed, the backup-recovery file name must be well specified for each problem. In this case, you must save to different files and correctly specify these files when resuming the calculation.
- **[sauvegarde_simple]** (*type: format_file*) The same keyword than Sauvegarde except, the last time step only is saved.
- **[reprise]** (*type: format_file*) Keyword to resume a calculation based on the name_file file (see the class format_file). If format_reprise is xyz, the name_file file should be the .xyz file created by the previous calculation. With this file, it is possible to resume a parallel calculation on P processors, whereas the previous calculation has been run on N (N<>P) processors. Should the calculation be resumed, values for the tinit (see schema_temps_base) time fields are taken from the name_file file. If there is no backup corresponding to this time in the name_file, TRUST exits in error.

- **[resume_last_time]** (*type: format_file*) Keyword to resume a calculation based on the name_file file, resume the calculation at the last time found in the file (tinit is set to last time of saved files).

---

pb_thermohydraulique_concentration

Inherits from: *pb_gen_base*

Resolution of Navier-Stokes/energy/multiple constituent transport equations.

Parameters are:

- **fluide_incompressible** (*type: fluide_incompressible*) The fluid medium associated with the problem.
- **[constituant]** (*type: constituant*) Constituents.
- **[navier_stokes_standard]** (*type: navier_stokes_standard*) Navier-Stokes equations.
- **[convection_diffusion_concentration]** (*type: convection_diffusion_concentration*) Constituent transport equations (concentration diffusion convection).
- **[convection_diffusion_temperature]** (*type: convection_diffusion_temperature*) Energy equation (temperature diffusion convection).
- **[milieu]** (*type: milieu_base*) The medium associated with the problem.
- **[postraitement | post_processing]** (*type: corps_postraitement*) One post-processing (without name).
- **[postraitements | post_processings]** (*type: postraitements*) List of Postraitement objects (with name).
- **[liste_de_postraitements]** (*type: liste_post_ok*) This
- **[liste_postraitements]** (*type: liste_post*) This block defines the output files to be written during the computation. The output format is lata in order to use OpenDX to draw the results. This block can be divided in one or several sub-blocks that can be written at different frequencies and in different directories. Attention. The directory lata used in this example should be created before running the computation or the lata files will be lost.
- **[sauvegarde]** (*type: format_file*) Keyword used when calculation results are to be backed up. When a coupling is performed, the backup-recovery file name must be well specified for each problem. In this case, you must save to different files and correctly specify these files when resuming the calculation.
- **[sauvegarde_simple]** (*type: format_file*) The same keyword than Sauvegarde except, the last time step only is saved.
- **[reprise]** (*type: format_file*) Keyword to resume a calculation based on the name_file file (see the class format_file). If format_reprise is xyz, the name_file file should be the .xyz file created by the previous calculation. With this file, it is possible to resume a parallel calculation on P processors, whereas the previous calculation has been run on N (N<>P) processors. Should the calculation be resumed, values for the tinit (see schema_temps_base) time fields are taken from the name_file file. If there is no backup corresponding to this time in the name_file, TRUST exits in error.
- **[resume_last_time]** (*type: format_file*) Keyword to resume a calculation based on the name_file file, resume the calculation at the last time found in the file (tinit is set to last time of saved files).

---

pb_thermohydraulique_concentration_scalaires_passifs

Inherits from: *pb_gen_base*

Resolution of Navier-Stokes/energy/multiple constituent transport equations, with the additional passive scalar equations.

Parameters are:

- **fluide_incompressible** (*type: fluide_incompressible*) The fluid medium associated with the problem.
- **[constituant]** (*type: constituant*) Constituents.
- **[navier_stokes_standard]** (*type: navier_stokes_standard*) Navier-Stokes equations.
- **[convection_diffusion_concentration]** (*type: convection_diffusion_concentration*) Constituent transport equations (concentration diffusion convection).
- **[convection_diffusion_temperature]** (*type: convection_diffusion_temperature*) Energy equations (temperature diffusion convection).
- **equations_scalaires_passifs** (*type: listeqn*) Passive scalar equations. The unknowns of the passive scalar equation number N are named temperatureN or concentrationN or fraction_massiqueN. This keyword is used to define initial conditions and the post processing fields. This kind of problem is very useful to test in only one data file (and then only one calculation) different schemes or different boundary conditions for the scalar transport equation.
- **[milieu]** (*type: milieu_base*) The medium associated with the problem.
- **[postraitement | post_processing]** (*type: corps_postraitement*) One post-processing (without name).
- **[postraitements | post_processings]** (*type: postraitements*) List of Postraitement objects (with name).
- **[liste_de_postraitements]** (*type: liste_post_ok*) This
- **[liste_postraitements]** (*type: liste_post*) This block defines the output files to be written during the computation. The output format is lata in order to use OpenDX to draw the results. This block can be divided in one or several sub-blocks that can be written at different frequencies and in different directories. Attention. The directory lata used in this example should be created before running the computation or the lata files will be lost.
- **[sauvegarde]** (*type: format_file*) Keyword used when calculation results are to be backed up. When a coupling is performed, the backup-recovery file name must be well specified for each problem. In this case, you must save to different files and correctly specify these files when resuming the calculation.
- **[sauvegarde_simple]** (*type: format_file*) The same keyword than Sauvegarde except, the last time step only is saved.
- **[reprise]** (*type: format_file*) Keyword to resume a calculation based on the name_file file (see the class format_file). If format_reprise is xyz, the name_file file should be the .xyz file created by the previous calculation. With this file, it is possible to resume a parallel calculation on P processors, whereas the previous calculation has been run on N (N<>P) processors. Should the calculation be resumed, values for the tinit (see schema_temps_base) time fields are taken from the name_file file. If there is no backup corresponding to this time in the name_file, TRUST exits in error.
- **[resume_last_time]** (*type: format_file*) Keyword to resume a calculation based on the name_file file, resume the calculation at the last time found in the file (tinit is set to last time of saved files).

pb_thermohydraulique_concentration_turbulent

Inherits from: *pb_gen_base*

Resolution of Navier-Stokes/energy/multiple constituent transport equations, with turbulence modelling.

Parameters are:

- **fluide_incompressible** (*type: fluide_incompressible*) The fluid medium associated with the problem.
- **[constituant]** (*type: constituant*) Constituents.
- **[navier_stokes_turbulent]** (*type: navier_stokes_turbulent*) Navier-Stokes equations as well as the associated turbulence model equations.
- **[convection_diffusion_concentration_turbulent]** (*type: convection_diffusion_concentration_turbulent*) Constituent transport equations (concentration diffusion convection) as well as the associated turbulence model equations.
- **[convection_diffusion_temperature_turbulent]** (*type: convection_diffusion_temperature_turbulent*) Energy equation (temperature diffusion convection) as well as the associated turbulence model equations.
- **[milieu]** (*type: milieu_base*) The medium associated with the problem.
- **[postraitement | post_processing]** (*type: corps_postraitement*) One post-processing (without name).
- **[postraitements | post_processings]** (*type: postraitements*) List of Postraitement objects (with name).
- **[liste_de_postraitements]** (*type: liste_post_ok*) This
- **[liste_postraitements]** (*type: liste_post*) This block defines the output files to be written during the computation. The output format is lata in order to use OpenDX to draw the results. This block can be divided in one or several sub-blocks that can be written at different frequencies and in different directories. Attention. The directory lata used in this example should be created before running the computation or the lata files will be lost.
- **[sauvegarde]** (*type: format_file*) Keyword used when calculation results are to be backed up. When a coupling is performed, the backup-recovery file name must be well specified for each problem. In this case, you must save to different files and correctly specify these files when resuming the calculation.
- **[sauvegarde_simple]** (*type: format_file*) The same keyword than Sauvegarde except, the last time step only is saved.
- **[reprise]** (*type: format_file*) Keyword to resume a calculation based on the name_file file (see the class format_file). If format_reprise is xyz, the name_file file should be the .xyz file created by the previous calculation. With this file, it is possible to resume a parallel calculation on P processors, whereas the previous calculation has been run on N (N<>P) processors. Should the calculation be resumed, values for the tinit (see schema_temps_base) time fields are taken from the name_file file. If there is no backup corresponding to this time in the name_file, TRUST exits in error.
- **[resume_last_time]** (*type: format_file*) Keyword to resume a calculation based on the name_file file, resume the calculation at the last time found in the file (tinit is set to last time of saved files).

pb_thermohydraulique_concentration_turbulent_scalaires_passifs

Inherits from: *pb_gen_base*

Resolution of Navier-Stokes/energy/multiple constituent transport equations, with turbulence modelling and with the additional passive scalar equations.

Parameters are:

- **fluide_incompressible** (*type: fluide_incompressible*) The fluid medium associated with the problem.
- **[constituant]** (*type: constituant*) Constituents.
- **[navier_stokes_turbulent]** (*type: navier_stokes_turbulent*) Navier-Stokes equations as well as the associated turbulence model equations.
- **[convection_diffusion_concentration_turbulent]** (*type: convection_diffusion_concentration_turbulent*) Constituent transport equations (concentration diffusion convection) as well as the associated turbulence model equations.
- **[convection_diffusion_temperature_turbulent]** (*type: convection_diffusion_temperature_turbulent*) Energy equations (temperature diffusion convection) as well as the associated turbulence model equations.
- **equations_scalaires_passifs** (*type: listeqn*) Passive scalar equations. The unknowns of the passive scalar equation number N are named temperatureN or concentrationN or fraction_massiqueN. This keyword is used to define initial conditions and the post processing fields. This kind of problem is very useful to test in only one data file (and then only one calculation) different schemes or different boundary conditions for the scalar transport equation.
- **[milieu]** (*type: milieu_base*) The medium associated with the problem.
- **[postraitement | post_processing]** (*type: corps_postraitement*) One post-processing (without name).
- **[postraitements | post_processings]** (*type: postraitements*) List of Postraitement objects (with name).
- **[liste_de_postraitements]** (*type: liste_post_ok*) This
- **[liste_postraitements]** (*type: liste_post*) This block defines the output files to be written during the computation. The output format is lata in order to use OpenDX to draw the results. This block can be divided in one or several sub-blocks that can be written at different frequencies and in different directories. Attention. The directory lata used in this example should be created before running the computation or the lata files will be lost.
- **[sauvegarde]** (*type: format_file*) Keyword used when calculation results are to be backed up. When a coupling is performed, the backup-recovery file name must be well specified for each problem. In this case, you must save to different files and correctly specify these files when resuming the calculation.
- **[sauvegarde_simple]** (*type: format_file*) The same keyword than Sauvegarde except, the last time step only is saved.
- **[reprise]** (*type: format_file*) Keyword to resume a calculation based on the name_file file (see the class format_file). If format_reprise is xyz, the name_file file should be the .xyz file created by the previous calculation. With this file, it is possible to resume a parallel calculation on P processors, whereas the previous calculation has been run on N (N<>P) processors. Should the calculation be resumed, values for the tinit (see schema_temps_base) time fields are taken from the name_file file. If there is no backup corresponding to this time in the name_file, TRUST exits in error.
- **[resume_last_time]** (*type: format_file*) Keyword to resume a calculation based on the name_file file, resume the calculation at the last time found in the file (tinit is set to last time of saved files).

pb_thermohydraulique_especes_qc

Inherits from: *pb_gen_base*

Resolution of thermo-hydraulic problem for a multi-species quasi-compressible fluid.

Parameters are:

- **fluide_quasi_compressible** (*type: fluide_quasi_compressible*) The fluid medium associated with the problem.
- **navier_stokes_qc** (*type: navier_stokes_qc*) Navier-Stokes equation for a quasi-compressible fluid.
- **convection_diffusion_chaleur_qc** (*type: convection_diffusion_chaleur_qc*) Temperature equation for a quasi-compressible fluid.
- **equations_scalaires_passifs** (*type: listeqn*) Passive scalar equations. The unknowns of the passive scalar equation number N are named temperatureN or concentrationN or fraction_massiqueN. This keyword is used to define initial conditions and the post processing fields. This kind of problem is very useful to test in only one data file (and then only one calculation) different schemes or different boundary conditions for the scalar transport equation.
- **[milieu]** (*type: milieu_base*) The medium associated with the problem.
- **[constituant]** (*type: constituant*) Constituent.
- **[postraitement | post_processing]** (*type: corps_postraitement*) One post-processing (without name).
- **[postraitements | post_processings]** (*type: postraitements*) List of Postraitement objects (with name).
- **[liste_de_postraitements]** (*type: liste_post_ok*) This
- **[liste_postraitements]** (*type: liste_post*) This block defines the output files to be written during the computation. The output format is lata in order to use OpenDX to draw the results. This block can be divided in one or several sub-blocks that can be written at different frequencies and in different directories. Attention. The directory lata used in this example should be created before running the computation or the lata files will be lost.
- **[sauvegarde]** (*type: format_file*) Keyword used when calculation results are to be backed up. When a coupling is performed, the backup-recovery file name must be well specified for each problem. In this case, you must save to different files and correctly specify these files when resuming the calculation.
- **[sauvegarde_simple]** (*type: format_file*) The same keyword than Sauvegarde except, the last time step only is saved.
- **[reprise]** (*type: format_file*) Keyword to resume a calculation based on the name_file file (see the class format_file). If format_reprise is xyz, the name_file file should be the .xyz file created by the previous calculation. With this file, it is possible to resume a parallel calculation on P processors, whereas the previous calculation has been run on N (N<>P) processors. Should the calculation be resumed, values for the tinit (see schema_temps_base) time fields are taken from the name_file file. If there is no backup corresponding to this time in the name_file, TRUST exits in error.
- **[resume_last_time]** (*type: format_file*) Keyword to resume a calculation based on the name_file file, resume the calculation at the last time found in the file (tinit is set to last time of saved files).

pb_thermohydraulique_especes_turbulent_qc

Inherits from: *pb_gen_base*

Resolution of turbulent thermohydraulic problem under low Mach number with passive scalar equations.

Parameters are:

- **fluide_quasi_compressible** (*type: fluide_quasi_compressible*) The fluid medium associated with the problem.
- **navier_stokes_turbulent_qc** (*type: navier_stokes_turbulent_qc*) Navier-Stokes equations under low Mach number as well as the associated turbulence model equations.
- **convection_diffusion_chaleur_turbulent_qc** (*type: convection_diffusion_chaleur_turbulent_qc*) Energy equation under low Mach number as well as the associated turbulence model equations.
- **equations_scalaires_passifs** (*type: listeqn*) Passive scalar equations. The unknowns of the passive scalar equation number N are named temperatureN or concentrationN or fraction_massiqueN. This keyword is used to define initial conditions and the post processing fields. This kind of problem is very useful to test in only one data file (and then only one calculation) different schemes or different boundary conditions for the scalar transport equation.
- **[milieu]** (*type: milieu_base*) The medium associated with the problem.
- **[constituant]** (*type: constituant*) Constituent.
- **[postraitement | post_processing]** (*type: corps_postraitement*) One post-processing (without name).
- **[postraitements | post_processings]** (*type: postraitements*) List of Postraitement objects (with name).
- **[liste_de_postraitements]** (*type: liste_post_ok*) This
- **[liste_postraitements]** (*type: liste_post*) This block defines the output files to be written during the computation. The output format is lata in order to use OpenDX to draw the results. This block can be divided in one or several sub-blocks that can be written at different frequencies and in different directories. Attention. The directory lata used in this example should be created before running the computation or the lata files will be lost.
- **[sauvegarde]** (*type: format_file*) Keyword used when calculation results are to be backed up. When a coupling is performed, the backup-recovery file name must be well specified for each problem. In this case, you must save to different files and correctly specify these files when resuming the calculation.
- **[sauvegarde_simple]** (*type: format_file*) The same keyword than Sauvegarde except, the last time step only is saved.
- **[reprise]** (*type: format_file*) Keyword to resume a calculation based on the name_file file (see the class format_file). If format_reprise is xyz, the name_file file should be the .xyz file created by the previous calculation. With this file, it is possible to resume a parallel calculation on P processors, whereas the previous calculation has been run on N (N<>P) processors. Should the calculation be resumed, values for the tinit (see schema_temps_base) time fields are taken from the name_file file. If there is no backup corresponding to this time in the name_file, TRUST exits in error.
- **[resume_last_time]** (*type: format_file*) Keyword to resume a calculation based on the name_file file, resume the calculation at the last time found in the file (tinit is set to last time of saved files).

pb_thermohydraulique_especes_wc

Inherits from: *pb_gen_base*

Resolution of thermo-hydraulic problem for a multi-species weakly-compressible fluid.

Parameters are:

- **fluide_weakly_compressible** (*type: fluide_weakly_compressible*) The fluid medium associated with the problem.
- **navier_stokes_wc** (*type: navier_stokes_wc*) Navier-Stokes equation for a weakly-compressible fluid.
- **convection_diffusion_chaleur_wc** (*type: convection_diffusion_chaleur_wc*) Temperature equation for a weakly-compressible fluid.
- **equations_scalaires_passifs** (*type: listeqn*) Passive scalar equations. The unknowns of the passive scalar equation number N are named temperatureN or concentrationN or fraction_massiqueN. This keyword is used to define initial conditions and the post processing fields. This kind of problem is very useful to test in only one data file (and then only one calculation) different schemes or different boundary conditions for the scalar transport equation.
- **[milieu]** (*type: milieu_base*) The medium associated with the problem.
- **[constituant]** (*type: constituant*) Constituent.
- **[postraitement | post_processing]** (*type: corps_postraitement*) One post-processing (without name).
- **[postraitements | post_processings]** (*type: postraitements*) List of Postraitement objects (with name).
- **[liste_de_postraitements]** (*type: liste_post_ok*) This
- **[liste_postraitements]** (*type: liste_post*) This block defines the output files to be written during the computation. The output format is lata in order to use OpenDX to draw the results. This block can be divided in one or several sub-blocks that can be written at different frequencies and in different directories. Attention. The directory lata used in this example should be created before running the computation or the lata files will be lost.
- **[sauvegarde]** (*type: format_file*) Keyword used when calculation results are to be backed up. When a coupling is performed, the backup-recovery file name must be well specified for each problem. In this case, you must save to different files and correctly specify these files when resuming the calculation.
- **[sauvegarde_simple]** (*type: format_file*) The same keyword than Sauvegarde except, the last time step only is saved.
- **[reprise]** (*type: format_file*) Keyword to resume a calculation based on the name_file file (see the class format_file). If format_reprise is xyz, the name_file file should be the .xyz file created by the previous calculation. With this file, it is possible to resume a parallel calculation on P processors, whereas the previous calculation has been run on N (N<>P) processors. Should the calculation be resumed, values for the tinit (see schema_temps_base) time fields are taken from the name_file file. If there is no backup corresponding to this time in the name_file, TRUST exits in error.
- **[resume_last_time]** (*type: format_file*) Keyword to resume a calculation based on the name_file file, resume the calculation at the last time found in the file (tinit is set to last time of saved files).

pb_thermohydraulique_list_concentration

Inherits from: *pb_gen_base*

Resolution of Navier-Stokes/energy/multiple constituent transport equations.

Parameters are:

- **fluide_incompressible** (*type: fluide_incompressible*) The fluid medium associated with the problem.
- **[constituant]** (*type: constituant*) Constituents.
- **[navier_stokes_standard]** (*type: navier_stokes_standard*) Navier-Stokes equations.
- **[convection_diffusion_temperature]** (*type: convection_diffusion_temperature*) Energy equation (temperature diffusion convection).
- **list_equations** (*type: listeqn*) convection_diffusion_concentration equations. The unknown of the concentration equation number N is named concentrationN. This keyword is used to define initial conditions and the post processing fields. This kind of problem is very useful to test in only one data file (and then only one calculation) different schemes or different boundary conditions for the scalar transport equation.
- **[milieu]** (*type: milieu_base*) The medium associated with the problem.
- **[postraitement | post_processing]** (*type: corps_postraitement*) One post-processing (without name).
- **[postraitements | post_processings]** (*type: postraitements*) List of Postraitement objects (with name).
- **[liste_de_postraitements]** (*type: liste_post_ok*) This
- **[liste_postraitements]** (*type: liste_post*) This block defines the output files to be written during the computation. The output format is lata in order to use OpenDX to draw the results. This block can be divided in one or several sub-blocks that can be written at different frequencies and in different directories. Attention. The directory lata used in this example should be created before running the computation or the lata files will be lost.
- **[sauvegarde]** (*type: format_file*) Keyword used when calculation results are to be backed up. When a coupling is performed, the backup-recovery file name must be well specified for each problem. In this case, you must save to different files and correctly specify these files when resuming the calculation.
- **[sauvegarde_simple]** (*type: format_file*) The same keyword than Sauvegarde except, the last time step only is saved.
- **[reprise]** (*type: format_file*) Keyword to resume a calculation based on the name_file file (see the class format_file). If format_reprise is xyz, the name_file file should be the .xyz file created by the previous calculation. With this file, it is possible to resume a parallel calculation on P processors, whereas the previous calculation has been run on N (N<>P) processors. Should the calculation be resumed, values for the tinit (see schema_temps_base) time fields are taken from the name_file file. If there is no backup corresponding to this time in the name_file, TRUST exits in error.
- **[resume_last_time]** (*type: format_file*) Keyword to resume a calculation based on the name_file file, resume the calculation at the last time found in the file (tinit is set to last time of saved files).

pb_thermohydraulique_list_concentration_turbulent

Inherits from: *pb_gen_base*

Resolution of Navier-Stokes/energy/multiple constituent transport equations, with turbulence modelling.

Parameters are:

- **fluide_incompressible** (*type: fluide_incompressible*) The fluid medium associated with the problem.
- **[constituant]** (*type: constituant*) Constituents.
- **[navier_stokes_turbulent]** (*type: navier_stokes_turbulent*) Navier-Stokes equations as well as the associated turbulence model equations.
- **[convection_diffusion_temperature_turbulent]** (*type: convection_diffusion_temperature_turbulent*) Energy equation (temperature diffusion convection) as well as the associated turbulence model equations.
- **list_equations** (*type: listeqn*) convection_diffusion_concentration equations. The unknown of the concentration equation number N is named concentrationN. This keyword is used to define initial conditions and the post processing fields. This kind of problem is very useful to test in only one data file (and then only one calculation) different schemes or different boundary conditions for the scalar transport equation.
- **[milieu]** (*type: milieu_base*) The medium associated with the problem.
- **[postraitement | post_processing]** (*type: corps_postraitement*) One post-processing (without name).
- **[postraitements | post_processings]** (*type: postraitements*) List of Postraitement objects (with name).
- **[liste_de_postraitements]** (*type: liste_post_ok*) This
- **[liste_postraitements]** (*type: liste_post*) This block defines the output files to be written during the computation. The output format is lata in order to use OpenDX to draw the results. This block can be divided in one or several sub-blocks that can be written at different frequencies and in different directories. Attention. The directory lata used in this example should be created before running the computation or the lata files will be lost.
- **[sauvegarde]** (*type: format_file*) Keyword used when calculation results are to be backed up. When a coupling is performed, the backup-recovery file name must be well specified for each problem. In this case, you must save to different files and correctly specify these files when resuming the calculation.
- **[sauvegarde_simple]** (*type: format_file*) The same keyword than Sauvegarde except, the last time step only is saved.
- **[reprise]** (*type: format_file*) Keyword to resume a calculation based on the name_file file (see the class format_file). If format_reprise is xyz, the name_file file should be the .xyz file created by the previous calculation. With this file, it is possible to resume a parallel calculation on P processors, whereas the previous calculation has been run on N (N<>P) processors. Should the calculation be resumed, values for the tinit (see schema_temps_base) time fields are taken from the name_file file. If there is no backup corresponding to this time in the name_file, TRUST exits in error.
- **[resume_last_time]** (*type: format_file*) Keyword to resume a calculation based on the name_file file, resume the calculation at the last time found in the file (tinit is set to last time of saved files).

