

## Tdyn CFD+HT reference manual

Three-dimensional environment for multiphysics analysis.

<http://www.compassis.com>

[info@compassis.com](mailto:info@compassis.com)

November 2018

Version: 15.1.0

### 1. About this Manual

This is the user's reference manual of the Tdyn CFD+HT module of Tdyn. This module provides an entire engineering solution for solving problems involving fluid and porous media flow, heat transfer and multi-physics. The present manual is organized in four sections:

- An introduction, where the user can find an explanation of the more general aspects of the program. It is strongly recommended to read this section before to start using Tdyn CFD+HT.
- A second part, named Tdyn CFD+HT Reference, that gives an explanation of every option available in Tdyn CFD+HT.
- The third part, gives some recommendations and best practices for the use of Tdyn CFD+HT.
- The fourth part, is only intended for experienced users, and gives some details about the internal operation of the Tdyn CFD+HT module.

In this manual, different kinds of fonts are used to help the users follow all the possibilities offered by the program Tdyn CFD+HT:

- **font** is used for the options found in the menus, tree and windows.
- *font* is used for fields found in the tree entries of **Conditions & Initial Data**, **Materials**, and **General Data**.

### 2. Introduction

Tdyn CFD+HT is an environment for multi physics simulation, using a stabilised (FIC) finite element method.

Tdyn CFD+HT includes different modules that allow to solve Heat Transfer in both solids and fluids, Turbulence, Advection of Species in fluids and solids and Free Surface (Transpiration or OddLevelSet method) problems using the same stabilised scheme mentioned above. Tdyn CFD+HT includes a complete environment for geometry and data definition, and post-processing of the analysis results, based in the Custom GiD system.

By using Tdyn CFD+HT it is also possible to configure additional user defined partial differential equations (PDE) solvers in both fluid and solid materials and to couple them with any of the other problems.

Finally, it can also be used to simulate problems where mesh deformation (i.e. body movement) may occur.

The different analysis available in Tdyn CFD+HT are fully integrated and the complete environment is used as one single program.

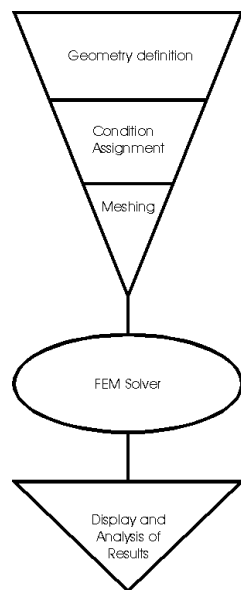
A more detailed explanation of the capabilities of every Tdyn CFD+HT module is given next.

- Fluid Flow module is able to solve incompressible or slightly compressible fluid flow problems, including turbulence effects (RANS equations). It is also able to solve porous media flow (Stokes) problem in solid materials. Physical properties used in this module can be defined in terms of any other variable of the problem.
- Heat Transfer module is able to solve heat transfer problems in fluids and solids. Physical properties used in this module can be defined in terms of any other variable of the problem.
- Species Advection module is able to solve species advection problems in fluids. It is also able to solve species diffusion problems in solids. This module allows to define a number of new species whose behaviour can be coupled among them or with any other variable (i.e. velocity, pressure, temperature, ...) used in Tdyn CFD+HT. Physical properties used in this module can be defined in terms of any other variable of the problem.
- PDE's solver module is able to solve user defined PDE problems in fluids and solids. This module allows to define a number of new variables (called  $\phi$ -phi problems) and specify and solve the differential equation that governs its behaviour. New user-defined problems can be coupled among them or with any other variable (i.e. velocity, pressure, temperature, ...) used in Tdyn CFD+HT.
- Mesh Deformation module includes all the necessary capabilities to solve problems with mesh updating techniques. This module includes several mesh updating techniques and arbitrary Lagrangian-Eulerian (ALE) algorithms for solving systems of equations. Note that ALE techniques are only available in fluid domain problems.
- Free Surface-Transpiration module is able to solve free surface equations, based on the transpiration technique. This module is specially adequate to solve naval hydrodynamics problems.
- Free Surface-Odd Level Set module can solve free surface problems by means of the overlapping domain decomposition level set technique. This module has been specially designed to solve large / violent free surface motions.
- RamSeries module features an advanced environment for

structural analysis, based on the Finite Element Method. RamSeries module includes 3D beam, shell and solid models, as well as a full support for nonlinear elements, nonlinear and linear material laws, and inelastic material models. RamSeries easily simulates even the largest and most intricate of structures. Furthermore, RamSeries features the latest technology for solving structural dynamics analyses, including contact-impact algorithms which permit the study of many common problems in engineering. As for composite materials, RamSeries allows to study laminated beam and shells, including the capability of setting predefined standard laminate sequences giving each ply direction, based on local or global frames of reference. A key feature here is the possibility of viewing the critical strains and stresses in each of the laminated material plies or tissues. RamSeries module allows to perform coupled analysis, ranging from thermo-mechanical analyses to the most sophisticated fluid-structure interaction problems.

The analysis of a problem by means of Tdyn CFD+HT consists of the following basic steps (see Figure 1):

- Pre-processing
- Creation (or importation) of the geometry to be analysed
- Assignment of the material properties, boundary conditions and definition of general data
- Mesh generation
- Calculation
- Post-processing



Steps involved in any FEM problem

## 2.1. Pre-processing

The first step for any analysis is the setting up of the problem. This includes the creation or importation from a CAD file of the problem's geometry (i.e. the control volume) the assignment of boundary and initial conditions, and the generation of the mesh. This will all be made in the PREPOST module of the Tdyn environment (based on the Custom GiD system).

*Remarks:*

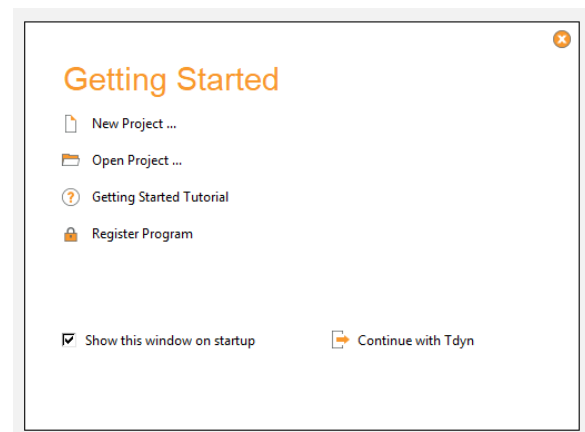
A full description of the CAD preprocessing

capabilities of the environment can be found in the GiD reference and user manuals (see Postprocessing section).

When Tdyn is run for the first time, the 'Getting Started' window shown below appears. It can be opened again at any moment by using the following command's sequence from the main menu of Tdyn.

### Help ► Open start page

This window provides the most basic options for a new user. From here, a new project can be started and an already existing project can be opened as well. This window also provides the opportunity to open the 'Getting Started Tutorial' of Tdyn which will guide the user through the necessary steps for running a simple simulation. Finally, the user can also access the passwords registration window which is a necessary step in order to run non-academic problems.



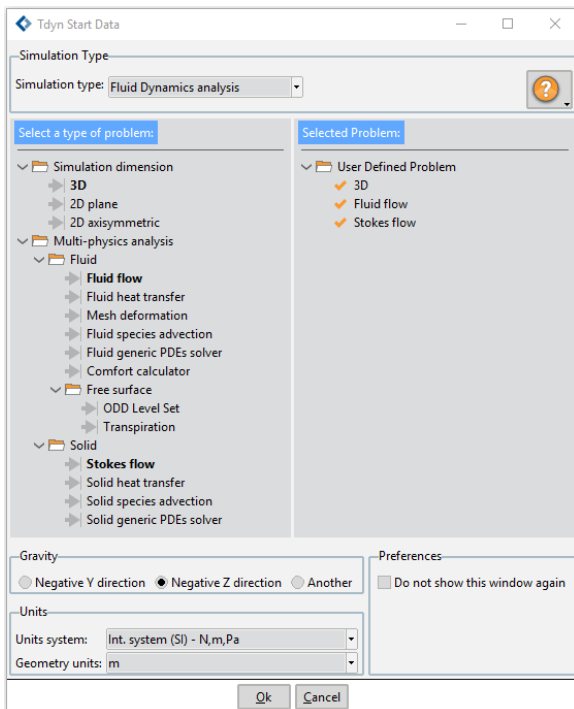
Getting Started window of Tdyn. Among other options, it provides the possibility to open the 'Getting Started Tutorial' which is a good starting point for any new user of the program

By invoking the 'New Project ...' option the 'Start Data' window shown below is started. This window allows the user to configure the type of analysis for the problem to be simulated. To modify the initial configuration of the analysis, this window can also be opened at any moment by selecting the following command's sequence from the main menu.

### Data ► Start Data

The first step to configure the analysis consists on the selection of the simulation type to be carried out. Available options are:

- Fluid Dynamics & Multi-physics
- Structural Analysis
- Seakeeping analysis
- Multiphysics analysis
- Coupled Seakeeping-Structural analysis
- Thermomechanical analysis
- Fluid-Structure interaction analysis



Start Data window of Tdyn used to quickly configure the type of analysis for the problem to be simulated

Once the simulation type has been selected, the Start Data window also allows to select what capabilities of Tdyn are going to be used. In this way, the graphic user interface (GUI) of Tdyn is simplified by removing those menus and options that are irrelevant for the type of analysis being considered.

When setting up a new analysis for calculation with Tdyn, the specific simulation type and the corresponding simulation dimension have to be selected. This can be done at any point from the creation of the new project to the final definition of the geometry, but has to be performed before the user can proceed to assigning boundary conditions and other problem parameters.

Depending on the simulation type selected for analysis, the following problem elements must be specified:

If 'Structural Analysis' option is selected:

- **Simulation Dimension:** Select the analysis dimension (2D Plane Strain, 2D Plane Stress or 3D).
- **Basic Element Type:** Different element combinations can be chosen (i.e: Beams + Shells), or just one.
- **Analysis Type:** Define the type of analysis to perform (Static or Dynamic).
- **Material Constitutive Model:** Define the type of analysis to perform (Linear or Non-linear).

If 'Multi-physics Analysis Type' option is selected:

- **Simulation Dimension:** Select the analysis dimension (2D Plane, 2D Axisymmetric or 3D).
- **Fluid:** Define data concerning solver in Fluid domain.

Fluid Flow

Heat Transfer

Mesh Deformation (it is a particular case of Fluid Flow analysis).

Species Advection

PDE's solver

Free Surface: Odd Level Set or Transpiration (it is a particular case of Fluid Flow analysis).

- **Solid:** Define data concerning solver in Solid domain.

Fluid Flow

Heat Transfer

Species Advection

PDE's solver

If 'Coupled Fluid-Structural Analysis' option is selected:

- **Simulation Dimension:** Select the analysis dimension (2D Plane or 3D).

Structural Analysis:

- **Basic Element Type:** Different element combinations can be chosen (i.e: Beams + Shells), or just one.
- **Analysis Type:** Define the type of analysis to perform: Static, Dynamic.
- **Material Constitutive Model:** Define the type of analysis to perform: Linear, Non-linear.

Multi-physics Analysis Type:

- **Fluid:** Define data concerning solver in Fluid domain.

Fluid Flow

Mesh Deformation (it is a particular case of Fluid Flow analysis).

- **Solid:** Define data concerning solver in Solid domain.

Heat Transfer

- **Gravity:** Choose the direction of the gravity for problems with Specific Weight  $\neq 0$ . This vector specifies the direction but not the magnitude of the gravity.

- **Units:** Choose the geometry units.

Remarks:

*Depending on the analysis preferences selected by the user some options are not available, these options are disabled or hidden in the Start Data window.*

Pre-processing part of the analysis consist of the following steps:













- Model geometry importation or generation
- Conditions & Boundaries assignment
- Materials definition
- General problem data insertion
- Solver data definition
- Units definition
- Mesh size assignment
- Mesh Generation

The first step for any analysis is the setting up of the problem. This includes the creation or importation from a CAD file of the problem's geometry (i.e. the control volume) the assignment of boundary and initial conditions, and the generation of the mesh. This will all be made in the Tdyn CFD+HT environment (based on the Custom GiD system).

Remarks:

A full description of the CAD preprocessing capabilities of the environment can be found in the GiD reference and user manuals.

The **Tdyn CFD+HT pre toolbar** has been designed to guide the user during the pre-processing part of the fluid dynamics and multi-physics analysis. The corresponding icons and their associated functions are summarized in what follows:

Tdyn CFD+HT preprocessor toolbar	
	Define fluid flow module data
	Define fluid heat transfer module data
	Define fluid species advection module data
	Define fluid generic PDEs solver module data
	Open mesh deformation conditions window
	Define free surface (Transpiration) module data
	Define free surface ODDLS module data
	Open fluid materials window
	Open solid materials window
	Open fluid boundaries window
	Open fluid dynamics and multi-physics data window
	Open modules and solvers data window



## 2.2. Calculate

After setting up the problem, the calculation can be started by using the following command's sequence from the main menu:

### Calculate ► Calculate window

When pressing the **Start** button in the **Calculate Window**, Tdyn will start the calculation. This can also be done automatically by using the corresponding icon.

Information about the evolution of the solution can be displayed by pressing the **Output View** button. This can also be done by using the corresponding toolbar icon (see Figure below).

	Calculate icon used to start the simulation
	Information icon used to check the evolution of the calculation at runtime

## 2.3. Post-processing

When the Tdyn calculation process is finished, the system displays the message *Process...name..., started on...date...has finished.* Then the results can be visualised by using the menu sequence:

### Postprocess ► Start

Note that the problem must still be loaded in the pre-process before starting the post-process; should this not be the case we first have to open the problem files again. Note that the intermediate results can be shown at any moment of the process even if the calculations are not finished.

However, in the case of large simulations that require much computing time and RAM memory, it is advisable to quit the pre-processing environment while the process is running. It is possible to close the program while having a Tdyn process running (before closing, a warning window asking whether running processes should be killed is displayed).

Once the problem is loaded into Tdyn environment and the post-process is activated, the results file will be loaded into the post-processing part of the environment. The results it contains can be visualised using the various post-processing options of the system, like turning on and off the element sets, using contour fills, vectors, iso-surfaces, cuts, graphs, animations, etc. See the [Post-process manual](#) for further details.

It is also possible to visualize some information and graph evolutions concerning forces and moments on bodies, body movements and convergence norms. Such kind of results can also be accessed during the calculation and without loading the post-processor. To this aim, just use the following menu sequences within Tdyn pre-processor:

### Utilities ► Forces on boundaries

### Utilities ► Forces graph

### Utilities ► Motions graph

### Utilities ► Norms graph

### 3. Tdyn Reference

This chapter gives a brief explanation of every option available in Tdyn CFD+HT.

#### 3.1. Item help

It is possible to obtain help for several items in the toolkit and windows by pressing right mouse button on them.

#### 3.2. Fluid Dyn. & Multi-Phy. Data

Fluid Dyn. & Multi-Phy data refers to all the information required for performing the analysis and it does not concern any particular module. Fluid Dyn. & Multi-Phy data also differs from the previous definitions of conditions and materials properties, which are assigned to different entities. Some examples of general Fluid Dyn. & Multi-Phy data are the type of solution algorithm used by the solver, the value of the time step, convergence conditions and so on.

##### 3.2.1. Options available in Problem section

This group of data refers to the selection of problems to be solved with Fluid Dynamics and Multi-physics.

*Solve fluid:* Select this option if you are going to solve any fluid problem. If this option is not selected, any defined fluid domain will be ignored in the solution of the problem. Several options exist for *Solve fluid*:

*Solve Fluid Flow:* Select this option if you are going to solve fluid flow (RANSE) problem. This option will only be available in Fluid Flow module

*Solve heat transfer:* Select this option to solve a heat transfer problem in a fluid. If this option is not selected, the temperature problem in fluid domains will be ignored in the solution process. This option will only be available in Heat Transfer module

*Solve Species advection:* Select this option to solve a species advection problem in fluid. If this option is not selected, the species advection problem in fluid domains will be ignored in the solution. This option will only be available in Species Advection module.

*Solve PDEs problems:* Select this option to solve any user defined PDE (phi variables) problem in fluid. If this option is not selected, the user defined PDE's problem in fluid domains will be ignored in the solution. This option will only be available in PDE's solver module.

*Solve free surface (ODDLS):* Select this option to solve free surface problems in fluid based on ODD level set. This option will only be available in ODDLS module.

*Solve free surface (Transpiration):* Select this option to solve a transpiration free surface problem in

fluid. If this option is not selected, the transpiration free surface problem in fluid domains will be ignored in the solution. This option is only available in Transpiration module of 3D analysis.

*Solve mesh deformation:* Select this option to apply mesh deformation algorithms and apply Arbitrary Lagrangian Eulerian (ALE) solvers in fluid. This option will only be available in Mesh Deformation module.

*Solve comfort:* Select this option to solve comfort problems in fluid domains. This option will only be available in Comfort module.

*Solve solid:* Select this option if you are going to solve any solid problem. If this option is not selected, any defined solid domain will be ignored in the solution of the problem. Several options exist for *Solve solid*:

*Solve Solid Flow:* Select this option if you are going to solve fluid flow problem in a solid (flow in porous media). This option will only be available in Flow in Solids module

*Solve heat transfer:* Select this option to solve a heat transfer problem in a solid. If this option is not selected, the temperature problem in solid domains will be ignored in the solution process. This option will only be available in Heat Transfer module

*Solve species advection:* Select this option to solve a species advection problem in solid. If this option is not selected, the species advection problem in solid domains will be ignored in the solution. This option will only be available in Species Advection module.

*Solve PDEs Problems:* Select this option to solve any user defined PDE (phi variables) problem in solid. If this option is not selected, the user defined PDE's problem in solid domains will be ignored in the solution. This option will only be available in PDE's solver module.

##### 3.2.2. Options available in Analysis section

*Number of Steps:* Number of steps of the simulation. Total physical time to be simulated will be Number of Steps x Time increment. Recommended values to achieve steady state is:

$$\text{Number of Steps} \geq 1000 \cdot dt \cdot V / L_D$$

where dt is the time increment and V,  $L_D$  the characteristic velocity and length.

*Time Increment:* Time step of the simulation. Total physical time to be simulated will be Number of Steps x . The recommended value is:



$$dt = C \cdot L_D / V$$

where  $dt$  is the time increment,  $V$ ,  $L_D$  the characteristic velocity and length and  $0.1 < C < 0.01$ .

In case a transient phenomenon of characteristic time or period ( $T$ ) is expected, then  $dt$  can be calculated as  $1/10$  to  $1/100$  of  $T$  of the problem is usually more appropriate.

*Remarks:*

In any case, it is important to verify if the  $dt$  calculated with the above formulae is adequate for the mesh used. This can be done by evaluating a characteristic mesh time as:  $dt_m = h/V$ , where  $h$  is the characteristic mesh size (usually the smallest element size). It is recommended the  $dt$  used in the calculations to be between  $2 \cdot dt_m < dt < 20 \cdot dt_m$ .

*Time increment* may also be defined by a global function (see Function Syntax section for further information). Units of the time step of the simulation are given in the menu next to this field.

*Max Iterations:* Maximum number of iterations of the non-linear algorithm for solution of the problem. Recommended values come from 3 to 10, depending on the value of the convergence norms (see Modules Data section).

*Remarks:*

In some cases the algorithm may not converge in the initial time-steps, due to the start up process, resulting in the appearing of a warning message *More than...number...iterations may be necessary*. If only the steady state is of interest, this message may be simply ignored, otherwise *Max Iterations* value should be increased.

*Initial Steps:* During first *Initial Steps* some controls are carried out in the algorithm in order to stabilise the problem during the start up process. It is strongly recommended to define *Initial Steps* about 10% of the *Number of Steps* in problems with free surface transpiration.

*Start Up Control:* if activated, during first *Initial Steps* the start up process is smoothed. This can be done by creating a adequate acceleration in the flow (*Speed*), by smoothly increasing the time increment (*Time*) or *Both*.

*Restart:* if *On*, the restart file is used to define the initial data. The Restart file taken will be 'ProblemName.flavia.rst'. This file is automatically written with the rest of the results. To restart a case, the *Number of Steps* must be increased in the number of new steps to be run.

*Remarks:*

Note that the *Number of Steps* must be set to a number greater than the last step reached in the previous calculation. For example, if one should want to restart and perform a calculation of 100 steps, and the previous calc. reached 600, the *Num. of Steps* should be set to 700.

*Processor unit:* it allows to choose between CPU and CPU+GPU processor options. If CPU mode is selected the entire calculation

is performed in the Computer Processor Unit. On the contrary, if CPU+GPU mode is selected the numerical solver runs on the Graphical Processor Unit if this type of device is available in the computer being used. CPU+GPU mode tries to benefit from the increasing computational power of modern GPU devices in order to increase solver performance.

*Multiprocessor mode:* it allows to select the parallel execution mode. By default, Parallel mode is used so that the program automatically makes use of the maximum available number of logical CPU cores. If Sequential mode is used instead, the solver runs sequentially so that the program executes in a single logical processor. If the User Defined option is chosen, the user is allowed to select the actual number of logical CPU cores to be used during the calculation.

*Number of CPU's:* Number of CPU's is the number of processors to be used on a parallel computation. It must be less or equal to the maximum number of available processors in the current computer. In multi-core CPU machines, *Number of CPU's* actually refers to the total number of independent cores.

*Use Hypre Solvers:* it allows to activate/deactivate the use of Hypre' solvers. Hypre is a software library of high performance preconditioners and solvers for the solution of large, sparse linear systems of equations on parallel computers developed by the LLNL (Hypr's site). It has been introduced in Tdyn to provide the capability of running parallel jobs using the message passing interface (MPI) paradigm.

*MPI:* it allows to activate/deactivate the message passing interface (MPI) parallel mode. It is only available when the *Hypr Solvers* option is active. Depending on the actual architecture and/or operating system of the computer, MPI execution may also require the installation of third party software responsible for the management of the parallel processes execution (see additional information on the [CompassIS](#) webpage).

*Number of MPI nodes:* when using the MPI parallel execution mode, this entry allows the user to specify the number of calculation nodes to be used for parallel execution. Based on this information, Tdyn will automatically perform the required domain decomposition before running the calculation.

*Steady State solver:* if *On*, it starts the calculation procedure for an automatic search of the steady state.

### 3.2.3. Options available in Results section

*Output Step:* Each *Output Step* time steps the results will be written to disk.

*Remarks:*

This value will control the size of the results file.

*Output Start:* The results will be written each *Output Step* time steps after *Output Start* steps.

*Remarks:*

This value will control the size of the results file.

*Results File:* Type of the results file (Binary, Binary2, ASCII or EnSightGold). Binary or Binary2 type should be selected in order to minimise the size of the results file. Binary2 must be used to take advantage of the CompassFEM custom postprocessor.

*Fluid Flow module:* Options available with Fluid Flow module

selected.

*Initial Data:* Mark to write in the results file the initial data of the problem.

*Velocity:* Mark to write velocity field in the results file.

*Velocity Stress Tensor:* Mark to write velocity stress tensor field in the results file.

*Pressure:* Mark to write pressure field in the results file.

*Pressure Gradient:* Mark to write pressure field in the results file.

*Total Pressure:* Mark to write total pressure field in the results file (including hydrostatic component).

*Density:* Mark to write fluid density field in the results file.

*Viscosity:* Mark to write viscosity field in the results file.

*Wall Law Traction:* Mark to write wall stress given by the Law of the Wall (if exists) in the results file.

*Tau Parameter:* Mark to write tau parameter (local Courant number) field in the results file.

*Eddy Viscosity:* Mark to write eddy viscosity field in the results file.

*Eddy Kinetic Energy:* Mark to write eddy kinetic energy field in the results file.

*Epsilon:* Mark to write epsilon (turbulence variable) field in the results file.

*Omega:* Mark to write omega (turbulence variable) field in the results file.

*K Tau:* Mark to write  $k\tau$  variable (of  $K_{KT}$  turbulence model) field in the results file.

*Heat Transfer module:* Options available with Heat Transfer module selected.

*Temperature:* Mark to write temperature field in

the results file.

*Temperature Gradient:* Mark to write temperature gradient field in the results file.

*Heat Flux:* Mark to write heat flux through the boundaries.

*Solid Density:* Mark to write solid density field in the results file.

*Species Advection module:* Options available with Species Advection module selected.

*Species Concentration:* Mark to write species concentration field in the results file.

*PDE's solver module:* Options available with PDE's solver module selected.

*Phi Variable:* Mark to write phi variables field in the results file.

*Mesh Deformation module:* Options available with Mesh Deformation module selected.

*Mesh Deformation:* Mark to write mesh deformation in the results file.

*ALE Velocity:* also referred as Eulerian velocity. Mark to write the velocity given in the moving reference frame.

*Free surface module:* options available with free surface module.

*Wave Elevation:* Mark to write wave elevation field in the results file.

*Wave Elevation Vector:* Mark to write wave elevation vector field in the results file.

*Comfort module:* options available with comfort module.

*PMV:* Mark to write PMV index results (*Predicted Mean Vote*).

*PPD:* Mark to write PPD index results (*Predicted Percentage Dissatisfied*).

*User defined functions module:* options available to provide user defined results. Each custom results may be written as a function of already available problem variables.

*Fluid Function #:* Mark to write the function field (only evaluated in fluid domain) in the results file.

The function field is written in IS units in the analysis group USERDEF. If this file is marked, two new field will be available.

*Name:* Name of the function. The corresponding field is identified with this name in the post-processing part.

*Function:* Insert the fluid function to be evaluated and written. See Function Syntax section for further information.

*Solid Function #:* Mark to write the function field (only evaluated in solid domain) in the results file. The function field is written in IS units in the analysis group USERDEF. If this file is marked, two new field will be available.

*Name:* Name of the function. The corresponding field is identified with this name in the post-processing part.

*Function:* Insert the fluid function to be evaluated and written. See Function Syntax section for further information.

### 3.2.4. Options available in Fluid Solver section

This group of data refers to all the information required to define the integration scheme and solver data of the problem/s to be analysed in the fluid domain.

*Flow Solver Model:* Flow solver model used in the fluid domain. Available options are *Incompressible*, *PrCompressible* (compressible algorithm using pressure as main variable) and *DnCompressible* (compressible algorithm using density as main variable).

*Incompressible* model is adequate for those problems where the compressibility effects are small, as happens in open flows with characteristic Mach number below 0.4. It can handle small compressibility effects using the *SlightlyIncompressible* fluid model algorithm. See Materials section for further information.

*PrCompressible* is a compressible using pressure as main variable. This model is suitable for most of the practical cases. However it can not handle shock waves.

*DnCompressible* model is the most suitable for those problems where compressible effects are quite relevant. It can even simulate shock waves.

*Time Integration:* Time integration scheme used in the solution process of the fluid problem:

*Backward Euler:* Implicit 1st order scheme.

*Crank Nicolson :* Implicit 2nd order scheme.

*Solver NonSymmetric:* Solver type used in the solution of the non-symmetric linear systems of equations.

*BiConjugateGradient:* Biconjugate gradient solver.

*StabBiConjugateGradient:* Stabilised biconjugate gradient solver.

*GMRes:* Generalised minimum residual solver.

*SquaredConjugateGradient:* Squared conjugate gradient solver.

*GaussSeidel:* Gauss Seidel solver.

*Direct:* Parallel Sparse Direct Solver.

*Tolerance:* Tolerance used in the solution of the non-symmetric linear systems of equations (see *Solver NonSymmetric*). A value smaller than  $1.0 \cdot 10^{-6}$  is recommended.

*Max. Iterations:* Maximum number of iterations of the non-symmetric linear systems of equations (see *Solver NonSymmetric*).

*Preconditioner:* Preconditioner used in the solution of the non-symmetric linear systems of equations (see *Solver NonSymmetric*).

*Remarks:*

In some cases using elements with high aspect ratio the diagonal preconditioner may work better than others.

*Krilov sp. dimension:* Dimension of internal direct solver used in GMRes solver (see *Solver NonSymmetric*). A value greater than 20 is recommended.

*Solver Symmetric:* Solver type used in the solution of the symmetric linear systems of equations.

*BiConjugateGradient:* Biconjugate gradient solver.

*StabBiConjugateGradient:* Stabilised biconjugate gradient solver.

*GMRes:* Generalised minimum residual solver.