---

pb_thermohydraulique_qc

Inherits from: *pb_gen_base*

Resolution of thermo-hydraulic problem for a quasi-compressible fluid.

Keywords for the unknowns other than pressure, velocity, temperature are :

masse_volumique : density

enthalpie : enthalpy

pression : reduced pressure

pression_tot : total pressure.

Parameters are:

- **fluide_quasi_compressible** (*type: fluide_quasi_compressible*) The fluid medium associated with the problem.
- **navier_stokes_qc** (*type: navier_stokes_qc*) Navier-Stokes equation for a quasi-compressible fluid.
- **convection_diffusion_chaleur_qc** (*type: convection_diffusion_chaleur_qc*) Temperature equation for a quasi-compressible fluid.
- **[milieu]** (*type: milieu_base*) The medium associated with the problem.
- **[constituant]** (*type: constituant*) Constituent.
- **[postraitement | post_processing]** (*type: corps_postraitement*) One post-processing (without name).
- **[postraitements | post_processings]** (*type: postraitements*) List of Postraitement objects (with name).
- **[liste_de_postraitements]** (*type: liste_post_ok*) This
- **[liste_postraitements]** (*type: liste_post*) This block defines the output files to be written during the computation. The output format is lata in order to use OpenDX to draw the results. This block can be divided in one or several sub-blocks that can be written at different frequencies and in different directories. Attention. The directory lata used in this example should be created before running the computation or the lata files will be lost.
- **[sauvegarde]** (*type: format_file*) Keyword used when calculation results are to be backed up. When a coupling is performed, the backup-recovery file name must be well specified for each problem. In this case, you must save to different files and correctly specify these files when resuming the calculation.
- **[sauvegarde_simple]** (*type: format_file*) The same keyword than Sauvegarde except, the last time step only is saved.
- **[reprise]** (*type: format_file*) Keyword to resume a calculation based on the name_file file (see the class format_file). If format_reprise is xyz, the name_file file should be the .xyz file created by the previous calculation. With this file, it is possible to resume a parallel calculation on P processors, whereas the previous calculation has been run on N (N<>P) processors. Should the calculation be resumed, values for the tinit (see schema_temps_base) time fields are taken from the name_file file. If there is no backup corresponding to this time in the name_file, TRUST exits in error.
- **[resume_last_time]** (*type: format_file*) Keyword to resume a calculation based on the name_file file, resume the calculation at the last time found in the file (tinit is set to last time of saved files).

pb_thermohydraulique_scalaires_passifs

Inherits from: *pb_gen_base*

Resolution of thermohydraulic problem, with the additional passive scalar equations.

Parameters are:

- **fluide_incompressible** (*type: fluide_incompressible*) The fluid medium associated with the problem.
- **[constituant]** (*type: constituant*) Constituents.
- **[navier_stokes_standard]** (*type: navier_stokes_standard*) Navier-Stokes equations.
- **[convection_diffusion_temperature]** (*type: convection_diffusion_temperature*) Energy equations (temperature diffusion convection).
- **equations_scalaires_passifs** (*type: listeqn*) Passive scalar equations. The unknowns of the passive scalar equation number N are named temperatureN or concentrationN or fraction_massiqueN. This keyword is used to define initial conditions and the post processing fields. This kind of problem is very useful to test in only one data file (and then only one calculation) different schemes or different boundary conditions for the scalar transport equation.
- **[milieu]** (*type: milieu_base*) The medium associated with the problem.
- **[postraitement | post_processing]** (*type: corps_postraitement*) One post-processing (without name).
- **[postraitements | post_processings]** (*type: postraitements*) List of Postraitement objects (with name).
- **[liste_de_postraitements]** (*type: liste_post_ok*) This
- **[liste_postraitements]** (*type: liste_post*) This block defines the output files to be written during the computation. The output format is lata in order to use OpenDX to draw the results. This block can be divided in one or several sub-blocks that can be written at different frequencies and in different directories. Attention. The directory lata used in this example should be created before running the computation or the lata files will be lost.
- **[sauvegarde]** (*type: format_file*) Keyword used when calculation results are to be backed up. When a coupling is performed, the backup-recovery file name must be well specified for each problem. In this case, you must save to different files and correctly specify these files when resuming the calculation.
- **[sauvegarde_simple]** (*type: format_file*) The same keyword than Sauvegarde except, the last time step only is saved.
- **[reprise]** (*type: format_file*) Keyword to resume a calculation based on the name_file file (see the class format_file). If format_reprise is xyz, the name_file file should be the .xyz file created by the previous calculation. With this file, it is possible to resume a parallel calculation on P processors, whereas the previous calculation has been run on N (N<>P) processors. Should the calculation be resumed, values for the tinit (see schema_temps_base) time fields are taken from the name_file file. If there is no backup corresponding to this time in the name_file, TRUST exits in error.
- **[resume_last_time]** (*type: format_file*) Keyword to resume a calculation based on the name_file file, resume the calculation at the last time found in the file (tinit is set to last time of saved files).

pb_thermohydraulique_turbulent

Inherits from: *pb_gen_base*

Resolution of thermohydraulic problem, with turbulence modelling.

Parameters are:

- **fluide_incompressible** (*type: fluide_incompressible*) The fluid medium associated with the problem.
- **navier_stokes_turbulent** (*type: navier_stokes_turbulent*) Navier-Stokes equations as well as the associated turbulence model equations.
- **convection_diffusion_temperature_turbulent** (*type: convection_diffusion_temperature_turbulent*) Energy equation (temperature diffusion convection) as well as the associated turbulence model equations.
- **[milieu]** (*type: milieu_base*) The medium associated with the problem.
- **[constituant]** (*type: constituant*) Constituent.
- **[postraitement | post_processing]** (*type: corps_postraitement*) One post-processing (without name).
- **[postraitements | post_processings]** (*type: postraitements*) List of Postraitement objects (with name).
- **[liste_de_postraitements]** (*type: liste_post_ok*) This
- **[liste_postraitements]** (*type: liste_post*) This block defines the output files to be written during the computation. The output format is lata in order to use OpenDX to draw the results. This block can be divided in one or several sub-blocks that can be written at different frequencies and in different directories. Attention. The directory lata used in this example should be created before running the computation or the lata files will be lost.
- **[sauvegarde]** (*type: format_file*) Keyword used when calculation results are to be backed up. When a coupling is performed, the backup-recovery file name must be well specified for each problem. In this case, you must save to different files and correctly specify these files when resuming the calculation.
- **[sauvegarde_simple]** (*type: format_file*) The same keyword than Sauvegarde except, the last time step only is saved.
- **[reprise]** (*type: format_file*) Keyword to resume a calculation based on the name_file file (see the class format_file). If format_reprise is xyz, the name_file file should be the .xyz file created by the previous calculation. With this file, it is possible to resume a parallel calculation on P processors, whereas the previous calculation has been run on N (N<>P) processors. Should the calculation be resumed, values for the tinit (see schema_temps_base) time fields are taken from the name_file file. If there is no backup corresponding to this time in the name_file, TRUST exits in error.
- **[resume_last_time]** (*type: format_file*) Keyword to resume a calculation based on the name_file file, resume the calculation at the last time found in the file (tinit is set to last time of saved files).

---

pb_thermohydraulique_turbulent_qc

Inherits from: *pb_gen_base*

Resolution of turbulent thermohydraulic problem under low Mach number.

Warning : Available for VDF and VEF P0/P1NC discretization only.

Parameters are:

- **fluide_quasi_compressible** (*type: fluide_quasi_compressible*) The fluid medium associated with the problem.
- **navier_stokes_turbulent_qc** (*type: navier_stokes_turbulent_qc*) Navier-Stokes equations under low Mach number as well as the associated turbulence model equations.

- **convection_diffusion_chaleur_turbulent_qc** (*type: convection_diffusion_chaleur_turbulent_qc*) Energy equation under low Mach number as well as the associated turbulence model equations.
- **[milieu]** (*type: milieu_base*) The medium associated with the problem.
- **[constituant]** (*type: constituant*) Constituent.
- **[postraitement | post_processing]** (*type: corps_postraitement*) One post-processing (without name).
- **[postraitements | post_processings]** (*type: postraitements*) List of Postraitement objects (with name).
- **[liste_de_postraitements]** (*type: liste_post_ok*) This
- **[liste_postraitements]** (*type: liste_post*) This block defines the output files to be written during the computation. The output format is lata in order to use OpenDX to draw the results. This block can be divided in one or several sub-blocks that can be written at different frequencies and in different directories. Attention. The directory lata used in this example should be created before running the computation or the lata files will be lost.
- **[sauvegarde]** (*type: format_file*) Keyword used when calculation results are to be backed up. When a coupling is performed, the backup-recovery file name must be well specified for each problem. In this case, you must save to different files and correctly specify these files when resuming the calculation.
- **[sauvegarde_simple]** (*type: format_file*) The same keyword than Sauvegarde except, the last time step only is saved.
- **[reprise]** (*type: format_file*) Keyword to resume a calculation based on the name_file file (see the class format_file). If format_reprise is xyz, the name_file file should be the .xyz file created by the previous calculation. With this file, it is possible to resume a parallel calculation on P processors, whereas the previous calculation has been run on N (N<>P) processors. Should the calculation be resumed, values for the tinit (see schema_temps_base) time fields are taken from the name_file file. If there is no backup corresponding to this time in the name_file, TRUST exits in error.
- **[resume_last_time]** (*type: format_file*) Keyword to resume a calculation based on the name_file file, resume the calculation at the last time found in the file (tinit is set to last time of saved files).

---

pb_thermohydraulique_turbulent_scalaires_passifs

Inherits from: *pb_gen_base*

Resolution of thermohydraulic problem, with turbulence modelling and with the additional passive scalar equations.

Parameters are:

- **fluide_incompressible** (*type: fluide_incompressible*) The fluid medium associated with the problem.
- **[constituant]** (*type: constituant*) Constituents.
- **[navier_stokes_turbulent]** (*type: navier_stokes_turbulent*) Navier-Stokes equations as well as the associated turbulence model equations.
- **[convection_diffusion_temperature_turbulent]** (*type: convection_diffusion_temperature_turbulent*) Energy equations (temperature diffusion convection) as well as the associated turbulence model equations.
- **equations_scalaires_passifs** (*type: listeqn*) Passive scalar equations. The unknowns of the passive scalar equation number N are named temperatureN or concentrationN or fraction_massiqueN. This keyword is used to define initial conditions and the post processing fields. This kind of problem is very useful to test in only one data file (and then only one calculation) different schemes or different boundary conditions for the scalar transport equation.
- **[milieu]** (*type: milieu_base*) The medium associated with the problem.

- **[postraitement | post_processing]** (*type: corps_postraitement*) One post-processing (without name).
- **[postraitements | post_processings]** (*type: postraitements*) List of Postraitement objects (with name).
- **[liste_de_postraitements]** (*type: liste_post_ok*) This
- **[liste_postraitements]** (*type: liste_post*) This block defines the output files to be written during the computation. The output format is lata in order to use OpenDX to draw the results. This block can be divided in one or several sub-blocks that can be written at different frequencies and in different directories. Attention. The directory lata used in this example should be created before running the computation or the lata files will be lost.
- **[sauvegarde]** (*type: format_file*) Keyword used when calculation results are to be backed up. When a coupling is performed, the backup-recovery file name must be well specified for each problem. In this case, you must save to different files and correctly specify these files when resuming the calculation.
- **[sauvegarde_simple]** (*type: format_file*) The same keyword than Sauvegarde except, the last time step only is saved.
- **[reprise]** (*type: format_file*) Keyword to resume a calculation based on the name_file file (see the class format_file). If format_reprise is xyz, the name_file file should be the .xyz file created by the previous calculation. With this file, it is possible to resume a parallel calculation on P processors, whereas the previous calculation has been run on N (N<>P) processors. Should the calculation be resumed, values for the tinit (see schema_temps_base) time fields are taken from the name_file file. If there is no backup corresponding to this time in the name_file, TRUST exits in error.
- **[resume_last_time]** (*type: format_file*) Keyword to resume a calculation based on the name_file file, resume the calculation at the last time found in the file (tinit is set to last time of saved files).

---

pb_thermohydraulique_wc

Inherits from: *pb_gen_base*

Resolution of thermo-hydraulic problem for a weakly-compressible fluid.

Keywords for the unknowns other than pressure, velocity, temperature are :

masse_volumique : density

pression : reduced pressure

pression_tot : total pressure

pression_hydro : hydro-static pressure

pression_eos : pressure used in state equation.

Parameters are:

- **fluide_weakly_compressible** (*type: fluide_weakly_compressible*) The fluid medium associated with the problem.
- **navier_stokes_wc** (*type: navier_stokes_wc*) Navier-Stokes equation for a weakly-compressible fluid.
- **convection_diffusion_chaleur_wc** (*type: convection_diffusion_chaleur_wc*) Temperature equation for a weakly-compressible fluid.
- **[milieu]** (*type: milieu_base*) The medium associated with the problem.
- **[constituant]** (*type: constituant*) Constituent.
- **[postraitement | post_processing]** (*type: corps_postraitement*) One post-processing (without name).
- **[postraitements | post_processings]** (*type: postraitements*) List of Postraitement objects (with name).

- **[liste_de_postraitements]** (*type: liste_post_ok*) This
- **[liste_postraitements]** (*type: liste_post*) This block defines the output files to be written during the computation. The output format is lata in order to use OpenDX to draw the results. This block can be divided in one or several sub-blocks that can be written at different frequencies and in different directories. Attention. The directory lata used in this example should be created before running the computation or the lata files will be lost.
- **[sauvegarde]** (*type: format_file*) Keyword used when calculation results are to be backed up. When a coupling is performed, the backup-recovery file name must be well specified for each problem. In this case, you must save to different files and correctly specify these files when resuming the calculation.
- **[sauvegarde_simple]** (*type: format_file*) The same keyword than Sauvegarde except, the last time step only is saved.
- **[reprise]** (*type: format_file*) Keyword to resume a calculation based on the name_file file (see the class format_file). If format_reprise is xyz, the name_file file should be the .xyz file created by the previous calculation. With this file, it is possible to resume a parallel calculation on P processors, whereas the previous calculation has been run on N (N<>P) processors. Should the calculation be resumed, values for the tinit (see schema_temps_base) time fields are taken from the name_file file. If there is no backup corresponding to this time in the name_file, TRUST exits in error.
- **[resume_last_time]** (*type: format_file*) Keyword to resume a calculation based on the name_file file, resume the calculation at the last time found in the file (tinit is set to last time of saved files).

---

pbc_med

Inherits from: *pb_gen_base*

Allows to read med files and post-process them.

Parameters are:

- **list_info_med** (*type: list_info_med*) not_set

---

problem_read_generic

Inherits from: *pb_gen_base*

The probleme_read_generic differs rom the rest of the TRUST code : The problem does not state the number of equations that are enclosed in the problem. As the list of equations to be solved in the generic read problem is declared in the data file and not pre-defined in the structure of the problem, each equation has to be distinctively associated with the problem with the Associate keyword.

Parameters are:

- **[milieu]** (*type: milieu_base*) The medium associated with the problem.
- **[constituant]** (*type: constituant*) Constituent.
- **[postraitement | post_processing]** (*type: corps_postraitement*) One post-processing (without name).
- **[postraitements | post_processings]** (*type: postraitements*) List of Postraitement objects (with name).
- **[liste_de_postraitements]** (*type: liste_post_ok*) This

- **[liste_postraitements]** (*type: liste_post*) This block defines the output files to be written during the computation. The output format is lata in order to use OpenDX to draw the results. This block can be divided in one or several sub-blocks that can be written at different frequencies and in different directories. Attention. The directory lata used in this example should be created before running the computation or the lata files will be lost.
- **[sauvegarde]** (*type: format_file*) Keyword used when calculation results are to be backed up. When a coupling is performed, the backup-recovery file name must be well specified for each problem. In this case, you must save to different files and correctly specify these files when resuming the calculation.
- **[sauvegarde_simple]** (*type: format_file*) The same keyword than Sauvegarde except, the last time step only is saved.
- **[reprise]** (*type: format_file*) Keyword to resume a calculation based on the name_file file (see the class format_file). If format_reprise is xyz, the name_file file should be the .xyz file created by the previous calculation. With this file, it is possible to resume a parallel calculation on P processors, whereas the previous calculation has been run on N (N<>P) processors. Should the calculation be resumed, values for the tinit (see schema_temps_base) time fields are taken from the name_file file. If there is no backup corresponding to this time in the name_file, TRUST exits in error.
- **[resume_last_time]** (*type: format_file*) Keyword to resume a calculation based on the name_file file, resume the calculation at the last time found in the file (tinit is set to last time of saved files).

2.1.49 Keywords derived from penalisation_l2_ftd

penalisation_l2_ftd

Synonyms: pp

not_set

2.1.50 Keywords derived from porosites

porosites

To define the volume porosity and surface porosity that are uniform in every direction in space on a sub-area.

Porosity was only usable in VDF discretization, and now available for VEF P1NC/P0.

Observations :

- Surface porosity values must be given in every direction in space (set this value

to 1 if there is no porosity),

- Prior to defining porosity, the problem must have been discretized.NL2 Can ‘t be

used in VEF discretization, use Porosites_champ instead.

Parameters are:

- **aco** (*type:* string into [“{“]) Opening curly bracket.
- **sous_zone | sous_zone1** (*type:* string) Name of the sub-area to which porosity are allocated.
- **bloc** (*type: bloc_lecture_poro*) Surface and volume porosity values.
- **[sous_zone2]** (*type:* string) Name of the 2nd sub-area to which porosity are allocated.

- **[bloc2]** (*type:* *bloc_lecture_poro*) Surface and volume porosity values.
- **acof** (*type:* string into ["}"]) Closing curly bracket.

2.1.51 Keywords derived from postraitements

postraitements

Synonyms: post_processings

Keyword to use several results files. List of objects of post-processing (with name).

2.1.52 Keywords derived from precond_base

ilu

Inherits from: *precond_base*

This preconditionner can be only used with the generic GEN solver.

Parameters are:

- **[type]** (*type:* int) values can be 0|1|2|3 for null|left|right|left-and-right preconditionning (default value = 2)
- **[filling]** (*type:* int) default value = 1.

---

precond_base

Basic class for preconditioning.

---

precondsolv

Inherits from: *precond_base*

not_set

Parameters are:

- **solveur** (*type:* *solveur_sys_base*) Solver type.

---

ssor

Inherits from: *precond_base*

Symmetric successive over-relaxation algorithm.

Parameters are:

- **[omega]** (*type:* double) Over-relaxation facteur (between 1 and 2, default value 1.6).

---

ssor_bloc

Inherits from: *precond_base*

not_set

Parameters are:

- **[alpha_0]** (*type:* double) not_set
- **[precond0]** (*type: precond_base*) not_set
- **[alpha_1]** (*type:* double) not_set
- **[precond1]** (*type: precond_base*) not_set
- **[alpha_a]** (*type:* double) not_set
- **[preconda]** (*type: precond_base*) not_set

2.1.53 Keywords derived from reactions

reactions

list of reactions

2.1.54 Keywords derived from schema_temps_base

euler_scheme

Synonyms: schema_euler_explicite, scheme_euler_explicit

Inherits from: *schema_temps_base*

This is the Euler explicit scheme.

Parameters are:

- **[tinit]** (*type:* double) Value of initial calculation time (0 by default).
- **[tmax]** (*type:* double) Time during which the calculation will be stopped (1e30s by default).
- **[tcpumax]** (*type:* double) CPU time limit (must be specified in hours) for which the calculation is stopped (1e30s by default).

- **[dt_min]** (*type:* double) Minimum calculation time step (1e-16s by default).
- **[dt_max]** (*type:* string) Maximum calculation time step as function of time (1e30s by default).
- **[dt_sauv]** (*type:* double) Save time step value (1e30s by default). Every dt_sauv, fields are saved in the .sauv file. The file contains all the information saved over time. If this instruction is not entered, results are saved only upon calculation completion. To disable the writing of the .sauv files, you must specify 0. Note that dt_sauv is in terms of physical time (not cpu time).
- **[dt_impr]** (*type:* double) Scheme parameter printing time step in time (1e30s by default). The time steps and the flux balances are printed (incorporated onto every side of processed domains) into the .out file.
- **[facsec]** (*type:* string) Value assigned to the safety factor for the time step (1. by default). It can also be a function of time. The time step calculated is multiplied by the safety factor. The first thing to try when a calculation does not converge with an explicit time scheme is to reduce the facsec to 0.5. Warning: Some schemes needs a facsec lower than 1 (0.5 is a good start), for example Schema_Adams_Bashforth_order_3.
- **[seuil_statio]** (*type:* double) Value of the convergence threshold (1e-12 by default). Problems using this type of time scheme converge when the derivatives dGi/dt NL1 of all the unknown transported values Gi have a combined absolute value less than this value. This is the keyword used to set the permanent rating threshold.
- **[residuals]** (*type: residuals*) To specify how the residuals will be computed (default max norm, possible to choose L2-norm instead).
- **[diffusion_implicite]** (*type:* int) Keyword to make the diffusive term in the Navier-Stokes equations implicit (in this case, it should be set to 1). The stability time step is then only based on the convection time step (dt=facsec*dt_convection). Thus, in some circumstances, an important gain is achieved with respect to the time step (large diffusion with respect to convection on tightened meshes). Caution: It is however recommended that the user avoids exceeding the convection time step by selecting a too large facsec value. Start with a facsec value of 1 and then increase it gradually if you wish to accelerate calculation. In addition, for a natural convection calculation with a zero initial velocity, in the first time step, the convection time is infinite and therefore dt=facsec*dt_max.
- **[seuil_diffusion_implicite]** (*type:* double) This keyword changes the default value (1e-6) of convergency criteria for the resolution by conjugate gradient used for implicit diffusion.
- **[impr_diffusion_implicite]** (*type:* int) Unactivate (default) or not the printing of the convergence during the resolution of the conjugate gradient.
- **[impr_extremums]** (*type:* int) Print unknowns extremas
- **[no_error_if_not_converged_diffusion_implicite]** (*type:* int) not_set
- **[no_conv_subiteration_diffusion_implicite]** (*type:* int) not_set
- **[dt_start]** (*type: dt_start*) dt_start dt_min : the first iteration is based on dt_min. dt_start dt_calc : the time step at first iteration is calculated in agreement with CFL condition. dt_start dt_fixe value : the first time step is fixed by the user (recommended when resuming calculation with Crank Nicholson temporal scheme to ensure continuity). By default, the first iteration is based on dt_calc.
- **[nb_pas_dt_max]** (*type:* int) Maximum number of calculation time steps (1e9 by default).
- **[niter_max_diffusion_implicite]** (*type:* int) This keyword changes the default value (number of unknowns) of the maximal iterations number in the conjugate gradient method used for implicit diffusion.
- **[precision_impr]** (*type:* int) Optional keyword to define the digit number for flux values printed into .out files (by default 3).
- **[periode_sauvegarde_securite_en_heures]** (*type:* double) To change the default period (23 hours) between the save of the fields in .sauv file.
- **[no_check_disk_space]** (*type:* flag) To disable the check of the available amount of disk space during the calculation.

- **[disable_progress]** (*type:* flag) To disable the writing of the .progress file.
- **[disable_dt_ev]** (*type:* flag) To disable the writing of the .dt_ev file.
- **[gnuplot_header]** (*type:* int) Optional keyword to modify the header of the .out files. Allows to use the column title instead of columns number.

---

leap_frog

Inherits from: *schema_temps_base*

This is the leap-frog scheme.

Parameters are:

- **[tinit]** (*type:* double) Value of initial calculation time (0 by default).
- **[tmax]** (*type:* double) Time during which the calculation will be stopped (1e30s by default).
- **[tcpumax]** (*type:* double) CPU time limit (must be specified in hours) for which the calculation is stopped (1e30s by default).
- **[dt_min]** (*type:* double) Minimum calculation time step (1e-16s by default).
- **[dt_max]** (*type:* string) Maximum calculation time step as function of time (1e30s by default).
- **[dt_sauv]** (*type:* double) Save time step value (1e30s by default). Every dt_sauv, fields are saved in the .sauv file. The file contains all the information saved over time. If this instruction is not entered, results are saved only upon calculation completion. To disable the writing of the .sauv files, you must specify 0. Note that dt_sauv is in terms of physical time (not cpu time).
- **[dt_impr]** (*type:* double) Scheme parameter printing time step in time (1e30s by default). The time steps and the flux balances are printed (incorporated onto every side of processed domains) into the .out file.
- **[facsec]** (*type:* string) Value assigned to the safety factor for the time step (1. by default). It can also be a function of time. The time step calculated is multiplied by the safety factor. The first thing to try when a calculation does not converge with an explicit time scheme is to reduce the facsec to 0.5. Warning: Some schemes needs a facsec lower than 1 (0.5 is a good start), for example Schema_Adams_Bashforth_order_3.
- **[seuil_statio]** (*type:* double) Value of the convergence threshold (1e-12 by default). Problems using this type of time scheme converge when the derivatives dGi/dt NL1 of all the unknown transported values Gi have a combined absolute value less than this value. This is the keyword used to set the permanent rating threshold.
- **[residuals]** (*type: residuals*) To specify how the residuals will be computed (default max norm, possible to choose L2-norm instead).
- **[diffusion_implicite]** (*type:* int) Keyword to make the diffusive term in the Navier-Stokes equations implicit (in this case, it should be set to 1). The stability time step is then only based on the convection time step (dt=facsec*dt_convection). Thus, in some circumstances, an important gain is achieved with respect to the time step (large diffusion with respect to convection on tightened meshes). Caution: It is however recommended that the user avoids exceeding the convection time step by selecting a too large facsec value. Start with a facsec value of 1 and then increase it gradually if you wish to accelerate calculation. In addition, for a natural convection calculation with a zero initial velocity, in the first time step, the convection time is infinite and therefore dt=facsec*dt_max.
- **[seuil_diffusion_implicite]** (*type:* double) This keyword changes the default value (1e-6) of convergency criteria for the resolution by conjugate gradient used for implicit diffusion.
- **[impr_diffusion_implicite]** (*type:* int) Unactivate (default) or not the printing of the convergence during the resolution of the conjugate gradient.

- **[impr_extremums]** (*type:* int) Print unknowns extremas
- **[no_error_if_not_converged_diffusion_implicite]** (*type:* int) not_set
- **[no_conv_subiteration_diffusion_implicite]** (*type:* int) not_set
- **[dt_start]** (*type: dt_start*) dt_start dt_min : the first iteration is based on dt_min. dt_start dt_calc : the time step at first iteration is calculated in agreement with CFL condition. dt_start dt_fixe value : the first time step is fixed by the user (recommended when resuming calculation with Crank Nicholson temporal scheme to ensure continuity). By default, the first iteration is based on dt_calc.
- **[nb_pas_dt_max]** (*type:* int) Maximum number of calculation time steps (1e9 by default).
- **[niter_max_diffusion_implicite]** (*type:* int) This keyword changes the default value (number of unknowns) of the maximal iterations number in the conjugate gradient method used for implicit diffusion.
- **[precision_impr]** (*type:* int) Optional keyword to define the digit number for flux values printed into .out files (by default 3).
- **[periode_sauvegarde_securite_en_heures]** (*type:* double) To change the default period (23 hours) between the save of the fields in .sauv file.
- **[no_check_disk_space]** (*type:* flag) To disable the check of the available amount of disk space during the calculation.
- **[disable_progress]** (*type:* flag) To disable the writing of the .progress file.
- **[disable_dt_ev]** (*type:* flag) To disable the writing of the .dt_ev file.
- **[gnuplot_header]** (*type:* int) Optional keyword to modify the header of the .out files. Allows to use the column title instead of columns number.

---

runge_kutta_ordre_2

Inherits from: *schema_temps_base*

This is a low-storage Runge-Kutta scheme of second order that uses 2 integration points. The method is presented by Williamson (case 1) in https://www.sciencedirect.com/science/article/pii/0021999180900339

Parameters are:

- **[tinit]** (*type:* double) Value of initial calculation time (0 by default).
- **[tmax]** (*type:* double) Time during which the calculation will be stopped (1e30s by default).
- **[tcpumax]** (*type:* double) CPU time limit (must be specified in hours) for which the calculation is stopped (1e30s by default).
- **[dt_min]** (*type:* double) Minimum calculation time step (1e-16s by default).
- **[dt_max]** (*type:* string) Maximum calculation time step as function of time (1e30s by default).
- **[dt_sauv]** (*type:* double) Save time step value (1e30s by default). Every dt_sauv, fields are saved in the .sauv file. The file contains all the information saved over time. If this instruction is not entered, results are saved only upon calculation completion. To disable the writing of the .sauv files, you must specify 0. Note that dt_sauv is in terms of physical time (not cpu time).
- **[dt_impr]** (*type:* double) Scheme parameter printing time step in time (1e30s by default). The time steps and the flux balances are printed (incorporated onto every side of processed domains) into the .out file.

- **[facsec]** (*type:* string) Value assigned to the safety factor for the time step (1. by default). It can also be a function of time. The time step calculated is multiplied by the safety factor. The first thing to try when a calculation does not converge with an explicit time scheme is to reduce the facsec to 0.5. Warning: Some schemes needs a facsec lower than 1 (0.5 is a good start), for example Schema_Adams_Bashforth_order_3.
- **[seuil_statio]** (*type:* double) Value of the convergence threshold (1e-12 by default). Problems using this type of time scheme converge when the derivatives dGi/dt NL1 of all the unknown transported values Gi have a combined absolute value less than this value. This is the keyword used to set the permanent rating threshold.
- **[residuals]** (*type: residuals*) To specify how the residuals will be computed (default max norm, possible to choose L2-norm instead).
- **[diffusion_implicite]** (*type:* int) Keyword to make the diffusive term in the Navier-Stokes equations implicit (in this case, it should be set to 1). The stability time step is then only based on the convection time step (dt=facsec*dt_convection). Thus, in some circumstances, an important gain is achieved with respect to the time step (large diffusion with respect to convection on tightened meshes). Caution: It is however recommended that the user avoids exceeding the convection time step by selecting a too large facsec value. Start with a facsec value of 1 and then increase it gradually if you wish to accelerate calculation. In addition, for a natural convection calculation with a zero initial velocity, in the first time step, the convection time is infinite and therefore dt=facsec*dt_max.
- **[seuil_diffusion_implicite]** (*type:* double) This keyword changes the default value (1e-6) of convergency criteria for the resolution by conjugate gradient used for implicit diffusion.
- **[impr_diffusion_implicite]** (*type:* int) Unactivate (default) or not the printing of the convergence during the resolution of the conjugate gradient.
- **[impr_extremums]** (*type:* int) Print unknowns extremas
- **[no_error_if_not_converged_diffusion_implicite]** (*type:* int) not_set
- **[no_conv_subiteration_diffusion_implicite]** (*type:* int) not_set
- **[dt_start]** (*type: dt_start*) dt_start dt_min : the first iteration is based on dt_min. dt_start dt_calc : the time step at first iteration is calculated in agreement with CFL condition. dt_start dt_fixe value : the first time step is fixed by the user (recommended when resuming calculation with Crank Nicholson temporal scheme to ensure continuity). By default, the first iteration is based on dt_calc.
- **[nb_pas_dt_max]** (*type:* int) Maximum number of calculation time steps (1e9 by default).
- **[niter_max_diffusion_implicite]** (*type:* int) This keyword changes the default value (number of unknowns) of the maximal iterations number in the conjugate gradient method used for implicit diffusion.
- **[precision_impr]** (*type:* int) Optional keyword to define the digit number for flux values printed into .out files (by default 3).
- **[periode_sauvegarde_securite_en_heures]** (*type:* double) To change the default period (23 hours) between the save of the fields in .sauv file.
- **[no_check_disk_space]** (*type:* flag) To disable the check of the available amount of disk space during the calculation.
- **[disable_progress]** (*type:* flag) To disable the writing of the .progress file.
- **[disable_dt_ev]** (*type:* flag) To disable the writing of the .dt_ev file.
- **[gnuplot_header]** (*type:* int) Optional keyword to modify the header of the .out files. Allows to use the column title instead of columns number.

runge_kutta_ordre_2_classique

Inherits from: *schema_temps_base*

This is a classical Runge-Kutta scheme of second order that uses 2 integration points.

Parameters are:

- **[tinit]** (*type:* double) Value of initial calculation time (0 by default).
- **[tmax]** (*type:* double) Time during which the calculation will be stopped (1e30s by default).
- **[tcpumax]** (*type:* double) CPU time limit (must be specified in hours) for which the calculation is stopped (1e30s by default).
- **[dt_min]** (*type:* double) Minimum calculation time step (1e-16s by default).
- **[dt_max]** (*type:* string) Maximum calculation time step as function of time (1e30s by default).
- **[dt_sauv]** (*type:* double) Save time step value (1e30s by default). Every dt_sauv, fields are saved in the .sauv file. The file contains all the information saved over time. If this instruction is not entered, results are saved only upon calculation completion. To disable the writing of the .sauv files, you must specify 0. Note that dt_sauv is in terms of physical time (not cpu time).
- **[dt_impr]** (*type:* double) Scheme parameter printing time step in time (1e30s by default). The time steps and the flux balances are printed (incorporated onto every side of processed domains) into the .out file.
- **[facsec]** (*type:* string) Value assigned to the safety factor for the time step (1. by default). It can also be a function of time. The time step calculated is multiplied by the safety factor. The first thing to try when a calculation does not converge with an explicit time scheme is to reduce the facsec to 0.5. Warning: Some schemes needs a facsec lower than 1 (0.5 is a good start), for example Schema_Adams_Bashforth_order_3.
- **[seuil_statio]** (*type:* double) Value of the convergence threshold (1e-12 by default). Problems using this type of time scheme converge when the derivatives dGi/dt NL1 of all the unknown transported values Gi have a combined absolute value less than this value. This is the keyword used to set the permanent rating threshold.
- **[residuals]** (*type: residuals*) To specify how the residuals will be computed (default max norm, possible to choose L2-norm instead).
- **[diffusion_implicite]** (*type:* int) Keyword to make the diffusive term in the Navier-Stokes equations implicit (in this case, it should be set to 1). The stability time step is then only based on the convection time step (dt=facsec*dt_convection). Thus, in some circumstances, an important gain is achieved with respect to the time step (large diffusion with respect to convection on tightened meshes). Caution: It is however recommended that the user avoids exceeding the convection time step by selecting a too large facsec value. Start with a facsec value of 1 and then increase it gradually if you wish to accelerate calculation. In addition, for a natural convection calculation with a zero initial velocity, in the first time step, the convection time is infinite and therefore dt=facsec*dt_max.
- **[seuil_diffusion_implicite]** (*type:* double) This keyword changes the default value (1e-6) of convergency criteria for the resolution by conjugate gradient used for implicit diffusion.
- **[impr_diffusion_implicite]** (*type:* int) Unactivate (default) or not the printing of the convergence during the resolution of the conjugate gradient.
- **[impr_extremums]** (*type:* int) Print unknowns extremas
- **[no_error_if_not_converged_diffusion_implicite]** (*type:* int) not_set
- **[no_conv_subiteration_diffusion_implicite]** (*type:* int) not_set
- **[dt_start]** (*type: dt_start*) dt_start dt_min : the first iteration is based on dt_min. dt_start dt_calc : the time step at first iteration is calculated in agreement with CFL condition. dt_start dt_fixe value : the first time step is

fixed by the user (recommended when resuming calculation with Crank Nicholson temporal scheme to ensure continuity). By default, the first iteration is based on dt_calc.

- **[nb_pas_dt_max]** (*type:* int) Maximum number of calculation time steps (1e9 by default).
- **[niter_max_diffusion_implicite]** (*type:* int) This keyword changes the default value (number of unknowns) of the maximal iterations number in the conjugate gradient method used for implicit diffusion.
- **[precision_impr]** (*type:* int) Optional keyword to define the digit number for flux values printed into .out files (by default 3).
- **[periode_sauvegarde_securite_en_heures]** (*type:* double) To change the default period (23 hours) between the save of the fields in .sauv file.
- **[no_check_disk_space]** (*type:* flag) To disable the check of the available amount of disk space during the calculation.
- **[disable_progress]** (*type:* flag) To disable the writing of the .progress file.
- **[disable_dt_ev]** (*type:* flag) To disable the writing of the .dt_ev file.
- **[gnuplot_header]** (*type:* int) Optional keyword to modify the header of the .out files. Allows to use the column title instead of columns number.

---

runge_kutta_ordre_3

Inherits from: *schema_temps_base*

This is a low-storage Runge-Kutta scheme of third order that uses 3 integration points. The method is presented by Williamson (case 7) in https://www.sciencedirect.com/science/article/pii/0021999180900339

Parameters are:

- **[tinit]** (*type:* double) Value of initial calculation time (0 by default).
- **[tmax]** (*type:* double) Time during which the calculation will be stopped (1e30s by default).
- **[tcpumax]** (*type:* double) CPU time limit (must be specified in hours) for which the calculation is stopped (1e30s by default).
- **[dt_min]** (*type:* double) Minimum calculation time step (1e-16s by default).
- **[dt_max]** (*type:* string) Maximum calculation time step as function of time (1e30s by default).
- **[dt_sauv]** (*type:* double) Save time step value (1e30s by default). Every dt_sauv, fields are saved in the .sauv file. The file contains all the information saved over time. If this instruction is not entered, results are saved only upon calculation completion. To disable the writing of the .sauv files, you must specify 0. Note that dt_sauv is in terms of physical time (not cpu time).
- **[dt_impr]** (*type:* double) Scheme parameter printing time step in time (1e30s by default). The time steps and the flux balances are printed (incorporated onto every side of processed domains) into the .out file.
- **[facsec]** (*type:* string) Value assigned to the safety factor for the time step (1. by default). It can also be a function of time. The time step calculated is multiplied by the safety factor. The first thing to try when a calculation does not converge with an explicit time scheme is to reduce the facsec to 0.5. Warning: Some schemes needs a facsec lower than 1 (0.5 is a good start), for example Schema_Adams_Bashforth_order_3.
- **[seuil_statio]** (*type:* double) Value of the convergence threshold (1e-12 by default). Problems using this type of time scheme converge when the derivatives dGi/dt NL1 of all the unknown transported values Gi have a combined absolute value less than this value. This is the keyword used to set the permanent rating threshold.

- **[residuals]** (*type: residuals*) To specify how the residuals will be computed (default max norm, possible to choose L2-norm instead).
- **[diffusion_implicite]** (*type:* int) Keyword to make the diffusive term in the Navier-Stokes equations implicit (in this case, it should be set to 1). The stability time step is then only based on the convection time step (dt=facsec*dt_convection). Thus, in some circumstances, an important gain is achieved with respect to the time step (large diffusion with respect to convection on tightened meshes). Caution: It is however recommended that the user avoids exceeding the convection time step by selecting a too large facsec value. Start with a facsec value of 1 and then increase it gradually if you wish to accelerate calculation. In addition, for a natural convection calculation with a zero initial velocity, in the first time step, the convection time is infinite and therefore dt=facsec*dt_max.
- **[seuil_diffusion_implicite]** (*type:* double) This keyword changes the default value (1e-6) of convergency criteria for the resolution by conjugate gradient used for implicit diffusion.
- **[impr_diffusion_implicite]** (*type:* int) Unactivate (default) or not the printing of the convergence during the resolution of the conjugate gradient.
- **[impr_extremums]** (*type:* int) Print unknowns extremas
- **[no_error_if_not_converged_diffusion_implicite]** (*type:* int) not_set
- **[no_conv_subiteration_diffusion_implicite]** (*type:* int) not_set
- **[dt_start]** (*type: dt_start*) dt_start dt_min : the first iteration is based on dt_min. dt_start dt_calc : the time step at first iteration is calculated in agreement with CFL condition. dt_start dt_fixe value : the first time step is fixed by the user (recommended when resuming calculation with Crank Nicholson temporal scheme to ensure continuity). By default, the first iteration is based on dt_calc.
- **[nb_pas_dt_max]** (*type:* int) Maximum number of calculation time steps (1e9 by default).
- **[niter_max_diffusion_implicite]** (*type:* int) This keyword changes the default value (number of unknowns) of the maximal iterations number in the conjugate gradient method used for implicit diffusion.
- **[precision_impr]** (*type:* int) Optional keyword to define the digit number for flux values printed into .out files (by default 3).
- **[periode_sauvegarde_securite_en_heures]** (*type:* double) To change the default period (23 hours) between the save of the fields in .sauv file.
- **[no_check_disk_space]** (*type:* flag) To disable the check of the available amount of disk space during the calculation.
- **[disable_progress]** (*type:* flag) To disable the writing of the .progress file.
- **[disable_dt_ev]** (*type:* flag) To disable the writing of the .dt_ev file.
- **[gnuplot_header]** (*type:* int) Optional keyword to modify the header of the .out files. Allows to use the column title instead of columns number.

runge_kutta_ordre_3_classique

Inherits from: *schema_temps_base*

This is a classical Runge-Kutta scheme of third order that uses 3 integration points.

Parameters are:

- **[tinit]** (*type:* double) Value of initial calculation time (0 by default).
- **[tmax]** (*type:* double) Time during which the calculation will be stopped (1e30s by default).
- **[tcpumax]** (*type:* double) CPU time limit (must be specified in hours) for which the calculation is stopped (1e30s by default).
- **[dt_min]** (*type:* double) Minimum calculation time step (1e-16s by default).
- **[dt_max]** (*type:* string) Maximum calculation time step as function of time (1e30s by default).
- **[dt_sauv]** (*type:* double) Save time step value (1e30s by default). Every dt_sauv, fields are saved in the .sauv file. The file contains all the information saved over time. If this instruction is not entered, results are saved only upon calculation completion. To disable the writing of the .sauv files, you must specify 0. Note that dt_sauv is in terms of physical time (not cpu time).
- **[dt_impr]** (*type:* double) Scheme parameter printing time step in time (1e30s by default). The time steps and the flux balances are printed (incorporated onto every side of processed domains) into the .out file.
- **[facsec]** (*type:* string) Value assigned to the safety factor for the time step (1. by default). It can also be a function of time. The time step calculated is multiplied by the safety factor. The first thing to try when a calculation does not converge with an explicit time scheme is to reduce the facsec to 0.5. Warning: Some schemes needs a facsec lower than 1 (0.5 is a good start), for example Schema_Adams_Bashforth_order_3.
- **[seuil_statio]** (*type:* double) Value of the convergence threshold (1e-12 by default). Problems using this type of time scheme converge when the derivatives dGi/dt NL1 of all the unknown transported values Gi have a combined absolute value less than this value. This is the keyword used to set the permanent rating threshold.
- **[residuals]** (*type: residuals*) To specify how the residuals will be computed (default max norm, possible to choose L2-norm instead).
- **[diffusion_implicite]** (*type:* int) Keyword to make the diffusive term in the Navier-Stokes equations implicit (in this case, it should be set to 1). The stability time step is then only based on the convection time step (dt=facsec*dt_convection). Thus, in some circumstances, an important gain is achieved with respect to the time step (large diffusion with respect to convection on tightened meshes). Caution: It is however recommended that the user avoids exceeding the convection time step by selecting a too large facsec value. Start with a facsec value of 1 and then increase it gradually if you wish to accelerate calculation. In addition, for a natural convection calculation with a zero initial velocity, in the first time step, the convection time is infinite and therefore dt=facsec*dt_max.
- **[seuil_diffusion_implicite]** (*type:* double) This keyword changes the default value (1e-6) of convergency criteria for the resolution by conjugate gradient used for implicit diffusion.
- **[impr_diffusion_implicite]** (*type:* int) Unactivate (default) or not the printing of the convergence during the resolution of the conjugate gradient.
- **[impr_extremums]** (*type:* int) Print unknowns extremas
- **[no_error_if_not_converged_diffusion_implicite]** (*type:* int) not_set
- **[no_conv_subiteration_diffusion_implicite]** (*type:* int) not_set
- **[dt_start]** (*type: dt_start*) dt_start dt_min : the first iteration is based on dt_min. dt_start dt_calc : the time step at first iteration is calculated in agreement with CFL condition. dt_start dt_fixe value : the first time step is

fixed by the user (recommended when resuming calculation with Crank Nicholson temporal scheme to ensure continuity). By default, the first iteration is based on dt_calc.

- **[nb_pas_dt_max]** (*type:* int) Maximum number of calculation time steps (1e9 by default).
- **[niter_max_diffusion_implicite]** (*type:* int) This keyword changes the default value (number of unknowns) of the maximal iterations number in the conjugate gradient method used for implicit diffusion.
- **[precision_impr]** (*type:* int) Optional keyword to define the digit number for flux values printed into .out files (by default 3).
- **[periode_sauvegarde_securite_en_heures]** (*type:* double) To change the default period (23 hours) between the save of the fields in .sauv file.
- **[no_check_disk_space]** (*type:* flag) To disable the check of the available amount of disk space during the calculation.
- **[disable_progress]** (*type:* flag) To disable the writing of the .progress file.
- **[disable_dt_ev]** (*type:* flag) To disable the writing of the .dt_ev file.
- **[gnuplot_header]** (*type:* int) Optional keyword to modify the header of the .out files. Allows to use the column title instead of columns number.

---

runge_kutta_ordre_4

Synonyms: runge_kutta_ordre_4_d3p

Inherits from: *schema_temps_base*

This is a low-storage Runge-Kutta scheme of fourth order that uses 3 integration points. The method is presented by Williamson (case 17) in https://www.sciencedirect.com/science/article/pii/0021999180900339

Parameters are:

- **[tinit]** (*type:* double) Value of initial calculation time (0 by default).
- **[tmax]** (*type:* double) Time during which the calculation will be stopped (1e30s by default).
- **[tcpumax]** (*type:* double) CPU time limit (must be specified in hours) for which the calculation is stopped (1e30s by default).
- **[dt_min]** (*type:* double) Minimum calculation time step (1e-16s by default).
- **[dt_max]** (*type:* string) Maximum calculation time step as function of time (1e30s by default).
- **[dt_sauv]** (*type:* double) Save time step value (1e30s by default). Every dt_sauv, fields are saved in the .sauv file. The file contains all the information saved over time. If this instruction is not entered, results are saved only upon calculation completion. To disable the writing of the .sauv files, you must specify 0. Note that dt_sauv is in terms of physical time (not cpu time).
- **[dt_impr]** (*type:* double) Scheme parameter printing time step in time (1e30s by default). The time steps and the flux balances are printed (incorporated onto every side of processed domains) into the .out file.
- **[facsec]** (*type:* string) Value assigned to the safety factor for the time step (1. by default). It can also be a function of time. The time step calculated is multiplied by the safety factor. The first thing to try when a calculation does not converge with an explicit time scheme is to reduce the facsec to 0.5. Warning: Some schemes needs a facsec lower than 1 (0.5 is a good start), for example Schema_Adams_Bashforth_order_3.
- **[seuil_statio]** (*type:* double) Value of the convergence threshold (1e-12 by default). Problems using this type of time scheme converge when the derivatives dGi/dt NL1 of all the unknown transported values Gi have a combined absolute value less than this value. This is the keyword used to set the permanent rating threshold.