*SquaredConjugateGradient:* Squared conjugate gradient solver.

*GaussSeidel:* Gauss Seidel solver.



*Direct:* Parallel Sparse Direct Solver.

*Tolerance:* Tolerance used in the solution of the symmetric linear systems of equations (see *Solver Symmetric*). A value smaller than  $1.0 \cdot 10^{-6}$  is recommended.

*Max. Iterations:* Maximum number of iterations of the symmetric linear systems of equations (see *Solver Symmetric*).

*Preconditioner:* Preconditioner used in the solution of the symmetric linear systems of equations (see *Solver Symmetric*).

*Remarks:*

In some cases using elements with high aspect ratio the diagonal preconditioner may work better than others.

*Krilov sp. dimension:* Dimension of internal direct solver used in GMRes solver (see *Solver Symmetric*). A value greater than 20 is recommended.

*Advection Norm:* Euclidean convergence norm of the velocity, used for recalculating or not advective terms. A value smaller than  $1.0 \cdot 10^{-5}$  is recommended.

*Steady State Norm:* Euclidean norm used to detect the steady state. If each variable increment is smaller than this norm, the problem is stopped and results are written to the disk.

*Increment Control:* This option activates a control that limits the maximum admissible increment of the variables for every iteration. The limit is taken as ratio of the convergence norm of the variable. Select None to switch this control off.

### 3.2.5. Options available in Solid Solver section

This group of data refers to all the information required to define the integration scheme and solver data of the problem/s to be analysed in the solid domain.

*Flow Solver Model:* Flow solver model used in the fluid domain. Available options are *Incompressible*, *PrCompressible* (compressible algorithm using pressure as main variable) and *DnCompressible* (compressible algorithm using density as main variable).

*Incompressible* model is adequate for those problems where the compressibility effects are small, as happens in open flows with characteristic Mach number below 0.4. It can handle small compressibility effects using the *SlightlyIncompressible* fluid model algorithm. See Materials section for further information. See Materials section for further information.

*PrCompressible* is a compressible using pressure as main variable. This model is suitable for most of the practical cases. However it can not handle shock waves.

*DnCompressible* model is the most suitable for those problems where compressible effects are quite relevant. It can even simulate shock waves.

*Time Integration:* Time integration scheme used in the solution process of the solid problem:

*Backward Euler:* Implicit 1st order scheme.

*Crank Nicolson :* Implicit 2nd order scheme.

*Solver Symmetric:* Solver type used in the solution of the

symmetric linear systems of equations.

*BiConjugateGradient:* Biconjugate gradient solver.

*StabBiConjugateGradient:* Stabilised biconjugate gradient solver.

*GMRes:* Generalised minimum residual solver.

*SquaredConjugateGradient:* Squared conjugate gradient solver.

*GaussSeidel:* Gauss Seidel solver.

*Direct:* Parallel Sparse Direct Solver.

*Tolerance:* Tolerance used in the solution of the symmetric linear systems of equations (see *Solver Symmetric*). A value smaller than  $1.0 \cdot 10^{-6}$  is recommended.

*Max. Iterations:* Maximum number of iterations of the symmetric linear systems of equations (see *Solver Symmetric*).

*Preconditioner:* Preconditioner used in the solution of the symmetric linear systems of equations (see *Solver Symmetric*).

*Remarks:*

In some cases using elements with high aspect ratio the diagonal preconditioner may work better than others.

*Krilov sp. dimension:* Dimension of internal direct solver used in GMRes solver (see *Solver Symmetric*). A value greater than 20 is recommended.

*Steady State Norm:* Norm used to detect the steady state. A value smaller than  $1.0 \cdot 10^{-5}$  is recommended.

*Increment Control:* This option activates a control that limits the maximum admissible increment of the variables for every iteration. The limit is taken as ratio of the convergence norm of the variable. Select None to switch this control off.

### 3.2.6. Options available in Other section

#### User Defined Integrals

Within this section of the data tree, the user can define fluid and solid volumetric integrals of any of the calculation variables. This integrals will be calculated for each time step.

#### Tcl data

*Use Tcl External Script:* If the check-box is selected, the Tcl extension is activated. The entry may indicate a Tcl script to be interpreted during execution. The Tcl script can define any of the standard program Tcl functions. See section Tcl Extension for further information about Tcl extension.

#### Other data

*Warn. Level:* If **None**, warning messages are not shown during the calculation process. Other possibilities are **Few**, **Some** or **All**.

*Multiple Runs & Additional Steps:* This options allows the user to create a vector of *factors*, which will affect the velocity field each additional run (which will run for # *additional\_steps*). For example, if *multiple\_runs*=[1.0 1.51.6] and *additional\_steps*=100,

then, when the first run finishes, a new run will start, for 100 steps, with the velocity field multiplied by 1.5. Again, when it finishes, the resulting vel. field will be multiplied by 1.6, and the calculation will run for another 100 steps.

*Mesh Refinement Type:* This option tells Tdyn to calculate some mesh correction parameters, while the analysis is running. These *corrections* are written in a file (background mesh, \*.bgm), and the mesher will read this file, modifying the mesh assigned sizes, in order to improve it.

### 3.3. Modules Data

Modules data refers to all the specific information needed to perform a particular Tdyn CFD+HT analysis. See Introduction section for more information about Tdyn CFD+HT modules.

**Tdyn CFD+HT Modules Data options** can be set from **Modules Data** in the tree.

#### 3.3.1. Fluid Flow data

##### General

*Use Total Pressure:* Mark if you want to use total pressure (including fluid-static term) as internal variable in the solution of the fluid flow problem.

*Remarks:*

In most of the cases the solution of the fluid problem without fluid-static term is the most accurate one. The "Use total pressure" option is hence deactivated by default since the pressure calculation algorithm is usually more precise. It is recommended to activate this option only when fluid-static (or more usually hydrostatic) effects are expected to have a significant effect in the problem under analysis. If this option is selected, please check the correctness of the pressure boundary conditions. You must ensure that such conditions take into account the fluid-static pressure term.

*Pressure reference location:* Mark if you want to define the origin of the fluid-static pressure term.

*Pressure Origin:* Coordinates of the total pressure origin.

*Xplane Symmetry in Fluid:* Mark if you want to define symmetry planes in the fluid problem, perpendicular to OX axis.

*Xplane Symmetry Position:* Position of the symmetry planes in the fluid problem, perpendicular to OX axis, given in the units of the geometry.

*Yplane Symmetry in Fluid:* Mark if you want to define symmetry planes in the fluid problem, perpendicular to OY axis.

*Yplane Symmetry Position:* Position of the symmetry planes in the fluid problem, perpendicular to OY axis, given in the units of the geometry.

*Zplane Symmetry in Fluid:* Mark if you want to define symmetry planes in the fluid problem, perpendicular to OZ axis.

*Zplane Symmetry Position:* Position of the symmetry planes in the fluid problem, perpendicular to OZ axis, given in the units of the geometry.

*Operating Pressure:* this is the reference pressure for compressible solvers. It is always taken into account to evaluate compressibility effects. The user must introduce pressure boundary conditions in accordance to the operating pressure value introduced in this field. In this sense, if for instance the atmospheric pressure is the actual value of the operating pressure introduced in this field, then you can fix the outlet boundary condition equal to zero and the inlet pressure equal to the actual inlet overpressure. On the contrary, if you use a zero pressure value as the operating pressure, then you must fix the outlet pressure boundary condition equal to the atmospheric pressure, and the inlet boundary condition equal to the atmospheric pressure plus the corresponding overpressure. In fact, both cases will provide the correct density, but only using the second approach you will also obtain the actual absolute pressure and total force over bodies.

### Algorithm

*Velocity Advect Stabilisation:* Order of the FIC advection stabilisation term in the Navier Stokes equations. Three available options are **Auto**, **4th\_Order** and **2nd\_Order**.

*Remarks:*

The 4<sup>th</sup> order term increases the accuracy of the solution and is recommended in most of the cases. However in some problems it may cause instabilities.

**Auto** mode will automatically switch between 4<sup>th</sup> and 2<sup>nd</sup> order scheme, depending on the smoothness of the solution.

*Velocity Control Level:* Level of control of instabilities (0 means Off). If instabilities are found in the velocity field when using the **2nd\_Order Velocity Advect Stabilisation**, first try to reduce *Time Increment*, then to increase this value. Note that high values may cause over-diffusive results.

*TauCalcType:* This indicates the method for calculating the stabilisation parameter *tau*. This should not be changed from its default value (*geometrical*), for most of the cases. Nevertheless, *analytical* method gives good results in cases where boundary layer mesh is involved.

*StabTauV MinRatio:* Minimum admissible ratio ( $\tau/dt$ , being  $dt$  the time increment) for the stabilisation parameter  $\tau$  of the velocity solver. It will be also used for temperature and advection of species problems.

*Remarks:*

Advection stabilisation term is proportional to the parameter  $\tau$ . In most of the cases, the minimum value of this parameter should not be fixed (i.e.  $\tau/dt = 0.0$ ), otherwise oscillations may appear.

*Velocity Inner Iterations:* Number of iterations of the inner nonlinear fluid flow momentum eq. solver (performed every external iteration).

*Velocity Norm:* Velocity Euclidean norm used to check convergence in the non-linear iteration loop.

*Velocity Boundary type:* AdvancedVBC implements specific treatment of boundary conditions for momentum equation in those boundaries with transient velocity conditions.

*Pressure Stabilisation:* Scheme to be used in the stabilisation of the Pressure solver of the Navier Stokes equations.

*StabTauP min. ratio:* Minimum admissible ratio ( $\tau/dt$ ) for the stabilization parameter  $\tau$  used in the pressure stabilization.

*StabTauP max. ratio:* Maximum admissible ratio ( $\tau/dt$ ) for the stabilization parameter  $\tau$  used in the pressure stabilization.

*Pressure Inner Iterations:* Number of iterations of the inner nonlinear Navier Stokes pressure solver (performed every external iteration).

*Pressure Norm:* Pressure Euclidean norm used to check convergence in the non-linear iteration loop.

**Pressure Boundary type:** AdvancedPBC implements specific treatment of boundary conditions for mass balance equation in those boundaries with transient velocity conditions.

**Initialise Flow Field:** If *Potential\_Flow* is selected, then the initial velocity and pressure field is taken from the adaptation of the solution of a potential flow problem, trying to the imposed boundary conditions. The available options are *None*, *Potential\_Flow* and *Stokes\_Flow*.

**Floatability by Density:** This must be activated if the floatability forces existing in fluids, due to changes in density, are to be simulated.

## Turbulence

**Turbulence Model:** Select the turbulence model to be used in the solution of the fluid flow problem.

**Laminar:** Navier Stokes equations are solved (i.e. Reynolds stress tensor is neglected and therefore only direct simulation of turbulence is done).

**Mixing\_Length:** Basic turbulence model based in the Prandtl hypothesis, where the turbulence length scale (L) is given in the *EddyLen Field* entry.

**Smagorinsky:** Basic large eddy simulation (LES) turbulence model. The implementation includes an eddy viscosity damping in the boundary layer area. See Turbulence Modelling section for further information.

**Kinetic\_Energy:** Prandtl's one equation (k) model for turbulent flows with integration to the wall, where the turbulence length scale (L) is given in the *EddyLen Field* entry.

**K\_Energy\_Two\_Layers:** Prandtl's one equation (k) model for turbulent flows with integration to the wall, where the turbulence length scale (L) is given in the *EddyLen Field* entry. The implementation of this model includes an eddy viscosity damping in the boundary layer area.

**K\_E\_High\_Reynolds:** Two-equation  $k-\varepsilon$  model for turbulent flows. The model implemented is based on the standard formulation with some modifications to be used with different wall boundary conditions. See Turbulence Modelling section for further information.

**K\_E\_Two\_Layers:** Two-equation  $k-\varepsilon$  model for turbulent flows with integration to the wall. This implementation uses the high-Re  $k-\varepsilon$  model only away from the wall in the fully turbulent region, and the near-wall viscosity affected layer is resolved with a one-equation model involving a length-scale prescription. See Turbulence Modelling section for further information.

**K\_E\_Lam\_Bremhorst:** Two-equation  $k-\varepsilon$  model for turbulent flows with integration to the wall. The model implemented is based on the description done by Lam-Bremhorst with some modifications to be used with different wall boundary conditions. See Turbulence Modelling section for further information.

**K\_E\_Launder\_Sharma:** Two-equation  $k-\varepsilon$  model for turbulent flows with integration to the wall. The model implemented is based on the description done by Launder and Sharma with some modifications to be used with different wall boundary conditions. See Turbulence Modelling section for further information.

**K\_Omega:** Two equation  $k-\omega$  model for turbulent flows with integration to the wall. The model implemented is based on the description done by Wilcox with some modifications to be used with different wall boundary conditions. See Turbulence Modelling section for further information.

**K\_Omega\_SST:** Two-equation model for turbulent flows with integration to the wall, expressed in terms of a  $k-\omega$  model formulation. The  $k-\omega$  SST shear-stress-transport model combines several desirable elements of standard  $k-\varepsilon$  and  $k-\omega$  models. See Turbulence Modelling section for further information.

**K\_KT:** Two-equation  $k-k\tau$  model for turbulent flows with integration to the wall. The model implemented is based on the description done by Wilcox with some modifications to be used with different wall boundary conditions. See Turbulence Modelling section for further information.

**Spalart\_Allmaras:** One equation model for turbulent flows with integration to the wall. See Turbulence Modelling section for further information.

**ILES:** Implicit LES model based on Finite Increment Calculus formulation.

**Remarks:**

*For further information about the turbulence models and how to solve turbulence flows, please consult the Tdyn's Turbulence Handbook.*

**Turbulence Advect Stabilisation:** Order of the FIC advection stabilisation term in the turbulence equations. Three available options are *Auto*, *4th\_Order* and *2nd\_Order*.

**Remarks:**

The 4<sup>th</sup> order term increases the accuracy of the solution and is recommended in most of the cases. However in some problems it may cause instabilities.

*Auto* mode will automatically switch between 4<sup>th</sup> and 2<sup>nd</sup> order scheme, depending on the smoothness of the solution.

**Turbulence Control Level:** Level of control of instabilities for turbulence (0 means Off). If unstabilities are found in the eddy viscosity field when using the *2nd\_Order Turbulence Advect Stabilisation*, first try to reduce *Time Increment* and refine the mesh when possible, then to increase this value. Note that high values may cause over-diffusive results.

**Turbulence Inner Iterations:** Number of iterations of the inner nonlinear turbulence solver (performed every external

iteration).

Advanced turbulence options can be accessed by using the following option of the tree to open the *Ransol module Advanced data window*:

**Modules data ► Fluid flow ► Turbulence ► More...**

The advanced options available in the contextual window are detailed in what follows:

**Fix Turbulence on Bodies:** If Yes is selected, turbulence variables will have a fixed value, given by the selected law of the wall on the bodies surfaces. If No is selected, natural boundary condition will be applied.

**Tvisco Min Ratio:** Eddy viscosity ratio with the minimum of initial values of the eddy viscosity, used to calculate the minimum admissible value.

**Tvisco Max Ratio:** Eddy viscosity ratio with the maximum of initial values of the eddy viscosity, used to calculate the maximum admissible value (> 1.0).

**Kenergy Min Ratio:** Eddy kinetic energy ( $k$ ) ratio with maximum of the initial values of  $k$ , used to calculate the minimum admissible value.

**Kenergy Max Ratio:** Eddy kinetic energy ( $k$ ) ratio with maximum of the initial values of  $k$ , used to calculate the maximum admissible value.

**Epsilon Min Ratio:** Epsilon ( $\epsilon$ ) ratio with the maximum of initial values of  $\epsilon$ , used to calculate the minimum admissible value.

**Epsilon Max Ratio:** Epsilon ( $\epsilon$ ) ratio with the maximum of initial values of  $\epsilon$ , used to calculate the maximum admissible value.

**Omega Min Ratio:** Omega ( $\omega$ ) ratio with the maximum of initial values of  $\omega$ , used to calculate the minimum admissible value.

**Omega Max Ratio:** Omega ( $\omega$ ) ratio with the maximum of initial values of  $\omega$ , used to calculate the maximum admissible value (> 1.0).

**K Tau Min Ratio:** K Tau ( $k\tau$ ) ratio with the maximum of initial values of  $k\tau$ , used to calculate the minimum admissible value.

**K Tau Max Ratio:** K Tau ( $k\tau$ ) ratio with the maximum of initial values of  $k\tau$ , used to calculate the maximum admissible value.

**Turbulence\_Control\_Level:** Level of turbulence stabilisation control (0 means Off). If instabilities are found in the eddy viscosity field, refine the mesh when possible, reduce Time\_Increment or increase this value. Note that too high values may cause over-diffusive eddy viscosity results. Recommended value is 2.

**EddyKEnergy\_Production\_Limit:** Maximum ratio between the Eddy kinetic energy production and reaction term. This limiter may prevent the unrealistic buildup of eddy viscosity in the stagnation region of the bodies. Recommended value is 20.0.

**Epsilon\_Production\_Limit:** Maximum ratio between the Epsilon production and reaction term.

**Epsilon\_Reaction\_Limit:** Maximum ratio between the Epsilon reaction and production term.

**Omega\_Production\_Limit:** Maximum ratio between the Omega production and reaction term.

**Omega\_Reaction\_Limit:** Maximum ratio between the Omega reaction and production term.

**EddyViscoT\_Production\_Limit:** Maximum ratio between the

Spallart-Almarax model production and reaction term.

**EddyViscoT\_Reaction\_Limit:** Maximum ratio between the Spallart-Almarax model reaction and production term.

### 3.3.2. Mesh Deformation data

**Fluid Mesh Deformation:** Mesh updating in fluid domain may be done by three different procedures:

**Lagrangian update:** Mesh deformation is performed following the velocity of the fluid. The following equations must be entered in Fluid deformation increment:

- OX:  $v_x \cdot dt$
- OY:  $v_y \cdot dt$
- OZ:  $v_z \cdot dt$

**ByBodies:** Mesh deformation only takes into account the movement of the defined bodies.

**ByFunctions:** Mesh deformation is performed following the values given in the *Fluid Deformation Increment* field.

**ByAllData:** Mesh deformation algorithm try to fulfil all the requirements (movement of bodies, deformation given in *Fluid Deformation Increment* field and boundary conditions).

**Update fluid mesh every (steps):** Mesh updating in fluid domain is carried out every *Update fluid mesh every (steps)* steps. If set to zero, the mesh deformation is just done before the first time step.

**Fluid Deformation Increment:** Functions defining fluid mesh deformation have to be inserted here. These functions must define the deformation increment for every time step. For example, a planar rotation around origin (0,0,0) may be defined by inserting the functions  $xrot(w \cdot dt)$ ,  $yrot(w \cdot dt)$ , 0.0, being  $w$  the angular velocity. Note that if the geometry units and the deformation units are different, since  $xrot$  and  $yrot$  are evaluated using internal units, the returning values have to be multiplied by the unit conversion factor.

**Solid Mesh Deformation:** Mesh updating in solid domain may be done by three different procedures:

**ByBodies:** Mesh deformation only takes into account the movement of the defined bodies.

**ByFunctions:** Mesh deformation is performed following the values given in the *Solid Deformation Increment* field.

**ByAllData:** Mesh deformation algorithm try to fulfil all the requirements (movement of bodies, deformation given in *Solid Deformation Increment* field and boundary conditions).

**Update solid mesh every (steps):** Mesh updating in solid domain is carried out every *Update solid mesh every (steps)* steps.

**Solid Deformation Increment:** Functions defining solid mesh deformation have to be inserted here. These functions must define the deformation increment for every time step. For example, a planar rotation around origin (0,0,0) may be defined by inserting the functions  $xrot(w \cdot dt)$ ,  $yrot(w \cdot dt)$ , 0.0, being  $w$  the angular velocity. Note that if the geometry units and the deformation units are different, since  $xrot$  and  $yrot$  are evaluated using internal units, the returning values have to be multiplied by the unit conversion factor.

**Movement stabilisation factor:** This factor is used to increase stability of the body movement. Higher values (>0.1) can produce compressibility effects which are necessary in the case of impact problems. Increase of this parameter, should be followed by an increase of *Pressure Inner Iterations* value.



### 3.3.3. Heat Transfer data

*Temp. Advect Stabilisation:* Order of the FIC advection stabilisation term in the temperature equation. Three available options are *Auto*, *4th\_Order* and *2nd\_Order*.

*Remarks:*

The 4<sup>th</sup> order term increases the accuracy of the solution and is recommended in most of the cases. However in some problems it may cause instabilities.

*Auto* mode will automatically switch between 4<sup>th</sup> and 2<sup>nd</sup> order scheme, depending on the smoothness of the solution.

*Temp. Control Level:* Level of control of instabilities (0 means Off). If instabilities are found in the velocity field when using the *2nd\_Order Temp. Advect Stabilisation*, first try to reduce *Time Increment*, then to increase this value. Note that high values may cause over-diffusive results.

*Temp. Inner Iterations:* Number of iterations of the inner (nonlinear) temperature eq. solver (performed every external iteration).

*Temperature Norm:* Temperature Euclidean norm used to check convergence in the non-linear iteration loop.

*Prandtl Number:* Prandtl number used to include turbulence effects in the temperature calculations.

*Radiation model:* model to be used in problems that involve heat transfer by radiation. There are currently two different radiation models available in Tdyn CFD+HT, P-1 and surface to surface (S2S) radiation models. The P-1 radiation is the simplest case of the more general P-N model. It is intended to be used in modelling problems that involve participating media, since it includes the effect of scattering. On the other hand, the S2S model is provided to take into account the radiation exchange in an enclosure of gray-diffuse surfaces that depends on the size, separation distance and orientation of the emitting surfaces. It implies the usage of the view factor geometric function.

*Elements per patch:* this option only concerns the S2S heat transfer by radiation model. It determines the number of elements per patch to be used in the calculation of the view factor matrix. A large number of elements per patch will reduce the computation time for the evaluation of the view factor matrix at the expense of the accuracy.

### 3.3.4. Free Surface (Transpiration) data

*Wave Elevation Norm:* Euclidean norm of wave elevation field used to check convergence in the non-linear iteration loop.

### 3.3.5. Free Surface (ODDLS) data

*Advect stabilisation:* Order of the FIC advection stabilisation term in the variable equation. Three available options are *Auto*, *4th\_Order* and *2nd\_Order*.

*Remarks:*

The 4<sup>th</sup> order term increases the accuracy of the solution and is recommended in most of the cases. However in some problems it may cause instabilities.

*Auto* mode will automatically switch between 4<sup>th</sup> and 2<sup>nd</sup> order scheme, depending on the smoothness of the solution.

*Control level:* Level of control of instabilities (0 means Off). If

instabilities are found in the variable field when using the *2nd\_OrderAdvect Stabilisation*, first try to reduce *Time Increment*, then to increase this value. Note that high values may cause over-diffusive results.

*Number of Phases:* Set to *One\_Phase* to just simulate the evolution of the primary phase (mono-phase flow with free surface). This option is useful in many cases of water-air flows, where the influence of the air movement in the water flow is negligible. *Two\_Phases* option will run the two phases flow algorithm.

*Convergence norm:* Euclidean norm used to check convergence in the non-linear iteration loop.

*Solver Scheme:* Set to *Naval* to increase the accuracy in those problems where the pressure distribution is mainly hydrostatic (it requires vertical coordinate to be parallel to Z axis).

*Reinitialisation Every (Steps):* In most of the cases, reinitialisation of the level set field is not necessary to be done every time step. This option sets the number of time steps to wait to the next reinitialisation.

*Boundary type:* AdvancedOBC implements specific treatment of boundary conditions for odd level set equation in those boundaries with fixed velocity conditions.

*Mass conservation:* If *On* is selected, additional conservation of primary phase is enforced. If *Fixed* is selected the *Mass increment* per time step is defined in the next entry. In those cases which the *Mass increment* is known, the accuracy of the results will be increases selecting the *Fixed* option and inserting the correct value in the *Mass increment* entry.

*Mass increment:* *Mass increment* per time step used to apply additional volume conservation.

*Units of the Mass increment* may be defined in the menu next to the *Mass increment* entry. It is possible to define additional units by entering new dimensionally correct units in the box (see Units Syntax section for further information).

### 3.3.6. Comfort

This data group includes different entries required for the thermal comfort module of Tdyn based on the Fanger's method.

The Fanger's method, through the calculation of the *Predicted Mean Vote* (PMV), predicts the thermal comfort, and extends the PMV to predict the proportion of dissatisfied people with the environment in terms of their comfort vote, *Predicted Percentage Dissatisfied* (PPD).

The PMV index predicts the mean response of a large group of people according to the *American Society of Heating, Refrigerating and Air-Conditioning Engineers* (ASHRAE) thermal sensation scale, from -3 (cold) to +3 (hot), where 0 represents a neutral thermal feeling.

*Mean Radiant Temperature:* the uniform temperature of an imaginary enclosure in which the radiation from the occupant equals the radiant heat transfer in the actual non-uniform enclosure.

*Relative Humidity:* a term used to describe the amount of water vapor in a mixture of air and water vapor expressed as a percentage:

$$\Phi = e_w / e_w^* \times 100\%$$



with  $e_w$ =partial pressure of water vapor ( $H_2O$ ) and  $e_w^*$ = saturated vapor pressure of water.

*Clothing Factor*: insulation of clothes measured with the unit "clo", where  $1\ clo = 0.155\ m^2K/W$ .

*Metabolic Rate*: human body heat production measured in the unit "met", where  $1\ met = 58\ W/m^2$ .

### 3.4. Conditions and Initial Data

Conditions are all properties of a problem (except **Materials**) that can be assigned to an entity, in order to define the basic boundary conditions of a problem. Conditions should be used to define inflow and outflow boundary conditions, symmetry or far field conditions, as well as complex boundary conditions like body wall type (i.e. law of the wall) or free surface.

In Tdyn CFD+HT conditions are available through the Tdyn data tree that can be accessed by using the following menu sequence:

**Data ► Data ► Conditions and Initial Data**

If a mesh has already been generated, any change in the condition assignments, requires meshing again to transfer these new conditions to the mesh. If the conditions were changed and a new mesh was not generated, the user will be warned, when the data for the analysis is being written.

#### 3.4.1. Fluid Flow Analysis Conditions

The following boundary conditions are available in the Fluid Flow Analysis of Tdyn CFD+HT, beneath the following section of the data tree.

**Conditions and Initial Data ► Fluid Flow**

#### Wall/Bodies

**Wall/Bodies** boundaries allow the user to define special boundary conditions, representing physical walls or bodies. The options available include analytical Law of the Wall as well as body motion properties. These properties can be assigned to lines (2D plane or 2D Axisymmetric), surfaces (3D) or boundary meshes.

##### Remarks:

If any entity is defined as a **Wall/Body** the graphs of the reaction forces on the fluid will be available in the post-process of Tdyn CFD+HT.

If any entity is defined as a **Wall/Body** and any movement is enabled (Mesh Deformation Analysis activated), the graphs of this movement will be available in the post-process of Tdyn CFD+HT.

In order to transfer **Wall/Body** data to the mesh, Meshing Criteria must be fixed to Yes in the corresponding geometrical entities. Note that this action is automatically done by Tdyn CFD+HT in most of the cases.

#### Options available in Wall Type page

**Fluid/Solid Wall:** Choose if the boundary condition is going to be applied either to a Fluid or a Solid domain boundary.

**BoundType:** Type of the wall boundary. Several options are available:

**InvisWall:** Impose the slipping boundary condition (i.e. wall normal velocity component will be zero).

**V\_fixWall:** Impose the null velocity condition on the boundary (i.e. velocity on the wall will be zero).

**None\_Wall:** No conditions will be applied to the boundary. This boundary type can be used to calculate forces on different parts of a body (in that case, the condition will be superimposed on the standard body condition).

**RoughWall:** Law of the wall condition, taking wall roughness into account, is applied at the wall distance  $\delta$ . See Near wall-modelling chapter below. The fluid stress (traction) given by the law of the wall at a wall distance  $\delta$  will be applied as boundary condition in the fluid solver. The wall distance must be inserted in the field *Delta* (see below).

**DeltaWall:** Extended law of the wall condition is applied on the boundary at the wall distance  $\delta$ . See Near wall-modelling chapter below. The fluid stress (traction) given by the law of the wall at a wall distance  $\delta$  will be applied as boundary condition in the fluid solver. The wall distance must be inserted in the field *Delta* (see below).