- **[residuals]** (*type: residuals*) To specify how the residuals will be computed (default max norm, possible to choose L2-norm instead).
- **[diffusion_implicite]** (*type:* int) Keyword to make the diffusive term in the Navier-Stokes equations implicit (in this case, it should be set to 1). The stability time step is then only based on the convection time step (dt=facsec*dt_convection). Thus, in some circumstances, an important gain is achieved with respect to the time step (large diffusion with respect to convection on tightened meshes). Caution: It is however recommended that the user avoids exceeding the convection time step by selecting a too large facsec value. Start with a facsec value of 1 and then increase it gradually if you wish to accelerate calculation. In addition, for a natural convection calculation with a zero initial velocity, in the first time step, the convection time is infinite and therefore dt=facsec*dt_max.
- **[seuil_diffusion_implicite]** (*type:* double) This keyword changes the default value (1e-6) of convergency criteria for the resolution by conjugate gradient used for implicit diffusion.
- **[impr_diffusion_implicite]** (*type:* int) Unactivate (default) or not the printing of the convergence during the resolution of the conjugate gradient.
- **[impr_extremums]** (*type:* int) Print unknowns extremas
- **[no_error_if_not_converged_diffusion_implicite]** (*type:* int) not_set
- **[no_conv_subiteration_diffusion_implicite]** (*type:* int) not_set
- **[dt_start]** (*type: dt_start*) dt_start dt_min : the first iteration is based on dt_min. dt_start dt_calc : the time step at first iteration is calculated in agreement with CFL condition. dt_start dt_fixe value : the first time step is fixed by the user (recommended when resuming calculation with Crank Nicholson temporal scheme to ensure continuity). By default, the first iteration is based on dt_calc.
- **[nb_pas_dt_max]** (*type:* int) Maximum number of calculation time steps (1e9 by default).
- **[niter_max_diffusion_implicite]** (*type:* int) This keyword changes the default value (number of unknowns) of the maximal iterations number in the conjugate gradient method used for implicit diffusion.
- **[precision_impr]** (*type:* int) Optional keyword to define the digit number for flux values printed into .out files (by default 3).
- **[periode_sauvegarde_securite_en_heures]** (*type:* double) To change the default period (23 hours) between the save of the fields in .sauv file.
- **[no_check_disk_space]** (*type:* flag) To disable the check of the available amount of disk space during the calculation.
- **[disable_progress]** (*type:* flag) To disable the writing of the .progress file.
- **[disable_dt_ev]** (*type:* flag) To disable the writing of the .dt_ev file.
- **[gnuplot_header]** (*type:* int) Optional keyword to modify the header of the .out files. Allows to use the column title instead of columns number.

runge_kutta_ordre_4_classique_3_8

Inherits from: *schema_temps_base*

This is a classical Runge-Kutta scheme of fourth order that uses 4 integration points and the 3/8 rule.

Parameters are:

- **[tinit]** (*type:* double) Value of initial calculation time (0 by default).
- **[tmax]** (*type:* double) Time during which the calculation will be stopped (1e30s by default).
- **[tcpumax]** (*type:* double) CPU time limit (must be specified in hours) for which the calculation is stopped (1e30s by default).
- **[dt_min]** (*type:* double) Minimum calculation time step (1e-16s by default).
- **[dt_max]** (*type:* string) Maximum calculation time step as function of time (1e30s by default).
- **[dt_sauv]** (*type:* double) Save time step value (1e30s by default). Every dt_sauv, fields are saved in the .sauv file. The file contains all the information saved over time. If this instruction is not entered, results are saved only upon calculation completion. To disable the writing of the .sauv files, you must specify 0. Note that dt_sauv is in terms of physical time (not cpu time).
- **[dt_impr]** (*type:* double) Scheme parameter printing time step in time (1e30s by default). The time steps and the flux balances are printed (incorporated onto every side of processed domains) into the .out file.
- **[facsec]** (*type:* string) Value assigned to the safety factor for the time step (1. by default). It can also be a function of time. The time step calculated is multiplied by the safety factor. The first thing to try when a calculation does not converge with an explicit time scheme is to reduce the facsec to 0.5. Warning: Some schemes needs a facsec lower than 1 (0.5 is a good start), for example Schema_Adams_Bashforth_order_3.
- **[seuil_statio]** (*type:* double) Value of the convergence threshold (1e-12 by default). Problems using this type of time scheme converge when the derivatives dGi/dt NL1 of all the unknown transported values Gi have a combined absolute value less than this value. This is the keyword used to set the permanent rating threshold.
- **[residuals]** (*type: residuals*) To specify how the residuals will be computed (default max norm, possible to choose L2-norm instead).
- **[diffusion_implicite]** (*type:* int) Keyword to make the diffusive term in the Navier-Stokes equations implicit (in this case, it should be set to 1). The stability time step is then only based on the convection time step (dt=facsec*dt_convection). Thus, in some circumstances, an important gain is achieved with respect to the time step (large diffusion with respect to convection on tightened meshes). Caution: It is however recommended that the user avoids exceeding the convection time step by selecting a too large facsec value. Start with a facsec value of 1 and then increase it gradually if you wish to accelerate calculation. In addition, for a natural convection calculation with a zero initial velocity, in the first time step, the convection time is infinite and therefore dt=facsec*dt_max.
- **[seuil_diffusion_implicite]** (*type:* double) This keyword changes the default value (1e-6) of convergency criteria for the resolution by conjugate gradient used for implicit diffusion.
- **[impr_diffusion_implicite]** (*type:* int) Unactivate (default) or not the printing of the convergence during the resolution of the conjugate gradient.
- **[impr_extremums]** (*type:* int) Print unknowns extremas
- **[no_error_if_not_converged_diffusion_implicite]** (*type:* int) not_set
- **[no_conv_subiteration_diffusion_implicite]** (*type:* int) not_set
- **[dt_start]** (*type: dt_start*) dt_start dt_min : the first iteration is based on dt_min. dt_start dt_calc : the time step at first iteration is calculated in agreement with CFL condition. dt_start dt_fixe value : the first time step is

fixed by the user (recommended when resuming calculation with Crank Nicholson temporal scheme to ensure continuity). By default, the first iteration is based on dt_calc.

- **[nb_pas_dt_max]** (*type:* int) Maximum number of calculation time steps (1e9 by default).
- **[niter_max_diffusion_implicite]** (*type:* int) This keyword changes the default value (number of unknowns) of the maximal iterations number in the conjugate gradient method used for implicit diffusion.
- **[precision_impr]** (*type:* int) Optional keyword to define the digit number for flux values printed into .out files (by default 3).
- **[periode_sauvegarde_securite_en_heures]** (*type:* double) To change the default period (23 hours) between the save of the fields in .sauv file.
- **[no_check_disk_space]** (*type:* flag) To disable the check of the available amount of disk space during the calculation.
- **[disable_progress]** (*type:* flag) To disable the writing of the .progress file.
- **[disable_dt_ev]** (*type:* flag) To disable the writing of the .dt_ev file.
- **[gnuplot_header]** (*type:* int) Optional keyword to modify the header of the .out files. Allows to use the column title instead of columns number.

---

runge_kutta_ordre_4_classique

Inherits from: *schema_temps_base*

This is a classical Runge-Kutta scheme of fourth order that uses 4 integration points.

Parameters are:

- **[tinit]** (*type:* double) Value of initial calculation time (0 by default).
- **[tmax]** (*type:* double) Time during which the calculation will be stopped (1e30s by default).
- **[tcpumax]** (*type:* double) CPU time limit (must be specified in hours) for which the calculation is stopped (1e30s by default).
- **[dt_min]** (*type:* double) Minimum calculation time step (1e-16s by default).
- **[dt_max]** (*type:* string) Maximum calculation time step as function of time (1e30s by default).
- **[dt_sauv]** (*type:* double) Save time step value (1e30s by default). Every dt_sauv, fields are saved in the .sauv file. The file contains all the information saved over time. If this instruction is not entered, results are saved only upon calculation completion. To disable the writing of the .sauv files, you must specify 0. Note that dt_sauv is in terms of physical time (not cpu time).
- **[dt_impr]** (*type:* double) Scheme parameter printing time step in time (1e30s by default). The time steps and the flux balances are printed (incorporated onto every side of processed domains) into the .out file.
- **[facsec]** (*type:* string) Value assigned to the safety factor for the time step (1. by default). It can also be a function of time. The time step calculated is multiplied by the safety factor. The first thing to try when a calculation does not converge with an explicit time scheme is to reduce the facsec to 0.5. Warning: Some schemes needs a facsec lower than 1 (0.5 is a good start), for example Schema_Adams_Bashforth_order_3.
- **[seuil_statio]** (*type:* double) Value of the convergence threshold (1e-12 by default). Problems using this type of time scheme converge when the derivatives dGi/dt NL1 of all the unknown transported values Gi have a combined absolute value less than this value. This is the keyword used to set the permanent rating threshold.
- **[residuals]** (*type:* *residuals*) To specify how the residuals will be computed (default max norm, possible to choose L2-norm instead).

- **[diffusion_implicite]** (*type:* int) Keyword to make the diffusive term in the Navier-Stokes equations implicit (in this case, it should be set to 1). The stability time step is then only based on the convection time step (dt=facsec*dt_convection). Thus, in some circumstances, an important gain is achieved with respect to the time step (large diffusion with respect to convection on tightened meshes). Caution: It is however recommended that the user avoids exceeding the convection time step by selecting a too large facsec value. Start with a facsec value of 1 and then increase it gradually if you wish to accelerate calculation. In addition, for a natural convection calculation with a zero initial velocity, in the first time step, the convection time is infinite and therefore dt=facsec*dt_max.
- **[seuil_diffusion_implicite]** (*type:* double) This keyword changes the default value (1e-6) of convergency criteria for the resolution by conjugate gradient used for implicit diffusion.
- **[impr_diffusion_implicite]** (*type:* int) Unactivate (default) or not the printing of the convergence during the resolution of the conjugate gradient.
- **[impr_extremums]** (*type:* int) Print unknowns extremas
- **[no_error_if_not_converged_diffusion_implicite]** (*type:* int) not_set
- **[no_conv_subiteration_diffusion_implicite]** (*type:* int) not_set
- **[dt_start]** (*type: dt_start*) dt_start dt_min : the first iteration is based on dt_min. dt_start dt_calc : the time step at first iteration is calculated in agreement with CFL condition. dt_start dt_fixe value : the first time step is fixed by the user (recommended when resuming calculation with Crank Nicholson temporal scheme to ensure continuity). By default, the first iteration is based on dt_calc.
- **[nb_pas_dt_max]** (*type:* int) Maximum number of calculation time steps (1e9 by default).
- **[niter_max_diffusion_implicite]** (*type:* int) This keyword changes the default value (number of unknowns) of the maximal iterations number in the conjugate gradient method used for implicit diffusion.
- **[precision_impr]** (*type:* int) Optional keyword to define the digit number for flux values printed into .out files (by default 3).
- **[periode_sauvegarde_securite_en_heures]** (*type:* double) To change the default period (23 hours) between the save of the fields in .sauv file.
- **[no_check_disk_space]** (*type:* flag) To disable the check of the available amount of disk space during the calculation.
- **[disable_progress]** (*type:* flag) To disable the writing of the .progress file.
- **[disable_dt_ev]** (*type:* flag) To disable the writing of the .dt_ev file.
- **[gnuplot_header]** (*type:* int) Optional keyword to modify the header of the .out files. Allows to use the column title instead of columns number.

---

runge_kutta_rationnel_ordre_2

Inherits from: *schema_temps_base*

This is the Runge-Kutta rational scheme of second order. The method is described in the note: Wambeck - Rational Runge-Kutta methods for solving systems of ordinary differential equations, at the link: https://link.springer.com/article/10.1007/BF02252381. Although rational methods require more computational work than linear ones, they can have some other properties, such as a stable behaviour with explicitness, which make them preferable. The CFD application of this RRK2 scheme is described in the note: https://link.springer.com/content/pdf/10.1007%2F3-540-13917-6_112.pdf.

Parameters are:

- **[tinit]** (*type:* double) Value of initial calculation time (0 by default).
- **[tmax]** (*type:* double) Time during which the calculation will be stopped (1e30s by default).
- **[tcpumax]** (*type:* double) CPU time limit (must be specified in hours) for which the calculation is stopped (1e30s by default).
- **[dt_min]** (*type:* double) Minimum calculation time step (1e-16s by default).
- **[dt_max]** (*type:* string) Maximum calculation time step as function of time (1e30s by default).
- **[dt_sauv]** (*type:* double) Save time step value (1e30s by default). Every dt_sauv, fields are saved in the .sauv file. The file contains all the information saved over time. If this instruction is not entered, results are saved only upon calculation completion. To disable the writing of the .sauv files, you must specify 0. Note that dt_sauv is in terms of physical time (not cpu time).
- **[dt_impr]** (*type:* double) Scheme parameter printing time step in time (1e30s by default). The time steps and the flux balances are printed (incorporated onto every side of processed domains) into the .out file.
- **[facsec]** (*type:* string) Value assigned to the safety factor for the time step (1. by default). It can also be a function of time. The time step calculated is multiplied by the safety factor. The first thing to try when a calculation does not converge with an explicit time scheme is to reduce the facsec to 0.5. Warning: Some schemes needs a facsec lower than 1 (0.5 is a good start), for example Schema_Adams_Bashforth_order_3.
- **[seuil_statio]** (*type:* double) Value of the convergence threshold (1e-12 by default). Problems using this type of time scheme converge when the derivatives dGi/dt NL1 of all the unknown transported values Gi have a combined absolute value less than this value. This is the keyword used to set the permanent rating threshold.
- **[residuals]** (*type: residuals*) To specify how the residuals will be computed (default max norm, possible to choose L2-norm instead).
- **[diffusion_implicite]** (*type:* int) Keyword to make the diffusive term in the Navier-Stokes equations implicit (in this case, it should be set to 1). The stability time step is then only based on the convection time step (dt=facsec*dt_convection). Thus, in some circumstances, an important gain is achieved with respect to the time step (large diffusion with respect to convection on tightened meshes). Caution: It is however recommended that the user avoids exceeding the convection time step by selecting a too large facsec value. Start with a facsec value of 1 and then increase it gradually if you wish to accelerate calculation. In addition, for a natural convection calculation with a zero initial velocity, in the first time step, the convection time is infinite and therefore dt=facsec*dt_max.
- **[seuil_diffusion_implicite]** (*type:* double) This keyword changes the default value (1e-6) of convergency criteria for the resolution by conjugate gradient used for implicit diffusion.
- **[impr_diffusion_implicite]** (*type:* int) Unactivate (default) or not the printing of the convergence during the resolution of the conjugate gradient.
- **[impr_extremums]** (*type:* int) Print unknowns extremas
- **[no_error_if_not_converged_diffusion_implicite]** (*type:* int) not_set
- **[no_conv_subiteration_diffusion_implicite]** (*type:* int) not_set
- **[dt_start]** (*type: dt_start*) dt_start dt_min : the first iteration is based on dt_min. dt_start dt_calc : the time step at first iteration is calculated in agreement with CFL condition. dt_start dt_fixe value : the first time step is fixed by the user (recommended when resuming calculation with Crank Nicholson temporal scheme to ensure continuity). By default, the first iteration is based on dt_calc.
- **[nb_pas_dt_max]** (*type:* int) Maximum number of calculation time steps (1e9 by default).
- **[niter_max_diffusion_implicite]** (*type:* int) This keyword changes the default value (number of unknowns) of the maximal iterations number in the conjugate gradient method used for implicit diffusion.

- **[precision_impr]** (*type:* int) Optional keyword to define the digit number for flux values printed into .out files (by default 3).
- **[periode_sauvegarde_securite_en_heures]** (*type:* double) To change the default period (23 hours) between the save of the fields in .sauv file.
- **[no_check_disk_space]** (*type:* flag) To disable the check of the available amount of disk space during the calculation.
- **[disable_progress]** (*type:* flag) To disable the writing of the .progress file.
- **[disable_dt_ev]** (*type:* flag) To disable the writing of the .dt_ev file.
- **[gnuplot_header]** (*type:* int) Optional keyword to modify the header of the .out files. Allows to use the column title instead of columns number.

---

sch_cn_ex_iteratif

Inherits from: *schema_temps_base*

This keyword also describes a Crank-Nicholson method of second order accuracy but here, for scalars, because of instablities encountered when dt>dt_CFL, the Crank Nicholson scheme is not applied to scalar quantities. Scalars are treated according to Euler-Explicite scheme at the end of the CN treatment for velocity flow fields (by doing p Euler explicite under-iterations at dt<=dt_CFL). Parameters are the sames (but default values may change) compare to the Sch_CN_iterative scheme plus a relaxation keyword: niter_min (2 by default), niter_max (6 by default), niter_avg (3 by default), facsec_max (20 by default), seuil (0.05 by default)

Parameters are:

- **[omega]** (*type:* double) relaxation factor (0.1 by default)
- **[niter_min]** (*type:* int) minimal number of p-iterations to satisfy convergence criteria (2 by default)
- **[niter_max]** (*type:* int) number of maximum p-iterations allowed to satisfy convergence criteria (6 by default)
- **[niter_avg]** (*type:* int) threshold of p-iterations (3 by default). If the number of p-iterations is greater than niter_avg, facsec is reduced, if lesser than niter_avg, facsec is increased (but limited by the facsec_max value).
- **[facsec_max]** (*type:* double) maximum ratio allowed between dynamical time step returned by iterative process and stability time returned by CFL condition (2 by default).
- **[seuil]** (*type:* double) criteria for ending iterative process (Max(|| u(p) - u(p-1)||/Max || u(p) ||) < seuil) (0.001 by default)
- **[tinit]** (*type:* double) Value of initial calculation time (0 by default).
- **[tmax]** (*type:* double) Time during which the calculation will be stopped (1e30s by default).
- **[tcpumax]** (*type:* double) CPU time limit (must be specified in hours) for which the calculation is stopped (1e30s by default).
- **[dt_min]** (*type:* double) Minimum calculation time step (1e-16s by default).
- **[dt_max]** (*type:* string) Maximum calculation time step as function of time (1e30s by default).
- **[dt_sauv]** (*type:* double) Save time step value (1e30s by default). Every dt_sauv, fields are saved in the .sauv file. The file contains all the information saved over time. If this instruction is not entered, results are saved only upon calculation completion. To disable the writing of the .sauv files, you must specify 0. Note that dt_sauv is in terms of physical time (not cpu time).
- **[dt_impr]** (*type:* double) Scheme parameter printing time step in time (1e30s by default). The time steps and the flux balances are printed (incorporated onto every side of processed domains) into the .out file.

- **[facsec]** (*type:* string) Value assigned to the safety factor for the time step (1. by default). It can also be a function of time. The time step calculated is multiplied by the safety factor. The first thing to try when a calculation does not converge with an explicit time scheme is to reduce the facsec to 0.5. Warning: Some schemes needs a facsec lower than 1 (0.5 is a good start), for example Schema_Adams_Bashforth_order_3.
- **[seuil_statio]** (*type:* double) Value of the convergence threshold (1e-12 by default). Problems using this type of time scheme converge when the derivatives dGi/dt NL1 of all the unknown transported values Gi have a combined absolute value less than this value. This is the keyword used to set the permanent rating threshold.
- **[residuals]** (*type: residuals*) To specify how the residuals will be computed (default max norm, possible to choose L2-norm instead).
- **[diffusion_implicite]** (*type:* int) Keyword to make the diffusive term in the Navier-Stokes equations implicit (in this case, it should be set to 1). The stability time step is then only based on the convection time step (dt=facsec*dt_convection). Thus, in some circumstances, an important gain is achieved with respect to the time step (large diffusion with respect to convection on tightened meshes). Caution: It is however recommended that the user avoids exceeding the convection time step by selecting a too large facsec value. Start with a facsec value of 1 and then increase it gradually if you wish to accelerate calculation. In addition, for a natural convection calculation with a zero initial velocity, in the first time step, the convection time is infinite and therefore dt=facsec*dt_max.
- **[seuil_diffusion_implicite]** (*type:* double) This keyword changes the default value (1e-6) of convergency criteria for the resolution by conjugate gradient used for implicit diffusion.
- **[impr_diffusion_implicite]** (*type:* int) Unactivate (default) or not the printing of the convergence during the resolution of the conjugate gradient.
- **[impr_extremums]** (*type:* int) Print unknowns extremas
- **[no_error_if_not_converged_diffusion_implicite]** (*type:* int) not_set
- **[no_conv_subiteration_diffusion_implicite]** (*type:* int) not_set
- **[dt_start]** (*type: dt_start*) dt_start dt_min : the first iteration is based on dt_min. dt_start dt_calc : the time step at first iteration is calculated in agreement with CFL condition. dt_start dt_fixe value : the first time step is fixed by the user (recommended when resuming calculation with Crank Nicholson temporal scheme to ensure continuity). By default, the first iteration is based on dt_calc.
- **[nb_pas_dt_max]** (*type:* int) Maximum number of calculation time steps (1e9 by default).
- **[niter_max_diffusion_implicite]** (*type:* int) This keyword changes the default value (number of unknowns) of the maximal iterations number in the conjugate gradient method used for implicit diffusion.
- **[precision_impr]** (*type:* int) Optional keyword to define the digit number for flux values printed into .out files (by default 3).
- **[periode_sauvegarde_securite_en_heures]** (*type:* double) To change the default period (23 hours) between the save of the fields in .sauv file.
- **[no_check_disk_space]** (*type:* flag) To disable the check of the available amount of disk space during the calculation.
- **[disable_progress]** (*type:* flag) To disable the writing of the .progress file.
- **[disable_dt_ev]** (*type:* flag) To disable the writing of the .dt_ev file.
- **[gnuplot_header]** (*type:* int) Optional keyword to modify the header of the .out files. Allows to use the column title instead of columns number.

sch_cn_iteratif

Inherits from: *schema_temps_base*

The Crank-Nicholson method of second order accuracy. A mid-point rule formulation is used (Euler-centered scheme). The basic scheme is: $$u(t+1) = u(t) + du/dt(t+1/2)*dt$$ The estimation of the time derivative du/dt at the level (t+1/2) is obtained either by iterative process. The time derivative du/dt at the level (t+1/2) is calculated iteratively with a simple under-relaxations method. Since the method is implicit, neither the cfl nor the fourier stability criteria must be respected. The time step is calculated in a way that the iterative procedure converges with the less iterations as possible.

Remark : for stationary or RANS calculations, no limitation can be given for time step through high value of facsec_max parameter (for instance : facsec_max 1000). In counterpart, for LES calculations, high values of facsec_max may engender numerical instabilities.

Parameters are:

- **[niter_min]** (*type:* int) minimal number of p-iterations to satisfy convergence criteria (2 by default)
- **[niter_max]** (*type:* int) number of maximum p-iterations allowed to satisfy convergence criteria (6 by default)
- **[niter_avg]** (*type:* int) threshold of p-iterations (3 by default). If the number of p-iterations is greater than niter_avg, facsec is reduced, if lesser than niter_avg, facsec is increased (but limited by the facsec_max value).
- **[facsec_max]** (*type:* double) maximum ratio allowed between dynamical time step returned by iterative process and stability time returned by CFL condition (2 by default).
- **[seuil]** (*type:* double) criteria for ending iterative process (Max(|| u(p) - u(p-1)||/Max || u(p) ||) < seuil) (0.001 by default)
- **[tinit]** (*type:* double) Value of initial calculation time (0 by default).
- **[tmax]** (*type:* double) Time during which the calculation will be stopped (1e30s by default).
- **[tcpumax]** (*type:* double) CPU time limit (must be specified in hours) for which the calculation is stopped (1e30s by default).
- **[dt_min]** (*type:* double) Minimum calculation time step (1e-16s by default).
- **[dt_max]** (*type:* string) Maximum calculation time step as function of time (1e30s by default).
- **[dt_sauv]** (*type:* double) Save time step value (1e30s by default). Every dt_sauv, fields are saved in the .sauv file. The file contains all the information saved over time. If this instruction is not entered, results are saved only upon calculation completion. To disable the writing of the .sauv files, you must specify 0. Note that dt_sauv is in terms of physical time (not cpu time).
- **[dt_impr]** (*type:* double) Scheme parameter printing time step in time (1e30s by default). The time steps and the flux balances are printed (incorporated onto every side of processed domains) into the .out file.
- **[facsec]** (*type:* string) Value assigned to the safety factor for the time step (1. by default). It can also be a function of time. The time step calculated is multiplied by the safety factor. The first thing to try when a calculation does not converge with an explicit time scheme is to reduce the facsec to 0.5. Warning: Some schemes needs a facsec lower than 1 (0.5 is a good start), for example Schema_Adams_Bashforth_order_3.
- **[seuil_statio]** (*type:* double) Value of the convergence threshold (1e-12 by default). Problems using this type of time scheme converge when the derivatives dGi/dt NL1 of all the unknown transported values Gi have a combined absolute value less than this value. This is the keyword used to set the permanent rating threshold.
- **[residuals]** (*type: residuals*) To specify how the residuals will be computed (default max norm, possible to choose L2-norm instead).
- **[diffusion_implicite]** (*type:* int) Keyword to make the diffusive term in the Navier-Stokes equations implicit (in this case, it should be set to 1). The stability time step is then only based on the convection time step (dt=facsec*dt_convection). Thus, in some circumstances, an important gain is achieved with respect to the time

step (large diffusion with respect to convection on tightened meshes). Caution: It is however recommended that the user avoids exceeding the convection time step by selecting a too large facsec value. Start with a facsec value of 1 and then increase it gradually if you wish to accelerate calculation. In addition, for a natural convection calculation with a zero initial velocity, in the first time step, the convection time is infinite and therefore dt=facsec*dt_max.

- **[seuil_diffusion_implicite]** (*type:* double) This keyword changes the default value (1e-6) of convergency criteria for the resolution by conjugate gradient used for implicit diffusion.
- **[impr_diffusion_implicite]** (*type:* int) Unactivate (default) or not the printing of the convergence during the resolution of the conjugate gradient.
- **[impr_extremums]** (*type:* int) Print unknowns extremas
- **[no_error_if_not_converged_diffusion_implicite]** (*type:* int) not_set
- **[no_conv_subiteration_diffusion_implicite]** (*type:* int) not_set
- **[dt_start]** (*type: dt_start*) dt_start dt_min : the first iteration is based on dt_min. dt_start dt_calc : the time step at first iteration is calculated in agreement with CFL condition. dt_start dt_fixe value : the first time step is fixed by the user (recommended when resuming calculation with Crank Nicholson temporal scheme to ensure continuity). By default, the first iteration is based on dt_calc.
- **[nb_pas_dt_max]** (*type:* int) Maximum number of calculation time steps (1e9 by default).
- **[niter_max_diffusion_implicite]** (*type:* int) This keyword changes the default value (number of unknowns) of the maximal iterations number in the conjugate gradient method used for implicit diffusion.
- **[precision_impr]** (*type:* int) Optional keyword to define the digit number for flux values printed into .out files (by default 3).
- **[periode_sauvegarde_securite_en_heures]** (*type:* double) To change the default period (23 hours) between the save of the fields in .sauv file.
- **[no_check_disk_space]** (*type:* flag) To disable the check of the available amount of disk space during the calculation.
- **[disable_progress]** (*type:* flag) To disable the writing of the .progress file.
- **[disable_dt_ev]** (*type:* flag) To disable the writing of the .dt_ev file.
- **[gnuplot_header]** (*type:* int) Optional keyword to modify the header of the .out files. Allows to use the column title instead of columns number.

---

schema_adams_bashforth_order_2

Inherits from: *schema_temps_base*

not_set

Parameters are:

- **[tinit]** (*type:* double) Value of initial calculation time (0 by default).
- **[tmax]** (*type:* double) Time during which the calculation will be stopped (1e30s by default).
- **[tcpumax]** (*type:* double) CPU time limit (must be specified in hours) for which the calculation is stopped (1e30s by default).
- **[dt_min]** (*type:* double) Minimum calculation time step (1e-16s by default).
- **[dt_max]** (*type:* string) Maximum calculation time step as function of time (1e30s by default).

- **[dt_sauv]** (*type:* double) Save time step value (1e30s by default). Every dt_sauv, fields are saved in the .sauv file. The file contains all the information saved over time. If this instruction is not entered, results are saved only upon calculation completion. To disable the writing of the .sauv files, you must specify 0. Note that dt_sauv is in terms of physical time (not cpu time).
- **[dt_impr]** (*type:* double) Scheme parameter printing time step in time (1e30s by default). The time steps and the flux balances are printed (incorporated onto every side of processed domains) into the .out file.
- **[facsec]** (*type:* string) Value assigned to the safety factor for the time step (1. by default). It can also be a function of time. The time step calculated is multiplied by the safety factor. The first thing to try when a calculation does not converge with an explicit time scheme is to reduce the facsec to 0.5. Warning: Some schemes needs a facsec lower than 1 (0.5 is a good start), for example Schema_Adams_Bashforth_order_3.
- **[seuil_statio]** (*type:* double) Value of the convergence threshold (1e-12 by default). Problems using this type of time scheme converge when the derivatives dGi/dt NL1 of all the unknown transported values Gi have a combined absolute value less than this value. This is the keyword used to set the permanent rating threshold.
- **[residuals]** (*type: residuals*) To specify how the residuals will be computed (default max norm, possible to choose L2-norm instead).
- **[diffusion_implicite]** (*type:* int) Keyword to make the diffusive term in the Navier-Stokes equations implicit (in this case, it should be set to 1). The stability time step is then only based on the convection time step (dt=facsec*dt_convection). Thus, in some circumstances, an important gain is achieved with respect to the time step (large diffusion with respect to convection on tightened meshes). Caution: It is however recommended that the user avoids exceeding the convection time step by selecting a too large facsec value. Start with a facsec value of 1 and then increase it gradually if you wish to accelerate calculation. In addition, for a natural convection calculation with a zero initial velocity, in the first time step, the convection time is infinite and therefore dt=facsec*dt_max.
- **[seuil_diffusion_implicite]** (*type:* double) This keyword changes the default value (1e-6) of convergency criteria for the resolution by conjugate gradient used for implicit diffusion.
- **[impr_diffusion_implicite]** (*type:* int) Unactivate (default) or not the printing of the convergence during the resolution of the conjugate gradient.
- **[impr_extremums]** (*type:* int) Print unknowns extremas
- **[no_error_if_not_converged_diffusion_implicite]** (*type:* int) not_set
- **[no_conv_subiteration_diffusion_implicite]** (*type:* int) not_set
- **[dt_start]** (*type: dt_start*) dt_start dt_min : the first iteration is based on dt_min. dt_start dt_calc : the time step at first iteration is calculated in agreement with CFL condition. dt_start dt_fixe value : the first time step is fixed by the user (recommended when resuming calculation with Crank Nicholson temporal scheme to ensure continuity). By default, the first iteration is based on dt_calc.
- **[nb_pas_dt_max]** (*type:* int) Maximum number of calculation time steps (1e9 by default).
- **[niter_max_diffusion_implicite]** (*type:* int) This keyword changes the default value (number of unknowns) of the maximal iterations number in the conjugate gradient method used for implicit diffusion.
- **[precision_impr]** (*type:* int) Optional keyword to define the digit number for flux values printed into .out files (by default 3).
- **[periode_sauvegarde_securite_en_heures]** (*type:* double) To change the default period (23 hours) between the save of the fields in .sauv file.
- **[no_check_disk_space]** (*type:* flag) To disable the check of the available amount of disk space during the calculation.
- **[disable_progress]** (*type:* flag) To disable the writing of the .progress file.
- **[disable_dt_ev]** (*type:* flag) To disable the writing of the .dt_ev file.

- **[gnuplot_header]** (*type:* int) Optional keyword to modify the header of the .out files. Allows to use the column title instead of columns number.

---

schema_adams_bashforth_order_3

Inherits from: *schema_temps_base*

not_set

Parameters are:

- **[tinit]** (*type:* double) Value of initial calculation time (0 by default).
- **[tmax]** (*type:* double) Time during which the calculation will be stopped (1e30s by default).
- **[tcpumax]** (*type:* double) CPU time limit (must be specified in hours) for which the calculation is stopped (1e30s by default).
- **[dt_min]** (*type:* double) Minimum calculation time step (1e-16s by default).
- **[dt_max]** (*type:* string) Maximum calculation time step as function of time (1e30s by default).
- **[dt_sauv]** (*type:* double) Save time step value (1e30s by default). Every dt_sauv, fields are saved in the .sauv file. The file contains all the information saved over time. If this instruction is not entered, results are saved only upon calculation completion. To disable the writing of the .sauv files, you must specify 0. Note that dt_sauv is in terms of physical time (not cpu time).
- **[dt_impr]** (*type:* double) Scheme parameter printing time step in time (1e30s by default). The time steps and the flux balances are printed (incorporated onto every side of processed domains) into the .out file.
- **[facsec]** (*type:* string) Value assigned to the safety factor for the time step (1. by default). It can also be a function of time. The time step calculated is multiplied by the safety factor. The first thing to try when a calculation does not converge with an explicit time scheme is to reduce the facsec to 0.5. Warning: Some schemes needs a facsec lower than 1 (0.5 is a good start), for example Schema_Adams_Bashforth_order_3.
- **[seuil_statio]** (*type:* double) Value of the convergence threshold (1e-12 by default). Problems using this type of time scheme converge when the derivatives dGi/dt NL1 of all the unknown transported values Gi have a combined absolute value less than this value. This is the keyword used to set the permanent rating threshold.
- **[residuals]** (*type: residuals*) To specify how the residuals will be computed (default max norm, possible to choose L2-norm instead).
- **[diffusion_implicite]** (*type:* int) Keyword to make the diffusive term in the Navier-Stokes equations implicit (in this case, it should be set to 1). The stability time step is then only based on the convection time step (dt=facsec*dt_convection). Thus, in some circumstances, an important gain is achieved with respect to the time step (large diffusion with respect to convection on tightened meshes). Caution: It is however recommended that the user avoids exceeding the convection time step by selecting a too large facsec value. Start with a facsec value of 1 and then increase it gradually if you wish to accelerate calculation. In addition, for a natural convection calculation with a zero initial velocity, in the first time step, the convection time is infinite and therefore dt=facsec*dt_max.
- **[seuil_diffusion_implicite]** (*type:* double) This keyword changes the default value (1e-6) of convergency criteria for the resolution by conjugate gradient used for implicit diffusion.
- **[impr_diffusion_implicite]** (*type:* int) Unactivate (default) or not the printing of the convergence during the resolution of the conjugate gradient.
- **[impr_extremums]** (*type:* int) Print unknowns extremas
- **[no_error_if_not_converged_diffusion_implicite]** (*type:* int) not_set

- **[no_conv_subiteration_diffusion_implicite]** (*type:* int) not_set
- **[dt_start]** (*type: dt_start*) dt_start dt_min : the first iteration is based on dt_min. dt_start dt_calc : the time step at first iteration is calculated in agreement with CFL condition. dt_start dt_fixe value : the first time step is fixed by the user (recommended when resuming calculation with Crank Nicholson temporal scheme to ensure continuity). By default, the first iteration is based on dt_calc.
- **[nb_pas_dt_max]** (*type:* int) Maximum number of calculation time steps (1e9 by default).
- **[niter_max_diffusion_implicite]** (*type:* int) This keyword changes the default value (number of unknowns) of the maximal iterations number in the conjugate gradient method used for implicit diffusion.
- **[precision_impr]** (*type:* int) Optional keyword to define the digit number for flux values printed into .out files (by default 3).
- **[periode_sauvegarde_securite_en_heures]** (*type:* double) To change the default period (23 hours) between the save of the fields in .sauv file.
- **[no_check_disk_space]** (*type:* flag) To disable the check of the available amount of disk space during the calculation.
- **[disable_progress]** (*type:* flag) To disable the writing of the .progress file.
- **[disable_dt_ev]** (*type:* flag) To disable the writing of the .dt_ev file.
- **[gnuplot_header]** (*type:* int) Optional keyword to modify the header of the .out files. Allows to use the column title instead of columns number.

---

schema_adams_moulton_order_2

Inherits from: *schema_temps_base*

not_set

Parameters are:

- **[facsec_max]** (*type:* double) Maximum ratio allowed between time step and stability time returned by CFL condition. The initial ratio given by facsec keyword is changed during the calculation with the implicit scheme but it couldn't be higher than facsec_max value.NL2 Warning: Some implicit schemes do not permit high facsec_max, example Schema_Adams_Moulton_order_3 needs facsec=facsec_max=1. Advice:NL2 The calculation may start with a facsec specified by the user and increased by the algorithm up to the facsec_max limit. But the user can also choose to specify a constant facsec (facsec_max will be set to facsec value then). Faster convergence has been seen and depends on the kind of calculation: NL2-Hydraulic only or thermal hydraulic with forced convection and low coupling between velocity and temperature (Boussinesq value beta low), facsec between 20-30NL2-Thermal hydraulic with forced convection and strong coupling between velocity and temperature (Boussinesq value beta high), facsec between 90-100 NL2-Thermohydralic with natural convection, facsec around 300NL2 -Conduction only, facsec can be set to a very high value (1e8) as if the scheme was unconditionally stableNL2These values can also be used as rule of thumb for initial facsec with a facsec_max limit higher.
- **[max_iter_implicite]** (*type:* int) Maximum number of iterations allowed for the solver (by default 200).
- **solveur** (*type: solveur_implicite_base*) This keyword is used to designate the solver selected in the situation where the time scheme is an implicit scheme. solver is the name of the solver that allows equation diffusion and convection operators to be set as implicit terms. Keywords corresponding to this functionality are Simple (SIMPLE type algorithm), Simpler (SIMPLER type algorithm) for incompressible systems, Piso (Pressure Implicit with Split Operator), and Implicite (similar to PISO, but as it looks like a simplified solver, it will use fewer timesteps, and ICE (for PB_multiphase). But it may run faster because the pressure matrix is not re-assembled and thus provides CPU gains. Advice: Since the 1.6.0 version, we recommend to use first the Implicite or Simple,

then Piso, and at least Simpler. Because the two first give a fastest convergence (several times) than Piso and the Simpler has not been validated. It seems also than Implicite and Piso schemes give better results than the Simple scheme when the flow is not fully stationary. Thus, if the solution obtained with Simple is not stationary, it is recommended to switch to Piso or Implicite scheme.

- **[tinit]** (*type:* double) Value of initial calculation time (0 by default).
- **[tmax]** (*type:* double) Time during which the calculation will be stopped (1e30s by default).
- **[tcpumax]** (*type:* double) CPU time limit (must be specified in hours) for which the calculation is stopped (1e30s by default).
- **[dt_min]** (*type:* double) Minimum calculation time step (1e-16s by default).
- **[dt_max]** (*type:* string) Maximum calculation time step as function of time (1e30s by default).
- **[dt_sauv]** (*type:* double) Save time step value (1e30s by default). Every dt_sauv, fields are saved in the .sauv file. The file contains all the information saved over time. If this instruction is not entered, results are saved only upon calculation completion. To disable the writing of the .sauv files, you must specify 0. Note that dt_sauv is in terms of physical time (not cpu time).
- **[dt_impr]** (*type:* double) Scheme parameter printing time step in time (1e30s by default). The time steps and the flux balances are printed (incorporated onto every side of processed domains) into the .out file.
- **[facsec]** (*type:* string) Value assigned to the safety factor for the time step (1. by default). It can also be a function of time. The time step calculated is multiplied by the safety factor. The first thing to try when a calculation does not converge with an explicit time scheme is to reduce the facsec to 0.5. Warning: Some schemes needs a facsec lower than 1 (0.5 is a good start), for example Schema_Adams_Bashforth_order_3.
- **[seuil_statio]** (*type:* double) Value of the convergence threshold (1e-12 by default). Problems using this type of time scheme converge when the derivatives dGi/dt NL1 of all the unknown transported values Gi have a combined absolute value less than this value. This is the keyword used to set the permanent rating threshold.
- **[residuals]** (*type: residuals*) To specify how the residuals will be computed (default max norm, possible to choose L2-norm instead).
- **[diffusion_implicite]** (*type:* int) Keyword to make the diffusive term in the Navier-Stokes equations implicit (in this case, it should be set to 1). The stability time step is then only based on the convection time step (dt=facsec*dt_convection). Thus, in some circumstances, an important gain is achieved with respect to the time step (large diffusion with respect to convection on tightened meshes). Caution: It is however recommended that the user avoids exceeding the convection time step by selecting a too large facsec value. Start with a facsec value of 1 and then increase it gradually if you wish to accelerate calculation. In addition, for a natural convection calculation with a zero initial velocity, in the first time step, the convection time is infinite and therefore dt=facsec*dt_max.
- **[seuil_diffusion_implicite]** (*type:* double) This keyword changes the default value (1e-6) of convergency criteria for the resolution by conjugate gradient used for implicit diffusion.
- **[impr_diffusion_implicite]** (*type:* int) Unactivate (default) or not the printing of the convergence during the resolution of the conjugate gradient.
- **[impr_extremums]** (*type:* int) Print unknowns extremas
- **[no_error_if_not_converged_diffusion_implicite]** (*type:* int) not_set
- **[no_conv_subiteration_diffusion_implicite]** (*type:* int) not_set
- **[dt_start]** (*type: dt_start*) dt_start dt_min : the first iteration is based on dt_min. dt_start dt_calc : the time step at first iteration is calculated in agreement with CFL condition. dt_start dt_fixe value : the first time step is fixed by the user (recommended when resuming calculation with Crank Nicholson temporal scheme to ensure continuity). By default, the first iteration is based on dt_calc.
- **[nb_pas_dt_max]** (*type:* int) Maximum number of calculation time steps (1e9 by default).

- **[niter_max_diffusion_implicite]** (*type:* int) This keyword changes the default value (number of unknowns) of the maximal iterations number in the conjugate gradient method used for implicit diffusion.
- **[precision_impr]** (*type:* int) Optional keyword to define the digit number for flux values printed into .out files (by default 3).
- **[periode_sauvegarde_securite_en_heures]** (*type:* double) To change the default period (23 hours) between the save of the fields in .sauv file.
- **[no_check_disk_space]** (*type:* flag) To disable the check of the available amount of disk space during the calculation.
- **[disable_progress]** (*type:* flag) To disable the writing of the .progress file.
- **[disable_dt_ev]** (*type:* flag) To disable the writing of the .dt_ev file.
- **[gnuplot_header]** (*type:* int) Optional keyword to modify the header of the .out files. Allows to use the column title instead of columns number.

---

schema_adams_moulton_order_3

Inherits from: *schema_temps_base*

not_set

Parameters are:

- **[facsec_max]** (*type:* double) Maximum ratio allowed between time step and stability time returned by CFL condition. The initial ratio given by facsec keyword is changed during the calculation with the implicit scheme but it couldn't be higher than facsec_max value.NL2 Warning: Some implicit schemes do not permit high facsec_max, example Schema_Adams_Moulton_order_3 needs facsec=facsec_max=1. Advice:NL2 The calculation may start with a facsec specified by the user and increased by the algorithm up to the facsec_max limit. But the user can also choose to specify a constant facsec (facsec_max will be set to facsec value then). Faster convergence has been seen and depends on the kind of calculation: NL2-Hydraulic only or thermal hydraulic with forced convection and low coupling between velocity and temperature (Boussinesq value beta low), facsec between 20-30NL2-Thermal hydraulic with forced convection and strong coupling between velocity and temperature (Boussinesq value beta high), facsec between 90-100 NL2-Thermohydralic with natural convection, facsec around 300NL2 -Conduction only, facsec can be set to a very high value (1e8) as if the scheme was unconditionally stableNL2These values can also be used as rule of thumb for initial facsec with a facsec_max limit higher.
- **[max_iter_implicite]** (*type:* int) Maximum number of iterations allowed for the solver (by default 200).
- **solveur** (*type:* *solveur_implicite_base*) This keyword is used to designate the solver selected in the situation where the time scheme is an implicit scheme. solver is the name of the solver that allows equation diffusion and convection operators to be set as implicit terms. Keywords corresponding to this functionality are Simple (SIMPLE type algorithm), Simpler (SIMPLER type algorithm) for incompressible systems, Piso (Pressure Implicit with Split Operator), and Implicite (similar to PISO, but as it looks like a simplified solver, it will use fewer timesteps, and ICE (for PB_multiphase). But it may run faster because the pressure matrix is not re-assembled and thus provides CPU gains. Advice: Since the 1.6.0 version, we recommend to use first the Implicite or Simple, then Piso, and at least Simpler. Because the two first give a fastest convergence (several times) than Piso and the Simpler has not been validated. It seems also than Implicite and Piso schemes give better results than the Simple scheme when the flow is not fully stationary. Thus, if the solution obtained with Simple is not stationary, it is recommended to switch to Piso or Implicite scheme.
- **[tinit]** (*type:* double) Value of initial calculation time (0 by default).
- **[tmax]** (*type:* double) Time during which the calculation will be stopped (1e30s by default).