**YplusWall:** Extended Law of the wall condition is applied on the boundary at the non-dimensional wall distance  $y^+$ . See Near wall-modelling chapter below. The fluid stress (traction) given by the law of the wall at a non-dimensional wall distance  $y^+$  will be applied as boundary condition in the fluid solver. The non-dimensional wall distance must be inserted in the field *Yplus* (see below).

**Cw\_U2Wall:** A traction given by  $C_w \cdot V^2$ , where  $C_w$  is a constant and  $V$  the fluid velocity, is imposed on the boundary. The constant  $C_w$  must be inserted in the field *Cw* (see below).

**ITTC Wall:** Extended Law of the wall condition is applied on the boundary at the non-dimensional wall distance  $y^+$ . The fluid stress (traction) given by the law of the wall at a non-dimensional wall distance  $y^+$  will be applied as boundary condition in the fluid solver. This traction is corrected according to the ITTC 57 friction law. The non-dimensional wall distance must be inserted in the field *Yplus* (see below).

**User Wall:** Law of the wall formulation that can be defined by the user. It requires explicit formulation of the wall traction (see below *FTau Field*), eddy kinetic energy (see below *KEnr Field*) and the turbulence length scale (see below *Elen Field*).

**Yplus:** If *YplusWall* is selected, wall law assumption is taken up to the non-dimensional wall distance  $y^+$  given here. The fluid stress (traction) given by the law of the wall will be then applied as a boundary condition in the fluid solver. See Near wall-modelling chapter below.

**Delta:** If *DeltaWall* or *RoughWall* is selected, wall law assumption is taken up to the dimensional wall distance  $\delta$  specified here. The fluid stress (traction) given by the law of the wall will be then applied as a boundary condition in the fluid solver. See Near wall-modelling chapter below.

**Delta Units:** Units of the dimensional wall distance  $\delta$  given in the previous field.

**Roughness:** Roughness of the wall (only used if *BoundType* | *RoughWall* is selected).

**Roughness Units:** Units of the dimensional wall distance  $\delta$  given

in the previous field.

**Cw:** Constant used in the definition of *BoundType* | *Cw\_U2Wall*.

**VelX/Y/Z Field:** Field used for defining the velocity profile on the boundary for *V\_fixBound BoundType*.

**VelN Field:** Field used for defining the normal velocity profile on the boundary for *VnfixBound BoundType*.

**FTau Field:** Field of wall traction used in the definition of *BoundType* | *User Wall*. It should be a explicit function of the variables used in Tdyn CFD+HT (see Function Syntax section for further information).

**KEnr Field:** Field of eddy kinetic energy used in the definition of *BoundType* | *User Wall*. It should be a explicit function of the variables used in Tdyn CFD+HT (see Function Syntax section for further information).

**ELen Field:** Field of turbulence length scale used in the definition of *BoundType* | *User Wall*. It should be a explicit function of the variables used in Tdyn CFD+HT (see Function Syntax section for further information).

**Sharp Angle:** The slipping boundary condition for the velocity will be corrected if any internal angle of this *Fluid Body* geometry is smaller than the one inserted here (see Figure 7). In those points, the boundary condition for the velocity is ignored. This condition can be used for automatic correction of boundary conditions, in those complex geometries with trailing edges, where the *Fluid Body* normal vector is undefined.



Figure 7. Example of application of the Sharp Angle option.

**Line Fix Angle:** *Fix Velocity Direction* boundary condition will be automatically applied as boundary condition if an external angle of this *Fluid Body* geometry is smaller than the one inserted here (see Figure 8). This condition should be used to automatically impose *Fix Velocity Direction* boundary conditions, in those complex geometries with edges or significant dihedral angles, where boundary conditions imposition by hand, may take too much time.

**Remarks:**

In Tdyn CFD+HT 2D Plane a null velocity is imposed (instead of *FixVelocity Direction* condition) if an external angle of a *Fluid Body* is smaller than the one inserted here.

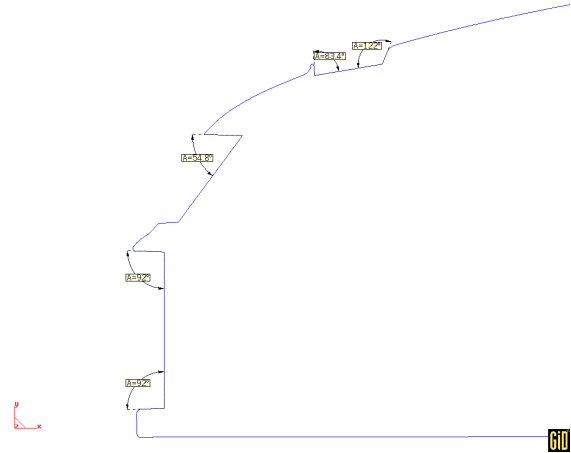


Figure 8. Example of the places where the Line Fix Angle option could be useful.

**SternC Angle:** A control for stern of bodies (in the free surface transpiration problem) is carried out. This control will be applied in those points of the floating line of the body, where the angle between the normal and the velocity is greater that the value inserted here. See Stern flow modelling in transpiration problem section.

**Remarks:**

This option is only available if the Free Surface (Transpiration) Analysis is activated.

## Options available in Body page

**Body Mass:** Mass of the body. Units of the mass field may be defined in the menu next to this entry. It is possible to define additional units by entering new dimensionally correct units in the box (see Units Syntax section for further information).

**Remarks:**

If the check box next to the *Body Mass* entry is selected, mass of the body will be estimated by Tdyn CFD+HT, based on a initial equilibrium of forces.

*Body Mass* entry will be available for all the modules of Tdyn CFD+HT, not only when the Mesh Deformation Analysis is activated.

**Center of Gravity:** Vector giving the center of gravity of the body. Units of the center of gravity may be defined in the units section of the data tree.

### General data ► Units ► Geometry units

**Remarks:**

*Center of Gravity* entry will be available for all the modules of Tdyn CFD+HT, not only when the Mesh Deformation Analysis is activated.

*Center of Gravity* can be defined by a time dependant function.

*Center of Gravity* position will be automatically updated with the movement of the Wall/Body.

**Radi-us/-i of Gyration:** Vector giving the radii of gyration of the body. Units of the radii of gyration may be defined in the units section of the data tree

### General data ► Units ► Geometry units

## Options available in Motions

**Displacement Options:** For every displacement degree of freedom there exists four possible options:

**Off:** the corresponding degree of freedom is disabled (movement is not allowed).

**On:** the corresponding degree of freedom is enabled (movement is allowed). If the value inserted in the corresponding *Displacement Values* field is different from zero for  $t = 0$ , this value will be used to define an initial movement of the body.

**Fix:** the corresponding degree of freedom is fixed to the time dependant function given by the *Displacement Values* field (movement is prescribed). This function is useful to impose rigid body motions.

**Field:** the corresponding degree of freedom is fixed to the generic function given by the *Displacement Values* field (movement is prescribed). This function is useful to impose body deformations.

**Displacement Values:** For every displacement degree of freedom, this vector gives the total displacement of the body. The corresponding fields will only be available if the *Displacement Options* field is selected as *Fix*. Units of the displacement values vector may be defined in units section of the data tree:

### General data ► Units ► Geometry units

**Rotation Options:** For every rotational degree of freedom there exists three possible options:

**Off:** the corresponding degree of freedom is disabled (rotation is not allowed).

**On:** the corresponding degree of freedom is enabled (rotation is allowed). If the value inserted in the corresponding *Rotation Values* field is different from zero for  $t = 0$ , this value will be used to define an initial movement of the body.

**Fix:** the corresponding degree of freedom is fixed to the value given by the *Rotation Values* field (rotation is prescribed). This function is useful to impose rigid body motions.

**Rotation Values:** For every rotational degree of freedom, this vector gives the total rotation of the body. The corresponding fields will only be available if the *Rotation Options* field is selected as *Fix*.

## Options available in Actions

**External Forces:** This vector defines the additional external forces (gravity forces are not included) acting on the center of gravity of the body.

**External Moments:** This vector defines the additional external moments acting on the center of gravity of the body.

**Relaxation factor:** This factor is used to increase stability of the body movement. The evaluation of the forces is relaxed, by averaging the previous and current value. Recommended value is 0.25.

**Damping factor:** Damping added to the movement of the body. If the objective of the analysis is the steady state, it is recommended to increase the value of this factor (recommended value in those cases is 0.75).

## Inlet

**Inlet** boundary condition is designed to represent a flow inlet. Use if the boundary condition is going to be applied either to a Fluid or a Solid entity.

**Inlet of:** determines whether the inlet boundary condition corresponds to a fluid or a solid.

**Boundary Type:** Type of inlet boundary. Three options are available:

**Inlet Velc:** Allows defining a velocity profile on the boundary. This profile is defined by inserting the velocity components in the fields *VelX\_Field*, *VelY\_Field* and *VelZ\_Field*.

**Inlet VelN:** Allows defining the normal velocity on the boundary, while tangential velocity is fixed to zero. This profile is defined by inserting the normal velocity component in the field *VelN\_Field*.

**Inlet Pres:** Allows defining a pressure field on the inlet boundary.

## Outlet

**Outlet** boundary condition is designed to represent a flow outlet. Use if the boundary condition is going to be applied either to a Fluid or a Solid entity.

**Outlet of:** determines whether the outlet boundary condition corresponds to a fluid or a solid.

**Boundary Type:** Type of outlet boundary. Two options are available:

**OutletPres:** Pressure field is defined at the outlet boundary.

**OutletNewm:** A Newmann boundary condition on the velocity (null derivative) is defined at the outlet boundary.

## Fix Pressure

This condition is assigned to geometrical/mesh entities or layers and is used to fix the pressure at the given value.

**Remarks:**

When the selected flow model is *PrCompressible* (compressible based on pressure) both pressure and density are prescribed when this condition is applied.

**Fields:**

**Pressure value:** value (real) of the pressure.

**Remarks:**

For most of the problems it is strongly recommended to fix the pressure in at least one point of the domain.

When the pressure is not specified in the analysis either directly or indirectly, no reference for the pressure exists, and the resulting distribution can only be used with relative values.

The pressure can be specified at a far field boundary (i.e. at the outflow boundary of the domain, when the boundary is far enough from the region of interest).

The pressure can also be specified at the inlet boundary. If the conditions at inlet are not well known, it is effective to move the boundary as far from the region of interest as possible.

When the selected flow model is *PrCompressible* (compressible based on pressure) both pressure and density are prescribed when this condition is applied.

## Conditional Pressure

This condition is assigned to geometrical/mesh entities and layers and is used to specify the pressure at the value given by the pressure entry of the conditional data (only if the *Pressure FuncCond* data field is greater than 0):

### Conditions and Initial data ► Conditional data ► PressureFuncCond

If the evaluation of the *Pressure FuncCond* field results in a value less than 0, the boundary conditions will not be applied. If the value is 0, the boundary condition will be applied only if it was applied in the previous time step.

#### Remarks:

When the selected flow model is *PrCompressible* (compressible based on pressure) both pressure and density are prescribed when this condition is applied.

When **ConditionalPressure** condition is assigned, *Pressure Field* entry is used both for defining initial values ( $t = 0$ ) when starting calculation and for evaluating the assigned condition the rest of time steps.

Entries of **Conditional** data may be defined by functions (see Function Syntax section).

*Pressure FuncCond* and *Pressure Field* entries are common for every **Conditional Pressure** condition and can only be modified in the **Conditional** data of the **Fluid Flow** conditions

In most of the problems, it is recommended to fix the pressure in at least one point of the domain. When the pressure is not specified in the analysis either directly or indirectly, no reference for the pressure exists, and the resulting distribution can only be used with relative values.

Sometimes the pressure is specified at a far field boundary (i.e. at the outflow boundary of the domain, when the boundary is far enough from the region of interest).

Sometimes the pressure is specified also at the inlet boundary. If the conditions at inlet are not well known, it is effective to move the boundary as far from the region of interest as possible.

## Pressure Field

This condition is assigned to geometrical/mesh entities and is used to fix the pressure at the value given by the *Pressure Field* entry of the initial and field data conditions in the data tree:

### Conditions and Initial data ► Initial and Conditional data ► Pressure Field

#### Remarks:

When the selected flow model is *PrCompressible*

(compressible based on pressure) both pressure and density are prescribed when this condition is applied.

#### Fields:

*Fix Initial:* The pressure will be fixed to the initial value (evaluated at  $t=0$ ) of the function inserted in the *Pressure Field* entry of the **Initial and Field** data conditions in the tree. *Pressure Field* entry is evaluated at the initial step ( $t=0$ ) and pressure is fixed to the resulting value for the rest of the execution.

*Fix Field:* The pressure will be fixed to the value (for every time step) of the function inserted in the *Pressure Field* of the **Initial and Field** data conditions in the tree. *Pressure Field* entry is evaluated every step and pressure is fixed to the resulting value.

#### Remarks:

In most of the problems, it is recommended to fix the pressure in at least one point of the domain. When the pressure is not specified in the analysis either directly or indirectly, no reference for the pressure exists, and the resulting distribution can only be used with relative values.

When **Pressure Field** condition is assigned, *Pressure Field* entry is used both for defining initial values ( $t = 0$ ) when starting calculation and for evaluating the assigned condition the rest of time steps.

*Pressure Field* value is common for every **Pressure Field** condition and can only be modified in the **Initial and Field data** conditions in the tree.

Sometimes the pressure is specified at a far field boundary (i.e. at the outflow boundary of the domain, when the boundary is far enough from the region of interest).

Sometimes the pressure is specified also at the inlet boundary. If the conditions at inlet are not well known, it is effective to move the boundary as far from the region of interest as possible.

This condition allows a definition of transient boundary conditions for the pressure. The analytical functions defining transient boundary conditions will be specified in the corresponding function inserted in the *Pressure Field* entry of the **Initial and Field data** conditions in the tree.

If the boundary conditions for the pressure are steady, this condition can be substituted by the **Fix Pressure** condition. The only difference between these two options in this case is, that when using **Pressure Field**, the value of the fixed pressure can be changed automatically in every entity by updating the corresponding function inserted in the *Pressure Field* entry of the **Initial and Field data** conditions in the tree.

Entries of the **Initial and Field data** conditions may be defined by functions (see Function Syntax section).

## Fix Velocity

This condition is assigned to geometrical/mesh entities or layers and is used to fix the velocity at the given value.

#### Fields:

*X Component:* Value (real) of the OX component of the velocity.

*Fix X:* Only if marked the OX component of the velocity will be fixed.

*Y Component:* Value (real) of the OY component of the velocity.

*Fix Y:* Only if marked the OY component of the velocity will be



fixed.

**Z Component:** Value (real) of the OZ component of the velocity.

**Fix Z:** Only if marked the OZ component of the velocity will be fixed.

**Remarks:**

The velocity has to be prescribed at the inlet boundary. If the conditions at inlet are not well known, it is effective to move the boundary as far from the region of interest as possible.

## Conditional Velocity

This condition is assigned to geometrical/mesh entities and layers and is used to specify the velocity at the value given by the corresponding *Velocity X/Y/Z Field entries* of the **Conditional data** conditions in the tree (only if the given conditional data field is greater than 0):

**Conditions and Initial data ► Conditional data ► Velocity X/Y/Z FuncCond**

If the evaluation of the corresponding *Velocity X/Y/Z FuncCond* field results in a value less than 0, the boundary conditions will not be applied. If the value is 0, the boundary condition will be applied only if it was applied in the previous time step.

**Remarks:**

When **Conditional Velocity** condition is assigned, *Velocity X/Y/Z Field* entries are used both for defining initial values ( $t = 0$ ) when starting calculation and for evaluating the assigned condition the rest of time steps.

Entries of the **Conditional data** may be defined by functions (see Function Syntax section).

*Velocity X/Y/Z FuncCond* entries are common for every **Conditional Velocity** condition and can only be modified in the **Conditional data**.

The velocity has to be prescribed at every inlet boundary. If the conditions at inlet are not well known, it is effective to move the boundary as far from the region of interest as possible.

## Velocity Field

This condition is assigned to geometrical/mesh entities and layers and is used to specify the velocity at the value given by the corresponding functions of the **Initial and Field data** section of the data tree:

**Conditions and Initial data ► Initial and Conditional data ► Initial and Field data**

**Remarks:**

Entries of the **Initial and Field data** may be defined by functions (see Function Syntax section).

**Fields:**

**Fix Initial X:** The OX component of the velocity will be fixed to the initial value (evaluated in  $t=0$ ) of the corresponding function of the **Initial and Field** data (*Velocity X Field* entry) only if the field is marked. *Velocity X Field* is evaluated at the initial step ( $t=0$ ) and velocity component is fixed to the resulting value for the rest of the execution.

**Fix Initial Y:** The OY component of the velocity will be fixed to the initial value (evaluated in  $t=0$ ) of the corresponding function of the **Initial and Field** data (*Velocity Y Field* entry) only if the field is marked. *Velocity Y Field* is evaluated at the initial step ( $t=0$ ) and velocity component is fixed to the resulting value for the rest of

the execution.

**Fix Initial Z:** The OZ component of the velocity will be fixed to the initial value (evaluated in  $t=0$ ) of the corresponding function of the **Initial and Field** data (*Velocity Z Field* entry) only if the field is marked. *Velocity Z Field* is evaluated at the initial step ( $t=0$ ) and velocity component is fixed to the resulting value for the rest of the execution.

**Fix Field X:** The OX component of the velocity will be fixed to the value (for every time step) of the corresponding function of **Initial and Field** data (*Velocity X Field* entry) only if the field is marked. *Velocity X Field* is evaluated every step and the corresponding velocity component is fixed to the resulting value.

**Fix Field Y:** The OY component of the velocity will be fixed to the value (for every time step) of the corresponding function of the **Initial and Field** data (*Velocity Y Field* entry) only if the field is marked. *Velocity Y Field* is evaluated every step and the corresponding velocity component is fixed to the resulting value.

**Fix Field Z:** The OZ component of the velocity will be fixed to the value (for every time step) of the corresponding function of the **Initial and Field** data (*Velocity Z Field* entry) only if the field is marked. *Velocity Z Field* is evaluated every step and the corresponding velocity component is fixed to the resulting value.

**Remarks:**

The velocity has to be prescribed at every inlet boundary. If the conditions at inlet are not well known, it is effective to move the boundary as far from the region of interest as possible.

When **Velocity Field** condition is assigned, *Velocity X/Y/Z Field* entries are used both for defining initial values ( $t = 0$ ) when starting calculation and for evaluating the assigned condition the rest of time steps.

This condition allows definitions of transient boundary conditions for the velocity. The analytical functions defining transient boundary conditions will be specified in the corresponding **Initial and Field** data section of the data tree.

If the boundary conditions for the velocity are steady, this condition can be substituted by the **Fix Velocity** condition. The only difference between these two options in this case is, that when using the **Velocity Field**, the value of the fixed velocity can be changed automatically in every entity by updating the corresponding **Initial and Field** data of the **Fluid Flow** conditions in the tree.

## Fix Turbulence

This condition is assigned to geometrical/mesh entities and layers and is used to fix the turbulence variables for those turbulence models based on the Reynolds extended analogy, at the initial (evaluated for  $t=0$ ) value given by the corresponding functions (*EddyKEner Field* and *EddyLength Field*) of the **Initial and Field data** section of the data tree:

**Conditions and Initial data ► Initial and Conditional data ► Initial and Field data ► EddyKEner field**

**Conditions and Initial data ► Initial and Conditional data ► Initial and Field data ► EddyLength field**

**Fields:**

**Fix:** The turbulence will be fixed to the initial value (evaluated in  $t=0$ ) of the corresponding function of the **Initial and Field data** section the tree only if this field is marked. *EddyKEner Field* and



*EddyLength Field* give the value of the eddy energy and turbulence length scale or mixing length (see Turbulence modelling section), respectively.

*Remarks:*

The turbulence variables have to be prescribed at every inlet boundary. In those entities where all components of the velocity have been prescribed, Tdyn CFD+HT automatically fixes all turbulence variables at the initial (evaluated for  $t=0$ ) value given in the *EddyKEner Field* and *EddyLength Field* entries of the **Initial and Field data** section of the tree. Therefore, assignment of this condition should not be necessary in most of the cases.

Fix Turbulence condition will only be useful for those turbulence models based on Reynolds extended analogy (see Turbulence Modelling section for further information).

## Remove Velocity Conditions

This condition is assigned to geometrical/mesh entities and layers and makes the program to ignore any specification on the velocity field.

*Fields:*

*Free Velocity:* The velocity will be removed only if this field is marked.

*Remarks:*

This option allows the solver to accomplish the Kutta-Jukowsky condition. In these cases the **Remove Velocity** condition will be assigned to the extreme point of the tail of a profile. It is also possible to automatically correct velocity impositions in these areas by using the **Wall/Bodies** options (see **Sharp Angle** option).

## Fix Velocity Direction

This condition is assigned to geometrical/mesh entities and may be used to define the direction of the velocity, according to the orientation of the skew system.

*Fields:*

*Local Axes:* Orientation of the Cartesian axes used to define the direction of the component of the velocity vector. These can be local axes of the geometry (**-Automatic-** option) or any user defined system.

*Type:* Axis of the **Local Axes** definition. The component of the velocity vector, parallel to this axis, will be fixed to the given value. The normal velocity component to a line or a surface can be fixed by selecting *Y\_Axis* or *Z\_Axis*. To see the defined **Local Axes**, press the button Draw.

*Remarks:*

Usually, the direction of the velocity has to be prescribed in some edges or areas with strong geometrical changes of the geometry.

The direction of the velocity can be automatically imposed by using the **Wall/Bodies** options (see **Fix Angle** option).

## Fix Velocity Component

This condition is assigned to geometrical/mesh and is used to specify the value of a component of the velocity vector.

*Fields:*

*Local Axes:* Orientation of the Cartesian axes used to define the direction of the component of the velocity vector. These can be

local axes of the geometry (**-Automatic-** option) or any user defined system.

*Type:* Axis of the **Local Axes** definition. The component of the velocity vector, parallel to this axis, will be fixed to the given value. The normal velocity component to a line or a surface can be fixed by selecting *Y\_Axis* or *Z\_Axis*. To see the defined **Local Axes**, press the button Draw.

*Value:* Value of the component of the velocity in the direction given by Type axis.

*Remarks:*

This option is used to prescribe slipping boundary conditions on a velocity field. Since the normal vector is sometimes undefined in some complex areas (i.e. dihedral angles) of the geometry, in some cases it is better to use the **Wall/Bodies** options instead.

## 3.4.2. Heat Transfer Analysis Conditions

The following boundary conditions are available when the Heat Transfer Analysis of Tdyn CFD+HT is activated.

### Fix Temperature

This condition is assigned to geometrical/mesh entities and layers and is used to fix the temperature in a geometrical entity or layer to the given value.

*Fields:*

*Value:* Value of the temperature.

### Conditional Temperature

This condition is assigned to geometrical/mesh entities and layers and is used to fix the temperature to the value given by the function inserted in the *Temperature FuncCond* entry of the **Conditional Data** section of the tree:

**Conditions and Initial data ► Initial and Conditional data ► Conditional data ► Temperature FuncCond**

*Fields:*

*Fix Initial:* The temperature will be fixed to the initial value (evaluated in  $t=0$ ) of the function inserted in the *Temperature FuncCond* entry of the **Conditional Data** only if the field is marked.

*Temperature Field* is evaluated initial step ( $t=0$ ) and temperature is fixed to the resulting value for the rest of the execution.

*Fix Field:* The temperature will be fixed to the value (for every time step) of the function inserted in the *Temperature FuncCond* entry of the **Conditional Data**. *Temperature FuncCond* entry is evaluated every step and temperature is fixed to the resulting value.

*Remarks:*

This condition allows definitions of transient boundary conditions for temperature. The analytical functions defining transient boundary conditions will be specified in the *Temperature FuncCond* entry of the **Conditional Data** in Heat Transfer Conditions entry of the tree.

If the boundary conditions for the temperature are steady, this condition can be substituted by the **Fix Temperature** condition. The only difference between these two options in this case is, that when using the **Temperature Field**, the value of the fixed temperature can be changed automatically in every entity by updating the *Temperature*

Field entry of the **Conditional Data**.

When **Temperature Field** condition is assigned, *Temperature Field* entry is used both for defining initial values ( $t = 0$ ) when starting calculation and for evaluating the assigned condition the rest of time steps.

## Temperature Field

This condition is assigned to geometrical/mesh entities and layers and is used to fix the temperature at the value given by the function inserted in the *Temperature Field* entry of the **Initial and Field Data** section of the tree:

**Conditions and Initial data ► Initial and Conditional data ► Initial and Field data ► Temperature field**

*Fields:*

*Fix Initial:* The temperature will be fixed to the initial value (evaluated in  $t=0$ ) of the function inserted in the *Temperature Field* entry of **Initial and Field Data** only if the field is marked.

*Temperature Field* is evaluated initial step ( $t=0$ ) and temperature is fixed to the resulting value for the rest of the execution.

*Fix Field:* The temperature will be fixed to the value (for every time step) of the function inserted in the *Temperature Field* entry of **Initial and Field Data**. *Temperature Field* entry is evaluated every step and temperature is fixed to the resulting value.

*Remarks:*

This condition allows definitions of transient boundary conditions for temperature. The analytical functions defining transient boundary conditions will be specified in the *Temperature Field* entry of **Initial and Field Data**.

If the boundary conditions for the temperature are steady, this condition can be substituted by the **Fix Temperature** condition. The only difference between these two options in this case is, that when using the **Temperature Field**, the value of the fixed temperature can be changed automatically in every entity by updating the *Temperature Field* entry of **Initial and Field Data**.

When **Temperature Field** condition is assigned, *Temperature Field* entry is used both for defining initial values ( $t = 0$ ) when starting calculation and for evaluating the assigned condition the rest of time steps.

## Heat Flux

*Heat Flux:* Heat flow (power) entering to domain through this **Fluid/Solid Boundary**. It may be a constant or a function. Units of the heat flux field may be defined in the menu next to this entry. It is possible to define additional units by entering new dimensionally correct units in the box (see Units Syntax section for further information). Note that positive values mean heat flow entering the domain.

*Reactive Heat Flux:* Factor of the reactive term of the heat flow (power) entering to the domain through this **Fluid/Solid Boundary**. The value here inserted will be multiplied by the current temperature to obtain the heat flow. It may be a constant or a function. Units of the reactive heat flux field may be defined in the menu next to this entry. It is possible to define additional units by entering new dimensionally correct units in the box (see Units Syntax section for further information).

*Remarks:*

Convection heat transfer may be simulated by inserting the function  $q - h \cdot (T_m - T_o)$  in the field Heat Flux, being  $q$  a defined heat flow,  $h$  the transmission coefficient and  $T_o$  the external

temperature. However it is recommended to split this flow in two terms, constant flow  $q + h \cdot T_o$  that should be inserted in the *Heat Flow* field and the coefficient of the temperature dependent term  $h$ , that should be entered in *Reactive Heat Flux* field.

## Radiation Flux

*Emissivity:* value of the emissivity to be used for the surfaces assigned to the corresponding radiation flux condition. In heat radiation problems this quantity measures the effectiveness of a material surface in emitting energy as thermal radiation. Quantitatively, emissivity is the ratio of the thermal radiation from a surface to the radiation from an ideal black surface at the same temperature as given by the Stefan-Boltzmann law. It must be a real value between 0.0 and 1.0

*Wall temperature:* value of temperature for the surfaces assigned to the corresponding radiation flux condition when using the P-1 radiation model.

### 3.4.3. Species Advection Analysis Conditions

#### Fix Concentration

This condition is assigned to geometrical/mesh entities and layers and is used to fix the value of the concentration of species (substances) at the given value.

*Fields:*

*Species Name:* Name of the species (see **Edit Species** description in section **Materials**) which concentration is to be fixed.

*Concentration:* Value of the concentration of the species.

*Remarks:*

The value of the concentration of every species should be prescribed at every inlet boundary.

#### Conditional concentration

This condition is assigned to geometrical/mesh entities and layers and is used to specify the concentration of species to the value given by the *Concentration Field* entry of the **Initial and Conditional** data section of the corresponding specie definition that is done within the Materials section of the data tree (only if the corresponding *Concentration Field* data is greater than 0):

**Materials ► Edit species ► Specie name ► Initial and Conditional**

If the evaluation of the *Concentration Field* results in a value less than 0, the boundary conditions will not be applied. If the value is 0, the boundary condition will be applied only if it was applied in the previous time step.