- **[tcpumax]** (*type:* double) CPU time limit (must be specified in hours) for which the calculation is stopped (1e30s by default).
- **[dt_min]** (*type:* double) Minimum calculation time step (1e-16s by default).
- **[dt_max]** (*type:* string) Maximum calculation time step as function of time (1e30s by default).
- **[dt_sauv]** (*type:* double) Save time step value (1e30s by default). Every dt_sauv, fields are saved in the .sauv file. The file contains all the information saved over time. If this instruction is not entered, results are saved only upon calculation completion. To disable the writing of the .sauv files, you must specify 0. Note that dt_sauv is in terms of physical time (not cpu time).
- **[dt_impr]** (*type:* double) Scheme parameter printing time step in time (1e30s by default). The time steps and the flux balances are printed (incorporated onto every side of processed domains) into the .out file.
- **[facsec]** (*type:* string) Value assigned to the safety factor for the time step (1. by default). It can also be a function of time. The time step calculated is multiplied by the safety factor. The first thing to try when a calculation does not converge with an explicit time scheme is to reduce the facsec to 0.5. Warning: Some schemes needs a facsec lower than 1 (0.5 is a good start), for example Schema_Adams_Bashforth_order_3.
- **[seuil_statio]** (*type:* double) Value of the convergence threshold (1e-12 by default). Problems using this type of time scheme converge when the derivatives dGi/dt NL1 of all the unknown transported values Gi have a combined absolute value less than this value. This is the keyword used to set the permanent rating threshold.
- **[residuals]** (*type: residuals*) To specify how the residuals will be computed (default max norm, possible to choose L2-norm instead).
- **[diffusion_implicite]** (*type:* int) Keyword to make the diffusive term in the Navier-Stokes equations implicit (in this case, it should be set to 1). The stability time step is then only based on the convection time step (dt=facsec*dt_convection). Thus, in some circumstances, an important gain is achieved with respect to the time step (large diffusion with respect to convection on tightened meshes). Caution: It is however recommended that the user avoids exceeding the convection time step by selecting a too large facsec value. Start with a facsec value of 1 and then increase it gradually if you wish to accelerate calculation. In addition, for a natural convection calculation with a zero initial velocity, in the first time step, the convection time is infinite and therefore dt=facsec*dt_max.
- **[seuil_diffusion_implicite]** (*type:* double) This keyword changes the default value (1e-6) of convergency criteria for the resolution by conjugate gradient used for implicit diffusion.
- **[impr_diffusion_implicite]** (*type:* int) Unactivate (default) or not the printing of the convergence during the resolution of the conjugate gradient.
- **[impr_extremums]** (*type:* int) Print unknowns extremas
- **[no_error_if_not_converged_diffusion_implicite]** (*type:* int) not_set
- **[no_conv_subiteration_diffusion_implicite]** (*type:* int) not_set
- **[dt_start]** (*type: dt_start*) dt_start dt_min : the first iteration is based on dt_min. dt_start dt_calc : the time step at first iteration is calculated in agreement with CFL condition. dt_start dt_fixe value : the first time step is fixed by the user (recommended when resuming calculation with Crank Nicholson temporal scheme to ensure continuity). By default, the first iteration is based on dt_calc.
- **[nb_pas_dt_max]** (*type:* int) Maximum number of calculation time steps (1e9 by default).
- **[niter_max_diffusion_implicite]** (*type:* int) This keyword changes the default value (number of unknowns) of the maximal iterations number in the conjugate gradient method used for implicit diffusion.
- **[precision_impr]** (*type:* int) Optional keyword to define the digit number for flux values printed into .out files (by default 3).
- **[periode_sauvegarde_securite_en_heures]** (*type:* double) To change the default period (23 hours) between the save of the fields in .sauv file.

- **[no_check_disk_space]** (*type:* flag) To disable the check of the available amount of disk space during the calculation.
- **[disable_progress]** (*type:* flag) To disable the writing of the .progress file.
- **[disable_dt_ev]** (*type:* flag) To disable the writing of the .dt_ev file.
- **[gnuplot_header]** (*type:* int) Optional keyword to modify the header of the .out files. Allows to use the column title instead of columns number.

---

schema_backward_differentiation_order_2

Inherits from: *schema_temps_base*

not_set

Parameters are:

- **[facsec_max]** (*type:* double) Maximum ratio allowed between time step and stability time returned by CFL condition. The initial ratio given by facsec keyword is changed during the calculation with the implicit scheme but it couldn't be higher than facsec_max value.NL2 Warning: Some implicit schemes do not permit high facsec_max, example Schema_Adams_Moulton_order_3 needs facsec=facsec_max=1. Advice:NL2 The calculation may start with a facsec specified by the user and increased by the algorithm up to the facsec_max limit. But the user can also choose to specify a constant facsec (facsec_max will be set to facsec value then). Faster convergence has been seen and depends on the kind of calculation: NL2-Hydraulic only or thermal hydraulic with forced convection and low coupling between velocity and temperature (Boussinesq value beta low), facsec between 20-30NL2-Thermal hydraulic with forced convection and strong coupling between velocity and temperature (Boussinesq value beta high), facsec between 90-100 NL2-Thermohydralic with natural convection, facsec around 300NL2 -Conduction only, facsec can be set to a very high value (1e8) as if the scheme was unconditionally stableNL2These values can also be used as rule of thumb for initial facsec with a facsec_max limit higher.
- **[max_iter_implicite]** (*type:* int) Maximum number of iterations allowed for the solver (by default 200).
- **solveur** (*type:* *solveur_implicite_base*) This keyword is used to designate the solver selected in the situation where the time scheme is an implicit scheme. solver is the name of the solver that allows equation diffusion and convection operators to be set as implicit terms. Keywords corresponding to this functionality are Simple (SIMPLE type algorithm), Simpler (SIMPLER type algorithm) for incompressible systems, Piso (Pressure Implicit with Split Operator), and Implicite (similar to PISO, but as it looks like a simplified solver, it will use fewer timesteps, and ICE (for PB_multiphase). But it may run faster because the pressure matrix is not re-assembled and thus provides CPU gains. Advice: Since the 1.6.0 version, we recommend to use first the Implicite or Simple, then Piso, and at least Simpler. Because the two first give a fastest convergence (several times) than Piso and the Simpler has not been validated. It seems also than Implicite and Piso schemes give better results than the Simple scheme when the flow is not fully stationary. Thus, if the solution obtained with Simple is not stationary, it is recommended to switch to Piso or Implicite scheme.
- **[tinit]** (*type:* double) Value of initial calculation time (0 by default).
- **[tmax]** (*type:* double) Time during which the calculation will be stopped (1e30s by default).
- **[tcpumax]** (*type:* double) CPU time limit (must be specified in hours) for which the calculation is stopped (1e30s by default).
- **[dt_min]** (*type:* double) Minimum calculation time step (1e-16s by default).
- **[dt_max]** (*type:* string) Maximum calculation time step as function of time (1e30s by default).

- **[dt_sauv]** (*type:* double) Save time step value (1e30s by default). Every dt_sauv, fields are saved in the .sauv file. The file contains all the information saved over time. If this instruction is not entered, results are saved only upon calculation completion. To disable the writing of the .sauv files, you must specify 0. Note that dt_sauv is in terms of physical time (not cpu time).
- **[dt_impr]** (*type:* double) Scheme parameter printing time step in time (1e30s by default). The time steps and the flux balances are printed (incorporated onto every side of processed domains) into the .out file.
- **[facsec]** (*type:* string) Value assigned to the safety factor for the time step (1. by default). It can also be a function of time. The time step calculated is multiplied by the safety factor. The first thing to try when a calculation does not converge with an explicit time scheme is to reduce the facsec to 0.5. Warning: Some schemes needs a facsec lower than 1 (0.5 is a good start), for example Schema_Adams_Bashforth_order_3.
- **[seuil_statio]** (*type:* double) Value of the convergence threshold (1e-12 by default). Problems using this type of time scheme converge when the derivatives dGi/dt NL1 of all the unknown transported values Gi have a combined absolute value less than this value. This is the keyword used to set the permanent rating threshold.
- **[residuals]** (*type: residuals*) To specify how the residuals will be computed (default max norm, possible to choose L2-norm instead).
- **[diffusion_implicite]** (*type:* int) Keyword to make the diffusive term in the Navier-Stokes equations implicit (in this case, it should be set to 1). The stability time step is then only based on the convection time step (dt=facsec*dt_convection). Thus, in some circumstances, an important gain is achieved with respect to the time step (large diffusion with respect to convection on tightened meshes). Caution: It is however recommended that the user avoids exceeding the convection time step by selecting a too large facsec value. Start with a facsec value of 1 and then increase it gradually if you wish to accelerate calculation. In addition, for a natural convection calculation with a zero initial velocity, in the first time step, the convection time is infinite and therefore dt=facsec*dt_max.
- **[seuil_diffusion_implicite]** (*type:* double) This keyword changes the default value (1e-6) of convergency criteria for the resolution by conjugate gradient used for implicit diffusion.
- **[impr_diffusion_implicite]** (*type:* int) Unactivate (default) or not the printing of the convergence during the resolution of the conjugate gradient.
- **[impr_extremums]** (*type:* int) Print unknowns extremas
- **[no_error_if_not_converged_diffusion_implicite]** (*type:* int) not_set
- **[no_conv_subiteration_diffusion_implicite]** (*type:* int) not_set
- **[dt_start]** (*type: dt_start*) dt_start dt_min : the first iteration is based on dt_min. dt_start dt_calc : the time step at first iteration is calculated in agreement with CFL condition. dt_start dt_fixe value : the first time step is fixed by the user (recommended when resuming calculation with Crank Nicholson temporal scheme to ensure continuity). By default, the first iteration is based on dt_calc.
- **[nb_pas_dt_max]** (*type:* int) Maximum number of calculation time steps (1e9 by default).
- **[niter_max_diffusion_implicite]** (*type:* int) This keyword changes the default value (number of unknowns) of the maximal iterations number in the conjugate gradient method used for implicit diffusion.
- **[precision_impr]** (*type:* int) Optional keyword to define the digit number for flux values printed into .out files (by default 3).
- **[periode_sauvegarde_securite_en_heures]** (*type:* double) To change the default period (23 hours) between the save of the fields in .sauv file.
- **[no_check_disk_space]** (*type:* flag) To disable the check of the available amount of disk space during the calculation.
- **[disable_progress]** (*type:* flag) To disable the writing of the .progress file.
- **[disable_dt_ev]** (*type:* flag) To disable the writing of the .dt_ev file.

- **[gnuplot_header]** (*type:* int) Optional keyword to modify the header of the .out files. Allows to use the column title instead of columns number.

---

schema_backward_differentiation_order_3

Inherits from: *schema_temps_base*

not_set

Parameters are:

- **[facsec_max]** (*type:* double) Maximum ratio allowed between time step and stability time returned by CFL condition. The initial ratio given by facsec keyword is changed during the calculation with the implicit scheme but it couldn't be higher than facsec_max value.NL2 Warning: Some implicit schemes do not permit high facsec_max, example Schema_Adams_Moulton_order_3 needs facsec=facsec_max=1. Advice:NL2 The calculation may start with a facsec specified by the user and increased by the algorithm up to the facsec_max limit. But the user can also choose to specify a constant facsec (facsec_max will be set to facsec value then). Faster convergence has been seen and depends on the kind of calculation: NL2-Hydraulic only or thermal hydraulic with forced convection and low coupling between velocity and temperature (Boussinesq value beta low), facsec between 20-30NL2-Thermal hydraulic with forced convection and strong coupling between velocity and temperature (Boussinesq value beta high), facsec between 90-100 NL2-Thermohydralic with natural convection, facsec around 300NL2 -Conduction only, facsec can be set to a very high value (1e8) as if the scheme was unconditionally stableNL2These values can also be used as rule of thumb for initial facsec with a facsec_max limit higher.
- **[max_iter_implicite]** (*type:* int) Maximum number of iterations allowed for the solver (by default 200).
- **solveur** (*type:* *solveur_implicite_base*) This keyword is used to designate the solver selected in the situation where the time scheme is an implicit scheme. solver is the name of the solver that allows equation diffusion and convection operators to be set as implicit terms. Keywords corresponding to this functionality are Simple (SIMPLE type algorithm), Simpler (SIMPLER type algorithm) for incompressible systems, Piso (Pressure Implicit with Split Operator), and Implicite (similar to PISO, but as it looks like a simplified solver, it will use fewer timesteps, and ICE (for PB_multiphase). But it may run faster because the pressure matrix is not re-assembled and thus provides CPU gains. Advice: Since the 1.6.0 version, we recommend to use first the Implicite or Simple, then Piso, and at least Simpler. Because the two first give a fastest convergence (several times) than Piso and the Simpler has not been validated. It seems also than Implicite and Piso schemes give better results than the Simple scheme when the flow is not fully stationary. Thus, if the solution obtained with Simple is not stationary, it is recommended to switch to Piso or Implicite scheme.
- **[tinit]** (*type:* double) Value of initial calculation time (0 by default).
- **[tmax]** (*type:* double) Time during which the calculation will be stopped (1e30s by default).
- **[tcpumax]** (*type:* double) CPU time limit (must be specified in hours) for which the calculation is stopped (1e30s by default).
- **[dt_min]** (*type:* double) Minimum calculation time step (1e-16s by default).
- **[dt_max]** (*type:* string) Maximum calculation time step as function of time (1e30s by default).
- **[dt_sauv]** (*type:* double) Save time step value (1e30s by default). Every dt_sauv, fields are saved in the .sauv file. The file contains all the information saved over time. If this instruction is not entered, results are saved only upon calculation completion. To disable the writing of the .sauv files, you must specify 0. Note that dt_sauv is in terms of physical time (not cpu time).
- **[dt_impr]** (*type:* double) Scheme parameter printing time step in time (1e30s by default). The time steps and the flux balances are printed (incorporated onto every side of processed domains) into the .out file.

- **[facsec]** (*type:* string) Value assigned to the safety factor for the time step (1. by default). It can also be a function of time. The time step calculated is multiplied by the safety factor. The first thing to try when a calculation does not converge with an explicit time scheme is to reduce the facsec to 0.5. Warning: Some schemes needs a facsec lower than 1 (0.5 is a good start), for example Schema_Adams_Bashforth_order_3.
- **[seuil_statio]** (*type:* double) Value of the convergence threshold (1e-12 by default). Problems using this type of time scheme converge when the derivatives dGi/dt NL1 of all the unknown transported values Gi have a combined absolute value less than this value. This is the keyword used to set the permanent rating threshold.
- **[residuals]** (*type: residuals*) To specify how the residuals will be computed (default max norm, possible to choose L2-norm instead).
- **[diffusion_implicite]** (*type:* int) Keyword to make the diffusive term in the Navier-Stokes equations implicit (in this case, it should be set to 1). The stability time step is then only based on the convection time step (dt=facsec*dt_convection). Thus, in some circumstances, an important gain is achieved with respect to the time step (large diffusion with respect to convection on tightened meshes). Caution: It is however recommended that the user avoids exceeding the convection time step by selecting a too large facsec value. Start with a facsec value of 1 and then increase it gradually if you wish to accelerate calculation. In addition, for a natural convection calculation with a zero initial velocity, in the first time step, the convection time is infinite and therefore dt=facsec*dt_max.
- **[seuil_diffusion_implicite]** (*type:* double) This keyword changes the default value (1e-6) of convergency criteria for the resolution by conjugate gradient used for implicit diffusion.
- **[impr_diffusion_implicite]** (*type:* int) Unactivate (default) or not the printing of the convergence during the resolution of the conjugate gradient.
- **[impr_extremums]** (*type:* int) Print unknowns extremas
- **[no_error_if_not_converged_diffusion_implicite]** (*type:* int) not_set
- **[no_conv_subiteration_diffusion_implicite]** (*type:* int) not_set
- **[dt_start]** (*type: dt_start*) dt_start dt_min : the first iteration is based on dt_min. dt_start dt_calc : the time step at first iteration is calculated in agreement with CFL condition. dt_start dt_fixe value : the first time step is fixed by the user (recommended when resuming calculation with Crank Nicholson temporal scheme to ensure continuity). By default, the first iteration is based on dt_calc.
- **[nb_pas_dt_max]** (*type:* int) Maximum number of calculation time steps (1e9 by default).
- **[niter_max_diffusion_implicite]** (*type:* int) This keyword changes the default value (number of unknowns) of the maximal iterations number in the conjugate gradient method used for implicit diffusion.
- **[precision_impr]** (*type:* int) Optional keyword to define the digit number for flux values printed into .out files (by default 3).
- **[periode_sauvegarde_securite_en_heures]** (*type:* double) To change the default period (23 hours) between the save of the fields in .sauv file.
- **[no_check_disk_space]** (*type:* flag) To disable the check of the available amount of disk space during the calculation.
- **[disable_progress]** (*type:* flag) To disable the writing of the .progress file.
- **[disable_dt_ev]** (*type:* flag) To disable the writing of the .dt_ev file.
- **[gnuplot_header]** (*type:* int) Optional keyword to modify the header of the .out files. Allows to use the column title instead of columns number.

schema_euler_implicite

Synonyms: scheme_euler_implicit

Inherits from: *schema_temps_base*

This is the Euler implicit scheme.

Parameters are:

- **[facsec_max]** (*type:* double) For old syntax, see the complete parameters of facsec for details
- **[resolution_monolithique]** (*type: bloc_lecture*) Activate monolithic resolution for coupled problems. Solves together the equations corresponding to the application domains in the given order. All aplication domains of the coupled equations must be given to determine the order of resolution. If the monolithic solving is not wanted for a specific application domain, an underscore can be added as prefix. For example, resolution_monolithique { dom1 { dom2 dom3 } _dom4 } will solve in a single matrix the equations having dom1 as application domain, then the equations having dom2 or dom3 as application domain in a single matrix, then the equations having dom4 as application domain in a sequential way (not in a single matrix).
- **[max_iter_implicite]** (*type:* int) Maximum number of iterations allowed for the solver (by default 200).
- **solveur** (*type: solveur_implicite_base*) This keyword is used to designate the solver selected in the situation where the time scheme is an implicit scheme. solver is the name of the solver that allows equation diffusion and convection operators to be set as implicit terms. Keywords corresponding to this functionality are Simple (SIMPLE type algorithm), Simpler (SIMPLER type algorithm) for incompressible systems, Piso (Pressure Implicit with Split Operator), and Implicite (similar to PISO, but as it looks like a simplified solver, it will use fewer timesteps, and ICE (for PB_multiphase). But it may run faster because the pressure matrix is not re-assembled and thus provides CPU gains. Advice: Since the 1.6.0 version, we recommend to use first the Implicite or Simple, then Piso, and at least Simpler. Because the two first give a fastest convergence (several times) than Piso and the Simpler has not been validated. It seems also than Implicite and Piso schemes give better results than the Simple scheme when the flow is not fully stationary. Thus, if the solution obtained with Simple is not stationary, it is recommended to switch to Piso or Implicite scheme.
- **[tinit]** (*type:* double) Value of initial calculation time (0 by default).
- **[tmax]** (*type:* double) Time during which the calculation will be stopped (1e30s by default).
- **[tcpumax]** (*type:* double) CPU time limit (must be specified in hours) for which the calculation is stopped (1e30s by default).
- **[dt_min]** (*type:* double) Minimum calculation time step (1e-16s by default).
- **[dt_max]** (*type:* string) Maximum calculation time step as function of time (1e30s by default).
- **[dt_sauv]** (*type:* double) Save time step value (1e30s by default). Every dt_sauv, fields are saved in the .sauv file. The file contains all the information saved over time. If this instruction is not entered, results are saved only upon calculation completion. To disable the writing of the .sauv files, you must specify 0. Note that dt_sauv is in terms of physical time (not cpu time).
- **[dt_impr]** (*type:* double) Scheme parameter printing time step in time (1e30s by default). The time steps and the flux balances are printed (incorporated onto every side of processed domains) into the .out file.
- **[facsec]** (*type:* string) Value assigned to the safety factor for the time step (1. by default). It can also be a function of time. The time step calculated is multiplied by the safety factor. The first thing to try when a calculation does not converge with an explicit time scheme is to reduce the facsec to 0.5. Warning: Some schemes needs a facsec lower than 1 (0.5 is a good start), for example Schema_Adams_Bashforth_order_3.
- **[seuil_statio]** (*type:* double) Value of the convergence threshold (1e-12 by default). Problems using this type of time scheme converge when the derivatives dGi/dt NL1 of all the unknown transported values Gi have a combined absolute value less than this value. This is the keyword used to set the permanent rating threshold.

- **[residuals]** (*type: residuals*) To specify how the residuals will be computed (default max norm, possible to choose L2-norm instead).
- **[diffusion_implicite]** (*type:* int) Keyword to make the diffusive term in the Navier-Stokes equations implicit (in this case, it should be set to 1). The stability time step is then only based on the convection time step (dt=facsec*dt_convection). Thus, in some circumstances, an important gain is achieved with respect to the time step (large diffusion with respect to convection on tightened meshes). Caution: It is however recommended that the user avoids exceeding the convection time step by selecting a too large facsec value. Start with a facsec value of 1 and then increase it gradually if you wish to accelerate calculation. In addition, for a natural convection calculation with a zero initial velocity, in the first time step, the convection time is infinite and therefore dt=facsec*dt_max.
- **[seuil_diffusion_implicite]** (*type:* double) This keyword changes the default value (1e-6) of convergency criteria for the resolution by conjugate gradient used for implicit diffusion.
- **[impr_diffusion_implicite]** (*type:* int) Unactivate (default) or not the printing of the convergence during the resolution of the conjugate gradient.
- **[impr_extremums]** (*type:* int) Print unknowns extremas
- **[no_error_if_not_converged_diffusion_implicite]** (*type:* int) not_set
- **[no_conv_subiteration_diffusion_implicite]** (*type:* int) not_set
- **[dt_start]** (*type: dt_start*) dt_start dt_min : the first iteration is based on dt_min. dt_start dt_calc : the time step at first iteration is calculated in agreement with CFL condition. dt_start dt_fixe value : the first time step is fixed by the user (recommended when resuming calculation with Crank Nicholson temporal scheme to ensure continuity). By default, the first iteration is based on dt_calc.
- **[nb_pas_dt_max]** (*type:* int) Maximum number of calculation time steps (1e9 by default).
- **[niter_max_diffusion_implicite]** (*type:* int) This keyword changes the default value (number of unknowns) of the maximal iterations number in the conjugate gradient method used for implicit diffusion.
- **[precision_impr]** (*type:* int) Optional keyword to define the digit number for flux values printed into .out files (by default 3).
- **[periode_sauvegarde_securite_en_heures]** (*type:* double) To change the default period (23 hours) between the save of the fields in .sauv file.
- **[no_check_disk_space]** (*type:* flag) To disable the check of the available amount of disk space during the calculation.
- **[disable_progress]** (*type:* flag) To disable the writing of the .progress file.
- **[disable_dt_ev]** (*type:* flag) To disable the writing of the .dt_ev file.
- **[gnuplot_header]** (*type:* int) Optional keyword to modify the header of the .out files. Allows to use the column title instead of columns number.

schema_implicite_base

Inherits from: *schema_temps_base*

Basic class for implicite time scheme.

Parameters are:

- **[max_iter_implicite]** (*type:* int) Maximum number of iterations allowed for the solver (by default 200).
- **solveur** (*type:* *solveur_implicite_base*) This keyword is used to designate the solver selected in the situation where the time scheme is an implicit scheme. solver is the name of the solver that allows equation diffusion and convection operators to be set as implicit terms. Keywords corresponding to this functionality are Simple (SIMPLE type algorithm), Simpler (SIMPLER type algorithm) for incompressible systems, Piso (Pressure Implicit with Split Operator), and Implicite (similar to PISO, but as it looks like a simplified solver, it will use fewer timesteps, and ICE (for PB_multiphase). But it may run faster because the pressure matrix is not re-assembled and thus provides CPU gains. Advice: Since the 1.6.0 version, we recommend to use first the Implicite or Simple, then Piso, and at least Simpler. Because the two first give a fastest convergence (several times) than Piso and the Simpler has not been validated. It seems also than Implicite and Piso schemes give better results than the Simple scheme when the flow is not fully stationary. Thus, if the solution obtained with Simple is not stationary, it is recommended to switch to Piso or Implicite scheme.
- **[tinit]** (*type:* double) Value of initial calculation time (0 by default).
- **[tmax]** (*type:* double) Time during which the calculation will be stopped (1e30s by default).
- **[tcpumax]** (*type:* double) CPU time limit (must be specified in hours) for which the calculation is stopped (1e30s by default).
- **[dt_min]** (*type:* double) Minimum calculation time step (1e-16s by default).
- **[dt_max]** (*type:* string) Maximum calculation time step as function of time (1e30s by default).
- **[dt_sauv]** (*type:* double) Save time step value (1e30s by default). Every dt_sauv, fields are saved in the .sauv file. The file contains all the information saved over time. If this instruction is not entered, results are saved only upon calculation completion. To disable the writing of the .sauv files, you must specify 0. Note that dt_sauv is in terms of physical time (not cpu time).
- **[dt_impr]** (*type:* double) Scheme parameter printing time step in time (1e30s by default). The time steps and the flux balances are printed (incorporated onto every side of processed domains) into the .out file.
- **[facsec]** (*type:* string) Value assigned to the safety factor for the time step (1. by default). It can also be a function of time. The time step calculated is multiplied by the safety factor. The first thing to try when a calculation does not converge with an explicit time scheme is to reduce the facsec to 0.5. Warning: Some schemes needs a facsec lower than 1 (0.5 is a good start), for example Schema_Adams_Bashforth_order_3.
- **[seuil_statio]** (*type:* double) Value of the convergence threshold (1e-12 by default). Problems using this type of time scheme converge when the derivatives dGi/dt NL1 of all the unknown transported values Gi have a combined absolute value less than this value. This is the keyword used to set the permanent rating threshold.
- **[residuals]** (*type:* *residuals*) To specify how the residuals will be computed (default max norm, possible to choose L2-norm instead).
- **[diffusion_implicite]** (*type:* int) Keyword to make the diffusive term in the Navier-Stokes equations implicit (in this case, it should be set to 1). The stability time step is then only based on the convection time step (dt=facsec*dt_convection). Thus, in some circumstances, an important gain is achieved with respect to the time step (large diffusion with respect to convection on tightened meshes). Caution: It is however recommended that the user avoids exceeding the convection time step by selecting a too large facsec value. Start with a facsec value of 1 and then increase it gradually if you wish to accelerate calculation. In addition, for a natural convection calculation with a zero initial velocity, in the first time step, the convection time is infinite and therefore dt=facsec*dt_max.

- **[seuil_diffusion_implicite]** (*type:* double) This keyword changes the default value (1e-6) of convergency criteria for the resolution by conjugate gradient used for implicit diffusion.
- **[impr_diffusion_implicite]** (*type:* int) Unactivate (default) or not the printing of the convergence during the resolution of the conjugate gradient.
- **[impr_extremums]** (*type:* int) Print unknowns extremas
- **[no_error_if_not_converged_diffusion_implicite]** (*type:* int) not_set
- **[no_conv_subiteration_diffusion_implicite]** (*type:* int) not_set
- **[dt_start]** (*type:* *dt_start*) dt_start dt_min : the first iteration is based on dt_min. dt_start dt_calc : the time step at first iteration is calculated in agreement with CFL condition. dt_start dt_fixe value : the first time step is fixed by the user (recommended when resuming calculation with Crank Nicholson temporal scheme to ensure continuity). By default, the first iteration is based on dt_calc.
- **[nb_pas_dt_max]** (*type:* int) Maximum number of calculation time steps (1e9 by default).
- **[niter_max_diffusion_implicite]** (*type:* int) This keyword changes the default value (number of unknowns) of the maximal iterations number in the conjugate gradient method used for implicit diffusion.
- **[precision_impr]** (*type:* int) Optional keyword to define the digit number for flux values printed into .out files (by default 3).
- **[periode_sauvegarde_securite_en_heures]** (*type:* double) To change the default period (23 hours) between the save of the fields in .sauv file.
- **[no_check_disk_space]** (*type:* flag) To disable the check of the available amount of disk space during the calculation.
- **[disable_progress]** (*type:* flag) To disable the writing of the .progress file.
- **[disable_dt_ev]** (*type:* flag) To disable the writing of the .dt_ev file.
- **[gnuplot_header]** (*type:* int) Optional keyword to modify the header of the .out files. Allows to use the column title instead of columns number.

---

schema_predictor_corrector

Inherits from: *schema_temps_base*

This is the predictor-corrector scheme (second order). It is more accurate and economic than MacCormack scheme. It gives best results with a second ordre convective scheme like quick, centre (VDF).

Parameters are:

- **[tinit]** (*type:* double) Value of initial calculation time (0 by default).
- **[tmax]** (*type:* double) Time during which the calculation will be stopped (1e30s by default).
- **[tcpumax]** (*type:* double) CPU time limit (must be specified in hours) for which the calculation is stopped (1e30s by default).
- **[dt_min]** (*type:* double) Minimum calculation time step (1e-16s by default).
- **[dt_max]** (*type:* string) Maximum calculation time step as function of time (1e30s by default).
- **[dt_sauv]** (*type:* double) Save time step value (1e30s by default). Every dt_sauv, fields are saved in the .sauv file. The file contains all the information saved over time. If this instruction is not entered, results are saved only upon calculation completion. To disable the writing of the .sauv files, you must specify 0. Note that dt_sauv is in terms of physical time (not cpu time).

- **[dt_impr]** (*type:* double) Scheme parameter printing time step in time (1e30s by default). The time steps and the flux balances are printed (incorporated onto every side of processed domains) into the .out file.
- **[facsec]** (*type:* string) Value assigned to the safety factor for the time step (1. by default). It can also be a function of time. The time step calculated is multiplied by the safety factor. The first thing to try when a calculation does not converge with an explicit time scheme is to reduce the facsec to 0.5. Warning: Some schemes needs a facsec lower than 1 (0.5 is a good start), for example Schema_Adams_Bashforth_order_3.
- **[seuil_statio]** (*type:* double) Value of the convergence threshold (1e-12 by default). Problems using this type of time scheme converge when the derivatives dGi/dt NL1 of all the unknown transported values Gi have a combined absolute value less than this value. This is the keyword used to set the permanent rating threshold.
- **[residuals]** (*type: residuals*) To specify how the residuals will be computed (default max norm, possible to choose L2-norm instead).
- **[diffusion_implicite]** (*type:* int) Keyword to make the diffusive term in the Navier-Stokes equations implicit (in this case, it should be set to 1). The stability time step is then only based on the convection time step (dt=facsec*dt_convection). Thus, in some circumstances, an important gain is achieved with respect to the time step (large diffusion with respect to convection on tightened meshes). Caution: It is however recommended that the user avoids exceeding the convection time step by selecting a too large facsec value. Start with a facsec value of 1 and then increase it gradually if you wish to accelerate calculation. In addition, for a natural convection calculation with a zero initial velocity, in the first time step, the convection time is infinite and therefore dt=facsec*dt_max.
- **[seuil_diffusion_implicite]** (*type:* double) This keyword changes the default value (1e-6) of convergency criteria for the resolution by conjugate gradient used for implicit diffusion.
- **[impr_diffusion_implicite]** (*type:* int) Unactivate (default) or not the printing of the convergence during the resolution of the conjugate gradient.
- **[impr_extremums]** (*type:* int) Print unknowns extremas
- **[no_error_if_not_converged_diffusion_implicite]** (*type:* int) not_set
- **[no_conv_subiteration_diffusion_implicite]** (*type:* int) not_set
- **[dt_start]** (*type: dt_start*) dt_start dt_min : the first iteration is based on dt_min. dt_start dt_calc : the time step at first iteration is calculated in agreement with CFL condition. dt_start dt_fixe value : the first time step is fixed by the user (recommended when resuming calculation with Crank Nicholson temporal scheme to ensure continuity). By default, the first iteration is based on dt_calc.
- **[nb_pas_dt_max]** (*type:* int) Maximum number of calculation time steps (1e9 by default).
- **[niter_max_diffusion_implicite]** (*type:* int) This keyword changes the default value (number of unknowns) of the maximal iterations number in the conjugate gradient method used for implicit diffusion.
- **[precision_impr]** (*type:* int) Optional keyword to define the digit number for flux values printed into .out files (by default 3).
- **[periode_sauvegarde_securite_en_heures]** (*type:* double) To change the default period (23 hours) between the save of the fields in .sauv file.
- **[no_check_disk_space]** (*type:* flag) To disable the check of the available amount of disk space during the calculation.
- **[disable_progress]** (*type:* flag) To disable the writing of the .progress file.
- **[disable_dt_ev]** (*type:* flag) To disable the writing of the .dt_ev file.
- **[gnuplot_header]** (*type:* int) Optional keyword to modify the header of the .out files. Allows to use the column title instead of columns number.

schema_temps_base

Basic class for time schemes. This scheme will be associated with a problem and the equations of this problem.

Parameters are:

- **[tinit]** (*type:* double) Value of initial calculation time (0 by default).
- **[tmax]** (*type:* double) Time during which the calculation will be stopped (1e30s by default).
- **[tcpumax]** (*type:* double) CPU time limit (must be specified in hours) for which the calculation is stopped (1e30s by default).
- **[dt_min]** (*type:* double) Minimum calculation time step (1e-16s by default).
- **[dt_max]** (*type:* string) Maximum calculation time step as function of time (1e30s by default).
- **[dt_sauv]** (*type:* double) Save time step value (1e30s by default). Every dt_sauv, fields are saved in the .sauv file. The file contains all the information saved over time. If this instruction is not entered, results are saved only upon calculation completion. To disable the writing of the .sauv files, you must specify 0. Note that dt_sauv is in terms of physical time (not cpu time).
- **[dt_impr]** (*type:* double) Scheme parameter printing time step in time (1e30s by default). The time steps and the flux balances are printed (incorporated onto every side of processed domains) into the .out file.
- **[facsec]** (*type:* string) Value assigned to the safety factor for the time step (1. by default). It can also be a function of time. The time step calculated is multiplied by the safety factor. The first thing to try when a calculation does not converge with an explicit time scheme is to reduce the facsec to 0.5. Warning: Some schemes needs a facsec lower than 1 (0.5 is a good start), for example Schema_Adams_Bashforth_order_3.
- **[seuil_statio]** (*type:* double) Value of the convergence threshold (1e-12 by default). Problems using this type of time scheme converge when the derivatives dGi/dt NL1 of all the unknown transported values Gi have a combined absolute value less than this value. This is the keyword used to set the permanent rating threshold.
- **[residuals]** (*type: residuals*) To specify how the residuals will be computed (default max norm, possible to choose L2-norm instead).
- **[diffusion_implicite]** (*type:* int) Keyword to make the diffusive term in the Navier-Stokes equations implicit (in this case, it should be set to 1). The stability time step is then only based on the convection time step (dt=facsec*dt_convection). Thus, in some circumstances, an important gain is achieved with respect to the time step (large diffusion with respect to convection on tightened meshes). Caution: It is however recommended that the user avoids exceeding the convection time step by selecting a too large facsec value. Start with a facsec value of 1 and then increase it gradually if you wish to accelerate calculation. In addition, for a natural convection calculation with a zero initial velocity, in the first time step, the convection time is infinite and therefore dt=facsec*dt_max.
- **[seuil_diffusion_implicite]** (*type:* double) This keyword changes the default value (1e-6) of convergency criteria for the resolution by conjugate gradient used for implicit diffusion.
- **[impr_diffusion_implicite]** (*type:* int) Unactivate (default) or not the printing of the convergence during the resolution of the conjugate gradient.
- **[impr_extremums]** (*type:* int) Print unknowns extremas
- **[no_error_if_not_converged_diffusion_implicite]** (*type:* int) not_set
- **[no_conv_subiteration_diffusion_implicite]** (*type:* int) not_set
- **[dt_start]** (*type: dt_start*) dt_start dt_min : the first iteration is based on dt_min. dt_start dt_calc : the time step at first iteration is calculated in agreement with CFL condition. dt_start dt_fixe value : the first time step is fixed by the user (recommended when resuming calculation with Crank Nicholson temporal scheme to ensure continuity). By default, the first iteration is based on dt_calc.

- **[nb_pas_dt_max]** (*type:* int) Maximum number of calculation time steps (1e9 by default).
- **[niter_max_diffusion_implicite]** (*type:* int) This keyword changes the default value (number of unknowns) of the maximal iterations number in the conjugate gradient method used for implicit diffusion.
- **[precision_impr]** (*type:* int) Optional keyword to define the digit number for flux values printed into .out files (by default 3).
- **[periode_sauvegarde_securite_en_heures]** (*type:* double) To change the default period (23 hours) between the save of the fields in .sauv file.
- **[no_check_disk_space]** (*type:* flag) To disable the check of the available amount of disk space during the calculation.
- **[disable_progress]** (*type:* flag) To disable the writing of the .progress file.
- **[disable_dt_ev]** (*type:* flag) To disable the writing of the .dt_ev file.
- **[gnuplot_header]** (*type:* int) Optional keyword to modify the header of the .out files. Allows to use the column title instead of columns number.

2.1.55 Keywords derived from solveur_implicite_base

ice

Inherits from: *solveur_implicite_base*

Implicit Continuous-fluid Eulerian solver which is useful for a multiphase problem. Robust pressure reduction resolution.

Parameters are:

- **[pression_degeneree]** (*type:* int) Set to 1 if the pressure field is degenerate (ex. : incompressible fluid with no imposed-pressure BCs). Default: autodetected
- **[reduction_pression | pressure_reduction]** (*type:* int) Set to 1 if the user wants a resolution with a pressure reduction. Otherwise, the flag is to be set to 0 so that the complete matrix is considered. The default value of this flag is 1.
- **[criteres_convergence]** (*type: bloc_criteres_convergence*) Set the convergence thresholds for each unknown (i.e: alpha, temperature, velocity and pressure). The default values are respectively 0.01, 0.1, 0.01 and 100
- **[iter_min]** (*type:* int) Number of minimum iterations
- **[seuil_convergence_implicite]** (*type:* double) Convergence criteria.
- **[nb_corrections_max]** (*type:* int) Maximum number of corrections performed by the PISO algorithm to achieve the projection of the velocity field. The algorithm may perform less corrections then nb_corrections_max if the accuracy of the projection is sufficient. (By default nb_corrections_max is set to 21).
- **[facsec_diffusion_for_sets]** (*type:* double) facsec to impose on the diffusion time step in sets while the total time step stays smaller than the convection time step.
- **[seuil_convergence_solveur]** (*type:* double) value of the convergence criteria for the resolution of the implicit system build by solving several times per time step the Navier_Stokes equation and the scalar equations if any. This value MUST be used when a coupling between problems is considered (should be set to a value typically of 0.1 or 0.01).
- **[seuil_generation_solveur]** (*type:* double) Option to create a GMRES solver and use vrel as the convergence threshold (implicit linear system Ax=B will be solved if residual error ||Ax-B|| is lesser than vrel).

- **[seuil_verification_solveur]** (*type:* double) Option to check if residual error ||Ax-B|| is lesser than vrel after the implicit linear system Ax=B has been solved.
- **[seuil_test_preliminaire_solveur]** (*type:* double) Option to decide if the implicit linear system Ax=B should be solved by checking if the residual error ||Ax-B|| is bigger than vrel.
- **[solveur]** (*type: solveur_sys_base*) Method (different from the default one, Gmres with diagonal preconditioning) to solve the linear system.
- **[no_qdm]** (*type:* flag) Keyword to not solve qdm equation (and turbulence models of these equation).
- **[nb_it_max]** (*type:* int) Keyword to set the maximum iterations number for the Gmres.
- **[controle_residu]** (*type:* flag) Keyword of Boolean type (by default 0). If set to 1, the convergence occurs if the residu suddenly increases.

---

implicite

Inherits from: *solveur_implicite_base*

similar to PISO, but as it looks like a simplified solver, it will use fewer timesteps. But it may run faster because the pressure matrix is not re-assembled and thus provides CPU gains.

Parameters are:

- **[seuil_convergence_implicite]** (*type:* double) Convergence criteria.
- **[nb_corrections_max]** (*type:* int) Maximum number of corrections performed by the PISO algorithm to achieve the projection of the velocity field. The algorithm may perform less corrections then nb_corrections_max if the accuracy of the projection is sufficient. (By default nb_corrections_max is set to 21).
- **[seuil_convergence_solveur]** (*type:* double) value of the convergence criteria for the resolution of the implicit system build by solving several times per time step the Navier_Stokes equation and the scalar equations if any. This value MUST be used when a coupling between problems is considered (should be set to a value typically of 0.1 or 0.01).
- **[seuil_generation_solveur]** (*type:* double) Option to create a GMRES solver and use vrel as the convergence threshold (implicit linear system Ax=B will be solved if residual error ||Ax-B|| is lesser than vrel).
- **[seuil_verification_solveur]** (*type:* double) Option to check if residual error ||Ax-B|| is lesser than vrel after the implicit linear system Ax=B has been solved.
- **[seuil_test_preliminaire_solveur]** (*type:* double) Option to decide if the implicit linear system Ax=B should be solved by checking if the residual error ||Ax-B|| is bigger than vrel.
- **[solveur]** (*type: solveur_sys_base*) Method (different from the default one, Gmres with diagonal preconditioning) to solve the linear system.
- **[no_qdm]** (*type:* flag) Keyword to not solve qdm equation (and turbulence models of these equation).
- **[nb_it_max]** (*type:* int) Keyword to set the maximum iterations number for the Gmres.
- **[controle_residu]** (*type:* flag) Keyword of Boolean type (by default 0). If set to 1, the convergence occurs if the residu suddenly increases.

---

piso

Inherits from: *solveur_implicite_base*

Piso (Pressure Implicit with Split Operator) - method to solve N_S.

Parameters are:

- **[seuil_convergence_implicite]** (*type:* double) Convergence criteria.
- **[nb_corrections_max]** (*type:* int) Maximum number of corrections performed by the PISO algorithm to achieve the projection of the velocity field. The algorithm may perform less corrections then nb_corrections_max if the accuracy of the projection is sufficient. (By default nb_corrections_max is set to 21).
- **[seuil_convergence_solveur]** (*type:* double) value of the convergence criteria for the resolution of the implicit system build by solving several times per time step the Navier_Stokes equation and the scalar equations if any. This value MUST be used when a coupling between problems is considered (should be set to a value typically of 0.1 or 0.01).
- **[seuil_generation_solveur]** (*type:* double) Option to create a GMRES solver and use vrel as the convergence threshold (implicit linear system Ax=B will be solved if residual error ||Ax-B|| is lesser than vrel).
- **[seuil_verification_solveur]** (*type:* double) Option to check if residual error ||Ax-B|| is lesser than vrel after the implicit linear system Ax=B has been solved.
- **[seuil_test_preliminaire_solveur]** (*type:* double) Option to decide if the implicit linear system Ax=B should be solved by checking if the residual error ||Ax-B|| is bigger than vrel.
- **[solveur]** (*type: solveur_sys_base*) Method (different from the default one, Gmres with diagonal preconditioning) to solve the linear system.
- **[no_qdm]** (*type:* flag) Keyword to not solve qdm equation (and turbulence models of these equation).
- **[nb_it_max]** (*type:* int) Keyword to set the maximum iterations number for the Gmres.
- **[controle_residu]** (*type:* flag) Keyword of Boolean type (by default 0). If set to 1, the convergence occurs if the residu suddenly increases.

---

sets

Inherits from: *solveur_implicite_base*

Stability-Enhancing Two-Step solver which is useful for a multiphase problem. Ref : J. H. MAHAFFY, A stability-enhancing two-step method for fluid flow calculations, Journal of Computational Physics, 46, 3, 329 (1982).

Parameters are:

- **[criteres_convergence]** (*type: bloc_criteres_convergence*) Set the convergence thresholds for each unknown (i.e: alpha, temperature, velocity and pressure). The default values are respectively 0.01, 0.1, 0.01 and 100
- **[iter_min]** (*type:* int) Number of minimum iterations
- **[seuil_convergence_implicite]** (*type:* double) Convergence criteria.
- **[nb_corrections_max]** (*type:* int) Maximum number of corrections performed by the PISO algorithm to achieve the projection of the velocity field. The algorithm may perform less corrections then nb_corrections_max if the accuracy of the projection is sufficient. (By default nb_corrections_max is set to 21).
- **[facsec_diffusion_for_sets]** (*type:* double) facsec to impose on the diffusion time step in sets while the total time step stays smaller than the convection time step.

- **[seuil_convergence_solveur]** (*type:* double) value of the convergence criteria for the resolution of the implicit system build by solving several times per time step the Navier_Stokes equation and the scalar equations if any. This value MUST be used when a coupling between problems is considered (should be set to a value typically of 0.1 or 0.01).
- **[seuil_generation_solveur]** (*type:* double) Option to create a GMRES solver and use vrel as the convergence threshold (implicit linear system Ax=B will be solved if residual error ||Ax-B|| is lesser than vrel).
- **[seuil_verification_solveur]** (*type:* double) Option to check if residual error ||Ax-B|| is lesser than vrel after the implicit linear system Ax=B has been solved.
- **[seuil_test_preliminaire_solveur]** (*type:* double) Option to decide if the implicit linear system Ax=B should be solved by checking if the residual error ||Ax-B|| is bigger than vrel.
- **[solveur]** (*type: solveur_sys_base*) Method (different from the default one, Gmres with diagonal preconditioning) to solve the linear system.
- **[no_qdm]** (*type:* flag) Keyword to not solve qdm equation (and turbulence models of these equation).
- **[nb_it_max]** (*type:* int) Keyword to set the maximum iterations number for the Gmres.
- **[controle_residu]** (*type:* flag) Keyword of Boolean type (by default 0). If set to 1, the convergence occurs if the residu suddenly increases.

---

simple

Inherits from: *solveur_implicite_base*

SIMPLE type algorithm

Parameters are:

- **[relax_pression]** (*type:* double) Value between 0 and 1 (by default 1), this keyword is used only by the SIMPLE algorithm for relaxing the increment of pressure.
- **[seuil_convergence_implicite]** (*type:* double) Convergence criteria.
- **[nb_corrections_max]** (*type:* int) Maximum number of corrections performed by the PISO algorithm to achieve the projection of the velocity field. The algorithm may perform less corrections then nb_corrections_max if the accuracy of the projection is sufficient. (By default nb_corrections_max is set to 21).
- **[seuil_convergence_solveur]** (*type:* double) value of the convergence criteria for the resolution of the implicit system build by solving several times per time step the Navier_Stokes equation and the scalar equations if any. This value MUST be used when a coupling between problems is considered (should be set to a value typically of 0.1 or 0.01).
- **[seuil_generation_solveur]** (*type:* double) Option to create a GMRES solver and use vrel as the convergence threshold (implicit linear system Ax=B will be solved if residual error ||Ax-B|| is lesser than vrel).
- **[seuil_verification_solveur]** (*type:* double) Option to check if residual error ||Ax-B|| is lesser than vrel after the implicit linear system Ax=B has been solved.
- **[seuil_test_preliminaire_solveur]** (*type:* double) Option to decide if the implicit linear system Ax=B should be solved by checking if the residual error ||Ax-B|| is bigger than vrel.
- **[solveur]** (*type: solveur_sys_base*) Method (different from the default one, Gmres with diagonal preconditioning) to solve the linear system.
- **[no_qdm]** (*type:* flag) Keyword to not solve qdm equation (and turbulence models of these equation).
- **[nb_it_max]** (*type:* int) Keyword to set the maximum iterations number for the Gmres.

- **[controle_residu]** (*type:* flag) Keyword of Boolean type (by default 0). If set to 1, the convergence occurs if the residu suddenly increases.