*Remarks:*

Entries of **Initial and Conditional** data may be defined by functions (see Function Syntax section).

*Concentration Field* and *Species Conditional* entries of a given specie definition are common for every **Conditional Concentration** condition and can only be modified in the **Initial and Conditional** data section of the specie definition.

When **Conditional Concentration** condition is assigned, *Conc. Field* entry is used both for defining initial values ( $t = 0$ ) when

starting calculation and for evaluating the assigned condition the rest of time steps.

## Concentration Field

This condition is assigned to geometrical/mesh entities and is used to fix the concentration of species to the value given by the *Concentration Field* entry of the **Initial and Conditional** data section of the corresponding specie definition beneath the Materials section:

**Materials ▶ Edit species ▶ Specie name ▶ Initial and Conditional**

*Fields:*

*Specie:* Name of the species (see **Edit Species** description in section **Materials**) whose concentration is going to be fixed.

*Fix Initial:* The concentration of the species will be fixed to the initial value (evaluated in  $t=0$ ) of the function inserted in the *Concentration Field* entry of the **Initial and Conditional** data section of the specie definition. *Concentration Field* entry is evaluated at the initial step ( $t=0$ ) and concentration is fixed to the resulting value for the rest of the execution.

*Fix Field:* The concentration will be fixed to the value (for every time step) of the function inserted in the *Concentration Field* entry of the **Initial and Conditional** data section of the specie definition. *Concentration Field* entry is evaluated every step and concentration is fixed to the resulting value.

*Remarks:*

The value of the concentration of each specie should be prescribed at every inlet boundary.

This condition allows definitions of transient boundary conditions for concentration. The analytical functions defining transient boundary conditions will be specified in the *Concentration Field* entry of the **Initial and Conditional** data section of the specie definition.

If the boundary conditions for the concentration are steady, this condition can be substituted by the **Fix Concentration** condition. The only difference between these two options in this case is, that when using the **Concentration Field**, the value of the fixed concentration can be changed automatically in every entity by updating the *Concentration Field* entry of the **Initial and Conditional** data section of the specie definition.

When **Concentration Field** condition is assigned, *Concentration Field* entry is used both for defining initial values ( $t = 0$ ) when starting calculation and for evaluating the assigned condition the rest of time steps.

## Advect Flux

*Advect Flux Solids/Fluids* allows to select among the different created species list, to assign to them a flux through a certain boundary.

*Fields:*

*Specie:* name of the specie to which the advect flux condition will refer to.

*SpecieFlux:* Flow of the species entering to the domain through this **Fluid/Solid Body**. It may be a constant or a function. Units of the flux specie field may be defined in the menu next to this entry. It is possible to define additional units by entering new dimensionally correct units in the box (see Units Syntax section for further information).

*Reactive Specie Flux:* Factor of the reactive term of the flow of the species entering to the domain through this **Fluid/Solid Body**. The value here inserted will be multiplied by the current species concentration to obtain the heat flow. It may be a constant or a function. Units of the reactive flux specie field may be defined in the menu next to this entry. It is possible to define additional units by entering new dimensionally correct units in the box (see Units Syntax section for further information).

*Remarks:*

Entering flow of species of the form  $h \cdot sp1$  should be inserted in the *Reactive Specie Flux* field as  $h$ .

Note that positive values means flow entering in the domain.

## 3.4.4. PDEs solver Conditions

### Fix Variable

This condition is assigned to geometrical/mesh entities and layers and is used to fix the value of the variable at the given value.

*Fields:*

*Variable:* Name of the variable (see material **Edit PDEs variables** description in section **Materials**) whose value is going to be fixed.

*Value:* Value of the variable.

*Remarks:*

The value of the variable should be prescribed at every inlet boundary.

### Conditional Variable

This condition is assigned to geometrical/mesh entities and layers and is used to specify the value of a variable at the value given by the *Variable Field* entry of the **Initial and Conditional** data section of the variable definition in the Materials section of the tree (only if the evaluation of the function defined in *Variable field* is greater than 0):

**Materials ▶ Edit PDEs variables ▶ Variable ▶ Initial and Conditional**

If the evaluation of the *Variable field* results in a value less than 0, the boundary conditions will not be applied. If the value is 0, the boundary condition will be applied only if it was applied in the previous time step.

*Remarks:*

Entries of the **Initial and Conditional** data section of the variable definition (within the Materials section) may be defined by functions (see Function Syntax section). *Variable field* and *Variable FuncCond.* entries are common for every **Conditional Variable** condition and can only be modified in the **Initial and Conditional** section of the variable definition.

When **Conditional Variable** condition is assigned the *Variable field* entry is used for defining both initial values ( $t = 0$ ) when starting calculation and for evaluating the assigned condition the rest of time steps.

### Variable Field

This condition is assigned to geometrical/mesh entities and is used to fix the value of a variable to the value given by the *Variable Field* entry of the **Initial and Conditional** data section of the corresponding variable definition (within the Materials section of the data tree):

**Materials ► Edit PDEs variables ► Variable ► Initial and Conditional**

*Fields:*

**Variable:** Name of the variable (see **Edit PDEs variables** description in section Materials) which value is to be fixed.

**Fix Initial:** The variable will be fixed to the initial value (evaluated in  $t=0$ ) of the function inserted in the **Variable Field** entry of the **Initial and Conditional** data section of the variable definition. **Variable Field** entry is evaluated at the initial step ( $t=0$ ) and the variable is fixed to the resulting value for the rest of the execution.

**Fix Field:** The variable will be fixed to the value (for every time step) of the function inserted in the **Variable Field** entry of the **Initial and Conditional** data section of the variable definition. **Variable Field** entry is evaluated every step and Variable is fixed to the resulting value.

*Remarks:*

The value of the variable should be prescribed at every inlet boundary.

This condition allows definitions of transient boundary conditions for variables. The analytical functions defining transient boundary conditions will be specified in the **Variable Field** entry of the **Initial and Conditional** data section of the variable definition.

If the boundary conditions for the variable are steady, this condition can be substituted by the **Fix Variable** condition. The only difference between these two options in this case is, that when using the **Variable Field** condition, the value of the fixed variable can be easily updated in every entity by changing the **Variable Field** entry of the **Initial and Conditional** data section of the variable definition.

When **Variable Field** condition is assigned **Variable Field** entry is used both for defining initial values ( $t = 0$ ) when starting calculation and for evaluating the assigned condition the rest of time steps.

## PDEs Variables Flux

**PDEs Variables Flux Solids/Fluids** allows to select among the list of created variables, to assign to them a flux through a certain boundary.

*Fields:*

**Variable Flux:** Flow of the variable entering to the domain through this **Fluid/Solid Body**. It may be a constant or a function. Units of the flux phi field may be defined in the menu next to this entry. It is possible to define additional units by entering new dimensionally correct units in the box (see Units Syntax section for further information).

**Reactive Variable Flux:** Factor of the reactive term of the flow of the species entering to the domain through this **Fluid/Solid Body**. The value here inserted will be multiplied by the current variable concentration to obtain the heat flow. It may be a constant or a function. Units of the reactive flux phi field may be defined in the menu next to this entry. It is possible to define additional units by entering new dimensionally correct units in the box (see Units Syntax section for further information).

*Remarks:*

Entering flow of the variable of the form  $h \cdot \phi_1$  should be inserted in the **Reactive Variable Flux** field as  $h$ .

Note that positive values means flow entering in the domain.

## 3.4.5. Mesh Deformation Conditions

### Fix Mesh Deformation

This condition is assigned to geometrical/mesh entities and layers. It is used to fix the value of the mesh deformation to zero or to the value given in the **Fluid Deformation Increment OX/OY/OZ** fields in the **Mesh deformation** section of the data tree. See the following **Modules Data** section of the tree:

**Modules data ► Fluid flow ► Mesh deformation**

*Fields:*

**Type:** Type of mesh deformation. If **Fix Field** is selected, imposed mesh deformation will be defined by **Fluid/Solid Deformation Increment OX/OY/OZ** fields (specified within the Modules data). If **Fix Null** is selected, mesh deformation is forced to be zero. Finally, if **No Fix** is selected, any other imposition on the mesh deformation field is ignored.

*Remarks:*

The type of *Fluid mesh deformation* to be imposed may be defined in the **Modules Data** section of the tree:

**Modules data ► Fluid flow ► Mesh deformation**

### Fix Mesh Velocity

This condition is assigned to geometrical/mesh entities. It is used to fix the value of the fluid flow on the selected entity to the value of the mesh deformation velocity.

**Conditions and Initial data ► Mesh deformation ► Fix mesh velocity**

*Fields:*

**Fix X:** mark this field in order to fix the X velocity component of the fluid flow to the value provided by the X component of the mesh deformation velocity.

**Fix Y:** mark this field in order to fix the Y velocity component of the fluid flow to the value provided by the Y component of the mesh deformation velocity.

**Fix Z:** mark this field in order to fix the Z velocity component of the fluid flow to the value provided by the Z component of the mesh deformation velocity.

## 3.4.6. Free Surface Conditions (ODDLS)

### ODDLS Field

This condition is assigned to geometrical/mesh entities and is used to fix the value of the level set function to the value given by the **OddLevelSet field** entry of the **Initial and Field data** section of the tree:

**Conditions and Initial data ► Initial and Conditional data ► Initial and Field data**

*Fields:*

**Fix Initial:** The level set function will be fixed to the initial value (evaluated in  $t=0$ ) of the function inserted in the **OddLevelSet field** entry of the **Initial and Field data** section of the tree. **OddLevelSet field** entry is evaluated at the initial step ( $t=0$ ) and level set function is fixed to the resulting value for the rest of the execution.

**Fix Field:** The level set function will be fixed to the value (evaluated every time step) of the function inserted in the **OddLevelSet Field** entry of **Initial and Field data** section of the



tree. *OddLevelSet field* entry is evaluated every time step and level set function is fixed to the resulting value.

*Remarks:*

The value of the level set function should be prescribed at every inlet boundary.

This condition allows definitions of transient boundary conditions for level set function. The analytical functions defining transient boundary conditions will be specified in the *OddLevelSet field* entry of **Initial and Field data**.

If the boundary conditions for the variable are steady, this condition can be substituted by the **Fix ODDLs** condition. The only difference between these two options in this case is, that when using the **ODDLs Field** the value of the fixed variable can be easily updated in every entity by changing the *OddLevelSet field* of **Initial and Field data**.

When **ODDLs Field** condition is assigned, *OddLevelSet field* entry is used both for defining initial values ( $t = 0$ ) when starting calculation and for evaluating the assigned condition the rest of time steps.

### Fix ODDLs

This condition is assigned to geometrical/mesh entities and layers and is used to fix the value of the level set function.

*Fields:*

*Value:* Value of the level set function. Positive values identifies the primary phase.

*Remarks:*

The value of the level set function should be prescribed at every inlet boundary.

### Conditional ODDLs

This condition is assigned to geometrical/mesh entities and layers and is used to specify the value of the level set function at the value given by the *OddLevelSet FuncCond* entry of the **Conditional data section of the tree** (only if the *OddLevelSet FuncCond* data field is greater than 0):

#### Conditions and Initial data ► Initial and Conditional data ► Conditional data ► OddLevelSet FuncCond

If the evaluation of the *OddLevelSet FuncCond* field results in a value less than 0, the boundary conditions will not be applied. If the value is 0, the boundary condition will be applied only if it was applied in the previous time step.

*Remarks:*

Entries of **Cond. Data** may be defined by functions (see Function Syntax section).

*OddLevelSet FuncCond* and *OddLevelSet Field* entries are common for every **Conditional ODDLs** condition and can only be modified in **Cond. Data**.

When **Conditional ODDLs** condition is assigned, *OddLevelSet Field* entry is used both for defining initial values ( $t = 0$ ) when starting calculation and for evaluating the assigned condition the rest of time steps.

### 3.4.7. Free Surface Conditions (Transpiration)

#### Fix Beta

This condition is assigned to geometrical/mesh entities and layers. It is used to fix the value of the wave elevation to its initial value. The initial value is the difference between the OZ coordinate of the point and the reference height of the free

surface. See the following Modules data section of the tree:

#### Modules data ► Fluid flow ► General ► Pressure reference location

*Fields:*

*Fix:* The wave elevation will be fixed to its initial value only if this field is marked.

*Remarks:*

This option is effective in the stern of some geometries to keep the stability of the free surface. In most of the cases it can be automatically imposed by using the **Wall/Body** options (see **Stern C Angle** option and Stern flow modelling in transpiration problem section).

### Free Surface

**Free Surface** boundary conditions identify a free surface boundary of a fluid in the analysis. These properties can be assigned to surfaces (3D).

*Remarks:*

**Free Surface** boundary condition is only available if the Free Surface (Transpiration) Analysis is activated.

In order to transfer **Free Surface** data to the mesh, Meshing Criteria must be fixed to Yes in the corresponding geometrical entities. Note that this action is automatically done by Tdyn CFD+HT in most of the cases.

*General fields:*

*Time Integration:* Time integration scheme used in the solution process of the free surface problem. The following options are available:

*Adams\_Bashforth\_2:* Explicit 2<sup>nd</sup> order Adams Bashforth scheme.

*Stabilised\_FIC:* Time stabilised FIC scheme.

*Backward\_Euler:* Implicit 1<sup>st</sup> order Backward Euler scheme.

*Forward\_Euler:* Explicit 1<sup>st</sup> order Forward Euler scheme.

*Crank\_Nicolson:* Implicit 2<sup>nd</sup> order Crank-Nicolson scheme.

*Length:* Characteristic length of the free surface problem (i.e. length of the **Fluid Body**).

*Damping length:* Relative damping length (total damping length is given by *Damping Length* x *Length*) to be used in this free surface calculation. The damping of the generated waves starts at a total damping length distance from the outlet of the free surface.

*Remarks:*

In most of the cases it is necessary to damp the wave elevation in order not to find bouncing effects in the boundaries.

*Damping factor:* Factor that controls the damping effect.

*Advanced fields:*

*Time factor:* Time integration security factor to be used in the explicit integration (i.e. Adams\_Bashforth\_2, Stabilised\_FIC and Forward\_Euler schemes) of this free surface.

*Step factor:* Time step ratio between free surface and fluid solver. It is possible to accelerate convergence by increasing this ratio, but may cause instability in the integration scheme. If chosen *Time Increment* is too high, reduce this value to achieve convergence.

*Remarks:*

Note that solutions with *Step factor*  $\neq 1$  will only give realistic results for the steady state.

*Advect\_Stabilisation:* The order of the FIC advection stabilisation term in the free surface equation. Two options are available *4th\_Order* and *2nd\_Order*.

*Remarks:*

The 4<sup>th</sup> order term increases the accuracy of the solution and is recommended in most of the cases, but in some problems may appear instabilities.

*StabTau\_MinRatio:* Minimum admissible ratio ( $\tau/dt$ , being  $dt$  the time increment) for the stabilisation parameter  $\tau$ .

*Remarks:*

Advection stabilisation term is proportional to the parameter  $\tau$ . In most of the cases, the minimum value of this parameter should not be fixed (i.e.  $\tau/dt = 0.0$ ), otherwise oscillations may appear.



## 3.5. Contacts

### 3.5.1. Solid-Solid Contact

**Solid-Solid Contact** boundaries identify a contact with continuity of the corresponding field between two disjoint solid domains. Contact properties can be defined and assigned to lines (2D Plane & 2D Axisymmetric analysis) or surfaces (3D analysis) or boundary meshes. Three contact types are available:

- **Interpolating contact:** this type of contact concerns the traditional contact algorithm. Contact surfaces must be coincident, although resulting contact meshes may be different in each side.
- **Distance contact:** this type of contact allows to correlate the results between two surfaces which are not truly in contact and that can be even moving apart from each other.
- **Periodic contact:** this type of contact is used to enforce periodic boundary conditions. When this type of contact is selected, X/Y/Z Distance fields become active so that distance functions can be defined in order to specify the periodicity of the contact.

#### Options available in Fluid Flow module

*Activate contact:* if this check button is activated, the contact algorithm will enforce continuity in both, velocity and pressure, across or between solid domains.

*Activate contact (only velocity):* if this check button is activated, the contact algorithm will enforce continuity only in velocity across or between solid domains.

Note that these two options are mutually exclusive.

#### Options available in Heat Transfer module

*Activate contact:* thermal contact is only active if the check-button is selected. If the check-button is not selected, thermal resistance is assumed to be infinite.

*Thermal resistance:* thermal resistance of the temperature contact between solid materials. If thermal resistance is null, contact is perfect. Thermal resistance  $R$  of an homogeneous layer of a solid material, can be calculated as  $R = e / k$ , being  $e$  the thickness of the layer, and  $k$  the thermal conductivity of the material.

#### Options available in Species Advection module

*Active species:* individual check buttons are available for each existing specie. The check button must be selected in order to activate the contact for the corresponding specie. If a check button is not selected, an infinite resistance is assumed to exist for that particular specie.

*Resistance:* Resistance term ( $R$ ), defining species flow through the contact as  $d\phi/dn = R \cdot (c_1 - c_2)$  being  $c_1$  and  $c_2$  the concentration (of the corresponding species) in domains 1 and 2 respectively. If resistance is null, contact is perfect.

#### Options available in PDE's solver module

*Active variable:* individual check buttons are available for each defined generic variable. The check button must be selected in order to activate the contact for the corresponding variable. If a check button is not selected, an infinite resistance is assumed to exist for that particular variable.

*Resistance:* Resistance term ( $R$ ), defining variables flow through the contact as  $d\phi/dn = R \cdot (\phi_1 - \phi_2)$  being  $\phi_1$  and  $\phi_2$  the concentration

(of the corresponding variables) in domains 1 and 2 respectively. If resistance is null, contact is perfect.

### Options available in Mesh Deformation module

*Activate Contact:* Select to activate the contact between two solid domains for the mesh deformation algorithm. Such an option is only effective if the *ByBodies* mesh deformation type has been selected in the data tree as follows:

**Modules data ► Mesh deformation ► Fluid mesh deformation ► ByBodies**

### 3.5.2. Fluid-Fluid Contact

**Fluid-Fluid Contact** boundaries identify a contact with continuity of the corresponding field between two disjoint fluid domains. Contact properties can be defined and assigned to lines (2D Plane & 2D Axisymmetric analysis) or surfaces (3D analysis) or boundary meshes. Three contact types are available:

- **Interpolating contact:** this type of contact concerns the traditional contact algorithm. Contact surfaces must be coincident, although resulting contact meshes may be different in each side.
- **Distance contact:** this type of contact allows to correlate the results between two surfaces which are not truly in contact and that can be even moving apart from each other.
- **Periodic contact:** this type of contact is used to enforce periodic boundary conditions. When this type of contact is selected, X/Y/Z Distance fields become active so that distance functions can be defined in order to specify the periodicity of the contact.

#### Options available in Fluid Flow module

*Activate contact:* if this check button is activated, the contact algorithm will enforce continuity in both, velocity and pressure, across or between fluid domains.

*Activate contact (only velocity):* if this check button is activated, the contact algorithm will enforce continuity only in velocity across or between fluid domains.

Note that these two options are mutually exclusive.

#### Options available in Heat Transfer module

*Activate contact:* thermal contact is only active if the check-button is selected. If the check-button is not selected, thermal resistance is assumed to be infinite.

*Thermal resistance:* thermal resistance of the temperature contact between fluid materials. If thermal resistance is null, contact is perfect. Thermal resistance  $R$  of an homogeneous layer of a solid material, can be calculated as  $R = e / k$ , being  $e$  the thickness of the layer, and  $k$  the thermal conductivity of the material.

#### Options available in Species Advection module

*Active species:* individual check buttons are available for each existing specie. The check button must be selected in order to activate the contact for the corresponding specie. If a check button is not selected, an infinite resistance is assumed to exist for that particular specie.

*Resistance:* Resistance term ( $R$ ), defining species flow through the contact as  $d\phi/dn = R \cdot (c_1 - c_2)$  being  $c_1$  and  $c_2$  the concentration (of the corresponding species) in domains 1 and 2 respectively. If resistance is null, contact is perfect.

## Options available in PDE's solver module

**Active variable:** individual check buttons are available for each defined generic variable. The check button must be selected in order to activate the contact for the corresponding variable. If a check button is not selected, an infinite resistance is assumed to exist for that particular variable.

**Resistance:** Resistance term (R), defining variables flow through the contact as  $d\varphi/dn=R(\varphi_1-\varphi_2)$  being  $\varphi_1$  and  $\varphi_2$  the concentration (of the corresponding variables) in domains 1 and 2 respectively. If resistance is null, contact is perfect.

## Options available in Mesh Deformation module

**Activate Contact:** Select to activate the contact between two fluid domains for the mesh deformation algorithm. Such and option is only effective if the *ByBodies* mesh deformation type has been selected in the data tree as follows:

**Modules data ► Mesh deformation ► Fluid mesh deformation ► ByBodies**

### 3.5.3. Fluid-Solid Contact

**Fluid-Solid Contact** boundaries identify a contact with continuity of the corresponding field between solid and fluid domains. Contact properties can be defined and assigned to lines (2D Plane & 2D Axisymmetric analysis) or surfaces (3D analysis) or boundary meshes. Three contact types are available:

- **Interpolating contact:** this type of contact concerns the traditional contact algorithm. Contact surfaces must be coincident, although resulting contact meshes may be different in each side.
- **Distance contact:** this type of contact allows to correlate the results between two surfaces which are not truly in contact and that can be even moving apart from each other.
- **Periodic contact:** this type of contact is used to enforce periodic boundary conditions. When this type of contact is selected, X/Y/Z Distance fields become active so that distance functions can be defined in order to specify the periodicity of the contact.

## Options available in Fluid Flow module

**Activate contact:** if this check button is activated, the contact algorithm will enforce continuity in both, velocity and pressure, across or between fluid and solid domains.

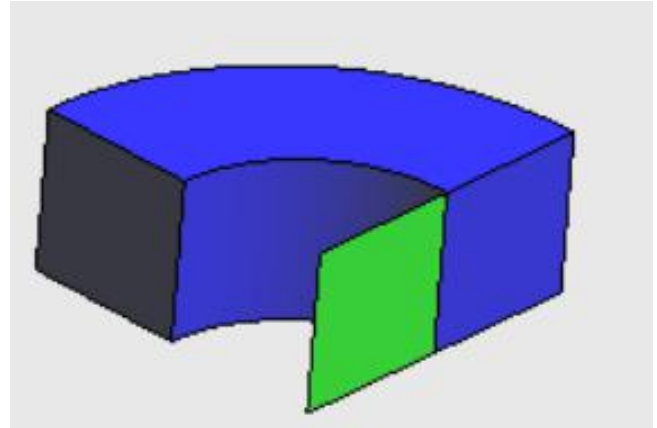
**Activate contact (only velocity):** if this check button is activated, the contact algorithm will enforce continuity only in velocity across or between fluid and solid domains.

Note that these two options are mutually exclusive.

## Options available in Heat Transfer module

**Activate contact:** thermal contact is only active if the check-button is selected. If the check-button is not selected, thermal resistance is assumed to be infinite.

**Thermal resistance:** thermal resistance of the temperature contact between solid and fluid materials. If thermal resistance is null, contact is perfect. Thermal resistance R of an homogeneous layer, can be calculated as  $R = e / k$ , being e the thickness of the layer, and k the thermal conductivity of the material.



## Options available in Species Advection module

**Active species:** individual check buttons are available for each existing specie. The check button must be selected in order to activate the contact for the corresponding specie. If a check button is not selected, an infinite resistance is assumed to exist for that particular specie.

**Resistance:** Resistance term (R), defining species flow through the contact as  $d\varphi/dn=R(c_s-c_f)$  being  $c_s$  and  $c_f$  the concentration (of the corresponding species) in solid and fluid domains respectively. If resistance is null, contact is perfect.

## Options available in PDE's solver module

**Active variable:** individual check buttons are available for each defined generic variable. The check button must be selected in order to activate the contact for the corresponding variable. If a check button is not selected, an infinite resistance is assumed to exist for that particular variable.

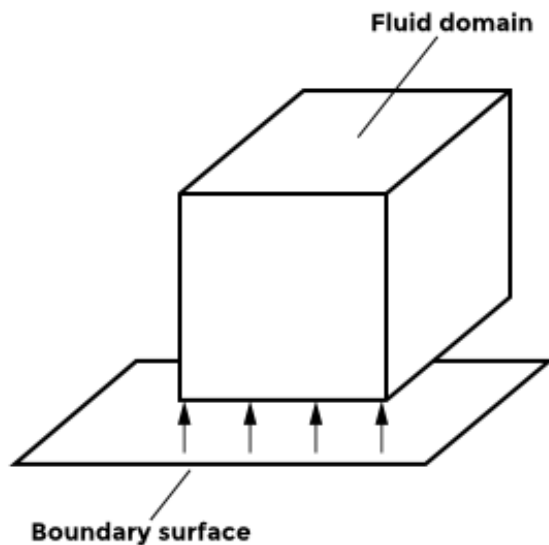
**Resistance:** Resistance term (R), defining variables flow through the contact as  $d\varphi/dn=R(\varphi_s-\varphi_f)$  being  $\varphi_s$  and  $\varphi_f$  the concentration (of the corresponding variables) in solid and fluid domains respectively. If resistance is null, contact is perfect.

### 3.5.4. Fluid-Boundary Contact

It allows to define a contact among a fluid and surfaces where fluid could be in contact. Three contact types are available:

- **Sticking:** fluid will not slide on the surface.
- **Frictionless:** fluid will slide on the surface without friction.
- **Coulomb:** fluid will slide on the surface with a coulomb friction. Friction coefficient must be defined.

Normals of boundary surface must point to the side of surface where fluid will contact.



Normals of boundary surface must point to the fluid domain

## 3.6. Materials

### 3.6.1. Fluids and Solids

Materials are groups of physical properties and other data that identify a material, fluid or solid to be used in the analysis.

For any problem that needs definition of materials, there is a database of existing materials that can be assigned to entities:

**Materials ► Physical properties ► Fluid | Solid**

The user can also create new materials derived from the existing ones and assign them as well:

**Materials ► Physical properties ► Generic Fluid | Solid**

To create a new Material, press "**CreateNew Material**" in the contextual menu of the abovementioned **Materials ► Physical Properties ► Generic Fluid | Solid** option, write a new name and change some of its properties. By pressing **Ok**, a new Material is created taking an existing one as a base Material, which means that the new Material will have the same fields as the base one. All new values for the fields can be entered when defining the new material. It is also possible to redefine existing Materials by entering new values directly in the fields.

*Remarks:*

If a mesh has already been generated and new materials are assigned to the geometry or some of the existing ones are removed, it is necessary to mesh again.

In this section only the main Materials will be presented. Therefore, Materials with other names can be found in the Materials database. Anyhow, all these Materials will be based on the ones shown here (i.e. they will have the same properties fields).

### Options available in Fluid Flow module

**Fluid Model:** this is the fluid model assumed to govern the behavior of the material. The fluid model must be one of *Incompressible*, *Slightly Compressible*, *Barotropic*, *Incompressible Ideal Gas* or *Ideal Gas*. But the actual available options will depend on the selected *Flow Solver Model*. The *Flow Solver Model* in turn can be selected using the following option of the data tree

### Fluid Dynamics and Multiphysics data ► Fluid Solver ► Flow Solver Model

**Density:** Density of the fluid. It may be a constant or a function (always greater than zero). Units of the density may be defined in the menu next to the density entry. It is possible to define additional units by entering new dimensionally correct units in the box (see Units Syntax section for further information).

**Viscosity:** Viscosity of the fluid. It may be a constant or a function (always greater than zero). Units of the viscosity may be defined in the menu next to the viscosity entry. It is possible to define additional units by entering new dimensionally correct units in the box (see Units Syntax section for further information).

**Create function for viscosity:** Tdyn offers the capability to analyse non-Newtonian fluids. The different models implemented can be chosen in the drop-down list of the window appearing by clicking on the button to the right of the viscosity entry of the generic non-Newtonian material (or any other existing non-Newtonian fluid) in the materials database.