---

simpler

Inherits from: *solveur_implicite_base*

Simpler method for incompressible systems.

Parameters are:

- **seuil_convergence_implicite** (*type:* double) Keyword to set the value of the convergence criteria for the resolution of the implicit system build to solve either the Navier_Stokes equation (only for Simple and Simpler algorithms) or a scalar equation. It is adviced to use the default value (1e6) to solve the implicit system only once by time step. This value must be decreased when a coupling between problems is considered.
- **[seuil_convergence_solveur]** (*type:* double) value of the convergence criteria for the resolution of the implicit system build by solving several times per time step the Navier_Stokes equation and the scalar equations if any. This value MUST be used when a coupling between problems is considered (should be set to a value typically of 0.1 or 0.01).
- **[seuil_generation_solveur]** (*type:* double) Option to create a GMRES solver and use vrel as the convergence threshold (implicit linear system Ax=B will be solved if residual error ||Ax-B|| is lesser than vrel).
- **[seuil_verification_solveur]** (*type:* double) Option to check if residual error ||Ax-B|| is lesser than vrel after the implicit linear system Ax=B has been solved.
- **[seuil_test_preliminaire_solveur]** (*type:* double) Option to decide if the implicit linear system Ax=B should be solved by checking if the residual error ||Ax-B|| is bigger than vrel.
- **[solveur]** (*type: solveur_sys_base*) Method (different from the default one, Gmres with diagonal preconditioning) to solve the linear system.
- **[no_qdm]** (*type:* flag) Keyword to not solve qdm equation (and turbulence models of these equation).
- **[nb_it_max]** (*type:* int) Keyword to set the maximum iterations number for the Gmres.
- **[controle_residu]** (*type:* flag) Keyword of Boolean type (by default 0). If set to 1, the convergence occurs if the residu suddenly increases.

---

solveur_implicite_base

Class for solver in the situation where the time scheme is the implicit scheme. Solver allows equation diffusion and convection operators to be set as implicit terms.

---

solveur_lineaire_std

Inherits from: *solveur_implicite_base*

not_set

Parameters are:

- **[solveur]** (*type:* *solveur_sys_base*) not_set

---

solveur_u_p

Inherits from: *solveur_implicite_base*

similar to simple.

Parameters are:

- **[relax_pression]** (*type:* double) Value between 0 and 1 (by default 1), this keyword is used only by the SIMPLE algorithm for relaxing the increment of pressure.
- **[seuil_convergence_implicite]** (*type:* double) Convergence criteria.
- **[nb_corrections_max]** (*type:* int) Maximum number of corrections performed by the PISO algorithm to achieve the projection of the velocity field. The algorithm may perform less corrections then nb_corrections_max if the accuracy of the projection is sufficient. (By default nb_corrections_max is set to 21).
- **[seuil_convergence_solveur]** (*type:* double) value of the convergence criteria for the resolution of the implicit system build by solving several times per time step the Navier_Stokes equation and the scalar equations if any. This value MUST be used when a coupling between problems is considered (should be set to a value typically of 0.1 or 0.01).
- **[seuil_generation_solveur]** (*type:* double) Option to create a GMRES solver and use vrel as the convergence threshold (implicit linear system Ax=B will be solved if residual error ||Ax-B|| is lesser than vrel).
- **[seuil_verification_solveur]** (*type:* double) Option to check if residual error ||Ax-B|| is lesser than vrel after the implicit linear system Ax=B has been solved.
- **[seuil_test_preliminaire_solveur]** (*type:* double) Option to decide if the implicit linear system Ax=B should be solved by checking if the residual error ||Ax-B|| is bigger than vrel.
- **[solveur]** (*type:* *solveur_sys_base*) Method (different from the default one, Gmres with diagonal preconditioning) to solve the linear system.
- **[no_qdm]** (*type:* flag) Keyword to not solve qdm equation (and turbulence models of these equation).
- **[nb_it_max]** (*type:* int) Keyword to set the maximum iterations number for the Gmres.
- **[controle_residu]** (*type:* flag) Keyword of Boolean type (by default 0). If set to 1, the convergence occurs if the residu suddenly increases.

2.1.56 Keywords derived from sondes

sondes

Synonyms: probes

List of probes.

2.1.57 Keywords derived from source_base

acceleration

Inherits from: *source_base*

Momentum source term to take in account the forces due to rotation or translation of a non Galilean referential R' (centre 0') into the Galilean referential R (centre 0).

Parameters are:

- **[vitesse]** (*type: field_base*) Keyword for the velocity of the referential R' into the R referential (dOO'/dt term [m.s-1]). The velocity is mandatory when you want to print the total cinetic energy into the non-mobile Galilean referential R (see Ec_dans_repere_fixe keyword).
- **[acceleration]** (*type: field_base*) Keyword for the acceleration of the referential R' into the R referential (d2OO'/dt2 term [m.s-2]). field_base is a time dependant field (eg: Champ_Fonc_t).
- **[omega]** (*type: field_base*) Keyword for a rotation of the referential R' into the R referential [rad.s-1]. field_base is a 3D time dependant field specified for example by a Champ_Fonc_t keyword. The time_field field should have 3 components even in 2D (In 2D: 0 0 omega).
- **[domegadt]** (*type: field_base*) Keyword to define the time derivative of the previous rotation [rad.s-2]. Should be zero if the rotation is constant. The time_field field should have 3 components even in 2D (In 2D: 0 0 domegadt).
- **[centre_rotation]** (*type: field_base*) Keyword to specify the centre of rotation (expressed in R' coordinates) of R' into R (if the domain rotates with the R' referential, the centre of rotation is 0'=(0,0,0)). The time_field should have 2 or 3 components according the dimension 2 or 3.
- **[option]** (*type:* string into ["terme_complet", "coriolis_seul", "entrainement_seul"]) Keyword to specify the kind of calculation: terme_complet (default option) will calculate both the Coriolis and centrifugal forces, coriolis_seul will calculate the first one only, entrainement_seul will calculate the second one only.

---

boussinesq_concentration

Inherits from: *source_base*

Class to describe a source term that couples the movement quantity equation and constituent transport equation with the Boussinesq hypothesis.

Parameters are:

- **c0** (*type:* list) Reference concentration field type. The only field type currently available is Champ_Uniforme (Uniform field).
- **[verif_boussinesq]** (*type:* int) Keyword to check (1) or not (0) the reference concentration in comparison with the mean concentration value in the domain. It is set to 1 by default.

---

boussinesq_temperature

Inherits from: *source_base*

Class to describe a source term that couples the movement quantity equation and energy equation with the Boussinesq hypothesis.

Parameters are:

- **t0** (*type:* string) Reference temperature value (oC or K). It can also be a time dependant function since the 1.6.6 version.
- **[verif_boussinesq]** (*type:* int) Keyword to check (1) or not (0) the reference temperature in comparison with the mean temperature value in the domain. It is set to 1 by default.

---

canal_perio

Inherits from: *source_base*

Momentum source term to maintain flow rate. The expression of the source term is:

S(t) = (2*(Q(0) - Q(t))-(Q(0)-Q(t-dt))/(coeff*dt*area)

NL2 Where:

coeff=damping coefficient

area=area of the periodic boundary

Q(t)=flow rate at time t

dt=time step

NL2 Three files will be created during calculation on a datafile named DataFile.data. The first file contains the flow rate evolution. The second file is useful for resuming a calculation with the flow rate of the previous stopped calculation, and the last one contains the pressure gradient evolution:

-DataFile_Channel_Flow_Rate_ProblemName_BoundaryName

-DataFile_Channel_Flow_Rate_repr_ProblemName_BoundaryName

-DataFile_Pressure_Gradient_ProblemName_BoundaryName

Parameters are:

- **bord** (*type:* string) The name of the (periodic) boundary normal to the flow direction.
- **[h]** (*type:* double) Half heigth of the channel.
- **[coeff]** (*type:* double) Damping coefficient (optional, default value is 10).
- **[debit_impose]** (*type:* double) Optional option to specify the aimed flow rate Q(0). If not used, Q(0) is computed by the code after the projection phase, where velocity initial conditions are slighlty changed to verify incompressibility.

---

coriolis

Inherits from: *source_base*

Keyword for a Coriolis term in hydraulic equation. Warning: Only available in VDF.

Parameters are:

- **omega** (*type:* string) Value of omega.

---

correction_antal

Inherits from: *source_base*

Antal correction source term for multiphase problem

---

darcy

Inherits from: *source_base*

Class for calculation in a porous media with source term of Darcy -nu/K*V. This keyword must be used with a permeability model. For the moment there are two models : permeability constant or Ergun's law. Darcy source term is available for quasi compressible calculation. A new keyword is aded for porosity (porosite).

Parameters are:

- **bloc** (*type: bloc_lecture*) Description.

---

dirac

Inherits from: *source_base*

Class to define a source term corresponding to a volume power release in the energy equation.

Parameters are:

- **position** (*type:* list) not_set
- **ch** (*type: field_base*) Thermal power field type. To impose a volume power on a domain sub-area, the Champ_Uniforme_Morceaux (partly_uniform_field) type must be used. Warning : The volume thermal power is expressed in W.m-3.

---

dispersion_bulles

Inherits from: *source_base*

Base class for source terms of bubble dispersion in momentum equation.

Parameters are:

- **[beta]** (*type:* double) Mutliplying factor for the output of the bubble dispersion source term.

---

dp_impose

Inherits from: *source_base*

Source term to impose a pressure difference according to the formula : DP = dp + dDP/dQ * (Q - Q0)

Parameters are:

- **aco** (*type:* string into [“{“]) Opening curly bracket.
- **dp_type** (*type: type_perte_charge_deriv*) mass flow rate (kg/s).
- **surface** (*type:* string into [“surface”]) not_set
- **bloc_surface** (*type: bloc_lecture*) Three syntaxes are possible for the surface definition block: For VDF and VEF: { X|Y|Z = location subzone_name } Only for VEF: { Surface surface_name }. For polymac { Surface surface_name Orientation champ_uniforme }.
- **acof** (*type:* string into [“}”]) Closing curly bracket.

---

flux_interfacial

Inherits from: *source_base*

Source term of mass transfer between phases connected by the saturation object defined in saturation_xxxx

---

forchheimer

Inherits from: *source_base*

Class to add the source term of Forchheimer -Cf/sqrt(K)*V2 in the Navier-Stokes equations. We must precise a permeability model : constant or Ergun’s law. Moreover we can give the constant Cf : by default its value is 1. Forchheimer source term is available also for quasi compressible calculation. A new keyword is aded for porosity (porosite).

Parameters are:

- **bloc** (*type: bloc_lecture*) Description.

---

frottement_interfacial

Inherits from: *source_base*

Source term which corresponds to the phases friction at the interface

Parameters are:

- **[a_res]** (*type:* double) void fraction at which the gas velocity is forced to approach liquid velocity (default alpha_evanescence*100)
- **[dv_min]** (*type:* double) minimal relative velocity used to linearize interfacial friction at low velocities
- **[exp_res]** (*type:* int) exponent that callibrates intensity of velocity convergence (default 2)

---

perte_charge_anisotrope

Inherits from: *source_base*

Anisotropic pressure loss.

Parameters are:

- **Lambda | lambda_u** (*type:* string) Function for loss coefficient which may be Reynolds dependant (Ex: 64/Re).
- **lambda_ortho** (*type:* string) Function for loss coefficient in transverse direction which may be Reynolds dependant (Ex: 64/Re).
- **diam_hydr** (*type:* *champ_don_base*) Hydraulic diameter value.
- **direction** (*type:* *champ_don_base*) Field which indicates the direction of the pressure loss.
- **[sous_zone]** (*type:* string) Optional sub-area where pressure loss applies.

---

perte_charge_circulaire

Inherits from: *source_base*

New pressure loss.

Parameters are:

- **Lambda | lambda_u** (*type:* string) Function f(Re_tot, Re_long, t, x, y, z) for loss coefficient in the longitudinal direction
- **lambda_ortho** (*type:* string) function: Function f(Re_tot, Re_ortho, t, x, y, z) for loss coefficient in transverse direction
- **diam_hydr** (*type:* *champ_don_base*) Hydraulic diameter value.
- **diam_hydr_ortho** (*type:* *champ_don_base*) Transverse hydraulic diameter value.
- **direction** (*type:* *champ_don_base*) Field which indicates the direction of the pressure loss.
- **[sous_zone]** (*type:* string) Optional sub-area where pressure loss applies.

---

perte_charge_directionnelle

Inherits from: *source_base*

Directional pressure loss.

Parameters are:

- **Lambda | lambda_u** (*type:* string) Function for loss coefficient which may be Reynolds dependant (Ex: 64/Re).
- **diam_hydr** (*type:* *champ_don_base*) Hydraulic diameter value.
- **direction** (*type:* *champ_don_base*) Field which indicates the direction of the pressure loss.
- **[sous_zone]** (*type:* string) Optional sub-area where pressure loss applies.

---

perte_charge_isotrope

Inherits from: *source_base*

Isotropic pressure loss.

Parameters are:

- **Lambda | lambda_u** (*type:* string) Function for loss coefficient which may be Reynolds dependant (Ex: 64/Re).
- **diam_hydr** (*type:* *champ_don_base*) Hydraulic diameter value.
- **[sous_zone]** (*type:* string) Optional sub-area where pressure loss applies.

---

perte_charge_reguliere

Inherits from: *source_base*

Source term modelling the presence of a bundle of tubes in a flow.

Parameters are:

- **spec** (*type:* *spec_pdcr_base*) Description of longitudinale or transversale type.
- **zone_name | name_of_zone** (*type:* string) Name of the sub-area occupied by the tube bundle. A Sous_Zone (Sub-area) type object called zone_name should have been previously created.

---

perte_charge_singuliere

Inherits from: *source_base*

Source term that is used to model a pressure loss over a surface area (transition through a grid, sudden enlargement) defined by the faces of elements located on the intersection of a subzone named subzone_name and a X,Y, or Z plane located at X,Y or Z = location.

Parameters are:

- **dir** (*type:* string into [“kx”, “ky”, “kz”, “k”]) KX, KY or KZ designate directional pressure loss coefficients for respectively X, Y or Z direction. Or in the case where you chose a target flow rate with regul. Use K for isotropic pressure loss coefficient

- **[coeff]** (*type:* double) Value (float) of friction coefficient (KX, KY, KZ).
- **[regul]** (*type: bloc_lecture*) option to have adjustable K with flowrate target { K0 valeur_initiale_de_k deb debit_cible eps intervalle_variation_mutiplicatif}.
- **surface** (*type: bloc_lecture*) Three syntaxes are possible for the surface definition block: For VDF and VEF: { X|Y|Z = location subzone_name } Only for VEF: { Surface surface_name }. For polymac { Surface surface_name Orientation champ_uniforme }

---

portance_interfaciale

Inherits from: *source_base*

Base class for source term of lift force in momentum equation.

Parameters are:

- **[beta]** (*type:* double) Multiplying factor for the bubble lift force source term.

---

puissance_thermique

Inherits from: *source_base*

Class to define a source term corresponding to a volume power release in the energy equation.

Parameters are:

- **ch** (*type: field_base*) Thermal power field type. To impose a volume power on a domain sub-area, the Champ_Uniforme_Morceaux (partly_uniform_field) type must be used. Warning : The volume thermal power is expressed in W.m-3 in 3D (in W.m-2 in 2D). It is a power per volume unit (in a porous media, it is a power per fluid volume unit).

---

radioactive_decay

Inherits from: *source_base*

Radioactive decay source term of the form $-lambda_i c_i$, where $0 leq i leq N$, N is the number of component of the constituent, c_i and $lambda_i$ are the concentration and the decay constant of the i-th component of the constituant.

Parameters are:

- **val** (*type:* list) n is the number of decay constants to read (int), and val1, val2... are the decay constants (double)

---

source_base

Basic class of source terms introduced in the equation.

---

source_constituant

Inherits from: *source_base*

Keyword to specify source rates, in [[C]/s], for each one of the nb constituents. [C] is the concentration unit.

Parameters are:

- **ch** (*type: field_base*) Field type.

---

source_generique

Inherits from: *source_base*

to define a source term depending on some discrete fields of the problem and (or) analytic expression. It is expressed by the way of a generic field usually used for post-processing.

Parameters are:

- **champ** (*type: champ_generique_base*) the source field

---

source_pdf

Inherits from: *source_base*

Source term for Penalised Direct Forcing (PDF) method.

Parameters are:

- **aire** (*type: field_base*) volumic field: a boolean for the cell (0 or 1) indicating if the obstacle is in the cell
- **rotation** (*type: field_base*) volumic field with 9 components representing the change of basis on cells (local to global). Used for rotating cases for example.
- **[transpose_rotation]** (*type:* flag) whether to transpose the basis change matrix.
- **modele** (*type: bloc_pdf_model*) model used for the Penalized Direct Forcing
- **[interpolation]** (*type: interpolation_ibm_base*) interpolation method

---

source_pdf_base

Inherits from: *source_base*

Base class of the source term for the Immersed Boundary Penalized Direct Forcing method (PDF)

Parameters are:

- **aire** (*type: field_base*) volumic field: a boolean for the cell (0 or 1) indicating if the obstacle is in the cell
- **rotation** (*type: field_base*) volumic field with 9 components representing the change of basis on cells (local to global). Used for rotating cases for example.
- **[transpose_rotation]** (*type:* flag) whether to transpose the basis change matrix.
- **modele** (*type: bloc_pdf_model*) model used for the Penalized Direct Forcing
- **[interpolation]** (*type: interpolation_ibm_base*) interpolation method

---

source_qdm

Inherits from: *source_base*

Momentum source term in the Navier-Stokes equations.

Parameters are:

- **ch | champ** (*type: field_base*) Field type.

---

source_qdm_lambdaup

Inherits from: *source_base*

This source term is a dissipative term which is intended to minimise the energy associated to non-conformscales u' (responsible for spurious oscillations in some cases). The equation for these scales can be seen as: du'/dt= -lambda. u' + grad P' where -lambda. u' represents the dissipative term, with lambda = a/Delta t For Crank-Nicholson temporal scheme, recommended value for a is 2.

Remark : This method requires to define a filtering operator.

Parameters are:

- **Lambda | lambda_u** (*type:* double) value of lambda
- **[lambda_min]** (*type:* double) value of lambda_min
- **[lambda_max]** (*type:* double) value of lambda_max
- **[ubar_umprim_cible]** (*type:* double) value of ubar_umprim_cible

---

source_robin

Inherits from: *source_base*

This source term should be used when a Paroi_decalee_Robin boundary condition is set in a hydraulic equation. The source term will be applied on the N specified boundaries. To post-process the values of tauw, u_tau and Reynolds_tau into the files tauw_robin.dat, reynolds_tau_robin.dat and u_tau_robin.dat, you must add a block Traitement_particulier { canal { } }

Parameters are:

- **bords** (*type: vect_nom*) not_set

---

source_robin_scalaire

Inherits from: *source_base*

This source term should be used when a Paroi_decalee_Robin boundary condition is set in a an energy equation. The source term will be applied on the N specified boundaries. The values temp_wall_valueI are the temperature specified on the Ith boundary. The last value dt_impr is a printing period which is mandatory to specify in the data file but has no effect yet.

Parameters are:

- **bords** (*type: listdeuxmots_sacc*) not_set

---

source_th_tdivu

Inherits from: *source_base*

This term source is dedicated for any scalar (called T) transport. Coupled with upwind (amont) or muscl scheme, this term gives for final expression of convection : div(U.T)-T.div (U)=U.grad(T) This ensures, in incompressible flow when divergence free is badly resolved, to stay in a better way in the physical boundaries.

Warning: Only available in VEF discretization.

---

terme_puissance_thermique_echange_impose

Inherits from: *source_base*

Source term to impose thermal power according to formula : P = himp * (T - Text). Where T is the Trust temperature, Text is the outside temperature with which energy is exchanged via an exchange coefficient himp

Parameters are:

- **himp** (*type: field_base*) the exchange coefficient
- **text** (*type: field_base*) the outside temperature
- **[pid_controler_on_targer_power]** (*type: bloc_lecture*) PID_controler_on_targer_power bloc with parameters target_power (required), Kp, Ki and Kd (at least one of them should be provided)

---

travail_pression

Inherits from: *source_base*

Source term which corresponds to the additional pressure work term that appears when dealing with compressible multiphase fluids

---

vitesse_derive_base

Inherits from: *source_base*

Source term which corresponds to the drift-velocity between a liquid and a gas phase

---

vitesse_relative_base

Inherits from: *source_base*

Basic class for drift-velocity source term between a liquid and a gas phase

---

2.1.58 Keywords derived from sources

sources

The sources.

2.1.59 Keywords derived from sous_zone

sous_zone

Synonyms: sous_domaine

It is an object type describing a domain sub-set.

A Sous_Zone (Sub-area) type object must be associated with a Domaine type object. The Read (Lire) interpretor is used to define the items comprising the sub-area.

Caution: The Domain type object nom_domaine must have been meshed (and triangulated or tetrahedralised in VEF) prior to carrying out the Associate (Associer) nom_sous_zone nom_domaine instruction; this instruction must always be preceded by the read instruction.

Parameters are:

- **[restriction]** (*type: sous_zone*) The elements of the sub-area nom_sous_zone must be included into the other sub-area named nom_sous_zone2. This keyword should be used first in the Read keyword.
- **[rectangle]** (*type: bloc_origine_cotes*) The sub-area will include all the domain elements whose centre of gravity is within the Rectangle (in dimension 2).

- **[segment]** (*type: bloc_origine_cotes*) not_set
- **[boite | box]** (*type: bloc_origine_cotes*) The sub-area will include all the domain elements whose centre of gravity is within the Box (in dimension 3).
- **[liste]** (*type:* int list) The sub-area will include n domain items, numbers No. 1 No. i No. n.
- **[fichier | filename]** (*type:* string) The sub-area is read into the file filename.
- **[intervalle]** (*type: deuxentiers*) The sub-area will include domain items whose number is between n1 and n2 (where n1<=n2).
- **[polynomes]** (*type: bloc_lecture*) A REPRENDRE
- **[couronne]** (*type: bloc_couronne*) In 2D case, to create a couronne.
- **[tube]** (*type: bloc_tube*) In 3D case, to create a tube.
- **[fonction_sous_zone | fonction_sous_domaine]** (*type:* string) Keyword to build a sub-area with the the elements included into the area defined by fonction>0.
- **[union | union_with]** (*type: sous_zone*) The elements of the sub-area nom_sous_zone3 will be added to the sub-area nom_sous_zone. This keyword should be used last in the Read keyword.

2.1.60 Keywords derived from turbulence_paroi_base

negligeable

Inherits from: *turbulence_paroi_base*

Keyword to suppress the calculation of a law of the wall with a turbulence model. The wall stress is directly calculated with the derivative of the velocity, in the direction perpendicular to the wall (tau_tan /rho= nu dU/dy).

Warning: This keyword is not available for k-epsilon models. In that case you must choose a wall law.

---

turbulence_paroi_base

Basic class for wall laws for Navier-Stokes equations.

2.1.61 Keywords derived from turbulence_paroi_scalaire_base

negligeable_scalaire

Inherits from: *turbulence_paroi_scalaire_base*

Keyword to suppress the calculation of a law of the wall with a turbulence model for thermohydraulic problems. The wall stress is directly calculated with the derivative of the velocity, in the direction perpendicular to the wall.

---

turbulence_paroi_scalaire_base

Basic class for wall laws for energy equation.

2.1.62 Keywords derived from vect_nom

vect_nom

Vect of name.