**Power law model:** if chosen in the drop-down list of the viscosity model window, a non-Newtonian flow will be modelled according to the power law model in (Bird 1976).

**Herschel-Bulkley model:** this is a three constant simple generalization of the Bingham plastic model to embrace the non-linear flow curve (see TdynCFD+HT theory manual for details).

**Carreau model (for pseudo-plastics):** when there are significant deviations from the power-law model at very high and very low shear rates, it is necessary to use a model that accounts for the limiting values of viscosities ( $\mu_0$  and  $\mu_\infty$ ). The Carreau model (Carreau 1972) attempts to describe a wide range of fluids by the establishment of a curve-fit to piecetogether functions for both Newtonian and Shear-thinning ( $n < 1$ ) non-Newtonian laws.

**Cross model:** four parameters model in (Cross 1965) which has gained popularity to describe the behavior of viscosity in the slow-shear-rate range.

**Consistency index ( $k$ ):** the consistency index of the power law model is a measure of the average viscosity of the fluid.

**Time constant ( $\lambda$ ):** time constant to be used in the Carreau viscosity model (see TdynCFD+HT theory manual for details on the Carreau model).

**Natural time ( $\lambda$ ):** inverse of the shear rate at which the fluid changes from Newtonian to power-law behavior in the Cross model (see TdynCFD+HT theory manual for details).

**Power-Law index ( $n$ ):** power law index of the power-law model. This value actually determines the class of the fluid.  $n = 1$  corresponds to a Newtonian fluid;  $n > 1$  corresponds to a shear-thickening (dilatant) fluid;  $n < 1$  corresponds to a shear-thinning (pseudo-plastic) fluid.

**$\mu_{Min.}$ :** minimum viscosity limit to be used in the power-law model.

**$\mu_{Max.}$ :** maximum viscosity limit to be used in the power-law model.

**$\mu_0$ :** zero-shear viscosity limit used in both the, Carreau model and the Cross model.

**$\mu_\infty$ :** infinite-shear viscosity limit used in both, the Carreau model and the Cross model.

**Yield stress threshold ( $\tau_0$ ):** yield stress in the Herschel-Bulkley model.

**Zero-shear viscosity ( $\mu_0$ ):** yielding viscosity (0 zero-shear viscosity) in the Herschel-Bulkley model.

**Temperature dependent:** this flag is used to choose between shear-rate dependent and shear-rate plus temperature dependent model.

**Reference temperature ( $T_0$ ):** reference temperature to be used for shear-rate and temperature dependent non-newtonian fluids (see TdynMPH theory manual for details on the Arrhenius law controlling the temperature dependence of viscosity).

**Activation energy/ $R(a)$ :** ratio of the activation energy to the thermodynamic constant for a shear-rate and temperature dependent non-newtonian fluid (see TdynMPH theory manual for details on the Arrhenius law controlling the temperature dependence of viscosity).

**Compressibility:** Compressibility factor of the fluid,  $\alpha$ , defined as the inverse of the square of the speed of the sound in the fluid. Here it can be defined by a constant or a function (always greater than zero). This option is only available for *Slightly Compressible* or *Barotropic* fluid models. Units of the compressibility may be defined in the menu next to this entry. It is possible to define additional units by entering new dimensionally correct units in the box (see Units Syntax section for further information).

**Remarks:**

Slightly compressible fluid model is defined by the following pressure/density relationship:  $\Delta p = \alpha \cdot \Delta p$ , where  $\alpha = 1/c^2$  is the compressibility of the fluid, being  $c$  the speed of sound in the medium.

*Barotropic* fluid model is defined by the following pressure/density relationship:  $p = A \cdot \rho^\gamma$  where the factor  $\gamma$  is related with the speed of sound in the medium by  $c = p/(\gamma \cdot p)$ .

**Molar Mass:** Molar mass of the gas. This option is only available for *Incompressible Ideal Gas* or *Ideal Gas* fluid models.

Units of the molar mass may be defined in the menu next to the this entry. It is possible to define additional units by entering new dimensionally correct units in the box (see Units Syntax section for further information).

**Darcy's law Resistance Matrix:** Coefficients of the matrix defining permeability resistance of the flow in a porous media (Darcy's law). Due to this effect, a pressure drop given by

$$\delta p_i = -(\mu \cdot D \cdot v)_i / v_i \cdot \delta x_i$$

will be added to the velocity momentum equations. Where  $\delta p_i$  is the pressure drop for the momentum equation in the  $x_i$  direction,  $\mu$  is the fluid viscosity,  $D$  is the *Darcy's law Resistance Matrix*, and  $v$  is the velocity vector of components  $v_i$ .

Units of the *Darcy's law Resistance Matrix* may be defined in the menu next to the these entries.

**Acceleration Field:** External acceleration vector acting on fluid. May be defined by constants or functions.

**Remarks:**

It is recommended to insert functions with a smoothed start up for this additional acceleration. Otherwise it can create oscillations in the solution.

Vertical field will be added to the vertical component of the gravity, as an additional acceleration.

## Options available in Heat Transfer module

**Density:** Density of the fluid. It may be a constant or a function (always greater than zero). Units of the density may be defined in the menu next to the density entry. It is possible to define additional units by entering new dimensionally correct units in the box (see Units Syntax section for further information).

**Remarks:**

Density entry in this window is the same for Fluid Flow and Heat Transfer data. When both modules are selected if Density entry is changed in Fluid Flow data, it will be automatically updated in Heat Transfer data.

**Specific Heat:** Specific heat ( $C_p$ ) of the fluid. It may be a constant or a function (always greater than zero). Units of the specific heat may be defined in the menu next to this entry. It is possible to define additional units by entering new dimensionally correct units in the box (see Units Syntax section for further information).

**Thermal Conductivity:** Thermal conductivity ( $k$ ) of the fluid. It may be a constant or a function (always greater than zero). Units of the thermal conductivity may be defined in the menu next to this entry. It is possible to define additional units by entering new dimensionally correct units in the box (see Units Syntax section for further information).

**Floatability:** Floatability effect of a fluid (Boussinesq type) due to small changes of density. May be a constant or a function. This property controls the buoyancy effect due to the variations of temperature in the fluid. Standard effects are modelled by inserting the function  $\beta \cdot (T - T_0)$  where  $\beta$  is the volume expansion of the fluid and  $T_0$  is the temperature of reference. In this case, buoyancy effect will be taken into account by a variation of density of the fluid proportional to the temperature ( $\rho = \rho_0 \cdot \beta \cdot (T - T_0)$ ). This term is undimensional.

**Remarks:**

Note that *Floatability* entry can also accept non-linear terms.

**Heat Source Field:** Volumetric heat source in the fluid. May be a constant or a function. Units of the heat source field may be defined in the menu next to this entry. It is possible to define additional units by entering new dimensionally correct units in the box (see Units Syntax section for further information).

**Heat Reaction Field:** Volumetric heat reaction in the fluid. This entry will be added as a reactive term in the system of equations (i.e. a source term depending linearly to the temperature). May be a constant or a function. Units of the heat reaction field may be defined in the menu next to this entry. It is possible to define additional units by entering new dimensionally correct units in the box (see Units Syntax section for further information).

## Options available in Free Surface (ODDLS) module

**Primary Phase:** Current material will be identified as primary phase for the ODDLS free surface analysis. The primary phase is the phase of interest of the analysis. Special care is taking into



account in order to improve the accuracy of the solution obtained for the primary phase.

*Secondary Phase:* Selected material will be identified as secondary phase for the ODDLS free surface analysis.

*Surface Tension:* Surface tension between primary and secondary phase.

Units of the surface tension can be defined in the menu next to the *Surface Tension* entry. It is possible to define additional units by entering new dimensionally correct units in the box (see Units Syntax section for further information).

### 3.6.2. Species

#### Species Materials Data

*Fluid/Solid Props.* tabs of Species Edition (see Figure 8) are split in two frames. Upper frame shows the standard equation that is solved for the species (equation is different in Fluids and Solids). Lower frame shows the entries of the coefficients of the differential equation of the selected species.

Species definition window

Options available are shown next:

*Advection f1:* Advection factor of the selected species (see Figure 8). This property may be defined by a constant or a function. Advection factor is undimensional.

*Diffusion (Fick's Law, in Fluids. For Solids, it is a matrix) f2:* Total diffusion of the selected species (see Figure 8). Please note that this value must include the turbulent and physical diffusion of species. This property may be defined by a constant or a function. Units of the total diffusion field may be defined in the menu next to this entry. It is possible to define additional units by entering new dimensionally correct units in the box (see Units Syntax section for further information).

*Degradation f3:* Reactive term of the selected species (see Figure 8). This property may be defined by a constant or a function. Units of the reactive field may be defined in the menu next to this entry. It is possible to define additional units by entering new dimensionally correct units in the box (see Units Syntax section for further information).

*Source f4:* Source of concentration of the selected species (see Figure 8). This property may be defined by a constant or a function. Units of the source of concentration field may be defined in the menu next to this entry. It is possible to define additional units by entering new dimensionally correct units in the box (see Units Syntax section for further information).

#### Species General Data

Options below have to be defined for every existing Species.

*Max limit:* Maximum acceptable value of the species concentration.

*Min limit:* Maximum acceptable value of the species concentration.

*Convergence norm:* Euclidean norm of species concentration used to check convergence in the non-linear iteration loop.

*Inner iterations:* Number of iterations of the inner (nonlinear) species concentration eq. solver (performed every external iteration).

*Advect stability:* Order of the FIC advection stabilisation term in the species concentration equation. Three available options are *Auto*, *4th\_Order* and *2nd\_Order*.

*Remarks:*

The 4<sup>th</sup> order term increases the accuracy of the solution and is recommended in most of the cases. However in some problems it may cause instabilities.

*Auto* mode will automatically switch between 4<sup>th</sup> and 2<sup>nd</sup> order scheme, depending on the smoothness of the solution.

*Stability control:* Level of control of instabilities (0 means Off). If instabilities are found in the species concentration field when using the *2nd\_Order Advect Stabilisation*, first try to reduce *Time Increment*, then to increase this value. Note that high values may cause over-diffusive results.

*Volume conservation:* If this box is selected, conservation of species concentration will be enforced.

#### Species Turbulence Data

*Schmidt number:* Schmidt number used to include turbulence effects in the species calculations.

#### Species Initial and Conditional Data

*Concentration Field:* Initial (t=0) and reference concentration field. May be a constant or a function (see Function Syntax section for further information).

*Remarks:*

If any **Concentration Field** condition has been assigned to any entity within this material, this field will be used as a base to calculate boundary conditions. If the corresponding *Fix Initial* field has been marked, the concentration of the species will be fixed to the initial value (evaluated in t = 0) of the function inserted here.

If the corresponding *Fix Field* has been marked, the concentration of the species will be fixed to the value (for every time step) of the function inserted here. It is possible to define transient boundary conditions for the concentration of the species this way.

**Species Conditional:** Conditional function used to define **Conditional Concentration** boundary conditions. **Conditional Concentration** boundary conditions will only be applied if the *Species Conditional* field value (resulting of the evaluation of the given function) is greater than 0. If the evaluation of the *Species Conditional* field results in a value less than 0, the boundary conditions will not be applied. If the value is 0, the boundary condition will be applied only if it was applied in the previous time step.

### 3.6.3. PDEs Variables

#### Variables Materials Data

*Fluid/Solid Props.* tabs of PDE's Variables Edition (see Figure 9) are split in two frames. Upper frame shows the standard equation that is solved for the species (equation is different in Fluids and Solids). Lower frame shows the entries of the coefficients of the differential equation of the selected species.

Options available for **Fluid Props.** are shown next:

*ft1:* Temporal factor of the selected variable (see Figure 9). This property may be defined by a constant or a function. Units of the temporal factor field may be defined in the menu next to this entry. It is possible to define additional units by entering new dimensionally correct units in the box (see Units Syntax section for further information).

Label	Value	Unit	Function
ft1:	1.0	1/m <sup>2</sup>	$f_x$
fc1:	1.0	1/m <sup>2</sup>	$f_x$
f2:	1.0	1/(m.s)	$f_x$
f3:	0.0	1/(m <sup>2</sup> .s)	$f_x$
f4:	0.0	mol/(m <sup>2</sup> .s)	$f_x$

PDEs Variables definition window

*fc1:* Advection factor of the selected variable (see Figure 9). This property may be defined by a constant or a function. Units of the advection factor field may be defined in the menu next to this entry. It is possible to define additional units by entering new dimensionally correct units in the box (see Units Syntax section for further information).

*f2:* Total diffusion of the selected variable (see Figure 9). Please note that this value must include the turbulent and physical diffusion of variable. This property may be defined by a constant or a function. Units of the total diffusion field may be defined in the menu next to this entry. It is possible to define additional units by entering new dimensionally correct units in the box (see Units Syntax section for further information).

*f3:* Reactive term of the selected variable (see Figure 9). This property may be defined by a constant or a function. Units of the reactive field may be defined in the menu next to this entry. It is possible to define additional units by entering new dimensionally correct units in the box (see Units Syntax section

for further information).

*f4:* Source of the selected variable (see Figure 9). This property may be defined by a constant or a function. Units of the source field may be defined in the menu next to this entry. It is possible to define additional units by entering new dimensionally correct units in the box (see Units Syntax section for further information).

Options available for **Solid Props.** are shown next:

*f1:* Temporal factor of the selected variable. This property may be defined by a constant or a function. Units of the temporal factor field may be defined in the menu next to this entry. It is possible to define additional units by entering new dimensionally correct units in the box (see Units Syntax section for further information).

*f2:* Total diffusion matrix of the selected variable. Please note that this value must include the turbulent and physical diffusion of variable. This property may be defined by a constant or a function. Units of the total diffusion field may be defined in the menu next to this entry. It is possible to define additional units by entering new dimensionally correct units in the box (see Units Syntax section for further information).

*Remarks:*

If only one value of the diffusion of the variable is available, it should be inserted in the diagonal terms of the matrix.

*f3:* Reactive term of the selected variable. This property may be defined by a constant or a function. Units of the reactive field may be defined in the menu next to this entry. It is possible to define additional units by entering new dimensionally correct units in the box (see Units Syntax section for further information).

*f4:* Source of the selected variable. This property may be defined by a constant or a function. Units of the source field may be defined in the menu next to this entry. It is possible to define additional units by entering new dimensionally correct units in the box (see Units Syntax section for further information).

#### Variables General Data

Options below have to be defined for every existing PDE's Variables.

*Max limit:* Maximum acceptable value of the variable field.

*Min limit:* Minimum acceptable value of the variable field.

*Convergence norm:* Euclidean norm of variable field used to check convergence in the non-linear iteration loop.

*Inner iterations:* Number of iterations of the inner (nonlinear) variable eq. solver (performed every external iteration).

*Variable stabilisation:* Order of the FIC advection stabilisation term in the variable equation. Three available options are *Auto*, *4th\_Order* and *2nd\_Order*.

*Remarks:*

The 4<sup>th</sup> order term increases the accuracy of the solution and is recommended in most of the cases. However in some problems it may cause instabilities.

*Auto* mode will automatically switch between 4<sup>th</sup> and 2<sup>nd</sup> order scheme, depending on the smoothness of the solution.



Stability control: Level of control of instabilities (0 means Off). If instabilities are found in the variable field when using the *2nd\_Order Advect Stabilisation*, first try to reduce *Time Increment*, then to increase this value. Note that high values may cause over-diffusive results.

*Volume conservation*: If this check-button is selected, conservation of variable field will be enforced.

## Variables Initial and Conditional Data

*Variable Field*: Initial ( $t=0$ ) and reference variable field. May be a constant or a function (see Function Syntax section for further information). There is one *Variable Field* entry for every variable.

*Remarks:*

If any **Variable Field** condition has been assigned to any entity, this field will be used as a base to calculate boundary conditions. If the corresponding *Fix Initial* field has been marked, the value of the variable will be fixed to the initial value (evaluated in  $t = 0$ ) of the function inserted here.

If the corresponding *Fix Field* has been marked, the value of the variable will be fixed to the value (for every time step) of the function inserted here. It is possible to define transient boundary conditions for the variables this way.

*Variable FuncCond*: Conditional function used to define **Conditional Variable** boundary conditions. **Conditional Variable** boundary conditions will only be applied if the *Variable FuncCond* field value (resulting of the evaluation of the given function) is greater than 0. If the evaluation of the *Variable FuncCond* field results in a value less than 0, the boundary conditions will not be applied. If the value is 0, the boundary condition will be applied only if it was applied in the previous time step. There is one *Vars. FuncCond* field for every variable.

Initialize variable: if this check-button is selected, the selected variable is re-initiated to a signed distance every time step.

### 3.7. Utilities

In this section, several specific utilities of Tdyn CFD+HT are introduced.

*Forces on Boundaries* (Pre-processor menu option **Utilities > Forces on Boundaries**, post-processor menu option **View Results > Forces on Boundaries**) shows the last value of the forces on the defined boundaries (see *Fluid and Solid Boundaries* reference for further information). The components of the acting force on the boundary are:

Pressure forces: force resulting of the integration of the pressure on the boundary.

Pressure moments: moments of the pressure forces, evaluated in the center of gravity of the boundary.

Static pressure force: force resulting of the integration of the fluid static force on the boundary (note that if the total pressure algorithm is selected, this component is null).

Static pressure moments: moments of the static pressure forces, evaluated in the center of gravity of the boundary.

Viscous forces: force resulting of the integration of the fluid traction on the boundary.

Viscous moments: moments of the viscous forces, evaluated in the center of gravity of the boundary.

Total forces: total forces acting on the boundary.

Total moments: total moments acting on the boundary, evaluated in the center of gravity of the boundary.

#### Remarks:

The units of the forces are based on the OutPut Units defined by the user (Newtons by default).

*Forces Graph*: (Pre-processor menu option **Utilities > Forces graph**, post-processor menu option **View Results > Forces graph**) shows a graph of the evolution of the forces on the defined boundaries (see *Fluid and Solid Boundaries* reference for further information).

*Motions Graph*: (Menu option **Utilities > Motions graph**, post-processor menu option **View Results > Motions graph**) shows a graph of the evolution of the movements on the defined boundaries (see *Fluid and Solid Boundaries* reference for further information).

*Norms Graph*: Time evolution graph of the different convergence norms involved in the problem. For each norm, normalized values of the increment ratios of the corresponding variable are plotted against time. When all variable increments become smaller than the *Steady State Norm* the simulation stops.

### 4. Free Surface Analysis with Transpiration Method

The Transpiration Free Surface Analysis of Tdyn CFD+HT has been specially designed for the simulation of the towing of ships test. For other free surface applications, the Overlapping Domain Decomposition Level Set (ODDLS) method should be used. Please refer to the tutorial examples of the ODDLS method of Tdyn CFD+HT for further information.

In particular, standard naval problems including ship movement

or analysis of complex transom stern flows can be easily analysed using ODDLS method. However, Transpiration method approach can be still interesting in some cases where capturing a small perturbation of the free surface is required. For those cases, the next sections present some specific tools of the Transpiration method for simulation of dynamic sinkage and trim effects and transom stern flows.

#### 4.1. Stern flow modelling in transpiration problem

It is well known that the standard solution of the advective equations as the free surface requires the imposition of Dirichlet conditions at the inlet boundaries.

Experimental analyses reveal that transom stern flow, occurring at a sufficient high speed, shows a local discontinuity in the wave elevation field. In those cases, the standard solution of the free surface equation close to this region is inconsistent with the convective nature of this equation. The trial of direct solving the free surface equation in this case results in instability in the wave height close to the transom region. This instability is found experimentally for low speeds, but the flow at a sufficient high speed is more stable and cannot be reproduced by using the standard techniques.

The conclusion of the above discussion is that it is necessary to determine and apply boundary conditions adequate for the free surface solution on the transom boundary. The obvious solution to this problem is to fix both the free surface elevation  $\beta$  and its derivative along the corresponding streamline. Its values will be approximately given by the transom position and the surface gradient. This option is available in Tdyn CFD+HT through the prescription of the wave elevation in the transom boundary (see **Fix Beta** conditions). However the direct imposition of the mentioned values can influence the transition between the transom flow and the lateral flow, resulting in inaccurate wave maps.

The solution developed in Tdyn CFD+HT, extends the free surface below the floater. The necessary Dirichlet boundary conditions imposed at the inlet of the domain are sufficient to achieve the well-possessed properties of the problem. It is interesting to note, that this imposition is not *ad hoc*, since the free surface equation have to be accomplished also in the wetted surface below the floater. Obviously proceeding this way will remain valid both for the wetted transom and for the dry transom flows and may also be applied to floaters with regular stern. Unfortunately, in the latter case, a very fine definition of the mesh is required in some cases, in order to capture the discontinuities that may appear in the wave elevation field.

In order to simplify the application of the above ideas, Tdyn CFD+HT includes a methodology to automatically detect when is necessary extend the free surface below the floater. The method is based on calculating the angle between the floating line and the local velocity. Only if this angle is greater than a given value (inserted in the field **SternC Angle**), the corresponding elements of the transom edge will have the necessary boundary condition.

### 5. Units Syntax

The default units system for Tdyn is the International System (IS). They are:

- Time in seconds (s)
- Lengths in meters (m)
- Masses in Kilograms (kg)
- Forces in Newton (N)

A derived unit is Pascal (Pa), where  $\text{Pa} = \text{N}/\text{m}^2$

These units can be changed in several parts of the program. Every window that asks for data for constraints, loads or properties has a field to choose the units for that window. The chosen units are only applicable in the data attached to it in the same window.

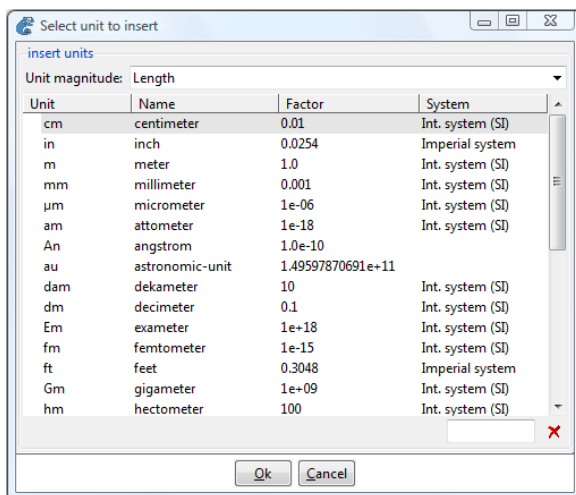
The units in which are expressed the coordinates of the geometrical model are chosen in the General data window:

Data -> General data -> Units -> Geometry units

Other predefined units that can be chosen are:

- Forces: Kilogram-force (kp)
- Lengths: centimeter (cm), millimeter (mm)

Different units can be selected using "**More Units**" option in the unit selection-box. A selection window is opened and selected units are inserted to create new ones:



These units can also be edited by selecting "**Edit mode>Edit field**" option in the unit selection-box. In edition mode new units can be written. After writing new units, they can be checked by selecting "**Edit mode>check**" option, if an edited unit is correctly written it will be signalled in green colour, if the new unit is wrongly written it will be signalled in red colour. Different operations can be inserted by selecting "**Edit mode>Insert**".

Result units can be expressed as:

- N-m-kg where:
- Displacements are in m
- Velocities are in m/s
- Accelerations are in  $\text{m}/\text{s}^2$
- Strengths are in  $\text{N}/\text{m}$  or  $\text{N}\cdot\text{m}/\text{m}$
- Stresses are in  $\text{Pa} = \text{N}/\text{m}^2$
- N-mm-kg where:
- Displacements are in mm
- Velocities are in  $\text{mm}/\text{s}$
- Accelerations are in  $\text{mm}/\text{s}^2$
- Strengths are in  $\text{N}/\text{mm}$  or  $\text{N}\cdot\text{mm}/\text{mm}$
- Stresses are in  $\text{N}/\text{mm}^2$
- Kp-cm-utm where:

Displacements are in cm

- Velocities are in  $\text{cm}/\text{s}$
- Accelerations are in  $\text{cm}/\text{s}^2$
- Strengths are in  $\text{Kp}/\text{cm}$  or  $\text{Kp}\cdot\text{cm}/\text{cm}$
- Stresses are in  $\text{Kp}/\text{cm}^2$
- KN,m,Mpa where:
- Displacements are in m
- Velocities are in  $\text{m}/\text{s}$
- Accelerations are in  $\text{m}/\text{s}^2$
- Strengths are in  $\text{kN}/\text{m}$  or  $\text{kN}\cdot\text{m}/\text{m}$  ( $\text{kN} = 10^3 \text{ N}$ )
- Stresses are in  $\text{Mpa} = 10^6 \text{ Pa}$

Note that the units in this system does not form a compatible system.

## 6. Create Report Tool

Tdyn CFD+HT includes a tool to comfortably create basic reports of the analysis, including text, images and graphs. This tool can be accessed both in the pre and post-processing part, through the menu option **Reports**.

'Open' creates an interactive report of the model.

The following commands are available in the '**Reports**' menu:

**Open:** creates the report of the model if it does not exist. Two options are shown:

**'Add default report pages':** The most relevant pages are automatically created.

**'Do not add. Only for notes':** None of the pages are created, they can be created manually at any time during the work.

**Delete:** deletes the existing report.

**Add automatic pages:** the most relevant pages are automatically created or updated if they already exist.

**Add current view:** current view can be added to the report in a new or an existing page.

Lognoter has been chosen for the report creation. Lognoter is a software to handle information organized by pages, the objective is to make easier the storage and management of information, notes and model information.

Lognoter can store textual formatted data, images and files all in the same place, providing a user friendly work environment.

## 7. Tcl Extension

Tdyn CFD+HT can be extended by using the Tcl scripting language. Tcl, or the "Tool Command Language", is a very simple, open-source-licensed programming language. Tcl provides basic language features such as variables, procedures, and control, and it runs on almost any modern OS, such as Unix, MacOS and Windows computers. But the key feature of Tcl is its extensibility.

You may find further information on Tcl at:

<http://wiki.tcl.tk/969>

Tdyn distribution includes a basic installation of Tcl, that allows to efficiently implement new capabilities in Tdyn. However full Tcl installation provides many tool-kits and libraries that can help in the implementation of above mentions Tdyn extensions.

The full Tcl version can be downloaded from:

<http://www.activestate.com/activetcl/>

Finally, you can use Ramdebbugger for editing and debugging Tcl code. Ramdebugger is free to use and can be downloaded from:

<http://www.compassis.com/ramdebugger>

## 7.1. Initiating Tcl extension

Tdyn CFD+HTTcl extension is initiated by selecting the *Use Tcl external script* option available in the **Fluid dynamics data > Other > Tcl data** page. If the check-box is selected, the Tcl extension of Tdyn CFD+HT is activated. The entry may indicate a Tcl script to be interpreted during Tdyn CFD+HT execution.

## 7.2. Tdyn Tcl event procedures

The Tcl script used for the Tdyn extension can optionally implement some of these Tcl event procedures (as well as other user-defined procedures). These procedures (listed below) are automatically called by Tdyn CFD+HT during execution, when the Tcl interface is activated. Their syntax corresponds to the standard Tcl language.

### • *TdynTcl\_InitiateProblem*

This procedure is called at the beginning of the execution, once all the data structures have been created.

### • *TdynTcl\_FinishProblem*

This procedure is called at the end of the execution of the current problem.

### • *TdynTcl\_StartNewFluidStep*

This procedure is called when a new time step is started in fluid domain.

### • *TdynTcl\_StartNewSolidStep*

This procedure is called when a new time step is started in solid domain.

### • *TdynTcl\_FinishFluidStep*

This procedure is called when a time step is finished in fluid domain.

### • *TdynTcl\_FinishSolidStep*

This procedure is called when a time step is finished in solid domain.

### • *TdynTcl\_FinishStep*

TdynTcl\_FinishStep is called once the current step is finished.

### • *TdynTcl\_AssembleFluidMomentumX,* *TdynTcl\_AssembleFluidMomentumY,* *TdynTcl\_AssembleFluidMomentumZ*

This procedure is invoked once the X/Y/Z component of the velocity momentum equation in the fluid domain has been assembled. It allows modifying Navier Stokes equations, for example by imposing boundary conditions for every

component of the velocity vector.

### • *TdynTcl\_AssembleSolidMomentumX,* *TdynTcl\_AssembleSolidMomentumY,* *TdynTcl\_AssembleSolidMomentumZ*

This procedure is invoked once the X/Y/Z component of the velocity momentum equation in the solid domain has been assembled. It allows modifying Navier Stokes equations, for example by imposing boundary conditions for every component of the velocity vector.

### • *TdynTcl\_AssembleFluidPressure*

This procedure is invoked once the continuity equation of the Navier Stokes equations in fluid domain has been assembled. It allows modifying Navier Stokes equations, for example by imposing boundary conditions for the pressure.

### • *TdynTcl\_AssembleSolidPressure*

This procedure is invoked once the continuity equation of the Navier Stokes equations in solid domain has been assembled. It allows modifying Navier Stokes equations, for example by imposing boundary conditions for the pressure.

### • *TdynTcl\_AssembleFluidODDLSPressure*

This procedure is invoked once the continuity equation of the Navier Stokes equations for ODD level set solver has been assembled. It allows modifying Navier Stokes equations, for example by imposing boundary conditions for the pressure.

### • *TdynTcl\_AssembleFluidTemperature*

This procedure is invoked once the temperature equation for the fluid has been assembled. It allows modifying temperature equation, for example by imposing boundary conditions for the fluid temperature field.

### • *TdynTcl\_AssembleSolidTemperature*

This procedure is invoked once the temperature equation for the solid has been assembled. It allows modifying temperature equation, for example by imposing boundary conditions for the solid temperature field.

### • *TdynTcl\_AssembleFluidSpecies index*

This procedure is invoked once the equation for the fluid species *index* has been assembled. The index of the species is sent to the procedure in the argument *index*. It allows modifying species equation, for example by imposing boundary conditions for the fluid species concentration field.

### • *TdynTcl\_AssembleSolidSpecies index*

This procedure is invoked once the equation for the solid species *index* has been assembled. The index of the species is sent to the procedure in the argument *index*. It allows modifying species equation, for example by imposing boundary conditions for the solid species concentration field.

### • *TdynTcl\_AssembleFluidPhiVariable index*

This procedure is invoked once the equation for the fluid phi variable defined by the index has been assembled. The index of the phi variable is sent to the procedure in the argument *index*. It allows modifying variable equation, for example by imposing boundary conditions for the fluid variable field.

### • *TdynTcl\_AssembleSolidPhiVariable index*

This procedure is invoked once the equation for the solid phi variable defined by the index has been assembled. The index of the phi variable is sent to the procedure in the argument *index*. It allows modifying variable equation, for example by imposing boundary conditions for the solid variable field.

- *TdynTcl\_CalculateDensityFromPressure*

This procedure is invoked once the density field has been updated in incompressible or slightly compressible flows.

- *TdynTcl\_CalculatePressureFromDensity*

This procedure is invoked once the pressure field has been updated in fully compressible flows.

- *TdynTcl\_AssembleFluidMeshDeformation*

This procedure is invoked once the structures required to perform the fluid mesh deformation are ready.

- *TdynTcl\_AssembleSolidMeshDeformation*

This procedure is invoked once the structures required to perform the solid mesh deformation are ready.

- *TdynTcl\_FinishFluidMeshDeformation*

This procedure is invoked once the fluid mesh deformation is done.

- *TdynTcl\_FinishSolidMeshDeformation*

This procedure is invoked once the solid mesh deformation is done.

- *TdynTcl\_MoveBody name*

This procedure is invoked once the body movement of body name is evaluated. It allows re-defining body movement.

### 7.3. Managing Tdyn data from the Tcl script

From the Tcl procedures defined in the Tdyn CFD+HT extension, it is possible to access Tdyn CFD+HT internal data. This access is done by means of the following functions:

*Remark:*

Unless otherwise indicated the arguments and returning values are given in internal units (those defined in the user interface).

The conversion factor to any other units can be obtained calling the procedures *TdynTcl\_UnitsToInternal* or *TdynTcl\_InternalToUnits*.

- *TdynTcl\_VecVal vec inode*

Returns the value of the variable identified by *vec*, corresponding to the node *inode*. *vec* must be one of the vector names defined in the section Function Syntax. Example: *TdynTcl\_VecVal Tm 10*

- *TdynTcl\_SetVecVal vec inode newvalue*

Set a single value of the vector identified by *vec*, corresponding to the node *inode*, to the value given by *newvalue*. *vec* must be one of the vector names defined in the section Function Syntax. Example: *TdynTcl\_SetVecVal Tm 10 0.0*

- *TdynTcl\_DVecVal vec inode idim*

Returns the value of the derivative in the direction given by *idim* (1 -x-, 2 -y-, 3 -z-) of variable identified by *vec*, corresponding to

the node *inode*. *vec* must be one of the vector names defined in the section Function Syntax. Example: *TdynTcl\_DVecVal Tm 10 2*

- *TdynTcl\_VecVals vec node\_list*

Returns a list with the values of the variable identified by *vec*, corresponding to the nodes in the list given by *node\_list*. *vec* must be one of the vector names defined in the section Function Syntax. Example: *TdynTcl\_VecVals Tm [list 10 12 14]*

- *TdynTcl\_VecValsAverage vec node\_list*

Returns the average of the values of the variable identified by *vec*, corresponding to the nodes in the list given by *node\_list*. *vec* must be one of the vector names defined in the section Function Syntax. Example: *TdynTcl\_VecValsAverage Tm [list 10 12 14]*

- *TdynTcl\_Coord inode idim*

Returns the coordinate *idim* (1 for x component, 2 for y component and 3 for z component) of the global node *inode*. The returning value is given in [m]. Example: *TdynTcl\_Coord 10 2*

- *TdynTcl\_Coords inode idom*

Returns the coordinates of the node (1 for x component, 2 for y component and 3 for z component) of the node *inode*. The returning value is given in [m]. The index of the can be global (*idom* = 0) fluid (*idom* = 1) or solid (*idom* = 2). Example: *TdynTcl\_Coords 10 0*

- *TdynTcl\_NNode itype*

Returns the number of nodes of the problem. If *itype* is 0, returns the total number of nodes, for *itype* 1 returns the number of fluid nodes and for *itype* 2 returns the number of solid nodes. Example: *TdynTcl\_NNode 0*

- *TdynTcl\_Dt*

Returns the current time increment. The returning value is given in [s]. Example: *TdynTcl\_Dt*

- *TdynTcl\_Time*

Returns the current physical time of the simulation (output time). The returning value is given in output units. Example: *TdynTcl\_Time*

- *TdynTcl\_PTime*

Returns the current physical time of the simulation (output time). The returning value is given in [s]. Example: *TdynTcl\_PTime*

- *TdynTcl\_Step*

Returns the current step of the simulation. Example: *TdynTcl\_Step*

- *TdynTcl\_Phase*

Returns 0 for during the *Initial Phase* of the calculation (if any) and 1 otherwise. Example: *TdynTcl\_Phase*

- *TdynTcl\_FixSystemRow idof val*

Fixes the *idof* row of the current system of equations to *val*. Example: *TdynTcl\_FixSystemRow 10 0.0*

- *TdynTcl\_DelSystemRow idof*

Sets to 0.0 all the entries of the *idof* row of the current system of equations. Example: *TdynTcl\_DelSystemRow 10*



### *TdynTcl\_GetSystemValue irow icol*

Returns the (irow, icol) position of the current system. Example: *TdynTcl\_GetSystemValue 10 15*

#### • *TdynTcl\_SetSystemValue irow icol val*

Prescribes to val the value of the (irow, icol) position of the current system. Example: *TdynTcl\_SetSystemValue 10 15 1.0*

#### • *TdynTcl\_GetRhs idof*

Returns the idof value of the right hand side vector of the current system of equations. Example: *TdynTcl\_GetRhs 10*

#### • *TdynTcl\_SetRhs idof val*

Sets the idof value of the right hand side vector of the current system of equations to val. Example: *TdynTcl\_SetRhs 10 0.0*

#### • *TdynTcl\_IsFluid inode*

Returns 1 if the index inode corresponds to a fluid node and 0 otherwise. Example: *TdynTcl\_IsFluid 10*

#### • *TdynTcl\_IsSolid idof*

Returns 1 if the index inode corresponds to a solid node and 0 otherwise. Example: *TdynTcl\_IsSolid 10*

#### • *TdynTcl\_GetFluidBodyNodes name*

Returns a list containing the indexes of the nodes of the FluidBody name. Example: *TdynTcl\_GetFluidBodyNodes fluid\_body*

#### • *TdynTcl\_GetFluidBodyElems name*

Returns a list containing the connectivities of the elements of the FluidBody name. Example: *TdynTcl\_GetFluidBodyElems fluid\_body*

#### • *TdynTcl\_GetSolidBodyNodes name*

Returns a list containing the indexes of the nodes of the SolidBody name. Example: *TdynTcl\_GetSolidBodyNodes solid\_body*

#### • *TdynTcl\_GetSolidBodyElems name*

Returns a list containing the connectivities of the elements the SolidBody name. Example: *TdynTcl\_GetSolidBodyElems solid\_body*

#### • *TdynTcl\_GetBodyArea name*

Returns the area of the body identified by name. Example: *TdynTcl\_GetBodyArea Fluid\_Body*

#### • *TdynTcl\_GetFluidNodes name*

Returns a list of the nodes belonging to the fluid Material identified by name. If no argument is given all nodes are returned. Example: *TdynTcl\_GetFluidNodes Fluid*

#### • *TdynTcl\_GetSolidNodes name*

Returns a list of the nodes belonging to the solid Material identified by name. If no argument is given all nodes are returned. Example: *TdynTcl\_GetSolidNodes Solid*

#### • *TdynTcl\_GetFluidElems name*

Returns a list containing the connectivities of the elements of the Fluid Material name. Example: *TdynTcl\_GetFluidElems fluid*

#### • *TdynTcl\_GetSolidElems name*

Returns a list containing the connectivities of the elements of the Solid Material name. Example: *TdynTcl\_GetSolidElems solid*

#### • *TdynTcl\_Message message type*

Print the notice or error given by message. type can be "error", "warning" or "notice". Error messages will stop calculation. Example: *TdynTcl\_Message "Tcl script executed correctly" notice*

#### • *TdynTcl\_UnitsConversor value in\_units out\_units magnitude*

Convert the value of the defined magnitude from the units given by in\_units to the units given by out\_units. Units format must follow the criteria defined in Units Syntax section. Example: *TdynTcl\_UnitsConversor 1.0 "[m]" "[mm]" Length*

#### • *TdynTcl\_SetGlobalVariable variable value*

Set the Tdyn variable to the given value. Available variables are: Number\_of\_Steps, Max\_Iterations, Total\_Time, OutPut\_Start, OutPut\_Step, and Gravity\_X/Y/Z.

Example: *TdynTcl\_SetGlobalVariable OutPut\_Start 10*

#### • *TdynTcl\_GetGlobalVariable variable*

Returns the value of the given Tdyn variable. Available variables are: Number\_of\_Steps, Max\_Iterations, Total\_Time, OutPut\_Start, OutPut\_Step, and Gravity\_X/Y/Z.

Example: *TdynTcl\_GetGlobalVariable OutPut\_Start*

#### • *TdynTcl\_GlobalToFluid inode*

Converts the global node index inode to local index in fluid domain. Example: *TdynTcl\_GlobalToFluid 10*

#### • *TdynTcl\_GlobalToSolid inode*

Converts the global node index inode to local index in solid domain. Example: *TdynTcl\_GlobalToSolid 10*

#### • *TdynTcl\_FluidToGlobal inode*

Converts the local node index inode in fluid domain to global index. Example: *TdynTcl\_FluidToGlobal 15*

#### • *TdynTcl\_SolidToGlobal inode*

Converts the local node index inode in solid domain to global index. Example: *TdynTcl\_SolidToGlobal 15*

#### • *TdynTcl\_SetFluidBodyVariable name variable value*

Set the variable of fluid body name to the given value. Available variables are: AccelerationX/Y/Z, RAccelerationX/Y/Z, DisplacementX/Y/Z, RotationX/Y/Z.

The value argument must be given in basic units [m],[s],[Kg],[N],[rad].

Example: *TdynTcl\_SetFluidBodyVariable fluid\_body DisplacementX 0.02*

#### • *TdynTcl\_GetFluidBodyVariable name variable*

Returns the value of the variable of fluid body name. Available variables are: ForceX/Y/Z, MomentX/Y/Z, AccelerationX/Y/Z, RAccelerationX/Y/Z, DisplacementX/Y/Z, RotationX/Y/Z. The returning variables are in basic units [m],[s],[Kg],[N],[rad].

Example: *TdynTcl\_GetFluidBodyVariable fluid\_body RAccelerationY*

#### • *TdynTcl\_SetSolidBodyVariable name variable value*

Set the variable of solid body name to the given value. Available variables are: AccelerationX/Y/Z, RAccelerationX/Y/Z, DisplacementX/Y/Z, RotationX/Y/Z.

The value argument must be given in basic units [m],[s],[Kg],[N],[rad].

Example: *TdynTcl\_SetSolidBodyVariable solid\_body DisplacementX 0.02*

• *TdynTcl\_GetSolidBodyVariable name variable*

Returns the value of the variable of solid body name. Available variables are: ForceX/Y/Z, MomentX/Y/Z, AccelerationX/Y/Z, RAccelerationX/Y/Z, DisplacementX/Y/Z, RotationX/Y/Z. The returning variables are in basic units [m],[s],[Kg],[N],[rad].

Example: *TdynTcl\_GetSolidBodyVariable solid\_body RAccelerationY*

• *TdynTcl\_GetFluidBodyInfo name info*

Returns some information of the fluid body name. Several options are available, depending on the value of the argument info:

- *nnormals*: a list of nodal normals for the body is returned.
- *enormals*: a list of normals of the body elements is returned.
- *nareas*: a list of the areas associated with every body node is returned.
- *eareas*: a list of the areas of every element of the body is returned.
- *area*: the area of the body is returned.

Example: *TdynTcl\_GetFluidBodyInfo fluid\_body nnormals*

• *TdynTcl\_GetSolidBodyInfo name info*

Returns some information of the solid body name. Several options are available, depending on the value of the argument info:

- *nnormals*: a list of nodal normals for the body is returned.
- *enormals*: a list of normals of the body elements is returned.
- *nareas*: a list of the areas associated with every body node is returned.
- *eareas*: a list of the areas of every element of the body is returned.
- *area*: the area of the body is returned.

Example: *TdynTcl\_GetSolidBodyInfo solid\_body eareas*

• *TdynTcl\_X/Y/Z*

Returns the x coordinate of the current node. This function can only be used in those tcl functions called from entries of Materials and Boundaries windows. The returning value is given in [m]. Example: *TdynTcl\_Y*

• *TdynTcl\_Index type*

Returns the index of the current node. Depending on type, the global (type = 0), fluid (type = 1) or solid (type = 2) index is returned. This function can only be used in those tcl functions called from entries of Materials and Boundaries windows.

• *TdynTcl\_ODDLSLevel inode*

Returns the level of the node inode given by the levelset function (i.e. nodes connected to free surface have level 1 for the fluid of interest and level -1 for the rest, the rest of the

nodes have a increasing (or decreasing) level given by the shortest path required to arrive to 1 or -1 nodes). Example: *TdynTcl\_ODDLSLevel 10*

• *TdynTcl\_Create\_Interpolator*

Creates an interpolator structure and returns its name. An interpolator can be used to interpolate data from nodal values of an initial mesh to a final mesh or the other way around.

Example: *set interpolator [TdynTcl\_Create\_Interpolator]*

• *TdynTcl\_Release\_Interpolator*

Deletes an interpolator structure. Example: *TdynTcl\_Release\_Interpolator \$interpolator*

• *TdynTcl\_Read\_Interpolator\_Mesh interpolator initial/final mesh\_file*

Reads a mesh file and inserts it in the interpolation structure. The arguments are the interpolator name, initial or final (i.e. original or final mesh) and the file name.

The mesh file must use the standard GiD ASCII format (see [http://www.gidhome.com/support\\_team/](http://www.gidhome.com/support_team/) for further information).

Example: *TdynTcl\_Read\_Interpolator\_Mesh \$interpolator initial {C:/Temp/meshi.msh}*

• *TdynTcl\_Insert\_Interpolator\_Mesh interpolator initial/final name*

Inserts a body mesh in the interpolation structure. The arguments are the interpolator name, initial or final (i.e. original or final mesh) and the body name or the keywords "fluid\_mesh" or "solid\_mesh" for inserting the full fluid/solid mesh.

Example: *TdynTcl\_Insert\_Interpolator\_Mesh \$interpolator initial fluid\_body*

• *TdynTcl\_OnInitial\_Interpolator interpolator vectorA vectorB*

Performs an interpolation of vectorA (nodal values of the final mesh) to vectorB (resulting nodal values of the initial mesh). The vectorB will be overwritten with the interpolated values.

Example: *TdynTcl\_OnInitial\_Interpolator \$interpolator \$vectorA \$vectorB*

• *TdynTcl\_OnFinal\_Interpolator interpolator vectorA vectorB*

Performs an interpolation of vectorA (nodal values of the initial mesh) to vectorB (resulting nodal values of the final mesh). The vectorB will be overwritten with the interpolated values.

Example: *TdynTcl\_OnFinal\_Interpolator \$interpolator \$vectorA \$vectorB*

• *TdynTcl\_InternalToUnits units value*

Convert a value from internal units to the units given as argument. If the value argument is omitted, the conversion factor for 1.0 is returned. See section [Units Syntax](#) for information about units syntax.

Example: *TdynTcl\_InternalToUnits {[m/s]}*

• *TdynTcl\_UnitsToInternal units value*

Converts value from the units given as argument to internal units. If the value argument is omitted, the conversion factor for 1.0 is returned. See section [Units Syntax](#) for information about

units syntax.

Example: `TdynTcl_InternalToUnits {[bar]}`

- `TdynTcl_SetSolveVariableOnvariable domain`

Activates the solution of the Tdyn problem associated to the specified variable (0 = within the total domain, 1 = within the Fluid, 2 = within the Solid). Example: `TdynTcl_SetSolveVariableOn velocity 0`

- `TdynTcl_SetSolveVariableOff variable domain`

Deactivates the solution of the Tdyn problem associated to the specified variable (0 = within the total domain, 1 = within the Fluid, 2 = within the Solid). Example: `TdynTcl_SetSolveVariableOff velocity 1`

- `TdynTcl_Clock codename start/end`

Starts or finish a time measurement procedure. The results are saved in the standard time table.

Example: `TdynTcl_Clock "Start time control" start`

- `TdynTcl_ProjectFileName:`

Returns the name of the current project. Example: `TdynTcl_ProjectFileName`

- `TdynTcl_GetFluidInfo coords/xcoords/ycoords/zcoords vector`

Returns coordinates of fluid material nodes, storing them in a vector. Different options are available, depending on the value of the argument info:

- `coords`: a list of nodal coordinates for the fluid is returned.
- `xcoords`: a list of nodal X coordinates for the fluid is returned.
- `ycoords`: a list of nodal Y coordinates for the fluid is returned.
- `zcoords`: a list of nodal Z coordinates for the fluid is returned.

Example:

```
set nnod [TdynTcl_NNode 1]

set x [::mather::mkvector $nnod 0.0]

TdynTcl_GetFluidInfo xcoords $x
```

- `TdynTcl_Create_DistanceToMesh body/file [fast/precise]`

Creates a structure to calculate distance to the mesh given in a body/boundary of mesh file in GiD format.

- `TdynTcl_Insert_DistanceToMesh mdistance body/file`

This procedure adds a new mesh into a previously created mdistance

- `TdynTcl_Update_DistanceToMesh mdistance body/file [mesh index]`

Updates mdistance's nodes position from a new file of due to the movement of body/boundary.

If any body movement is performed during calculation, this will only be taken into account after calling to this function.

- `TdynTcl_Calculate_DistanceToMesh mdistance body vector [numtype]`

Calculates the distance to the nodes of the given body/boundary. The results are stored in the given vector.

- `TdynTcl_Delete_DistanceToMesh [mdistance]`
- `TdynTcl_Release_DistanceToMesh [mdistance]`

Deletes a mdistance (all mdistances are released if no argument is given).

## 7.4. Calling Tcl procedures

Most of the material and data fields of Tdyn CFD+HT can be defined by Tcl procedures. This can be done by using the `tcl` function, that executes a Tcl script or procedure returning a double value. The syntax of this function is `tcl(.)` where the argument is the script to be executed. Examples:

- `tcl(set var)`: return the value of the Tcl variable var.
- `tcl(myproc arg)`: the procedure myproc is called with argument arg. Procedure myproc must be defined in the Tcl script selected in the `Tcl extension` entry and must return a double value.

Note that in order to use this function, Tcl extension must be activated by selecting `Tcl extension`.

## 7.5. Tdyn Tcl math library

Tdyn CFD+HT Tcl extension includes a basic library for vector operation and manipulation. The functions for vector manipulation can operate with temporal vector created from Tcl code and with internal Tdyn CFD+HT variable vectors. Internal Tdyn CFD+HT vectors can be accessed using the standard names for variables defined in Function Syntax section, but they have to be preceded by an 's' (for solid domain vectors) or 'f' (for fluid domain vectors). This way, the vector containing the x components of the velocity in the fluid domain can be accessed by 'fvx', and the temperature of the solid domain by 'stm'.

*Remark:*

Internal Tdyn CFD+HT vectors are in internal units (those defined in the user interface). The conversion factor to any other units can be obtained calling the procedures `TdynTcl_UnitsToInternal` or `TdynTcl_InternalToUnits`.

The basic functions of this library are described below.

- `vmexpr function(vectorname)`

Performs the operation given by function in the vector `vectorname`, returning a real. Functions available for `vmexpr` are those defined in `vmevaluate`. Example: `set ret [vmexpr sum_abs(fvx)]`

- `vmexpr vector=linear_operation or vmexpr temp=function(linear_operation)`

Calculates a linear combination of vectors, returning the results in `vector`. If `vector` is "temp" a new temporal vector is returned. A `function` can be applied to the results of the linear combination. Functions available for `vmexpr` are those defined in `mather_apply_vectors`.

Examples:

```
set temp [vmexpr temp=abs(2.0*fvy)]
```

```
set temp [vmexpr temp=2.0*stm]
```

```
vmexpr $phis=$phi2+$phi1
```

```
set temp [vmexpr temp=fpr-1.]
```

```
set temp [vmexpr temp=-1.0*$phi1+[expr  
$a+$b]*$phi2+3.0*$phi3+1.0]
```

The following functions are also available under the namespace `::mather::` with the same name, but without the prefix `wm`.

Example:

```
::mather::mkvectororvmmkvector.
```

- `vmmkvector vectorsize defaultvalue`

Creates a temporal vector of size `vectorsize`. This vector is initiated to `defaultvalue`. The function returns the name of the vector.

Example:

```
set temp [vmmkvector 100 0.0]
```

- `vmdelete vectorname`

Deletes the temporal vector named `vectorname`.

Example:

```
vmdelete vector2
```

- `vmvector_info name type`

Access to information of the vector given by name. `type` can be: `length` (length of the vector), `min_index` (index of the minimum of the elements), `max_index` (index of the maximum of the elements), `min_abs_index` (index of the minimum in absolute value of the elements), `max_abs_index` (index of the maximum in absolute value of the elements) or `values` (return a list with the elements of the vector).

Example:

```
vmvector_info $res values
```

### `vmdelete`

Deletes all the temporal vectors.

Example:

```
vmdelete
```

- `vmdimtemps isize`

Defines the dimension of the array of temporal vectors to `isize`. By default Tdyn CFD+HT only allows to use 100 temporal vectors. This function has to be called before to start using temporal vectors.

Example:

```
vmdimtemps 150
```

- `vmsetelem vectorname index newvalue`

Set a single element of a vector to a new value.

Example:

```
vmsetelem stm 150 0.0
```

- `vmgetelem vectorname index`

Returns a single element of a vector.

Example:

```
vmgetelem stm 150
```

- `vmvector_print vectorname`

Returns a string containing all the the index and value of every element of the vector.

Example:

```
vmvector_print fvy
```

- `vmapply function vectorname`

Apply a *function* to every value of the vector named `vectorname`. Available functions are `abs`, `sqrt`, `cos`, `sin`, `tan`, `acos`, `asin`, `atan`, `cosh`, `sinh`, `tanh`, `log`, `log10`, `exp`, `square`, `inverse` and `opposite`.

Example:

```
vmapply sqrt temp2
```

- `vmevaluate function vectorname`

Apply a function to a vector named *vectorname*. Available functions are *sum*, *sum\_abs*, *min\_val*, *max\_val*, *min\_abs*, *max\_abs*, *min\_signed*, *max\_signed*, *sumsq*, *norm*, *mean*, *fintegral*, *sintegral*, *fintegralnorm*, *sintegralnorm*.

Examples:

*vmevaluate min\_val temp2*

*vmevaluate sintegral stm*

*vmevaluate fintegral ftm*

- *vmupdate vectorname newvalue*

Sets the  $i^{\text{th}}$  value of *vectorname* to the  $i-1^{\text{th}}$ . The last value of *vectorname* is set to *newvalue*.

- *vmexists\_vector vectorname*

Returns 1 if the vector *vectorname* exists in the current problem. *vectorname* can be any of the standard vector names of Tdyn CFD+HT shown in Function Syntax section.

- *vmexists\_smatrix matrixname*

Returns 1 if the matrix *matrixname* exists in the current problem. The argument *matrixname* is generally composed of a key name and a prefix (f- for fluid domain and s- for solid domain):

$$N\_DNx: \text{matrix } \int_{\Omega} N_i \cdot dN_j / dN_x d\Omega$$

$$N\_DNy: \text{matrix } \int_{\Omega} N_i \cdot dN_j / dN_y d\Omega$$

$$N\_DNz: \text{matrix } \int_{\Omega} N_i \cdot dN_j / dN_z d\Omega$$

$$DN\_DN: \text{matrix } \int_{\Omega} \nabla N_i \cdot \nabla N_j d\Omega$$

$$DNx\_DNx: \text{matrix } \int_{\Omega} dN_i / dN_x \cdot dN_j / dN_x d\Omega$$

$$DNx\_DNy: \text{matrix } \int_{\Omega} dN_i / dN_x \cdot dN_j / dN_y d\Omega$$

$$DNx\_DNz: \text{matrix } \int_{\Omega} dN_i / dN_x \cdot dN_j / dN_z d\Omega$$

$$DNy\_DNy: \text{matrix } \int_{\Omega} dN_i / dN_y \cdot dN_j / dN_y d\Omega$$

$$DNy\_DNz: \text{matrix } \int_{\Omega} dN_i / dN_y \cdot dN_j / dN_z d\Omega$$

$$DNz\_DNz: \text{matrix } \int_{\Omega} dN_i / dN_z \cdot dN_j / dN_z d\Omega$$

$$SDNx\_DNy: \text{matrix } \int_{\Omega} dN_i / dN_x \cdot dN_j / dN_y d\Omega + \int_{\Omega}$$

$$dN_i / dN_j \cdot dN_j / dN_x d\Omega$$

$$SDNx\_DNz: \text{matrix } \int_{\Omega} dN_i / dN_x \cdot dN_j / dN_z d\Omega + \int_{\Omega} dN_i / dN_z \cdot dN_j / dN_x d\Omega$$

$$SDNy\_DNz: \text{matrix } \int_{\Omega} dN_i / dN_y \cdot dN_j / dN_z d\Omega + \int_{\Omega} dN_i / dN_z \cdot dN_j / dN_y d\Omega$$

$$N\_N: \text{matrix } \int_{\Omega} N_i N_j d\Omega$$

$$STAB: \text{matrix } \int_{\Omega} \nabla N_i (v \cdot \nabla) \nabla N_j d\Omega$$

Furthermore, the system matrix of a problem is accessed with the key name *SYS*, and the system matrix for every component of the velocity vector with the key name *VSYS*.

Finally, the mass matrix of the boundary mesh of any body can be accessed using the prefix f- or s- and the name of the corresponding body.

*Remark:*

Any of these matrices can be or not available depending on the problem.

Examples: *fn\_n*, *sbody*, *fdn\_dn*, *sdnx\_dnx*.

- *vmsmatrix\_getelem matrixname irow icol*

Returns the element of the matrix in the position *irow icol*.

- *vmsmatrix\_setelem matrixname irow icol value*

Sets the element of the matrix in the position *irow icol* to *value*.

- *vmmatrix\_copy matrixname matrixname2*

Copies *matrixname2* to *matrixname*.

- *vmmatrix\_add matrixname matrixname2*

Adds *matrixname2* to *matrixname*.

- *vmmult\_matrix\_per\_vec matrixname vectorname vectorname2*

Multiplies *matrixname* by *vectorname* and saves the result in *vectorname2*.

- *vmmult\_matrix\_per\_vec\_add matrixname vectorname vectorname2*

Multiplies *matrixname* by *vectorname* and adds the result to *vectorname2*.

- *vmmult\_matrix\_per\_vec\_sub matrixname vectorname vectorname2*

Multiplies *matrixname* by *vectorname* and subtracts the result from *vectorname2*.

- *vmmult\_matrixt\_per\_vec matrixname vectorname vectorname2*

Multiplies the transpose of *matrixname* by *vectorname* and saves the result in *vectorname2*.

- *vmmult\_matrixt\_per\_vec\_add matrixname vectorname vectorname2*



*vectorname2*

Multiplies the transpose of *matrixname* by *vectorname* and adds the result to *vectorname2*.

- *vmmult\_matrixt\_per\_vec\_sub matrixname vectorname vectorname2*

Multiplies the transpose of *matrixname* by *vectorname* and subtracts the result from *vectorname2*.

- *vmmatrix\_vector\_mult\_add matrixname vectorname matrixname2 {add\_arg}*

Multiplies *matrixname* by the elements of *vectorname* and adds the result to *matrixname2* ( $\mathbf{B}=\mathbf{A}\cdot\mathbf{v}$ ). If the additional *add\_arg* is given, other type of matrix-vector operations can be done. For *add\_arg*=1, the operation carried out is  $\mathbf{B}=1/2\cdot[\mathbf{A}\cdot\mathbf{v}+(\mathbf{A}^T\cdot\mathbf{v})^T]$ . For *add\_arg*=2, the operation done is similar to the previous case, but the result of the operation is limited if the element of the matrix is negative. For *add\_arg*=3, the operation performed is  $\mathbf{B}=(\mathbf{A}^T\cdot\mathbf{v})^T$ .

- *vmmatrix\_scalar\_mult\_add matrixname value matrixname2*

Multiplies *matrixname* by *value* and adds the result to *matrixname2*.

- *vmmult\_vector\_per\_vec vector vector2 resvector*

Multiplies two vectors (*vector* and *vector2*) component by component and saves the result in *resvector*.

- *vmquot\_vector\_per\_vec vector vector2 resvector*

Calculates the quotient of the components of two vectors (*vector* and *vector2*) and saves the result in *resvector*.

- *vmcreate\_function domain function*

*::mather::create\_function domain function*

Creates a Tdyn function to be used in the Tcl script. Domain must be fluid/solid/waves (waves is used for Seakeeping analysis). The functions returns the function name assigned.

Further information on the Tdyn functions syntax can be find at [Function Syntax](#).

- *vmevaluate\_function function [ipnt]*

*::mather::evaluate\_function function [ipnt]*

Evaluates a Tdyn function created in the Tcl script. If *ipnt* is given, the function is evaluated for the node of index *ipnt*.

- *vmdelete\_function function\_name*

*::mather::delete\_function [function\_name]*

Deletes a previously created function given by *function\_name*.

- *mather\_initcoupling type port timeout*

*::mather::initcoupling type port timeout*

Initiate coupling library of Tdyn. Coupling library allows to interchange information at memory level among two Tdyn instances.

type must be 0,1,2 indicating whether this instance will act as server (1), client (2) or any of them (0, randomly selected).

- *mather\_insertcouplingmesh name/file*

*::mather::insert\_coupling\_mesh name/file*

Inserts a mesh to interpolate data from one side of the communication interface to the other. The input data must be the internal name of a body mesh or a file containing a mesh in GiD ASCII format.

- *mather\_retrievecouplingheader*

*::mather::retrieve\_coupling\_header*

Starts a reading communication sequence. The execution of the current instance is paused until a confirmation of reception is received. This confirmation must be sent by other instance by a call to *::mather::send\_coupling\_vector*.

The procedure returns a Tcl list containing the data type id (integer), the corresponding time step (double) and the size of the vector that will be sent immediately (integer).

- *mather\_sendcouplingvector name type step*

*::mather::send\_coupling\_vector name type step*

Sends a vector given by name to other Tdyn instance.

- *mather\_retrievecouplingvector name*

*::mather::retrieve\_coupling\_vector name*

Reads the information sent by other Tdyn instance by means of *::mather::send\_coupling\_vector*, and saves the data to vector name. A previous call to *::mather::retrieve\_coupling\_header* is required.

- *mather\_retrievecouplingconvergence*

*::mather::retrieve\_coupling\_convergence*

Reads the convergence information sent by other Tdyn instance by means of *::mather::send\_coupling\_convergence*, and return 1 (convergence obtained) or 0 (not converged).

- *mather\_sendcouplingconvergence step 0/1*

*::mather::send\_coupling\_convergence*

Sends convergence information to other Tdyn instance. The arguments are the time step and the convergence status (1 for 'convergence obtained', or 0 for 'not converged').

## 7.6. Examples of scripts defining a Tcl extension

The scripts below show examples of procedures defining a Tdyn CFD+HT Tcl extension. In order to execute these procedures, they have to be saved to a file and the file has to be inserted in the *Tcl extension* entry in the **Fluid Dynamics & Multi-PhysicsData > Other | General data** page.

More practical examples of Tcl extension can be found in the Tdyn Tutorials manual: see Ekman's Spiral and Taylor-Couette flow tutorials.

### • Tcl script example 1: Write a notice in Tdyn's info file

The following script (*TdynTcl\_AssembleFluidSpecies*) is executed every time the system of equations for species problem is assembled. It just writes a message to the standard Tdyn CFD+HT output (**Calculate > View process info...**).

```
proc TdynTcl_AssembleFluidSpecies { ispecies } {  
    # Reading Tdyn CFD+HT internal time
```

```
set t [TdynTcl_Time]
# Writing a message in Tdyn CFD+HT info window
TdynTcl_Message "Executing TdynTcl_AssembleFluidSpecies
$ispecies: time $t" notice
}
```

#### • Tcl script example 2: Fix pressure values

The following script is invoked every time the pressure system of equations is assembled. It fixes the value of the pressure to  $y^2+1$ .

```
proc TdynTcl_AssembleFluidPressure {} {
    # Reading the number of total nodes
    set nnode [TdynTcl_NNode 0]
    # Imposing boundary conditions
    for {set i 1} {$i <= $nnode} {incr i} {
        # Check if the node is in the fluid domain

        if { [TdynTcl_IsFluid $i] } {

            # Read y coordinate of the node i

            set y [TdynTcl_Coord $i 2]

            # Fix degree of freedom i to  $y^2+1$ 

            TdynTcl_FixSystemRow $i [expr pow($y,2)+1]

        }
    }
}
```

#### • Tcl script example 3: Write results in a file

The following script is invoked once the current time step is finished. It writes the velocity values of several points to a file.

```
proc TdynTcl_FinishStep {} {
    # Open file "C:/Temp/writing_in_this_file" ...
    cd {C:/Temp}
    set fileid [open writing_in_this_file w+]
    # ... and writes some info for a list of nodes
    set nodelist [list 1 2 3 4 5]
    puts $fileid "Velocity info for time $t"
    foreach inode $nodelist {
        puts $fileid "Velocity of node $inode: \

        [TdynTcl_VecVal vx $inode] \

        [TdynTcl_VecVal vy $inode] \

        [TdynTcl_VecVal vz $inode]"
    }
}
```

```
# Finally close the file
close $fileid
}
```

#### • Tcl script example 4: Impose traction on fluid body

The following script is called every time the velocity system of equations is assembled. It imposes a traction on the nodes of the fluid body "wind" that is defined through the user interface in the standard way.

```
proc TdynTcl_AssembleFluidMomentumX {} {
    TdynTcl_Message "Imposing traction in fluid momentum X
equation" notice
    # Value of traction
    set value 1.0
    # Read the list of nodes of the fluid body "Wind"
    set nodes [TdynTcl_GetFluidBodyNodes wind]
    # Create a traction vector
    set nnode [TdynTcl_NNode 1]
    set tract [::mather::mkvector $nnode 0.0]
    foreach inode $nodes {
        set jnode [TdynTcl_GlobalToFluid $inode]
        ::mather::setelem $tract $jnode $value
    }
    # Calculate the FEM integral of the traction
    set temp [::mather::vmexpr temp=fwind*$tract]
    # Assemble the terms
    foreach inode $nodes {
        set jnode [TdynTcl_GlobalToFluid $inode]
        set ri [TdynTcl_GetRhs $jnode]
        set ti [::mather::getelem $temp $jnode]
        TdynTcl_SetRhs $jnode [expr $ri+$ti]
    }
    # Delete the temporal vectors
    ::mather::delete $tract
    ::mather::delete $temp
}
```

#### • Tcl script example 5: Loading Tk package

The following script loads Tk package. Tk is Tcl a library, including basic elements for building a graphical user interface. Once this library is loaded, graphic elements can be created from the Tdyn Tcl interface. In this example, a text window is created and then every time step, the text "Step \$Current\_Step\$" is printed in that window.

In the following code, the full Tcl installation is assumed to be in the directory "C:\Program Files\Tcl\lib". The full Tcl installation

can be downloaded from <http://www.activestate.com/activetcl/>.

*# Define directory of the Tcl installation*

*lappend ::auto\_path {C:\Program Files\Tcl\lib}*

*TdynTcl\_Message [package require Tk] notice*

*# Creates a text window*

*pack [text .t -width 70 -height 20]*

*# Prints information in the text window*

*proc TdynTcl\_StartNewFluidStep { } {*

*set step [TdynTcl\_Step]*

*.t ins end "Step \$step\n"; .t see end; update*

*}*

#### • **Tcl script example 6: Interpolates data from one mesh to other**

The following example interpolates the nodal values of the pressure from a Fluid Body mesh of Tdyn, on a mesh read from a file. Both meshes must represent the same geometry.

The mesh file read from a file must use the standard GiD ASCII format (see [http://www.gidhome.com/support\\_team/](http://www.gidhome.com/support_team/) for further information).

*proc TdynTcl\_FinishProblem { } {*

*# Creates an interpolator structure*

*set interpolator [TdynTcl\_Create\_Interpolator]*

*# Insert first mesh (A) - Fluid Body called "Wall/Bodies Auto1"*

*TdynTcl\_Insert\_Interpolator\_Mesh \$interpolator initial "Wall/Bodies Auto1"*

*# Insert second mesh (B) from file C:/Temp/bmesh.msh*

*TdynTcl\_Read\_Interpolator\_Mesh \$interpolator final {C:/Temp/bmesh.msh}*

*# Creates a vector (dimension of the number of nodes of mesh B is set to nnod)*

*set nnod 587*

*set res [::math::mkvector \$nnod 0.0]*

*# Performs an interpolation of the pressure values in mesh A to mesh B*

*TdynTcl\_OnFinal\_Interpolator \$interpolator fpr \$res*

*# Writes the resulting values in a file (GiD postprocessing format)*

*cd {C:/Temp}*

*set fileid [open output\_file w+]*

*puts \$fileid "GiD Post Results File 1.0"*

*puts \$fileid "Result Pressure Analysis 1 Scalar OnNodes"*

*puts \$fileid "Values"*

*puts \$fileid [::math::vector\_print \$res]*

*puts \$fileid "End Values"*

*close \$fileid*

*# Deletes interpolator structure*

*TdynTcl\_Release\_Interpolator \$interpolator*

*}*

#### • **Tcl script example 7: Debugging a Tcl script with Ramdebugger**

RamDebugger is a graphical debugger for Tcl-TK. With RamDebugger, it is possible to make Local Debugging, where the debugger starts the program to be debugged, and Remote debugging, where the program to debug is already started and RamDebugger connects to it. The latter option will be used in this case.

*Remark:*

In Windows, it is necessary to load the package comm (not the standard, but the one modified in RamDebugger), in order to debug the program remotely.

To debug the following example:

1. open it with Ramdebugger and set a breakpoint in the line "TdynTcl\_Message "Set a breakpoint in this line" notice".
2. Then execute Tdyn, and wait for a few seconds, until the execution freezes.
3. Go to Ramdebugger and select .

**File ► Debug on ► Tdyn**

4. Tdyn execution will restart until the breakpoint is find.

*Remark:*

If Tdyn is not included in the list of "Remote TCL debugging" programs, select to update the list.

**File ► Debug on ► Update remotes**

*# Load package commR*

*# Change the directory below with the one where Ramdebugger is intalled*

*lappend ::auto\_path {C:/Utils/RamDebugger7.0/addons}*

*TdynTcl\_Message [package require commR] notice*

*set ret [package require commR]*

*# This register Tdyn for debugging*

*comm::register Tdyn 1*

*# Add breakpoint beyond this point.*

*# breakpoints only work inside a proc.*

*proc TdynTcl\_FinishStep { } {*

*TdynTcl\_Message "Set a breakpoint in this line" notice*

*TdynTcl\_Message "Click the arrow button to go to this line" notice*

*TdynTcl\_Message "Click the arrow button to go to this line" notice*

*}*

*# Program gets stopped here waiting for the debugger to connect*

*commR::wait\_for\_debugger*

Tcl script example 8: Solving the wave equation using URSOLVER

This example shows how to solve the following wave equation:

$$d^2\phi/dt^2 - c^2 \cdot \Delta\phi = 0$$

The procedure to solve it requires to create a new phi variable (ph1) using URSOLVER module. All the properties of the new variable have to be set to zero except f2, that have to be set to the square of the wave velocity (phase speed). In the following picture f2 is set to the value corresponding to a gravity wave of 11 m length.

Definition of the properties of the phi variable.

The model has to be completed with the proper initial and boundary conditions.

The following script modifies the assembling process of the ph1 problem, by adding the temporal term of the wave equation.

```
set x1 ""
set x2 ""

proc TdynTcl_StartNewFluidStep {} {
    global x1 x2

    # Create two vectors to save ph1(t-dt) and ph1(t)

    set nnod [TdynTcl_NNode 1]

    # x1 is initiated with ph1' values

    if { $x1 eq "" } {
        set x1 [::mather::mkvector $nnod 0.0]
    }

    if { $x2 eq "" } {
        set x2 [vmexpr temp=fph1]
    }
}
```

```
proc TdynTcl_AssembleFluidPhiVariable { index } {
    global x1 x2

    if { $index != 1 } { return }

    set nnod [TdynTcl_NNode 1]

    set step [TdynTcl_Step]

    set dt_i [expr 1.0/[TdynTcl_Dt]]

    set dt2i [expr pow($dt_i,2.0)]

    # Assemble temporal term (ph1'')

    if { $step == 1 } {
        set x2_ [vmexpr temp=$dt2i*$x2]
        set x1_ [vmexpr temp=$dt_i*$x1]
    } else {
        set x2_ [vmexpr temp=2.0*$dt2i*$x2]
        set x1_ [vmexpr temp=-$dt2i*$x1]
    }

    # Assemble ph1(t+dt)/dt^2

    ::mather::matrix_scalar_mult_add fsys $dt2i fn_n

    # Assemble (right hand side) 2.0*ph1(t)/dt^2

    ::mather::mult_matrix_per_vec fn_n $x2_ frhs

    # Assemble (right hand side) -ph1(t-dt)/dt^2

    ::mather::mult_matrix_per_vec_add fn_n $x1_ frhs

    ::mather::delete $x1_
    ::mather::delete $x2_
}

proc TdynTcl_FinishFluidStep {} {
    global x1 x2

    # Update phi values

    vmexpr $x1=$x2
    vmexpr $x2=fph1
}
```

#### • Tcl script example 8: Distance to body calculation

The following example, creates the structure to calculate the distance to the body "group2". Afterwards, the distance is calculated every step and stored in the vector "vector0".

```
set myvec ""

set mydistance ""

proc TdynTcl_InitiateProblem {} {
    global myvec mydistance

    TdynTcl_Clock "Distance evaluation" start
```

```
set myvec [::math::mkvector vector0 [TdynTcl_NNode 2] 0.0 ]

set mydistance [TdynTcl_Create_DistanceToMesh group2
precise]

TdynTcl_Clock "Distance evaluation" end

}

proc TdynTcl_FinishSolidStep {} {

global myvec mydistance

if { [TdynTcl_Step] == 1 } {

set name "Distance evaluation 1"

} else {

set name "Distance evaluation"

}

TdynTcl_Clock $name start

# Note numtype argument (0,1 or 2) is optional in the following

TdynTcl_Update_DistanceToMesh $mydistance group2

TdynTcl_Calculate_DistanceToMesh $mydistance group1
$myvec

TdynTcl_Clock $name end

}
```

#### Tcl script example 9: Distance to body calculation

## 8. Some comments on Boundary Conditions

In many real applications, there is a frequent difficulty in defining some of the boundary conditions at the inlet and outlet of a calculation domain in the detail that is needed for an accurate simulation. A typical example is the specification of turbulence properties (turbulence intensity and length scale) at the inlet flow boundary, as these are rarely available for a new design configuration. Other examples are the specification of the boundary layer velocity profile on the walls at an inlet, or the precise distribution of a species concentration at an inlet boundary.

Next a brief list of recommendations about boundary conditions prescription is given.

### 8.1. General guidelines on boundary conditions

- The velocity and turbulence variables have to be prescribed at the inlet boundary. The pressure is sometimes also specified at the inlet boundary. Turbulence variables are automatically prescribed at the inlet boundaries in Tdyn CFD+HT.
- If the conditions at inlet are not well known, examine the possibilities of moving the domain boundaries to a position where boundary conditions are better identified.
- Check whether upstream or downstream obstacles (such as bends, contractions, diffusers, etc.) outside of the flow domain are present which could significantly affect on the flow distribution. Often, information about components upstream or downstream of the domain is lacking or not available at the beginning of a project.
- For each class of problem that is of interest, carry out a sensitivity analysis in which the boundary conditions are

systematically changed within certain limits to see the variation in results. Should any of these variations prove to have a stronger effect on the simulated results, and lead to large changes in the simulation, it is necessary to obtain more accurate data on the boundary conditions that are specified.

- Place open boundary conditions (outflow, or pressure prescription) as far away from the region of interest as possible and avoid open boundaries in regions of strong geometrical changes or in regions of re-circulation.
- Pay an extra attention to the orientation of outlet planes with regard to the mean flow, especially when the boundary condition consists of a constant pressure profile.
- Select the boundary conditions imposed at the outlet to have only a weak influence on the upstream flow. Extreme care is needed when specifying flow velocities and directions on the outlet plane.
- Particular care should be taken in strongly swirling flows where the pressure distribution on the outlet boundary is strongly influenced by the swirl. It is therefore not acceptable to specify constant pressure across the outlet.
- Be aware of the possibility of inlet flow inadvertently occurring at the outflow boundary, during the simulation process. This fact may lead to difficulties in obtaining a stable solution or even to an incorrect solution. If it is not possible to avoid this by relocating the position of the outlet boundary in the domain, try to avoid the problem by restricting the flow area at the outlet, provided so that the outflow boundary is not near the region of interest. If the outflow boundary condition allows the flow to re-enter the domain, the value of all transported variables should be imposed as in an inlet boundary.
- If there are multiple outlets, impose either pressure boundary conditions or velocity specifications depending on the known quantities.
- Tdyn CFD+HT allows not making any prescription in the pressure field. If no prescription of the pressure is done, some instabilities may appear.
- Be aware that Tdyn CFD+HT will impose default boundary conditions for regions of the domain boundaries, where the user has not specified anything. Tdyn CFD+HT also fixes all the turbulence variables in those entities where all components of the velocity have been prescribed. These prescriptions are done due to numerical stability reasons and in most of the cases will not affect the results.
- If possible, carry out a sensitivity analysis in which the key inlet boundary conditions are systematically changed within certain limits. Depending on the problem, the key parameters that might be examined are: inlet flow direction and magnitude, uniform distribution of a parameter or a profile specification (for example a uniform inlet velocity or an inlet velocity profile), physical parameters and turbulence properties at inlet.

## 9. Function Syntax

Most of the material and data fields of Tdyn CFD+HT may be defined by functions, through a "Function editor". Those data fields can be inserted by means of the function editor by pressing buttons shown in the following figure. Note that these buttons are only available in those pages where any function field exists.



Function editor button





The "Functions editor" allows for comfortable definition of complex functions and matrices. Such an editor includes some tools to help the insertion of functions as for instance a list of available variables and operators.

Next a brief explanation of the syntax of the functions used in Tdyn CFD+HT is given.

#### Variables:

The variables that can be used for the definition of the functions are presented next.

Global variables, available for any function:

- t* : Total (physical) time
- it* : Interval time. Time from the last restarting of the problem.
- dt* : Time increment.
- rand* : Random number.
- step* : Current step of the analysis.

Geometrical variables, available for functions in domain scope (fluid or solid domain functions):

- x* : X cartesian coordinate of the point.
- y* : Y cartesian coordinate of the point.
- z* : Z cartesian coordinate of the point.
- rhoc* :  $\rho$  corresponding to cylindrical coordinates of the point.  $\rho$  is given by  $\rho = \sqrt{x^2 + y^2}$
- thetac* :  $\theta$  corresponding to cylindrical coordinates of the point.  $\theta$  is given by  $\theta = \text{atan}(y/x)$ .
- rhos* :  $\rho$  corresponding to spherical coordinates of the point.  $\rho$  is given by  $\rho = \sqrt{x^2 + y^2 + z^2}$
- thetas* :  $\theta$  corresponding to spherical coordinates of the point.  $\theta$  is given by  $\theta = \text{atan}(y/x)$ .
- phis* :  $\varphi$  corresponding to spherical coordinates of the point.  $\varphi$  is given by  $\varphi = \text{atan}(\sqrt{x^2 + y^2}/z)$
- nx* : X component of the point normal.
- ny* : Y component of the point normal.
- nz* : Z component of the point normal.

Unknowns or physical properties that can be used in fluid domain scope:

#### Remarks:

*Unknowns variables or physical properties are evaluated by default within functions in general units. Furthermore it is possible to define the units to be used in the function. In these cases, the units must be inserted after the variable name between square brackets.*

#### Examples:

*vx[m/s]/2*

*pr[bar]+1*

*sp1[%]+10*

*It is also possible to obtain the value of the physical properties in a particular node.*

#### Examples:

*vx(#node)*

*pr(#node)*

*vx* : X velocity component at the point (Fluid Flow module).

*vy* : Y velocity component at the point (Fluid Flow module).

*vz* : Z velocity component at the point (Fluid Flow module).

*pr* : Pressure of the point (Fluid Flow module).

*pt* : Thermodynamic pressure (total pressure, including hydrostatic term plus operating pressure). Only available when solving fluid flow (Fluid Flow module).

*vt* : Eddy viscosity at the point (only available if turbulence model is selected) (Fluid Flow module).

*ke* : Eddy kinetic energy at the point (only available if turbulence model, based in the k equation is selected) (Fluid Flow module).

*ep* : Epsilon value at the point (only available if k-epsilon turbulence model is selected) (Fluid Flow module).

*om* : Omega value at the point (only available if k-omega turbulence model is selected) (Fluid Flow module).

*kt* : Ktau ( $k^*\tau$ ) value at the point (only available if k-ktau turbulence model is selected) (Fluid Flow module).

*bt* : Beta (wave elevation) value at the point (Free Surface-Transpiration module).

*dn* : Density of the point.

*vs* : Viscosity of the point (Fluid Flow module).

*kvs* : Kinematic viscosity of the point (Fluid Flow module).

*cm* : Compressibility factor at the point (Fluid Flow module).

*tm* : Temperature of the point (Fluid Flow module).

*cp* : Specific heat of the point (Fluid Flow module).

*kis* : Heat conductivity of the point for fluids (Fluid Flow module).

*sp* : Concentration of the corresponding species at the point (Species Advection module). Number of species must be indicated after sp (i.e. sp1, sp7)

*ph* : Value of the corresponding variable at the point (PDE's solver module). Number of variable must be indicated after ph (i.e. ph1, ph7)

*ds* : Distance of the point to the closest wall (Fluid Flow module).

*mx* : Accumulated mesh deformation of the point in the x direction (Mesh Deformation module).

*my* : Accumulated mesh deformation of the point in the y direction (Mesh Deformation module).

*mz* : Accumulated mesh deformation of the point in the z direction (Mesh Deformation module).

*ols* : Level set field defining free surface (Free Surface-Odd Level Set module).

#### Remarks:

*Variables are evaluated at the previous time step (t-dt).*

*Variables are only available when corresponding module is active.*

Unknown or physical properties functions that can be used in

solid domain scope:

*vx* : X velocity component at the point (Fluid Flow module).  
*vy* : Y velocity component at the point (Fluid Flow module).  
*vz* : Z velocity component at the point (Fluid Flow module).  
*pr* : Pressure of the point (Fluid Flow module).  
*pt* : Thermodynamic pressure (total pressure, including hydrostatic term plus operating pressure). Only available when solving fluid flow (Fluid Flow module).  
*dn* : Fluid density of the point (Fluid Flow module).  
*vs* : Fluid viscosity of the point (Fluid Flow module).  
*kvs* : Fluid kinematic viscosity of the point (Fluid Flow module).  
*sdn* : Solid density of the point.  
*tm* : Temperature of the point (Heat Transfer module).  
*cp* : Specific heat of the point (Heat Transfer module).  
*kxx* : Heat conductivity (xx) of the point (Heat Transfer module).  
*kxy* : Heat conductivity (xy) of the point (Heat Transfer module).  
*kxz* : Heat conductivity (xz) of the point (Heat Transfer module).  
*kyy* : Heat conductivity (yy) of the point (Heat Transfer module).  
*kyz* : Heat conductivity (yz) of the point (Heat Transfer module).  
*kzz* : Heat conductivity (zz) of the point (Heat Transfer module).  
*sp* : Concentration of the corresponding species at the point (Species Advection module). Number of species must be indicated after sp (i.e. sp1, sp7)  
*ph* : Value of the corresponding variable at the point (PDE's solver module). Number of variable must be indicated after ph (i.e. ph1, ph7)  
*ds* : Distance of the point to the closest wall (Fluid Flow module).  
*mx* : Accumulated mesh deformation of the point in the x direction (Mesh Deformation module).  
*my* : Accumulated mesh deformation of the point in the y direction (Mesh Deformation module).  
*mz* : Accumulated mesh deformation of the point in the z direction (Mesh Deformation module).

**Remarks:**

*Variables are evaluated at the previous time step (t-dt).*

*Variables are only available when corresponding module is active.*

**Examples:**

x+y  
0.1\*vx\*vx  
120.0\*(tm-25)  
0.1\*sp2  
sin(x-t)\*log(y^2)

**Constants:**

The constants defined for functions internal compilation are:

*pi* : 3.1415926535897932385

*exp(1)* : 2.7182818284590452354

*dt* : Time increment of the current step.

*op* : Operating pressure.

*infinite* : Infinite.

**Function operators:**

The function operators calculate the value of a standard function at the point defined by the given argument. The function operators that can be used for the definition of the Tdyn CFD+HT functions are:

*sqrt* : the sqrt function calculates the square root of the argument. Syntax: sqrt(.)

*abs* : the abs function calculates the absolute value of the argument. Syntax: abs(.)

*ln* : logarithm of the argument, e base. Syntax: ln(.)

*log* : logarithm of the argument, decimal base. Syntax: log(.)

*fac* : factorial of the argument. Syntax: fac(.)

*sin* : sine of the argument. Syntax: sin(.) (argument given in radians).

*cos* : cosine of the argument. Syntax: cos(.) (argument given in radians).

*tan* : tangent of the argument. Syntax: tan(.) (argument given in radians).

*asin* : The asin function returns the arcsine of the argument in the range -π/2 to π/2 radians. Syntax: asin(.).

*acos* : The acos function returns the arccosine of the argument in the range 0 to π radians. Syntax: acos(.).

*atan* : The atan function returns the arctangent of the argument in the range -π/2 to π/2 radians. Syntax: atan(.) (result given in radians).

*sinh* : hyperbolic sine of the argument. Syntax: sinh(.).

*cosh* : hyperbolic cosine of the argument. Syntax: cosh(.).

*tanh* : hyperbolic tangent of the argument. Syntax: tanh(.).

*exp* : the exp function calculates the exponential value of the argument. Syntax: exp(.).

*heaviside* : the heaviside function evaluates Hs defined as:

$$H_{\varepsilon}(\Phi) = 0\Phi < -\varepsilon$$

$$H_{\varepsilon}(\Phi) = \frac{1}{2} \left( 1 + \frac{\Phi}{\varepsilon} + \frac{1}{\pi} \sin(\pi * \frac{\Phi}{\varepsilon}) \right) |\Phi| < \varepsilon$$

$$H_{\varepsilon}(\Phi) = 1\Phi > \varepsilon$$

The syntax of the function is heaviside(...), where the first argument is and the second .

*Interpolate* : performs a linear interpolation, based on the given data. Two arguments are required: a list of pairs (ξ,η), defining a polylineal curve, and a function (f) defining the point (ξ) where the evaluation is to be done. Syntax: interpolate(#ξ<sub>1</sub>,η<sub>1</sub>,ξ<sub>2</sub>,η<sub>2</sub>,ξ<sub>3</sub>,η<sub>3</sub>,...#f).

*InterpolateSpline* : performs a spline interpolation, based on the given data. Two arguments are required: a list of pairs (ξ,η), defining a the curve, and a function (f) defining the the point (ξ) where the evaluation is to be done. Syntax: interpolatespline(#ξ<sub>1</sub>,η<sub>1</sub>,ξ<sub>2</sub>,η<sub>2</sub>,ξ<sub>3</sub>,η<sub>3</sub>,...#f).

*InterpolateFile* : performs a spline interpolation, based on the data given in a file. Two arguments are required: a file name where a list of pairs (ξ,η), defining a the curve, is given, and a function defining the the point (ξ) where the

evaluation is to be done. Syntax: Interpolatefile(filename, f), where the first argument in the filename, and the second a function (f) defining the value.

*srand* : The rand function returns a pseudorandom integer in the range 0 to 1, based on the argument given as seed. Syntax: srand(.).

*int* : Integer conversos. Syntax: int(.).

- : change sign operator. Syntax: (-expression).

*j0* : Calculates Bessel function of first kind and order 0, at the given point. Syntax: j0(.).

*j1* : Calculates Bessel function of first kind and order 1, at the given point. Syntax: j1(.).

*jn* : Calculates Bessel function of first kind and order n, at the given point. Syntax: jn(.,.), where the first argument is the evaluation point and the second is the order of the Bessel function.

*y0* : Calculates Bessel function of second kind and order 0, at the given point. Syntax: y0(.).

*y1* : Calculates Bessel function of second kind and order 1, at the given point. Syntax: y1(.).

*yn* : Calculates Bessel function of second kind and order n, at the given point. Syntax: yn(.,.), where the first argument is the evaluation point and the second is the order of the Bessel function.

*maxs* : Maximum of the surrounding values. For a given point, maximum of the values of the argument for the connected nodes is returned. Syntax maxs(.), where the argument is any unknown or physical property (see variables section for further information).

*mins* : Minimum of the surrounding values. For a given point, minimum of the values of the argument for the connected nodes is returned. Syntax mins(.), where the argument is any unknown or physical property (see variables section for further information).

*meds* : Average of the maximum and minimum of the surrounding values. For a given point, average of the maximum and minimum values of the argument for the connected nodes is returned. Syntax meds(.), where the argument is any unknown or physical property (see variables information).

*aver* : Average of the surrounding values. For a given point, average of the values of the argument for the connected nodes is returned. Syntax aver(.), where the argument is any unknown or physical property (see variables information). Weighted average is calculated by using standard FEM integration on the connected elements.

*maxvar* : Maximum of the values of the unknown or physical property given as argument (see variables section for further information). Syntax maxvar(.).

*minvar* : Minimum of the values of the unknown or physical property given as argument (see variables section for further information). Syntax minvar(.).

*imat* or *igroup* : Returns 1 if the point belongs to the material given as argument. Note that the material name is actually the name of the group used to assign the material. This function is case insensitive. Syntax imat(.) where the argument is a material's name (if the point belongs to two contiguous materials, it returns 1 in both cases).

*mat* or *group* : Returns 1 if the material given as argument is being used. Note that the material name is actually the name of the group used to assign the material. Syntax mat(.) where the argument is a material's name

*curv* : Curvature of the variable given as argument (see variables section for further information). Syntax curv(.).

*pfx* : X Component of the pressure force of the given Body. Syntax: pfx(.) where the argument is a body's name.

*pfy* : Y Component of the pressure force of the given Body. Syntax: pfy(.) where the argument is a body's name.

*pfz* : Z Component of the pressure force of the given Body. Syntax: pfz(.) where the argument is a body's name.

*vfx* : X Component of the viscous (stress) force of the given Body. Syntax: vfx(.) where the argument is a body's name.

*vfy* : Y Component of the viscous (stress) force of the given Body. Syntax: vfy(.) where the argument is a body's name.

*vfz* : Z Component of the viscous (stress) force of the given Body. Syntax: vfz(.) where the argument is a body's name.

*pmx* : X Component of the pressure moment of the given Body, evaluated at its center of gravity. Syntax: pmx(.) where the argument is a body's name.

*pmy* : Y Component of the pressure moment of the given Body, evaluated at its center of gravity. Syntax: pmy(.) where the argument is a body's name.

*pmz* : Z Component of the pressure moment of the given Body, evaluated at its center of gravity. Syntax: pmz(.) where the argument is a body's name.

*vmx* : X Component of the viscous (stress) moment of the given Body, evaluated at its center of gravity. Syntax: vmx(.) where the argument is a body's name.

*vmy* : Y Component of the viscous (stress) moment of the given Body, evaluated at its center of gravity. Syntax: vmy(.) where the argument is a body's name.

*vmz* : Z Component of the viscous (stress) moment of the given Body, evaluated at its center of gravity. Syntax: vmz(.) where the argument is a body's name.

*dsx* : X Component of the accumulated displacement vector of the given Body. Syntax: dsx(.) where the argument is a body's name.

*dsy* : Y Component of the accumulated displacement vector of the given Body. Syntax: dsy(.) where the argument is a body's name.

*dsz* : Z Component of the accumulated displacement vector of the given Body. Syntax: dsz(.) where the argument is a body's name.

*rtx* : X Component of the accumulated rotation vector of the given Body. Syntax: rtx(.) where the argument is a body's name.

*rtz* : Y Component of the accumulated rotation vector of the given Body. Syntax: rty(.) where the argument is a body's name.

*rtz* : Z Component of the accumulated rotation vector of the given Body. Syntax: rtz(.) where the argument is a body's name.

*vlx* : X Component of the velocity vector of the given Body. Syntax: vlx(.) where the argument is a body's name.

*vly* : Y Component of the velocity vector of the given Body. Syntax: vly(.) where the argument is a body's name.

*vlz* : Z Component of the velocity vector of the given Body. Syntax: vlz(.) where the argument is a body's name.

*vrx* : X Component of the rotational velocity vector of the given Body. Syntax: vrx(.) where the argument is a body's name.

*vry* : Y Component of the rotational velocity vector of the given Body. Syntax: vry(.) where the argument is a body's name.

**vrz** : Z Component of the rotational velocity vector of the given Body. Syntax: vrz(.) where the argument is a body's name.

**xcg** : X Component of the center of gravity of the given Body. Syntax: xcg(.) where the argument is a body's name.

**ycg** : Y Component of the center of gravity of the given Body. Syntax: ycg(.) where the argument is a body's name.

**zcg** : Z Component of the center of gravity of the given Body. Syntax: zcg(.) where the argument is a body's name.

**xrot** : X displacement corresponding to de planar rotation of a point (center of rotation is the point 0,0,0). Note that if the geometry units and function units are different, the returning value must be multiplied by the units conversion factor.

Syntax xrot(.) where the argument is the rotation angle.

**yrot** : Y displacement corresponding to de planar rotation of a point (center of rotation is the point 0,0,0). Note that if the geometry units and function units are different, the returning value must be multiplied by the units conversion factor.

Syntax yrot(.) where the argument is the rotation angle.

**Readfile** : Execute the function in the text file defined by the argument. The file must include a first line defining the maximum time to use the function and a second line containing the function to be executed. If the current time is greater than the one defined in the file, Tdyn CFD+HT waits until the file is updated.

Syntax readfile(.) where the argument is the path and name of the file. Example readfile(C:\Temp\velx.dat). Example of file format:

Time = 0.1;

Function = "interpolate(#0.0,1.1,1.0,2.0#t);";

**Tcl** : Executes a TCL script or procedure returning a double value.

Syntax tcl(.) where the argument is the script to be executed. Example tcl(set var) return the value of tcl variable var.

Note: In order to use this function, TCL extension must be enabled by activating Tcl extension.

**CloudOfDataFile** : performs a local interpolation based on the cloud of points (x,y,z) and data (.) given in a file. The argument is the path and name of the text file. Syntax: CloudOfDataFile(-), where the argument in the filename. Example CloudOfDataFile(C:\Temp\velx.dat). Example of file format:

0.0 0.0 0.0 1.0

0.0 1.0 0.0 0.5

1.0 1.0 1.0 2.0

5.0 2.5 2.0 5.0

**Examples:**

2\*sqrt(y)

x\*fac(5)

srand(0)

log(abs(x))

exp(5)

interpolate(#1.0,2.0,2.0,2.5,3.0,2.0#t^2)

maxvar(vx)

mat('Fluid')

**Operators:**

Operators that can be used for Tdyn CFD+HT functions definitions are:

**+** : adding operator.

Syntax: [adding\_expression] + [adding\_expression].

**-** : subtraction operator.

Syntax: [subtraction\_expression] - [subtraction\_expression].

**^** : exponent operator.

Syntax: [exponent\_expression] ^ [function\_expression].

**\*** : multiplicative operator.

Syntax: [multiplicative\_expression] \* [multiplicative\_expression].

**/** : division operator.

Syntax: [multiplicative\_expression] / [quotient\_expression].

**div** : integer division operator int(x/y+0.5).

Syntax: ([multiplicative\_expression]) div ([quotient\_expression]). Example: (x)div(2+y).

**idiv** : integer division operator int(x/y+0.5). Similar to div operator but with different syntax.

Syntax: idiv ([multiplicative\_expression], [quotient\_expression]). Example: idiv(x,2+y).

**mod** : integer division module operator int(x+0.5)%int(y+0.5).

Syntax: ([multiplicative\_expression]) mod ([quotient\_expression]). Example: (t)mod(2).

**imod** : integer division module operator int(x+0.5)%int(y+0.5). Similar to mod operator but with different syntax.

Syntax: imod ([multiplicative\_expression],[quotient\_expression]). Example: imod(t,2).

**rdiv** : real division operator int(x/y).

Syntax: ([multiplicative\_expression]) rdiv ([quotient\_expression]). Example: (t)rdiv(5).

**ddiv** : real division operator int(x/y). Similar to rdiv operator but with different syntax.

Syntax: ddiv ([multiplicative\_expression], [quotient\_expression]). Example: ddiv(t,5).

**rmod** : real division module operator x/y-int(x/y).

Syntax: ([multiplicative\_expression]) rmod ([quotient\_expression]). Example: (t)rmod(5).

**dmod** : real division module operator x/y-int(x/y). Similar to rmod operator but with different syntax.

Syntax: dmod ([multiplicative\_expression],[quotient\_expression]). Example: dmod(t,5).

**max** : maximum operator.

Syntax: max ([expression], [expression]). Example: max(x,y).

**min** : minimum operator.

Syntax: min ([expression], [expression]). Example: min(x,y).

**not** : not operator.

Syntax: not([function\_expression]).

**~** : not operator.

Syntax: ~([function\_expression]).

*Examples:*

(2\*y)  
(5\*(y+1))/2  
(z\*y)mod(5)  
imod(z\*y) (5)  
(5^4)

*Relational operators:*

The relational (binary) operators compare their first operand with their second operand to test validity of the specified relationship. The result of the relational expression is 1 if the tested relationship is true and 0 if it is false. The binary operators that can be used for Tdyn CFD+HT functions definitions are:

< : less than operator.

Syntax: [expression] < [expression].

< = : less or equal than operator.

Syntax: [expression] <= [expression].

>= : greater or equal than operator.

Syntax: [expression] >= [expression].

> : greater than operator.

Syntax: [expression] > [expression].

= : equal operator.

Syntax: [expression] = [expression].

~= : not equal operator.

Syntax: [expression] != [expression].

& : and operator.

Syntax: [expression] & [expression].

| : and operator.

Syntax: [expression] | [expression].

*Examples:*

(y>2)  
(x<=1)  
(x!=1)  
(y>2)&(x>2)&(x<3)&(y<3)

*Cartesian derivatives:*

It is also possible to use cartesian derivatives to evaluate functions in solid and fluid domains. The Cartesian derivative operators that can be used for Tdyn CFD+HT functions definitions are:

dx : X Cartesian derivative. The argument must be a variable defined in the whole domain (not a function).

dy : Y Cartesian derivative. The argument must be a variable defined in the whole domain (not a function).

dz : Z Cartesian derivative. The argument must be a variable defined in the whole domain (not a function).

grad : Norm of the gradient of a variable. The argument must be a variable defined in the whole domain (not a function).

*Examples:*

vx\*dx(vx)

nx\*dx(vx)+ny\*dy(vx)

grad(vx)

*if-else statement:*

The if statement controls conditional branching. The body of the if statement (elif\_expression) is executed if the value of the expression is non zero. The syntax for the if statement is the following:

if(expression)then(elif\_expression)else(next\_expression)endif

being elif\_expression an additional expression that may include an elif clause with next form:

(expression2)elif(elif\_expression2)then(next\_expression2)

*Examples:*

if(y>2)then(if(x<1)then(1)else(0)endif)else(0)endif  
if(y>2)then(1)elif(x<1)then(2)else(0)endif

*Remarks:*

*Dimensional variables used within the functions defined by the user are evaluated by default using general units. Nevertheless, it is also possible to define the units to be used for each variable within the function at hand. To this aim, the units must be inserted between square brackets after the variable name. The variables that can be defined with units are: t,it,dt,x,y,z and any other vector such as pr, vx, vy, tm, dn, ... It is also possible to define the units of any variable resulting from the application of a given operator over one of the variables described above. If this is the case, the units of the operation result must be indicated as well between brackets after the variable to which the operator under consideration operates.*

*Examples:*

*Units on variables: 2\*t[s], dt[ms], x[cm]+y[m]^2, pr[bar], dn[Kg/m3]*

*Units in operation results: dx(vx[1/s]), dxdy(tm[K/m2])*

Every data field in Tdyn includes several standard units definition, but the user can also enter additional units based on the basic units definition shown in the following table.



Unit	Dimension	Conversion factor to SI	Description
%	Undimensional	0.01	per-one-hundred
‰	Undimensional	0.001	per-one-thousand
pi	Undimensional	3.14159265	pi
e	Undimensional	2.71828183	ln-base
ppmC	Undimensional	1.00E-06	parts-per-million of phi
ppbC	Undimensional	1.00E-09	parts-per-billion of phi
U	Undimensional	1	phi units (user defined)
C	Undimensional	1	spc units (user defined)
m	Length	1	meter
Å	Length	1.00E-10	angstrom
in	Length	0.0254	inch
pica	Length	0.00423333	pica=1/6[in]
ft	Length	0.3048	feet=12[in]
yd	Length	0.9144	yard=3[ft]
mi	Length	1609.344	mile=5280[ft]
leg	Length	4828.032	league=3[mi]
Nmi	Length	1852	nautic-mile
au	Length	1.50E+11	astronomic-unit
s	Time	1	second
min	Time	60	minute
hour	Time	3600	hour=60[min]
day	Time	86400	day=24[hour]
week	Time	604800	week=7[day]
month	Time	2629743.83	month=1/12[year]
year	Time	31556926	year=365.24219879[day]
Kg	Mass	1	Kilogram
g	Mass	0.001	gram
qui	Mass	100	quintal
lb	Mass	0.45359237	libra
ton	Mass	907.18474	tonne=2000[lb]
oz	Mass	0.02834952	ounce=1/16[lb]
amu	Mass	1.66E-27	atomic-mass-unit
rad	Angle	1	radian
deg	Angle	0.01745329	degree=1/180[pi.rad]
grad	Angle	0.01570796	grade=0.9[deg]
turn	Angle	6.28318531	turn=[2.pi.rad]

Unit	Dimension	Conversion factor to SI	Description
Gt	Temperature	1	Grade-thermal
ha	Area	10000	hectare
acre	Area	4046.85642	acre=4840yd <sup>2</sup>
barn	Area	1.00E-28	barn
are	Area	100	are
lt	Volume	0.001	liter
gal	Volume	0.00378541	gallon=231[in3]
brgal	Volume	0.0045461	brt gallon=277.42[in3]
imp	Volume	1.20095	imperial
pt	Volume	0.00189271	pint=0.5[gal]
qt	Volume	0.00094635	quart=0.25[gal]
cc	Volume	1.00E-06	cubic-centimeter
knot	Velocity	0.51444444	knot=[Nmi/hour]
nudo	Velocity	0.51444444	nudo=knot
mach	Velocity	331.46	sound-speed
c	Velocity	3.00E+08	light-speed
G	Acceleration	9.80665	earth-gravity
N	Force	1	Newton=[Kg.m/s <sup>2</sup> ]
dn	Force	1.00E-05	dyne=[g.cm/s <sup>2</sup> ]
Kp	Force	9.80665	Kilopondio=[Kg.G]
lbf	Force	4.448222	lbra-ft=[lb.G]
tonf	Force	8896.44323	tonne-ft=[ton.G]
Pa	Tension	1	Pascal=[N/m <sup>2</sup> ]
mmHg	Tension	133.322	mm-of-Hg
atm	Tension	101325	Atmosphere
bar	Tension	1.00E+05	bar=750.0638[mmHg]
psi	Tension	6894.75789	psi=[lbf/in <sup>2</sup> ]
J	Energy	1	Joule=[N.m]
erg	Energy	1.00E-07	ergio=[dn.cm]
cal	Energy	4.1868	caloria
btu	Energy	1055.06	British-thermal-Unit
W	Potency	1	Watt=[N.m/s]
hp	Potency	745.6999	Horsepower
hz	Frequency	1	[1/s]
ppm	Density	1.00E-06	parts-per-million=[g/l]
ppb	Density	1.00E-09	parts-per-billion=[mg/l]
St	Kinematic Viscosity	1.00E-04	Stokes
P	Viscosity	1.00E-01	Poise

Examples:

W/m (watts per meter), N/m<sup>2</sup> (newtons per square meter), m<sup>3</sup> (cubic meter), g/cc (grams per cubic centimeter).

New units can also be created by adding standard modifiers to the basic unit. These modifiers are shown next:

	Value	Name
a	1.00E-18	ato
f	1.00E-15	femto
p	1.00E-12	pico
n	1.00E-09	nano
u	1.00E-06	micro
m	1.00E-03	mili
c	1.00E-02	centi
d	1.00E-01	deci
D	1.00E+01	Deca
H	1.00E+02	Hecto
K	1.00E+03	Kilo
M	1.00E+06	Mega
G	1.00E+09	Giga
T	1.00E+12	Tera
E	1.00E+15	Hexa
P	1.00E+18	Peta

## 10. Executing Tdyn Solver

### 10.1. Automatically executing tdyn

Tdyn CFD+HT solver can be comfortably started through Tdyn pre-processing environment **Calculate** menu. Once the analysed problem is defined (i.e. the geometry is created) the boundary conditions assigned, and the mesh is generated, the **Start** button in the **Calculate** menu (or the Calculate icon) can be pressed.

When the **Start** button is pressed, the system writes the input file for the calculation module (the 'calculation file') called `ProblemName.flavia` into the `ProblemName.gid` directory, and then the `compassfem.tdyn?d.win.bat` file from the Tdyn problemtype directory is executed, with the following arguments:

- Argument 1: Problem name
- Argument 2: Problem directory
- Argument 3: Problemtype directory

From within the `compassfem.tdyn?d.win.bat` batch file, the program executable `tdyn.exe` is called, with the name of the input file given as an argument. Now `tdyn.exe` is started and creates a number of output files (see section [Output files](#) for a brief description of the files generated during the execution).

### 10.2. Manually executing tdyn

Sometimes it can be interesting to run the Tdyn executable manually (without using the graphic user interface of the software). The necessary steps are described here.

From here in advance, the following notation will be employed for description purposes:

`$gidpath` : root directory of the installed program. It contains, among others, the `gid.exe` executable file called to run the GiD custom GUI.

`$CompassFEM_version` : CompassFEM problemtype version name.

`$inputpath` : directory that contains the tdyn input data file.

`$modelname` : name of the input data file.

First, the input data file required by the Tdyn executable must be generated before execution. To this aim, the corresponding model must be loaded to the GiD custom interface.

Next, the input data file must be exported, assuming that the model setup has been finished successfully (applying material properties and boundary conditions) and that the mesh has been already generated. In order to export the input file, the following menu sequence must be used:

**Files ► Export ► Calculation file...**

By doing this, the user will be asked for a file name (`$modelname`) and location (`$inputpath`). By default, `.dat` extension will be used to export de input file. If desired, `.flavia` extension can be also specified for instance, trying to mimic the file name convention used when running Tdyn automatically.

Before execution, you must ensure that the tdyn process is able to find a `password.txt` file containing a valid tdyn password. You can create such a text file manually and copying the password inside. Alternatively, the `password.txt` file can be copied from the directory `$gidpath\problemtypes\CompassFEM_version\compassfem.gid`, where it is automatically saved if tdyn

passwords have been previously registered through the GiD custom GUI. For manual execution of tdyn, the password file must be located either next to the `tdyn.exe` executable (this is on the previously mentioned `$gidpath\problemtypes\CompassFEM_version\compassfem.gid\exec` directory) or next to the input file.

After exporting the input file and copying a valid `password.txt` file, everything is ready to launch tdyn manually. To this aim, open a command shell and move to the location of the `tdyn.exe` executable. Such a location will be typically of the form:

`$gidpath\problemtypes\CompassFEM_version\compassfem.gid\exec`

Note that `tdyn.exe` may be also executed from an arbitrary location if the directory path above is conveniently added to the environment system variable `PATH`.

Finally, launch tdyn by using the following command line:

`\> tdyn.exe -name "$inputpath\%modelname"`

Optionally, the argument `-2D` or `-3D` can be passed to `tdyn.exe` indicating the spatial dimension of the problem under analysis. In any case, Tdyn will check against the input file the validity of the specified dimension. If the given argument is not consistent with the input file information, Tdyn process will end with an error.

### 10.3. Output files generated during process execution

`ProblemName.flavia.inf` : Text file containing global information as well as process information for each time step. The content of this file can also be accessed during calculation through the GUI by using the menu option *Calculate > View process info*.

`ProblemName.flavia.out` : Text file containing detailed information of the calculation for each time step.

`ProblemName.flavia.tim` : Text file that contains a timetable giving information on the CPU time consumption of the process. The timetable contains a report of the execution time used by different parts of the problem. This file is only available after the successful calculation of a problem.

`ProblemName.flavia.err` : Text file containing error message (file created only if Tdyn exits with an error).

`ProblemName.flavia.for` : Text file that contains time evolution of the various components of forces and moments calculated over each body. This forces and moments file contains the forces and moments that act on the surfaces with **Fluid Body** properties assigned. Forces are often used to evaluate the convergence of a simulation. The file can be opened from within *Custom GiD* interface, using the options in the *Utilities* menu or in a spreadsheet program (like MS Excel) or another plotting utility (like Gnuplot). The first few time steps should be excluded from the graph, because the forces can oscillate at very high values due to the start up process, which can change the order of magnitude of the graph scales.

`ProblemName.flavia.mov` : Text file that contains time evolution data of body movements.

`ProblemName.flavia.nor` : Text file that contains time evolution information of the convergence norms associated to each problem variable.

`ProblemName.flavia.res` : Results file that contains all field valued results. When pressing **Postprocess** in *Custom GiD*, this file is loaded, and the results it contains can be visualized in the

post-processing module. Also note that each calculation will delete a previous results file that might exist in this directory, unless it has been renamed before the new calculation process has been started.

ProblemName.flavia.rst : Restart file that contains the necessary results to be used when restarting a previous calculation. The restart file contains the velocities, pressures, and other relevant data of the last time-step that has been written into the restart file. If Tdyn is started with the option *Restart: On* (that have to be set in the **PROBLEM** window) the file ProblemName.flavia.rst is read and the calculation will be restarted using the last time-step data as initial condition. This file is updated each time-step that the results are written into the results file (ProblemName.flavia.res). Thus even a process that has not yet terminated can be killed and restarted with the option *Restart: On*, without having to recalculate all the time steps from the beginning. By killing the process, only the time steps from the last step that has been written into the results file onwards will be 'lost' and will have to be recalculated in case of a restart. Note that the restart file will be written in any case - even in the case that a calculation is started with the *Restart: Off* option. Also note that each time the restart file is updated, as well as with each new calculation, a restart file that might already exist in the problem directory will be deleted unless it has previously been renamed.

ProblemName.flavia.ram.res : Forces results for structural analysis using Ram-Series.

ProblemName.flavia.sat : Sink & Trim data.

ProblemName.flavia.ram.msh : Mesh for structural analysis using Ram-Series.

## 11. References

1. E. Oñate and J. García-Espinosa. Finite Element Analysis of Incompressible Flows with Free Surface Waves using a Finite Calculus Formulation. ECCOMAS 2001. Swansea UK 2001.
2. E. Oñate, and J. García-Espinosa, Advanced Finite Element Methods for Fluid Dynamic Analysis of Ships, MARNET-CFD Workshop, Crete, Athens, 2001.
3. E. Oñate, and J. García-Espinosa, A Finite Element Method for Fluid-Structure Interaction with Surface Waves Using a Finite Calculus Formulation, Comput. Methods Appl. Mech. Engrg. 191 (2001) 635-660.
4. E. Oñate and J. García-Espinosa. A methodology for analysis of fluid-structure interaction accounting for free surface waves. European Conference on Computational Mechanics (ECCM99). Munich, Germany, September 1999.
5. O. Soto, R. Löhner, J. Cebal and R. Codina. A Time Accurate Implicit-Monolithic Finite Element Scheme for Incompressible Flow. ECCOMAS 2001. Swansea UK 2001.
6. W. Rodi. Turbulence models and their application in hydraulics - a state of the art review. Institut für Hydromechanik and Sonderforschungsbereich, University of Karlsruhe, Germany, 1980.
7. J. Smagorinsky. General circulation model of the atmosphere. Mon. Weather Rev., 91:99-164, 1963.
8. V.C.Patel, W.Rodi & G.Scheurer, Turbulence models for near-wall and low-Reynolds-number flows: A review, AIAA J, Vol.23, No.9, p1308, (1984)
9. B. Launder and B. Sharma. Application of the Energy Dissipation model of Turbulence to the Calculation of Flow near a Spinning Disk. Letters in Heat and Mass Transfer, vol. 1, no. 2, 1974, pp. 131-138.
10. D. Wilcox. Turbulence Modelling for CFD. DCW Industries, Inc., 5354 Palm Drive, La Cañada, California, 1993.
11. J. Bardina, P. Huang and T. Coakley. Turbulence Modelling Validation, Testing, and Development. NASA Technical Memorandum 110446, April 1997.
12. ERCOFTAC (European Research Community On Flow, Turbulence And Combustion), Best Practice Guidelines for Industrial Computational Fluid Dynamics, United Kingdom, January 2000.
13. International Center for Numerical Methods in Engineering (CIMNE) and COMPASS Ingeniería y Sistemas. GiD The personal pre and postprocessing system. Documentation available at <http://gid.cimne.upc.es> or <http://www.compassis.com>.
14. E. Oñate, A. Valls, J. García. FIC/FEM formulation stabilizing terms for incompressible flows at low and high Reynolds numbers. Comput. Mech. 38:440-455, 2006.
15. J. García, A. Valls, E. Oñate. ODDLS: A new unstructured mesh finite element method for the analysis of free surface flow problems. Int. J. Numer. Meth. Engng. Published online, 2008.
16. R.B.Bird. Useful non-Newtonian models- Annu. Rev. Fluid MEch. 8 (1976)13.
17. P.J.Carreau. Rheological equations from molecular network theories. Trans. Soc. Rheol. 16 (1972)99.
18. M.M.Cross. Rheology of non-Newtonian fluids: a new flow equation for pseudoplastic systems. J.ColloidSci. 20 (1965)417