R-10-72

Darcy Tools version 3.4

User's Guide

Urban Svensson, Computer-aided Fluid Engineering AB

Michel Ferry, MFRDC, Orvault, France

December 2010

Svensk Kärnbränslehantering AB
Swedish Nuclear Fuel
and Waste Management Co

Box 250, SE-101 24 Stockholm
Phone +46 8 459 84 00

ISSN 1402-3091
SKB R-10-72

Darcy Tools version 3.4

User's Guide

Urban Svensson, Computer-aided Fluid Engineering AB

Michel Ferry, MFRDC, Orvault, France

December 2010

Preface

The User's Guide for DarcyTools V3.4 is intended to assist new and experienced users of DarcyTools. The Guide is far from complete and it has not been the ambition to write a manual that answers all questions a user may have.

The objectives of the Guide can be stated as follows:

– Give an overview of the code structure and how DarcyTools is used.

– Get familiar with the "Compact Input File", which is the main way to specify input data.

– Get familiar with the "Fortran Input File", which is the more advanced way to specify input data.

Contents

1	**Introduction**	7
1.1	Background	7
1.2	Objective	7
1.3	Intended use and user	7
1.4	Outline	8
2	**What DarcyTools does**	9
2.1	Situation in mind	9
2.2	Key features	11
3	**How DarcyTools operates**	13
3.1	Introduction	13
3.2	General structure	13
3.3	Mathematical Model	15
3.4	Finite volume equations and solver	17
3.5	Grid	17
3.6	Hardware, OS, etc	19
3.7	Directories, files and scripts	19
4	**Compact Input File (CIF)**	21
4.1	Introduction	21
4.2	Overview	23
4.3	Commands	25
5	**File Formats**	79
6	**Property Generation (PROPGEN)**	89
7	**Fortran Input File (FIF)**	93
7.1	Why needed?	93
7.2	How to use	93
8	**A Generic Äspö Model**	97
8.1	Introduction	97
8.2	The situation studied	97
8.3	The set-up	99
9	**References**	115
10	**Notation**	117
Appendix A	User Subroutines	119
Appendix B	Structure of SOLVE	133
Appendix C	Four CIF-examples	143
Appendix D	CIF commands and arguments	159

1 Introduction

1.1 Background

DarcyTools is a computer code for simulation of flow and transport in porous and/or fractured media. The fractured media in mind is a fractured rock and the porous media the soil cover on the top of the rock; it is hence groundwater flows, which is the class of flows in mind.

DarcyTools is developed by a collaborative effort by SKB (The Swedish Nuclear Fuel and Waste Management Company), MFRDC (Michel Ferry R& D Consulting) and CFE AB (Computer-aided Fluid Engineering AB). It builds upon earlier development of groundwater models, carried out by CFE during the last twenty years. In the earlier work the CFD code PHOENICS (Spalding 1981) was used as an equation solver. DarcyTools is based on a solver called MIGAL (Ferry 2002). It has however been carefully evaluated that the two solvers produce very similar solutions and the earlier work is thus still valid as a background for DarcyTools.

The present report will focus on the software that constitutes DarcyTools. Two accompanying reports cover other aspects:

– Concepts, Methods and Equations. (Svensson et al. 2010) (Hereafter denoted Report 1).

– Verification, Validation and Demonstration (Svensson 2010) (Hereafter denoted Report 2).

Two basic approaches in groundwater modelling can be identified; in one we define grid cell conductivities (sometimes called the continuum porous-medium (CPM) approach, i.e. Jackson et al. 2000), in the other we calculate the flow through the fracture network directly (DFN approach). Both approaches have their merits and drawbacks, which however will not be discussed here (for a discussion, see Sahimi 1995). In DarcyTools the two approaches are combined, meaning that we first generate a fracture network and then represent the network as grid cell properties. Further background information is given in the two reports mentioned.

1.2 Objective

The main objective of this User's Guide is to introduce and explain all parts of DarcyTools that a user needs to be concerned with.

1.3 Intended use and user

Commercially available computer codes often have a well structured way to specify input data. This may make the code easy to use and control. However, normally it also introduces a limit in the generality (in terms of applications) of the code. It may for example not be possible to add new source and sink terms, as formulated by the user. For a commercial code it may be necessary to limit the access to the source code, as "well meant improvements" by users will make user support impossible.

DarcyTools is not a commercial code and it is intended for a small group of users. The user will therefore have access to the source code needed for more "advanced" modifications. The user accessible parts are however well separated from the main codes and are also "as small as possible" (perhaps 1–2% of the total code); this to simplify the task for the user. For most problems it will however suffice to specify the input by way of data statements (details later).

The intended user is thus assumed to have a general knowledge about advanced computer codes and to be able to take the right decisions about possible additions to the code.

1.4 Outline

After a few introductory chapters (Chapter 1 to 3), the Compact Input File (CIF) (Chapter 4) is described. This is the main mode of input specification. If the CIF commands do not suffice (for example if complex transient boundary conditions need to be specified) a Fortran input file (FIF) (Chapter 7) is used. The FIF gives a user significantly more control, but it also requires a deeper understanding of the code and, of course, also the skill of programming in Fortran. Similarly, if the advanced option for fracture network representation is used, a Fortran file called PROPGEN (for property generation) (Chapter 6) needs to be considered. After these sections a generic application case is discussed. A deeper understanding of the code structure can be obtained by studying the Appendices. File formats are given in Chapter 5 and a summary of notations is found in Chapter 10.

2 What DarcyTools does

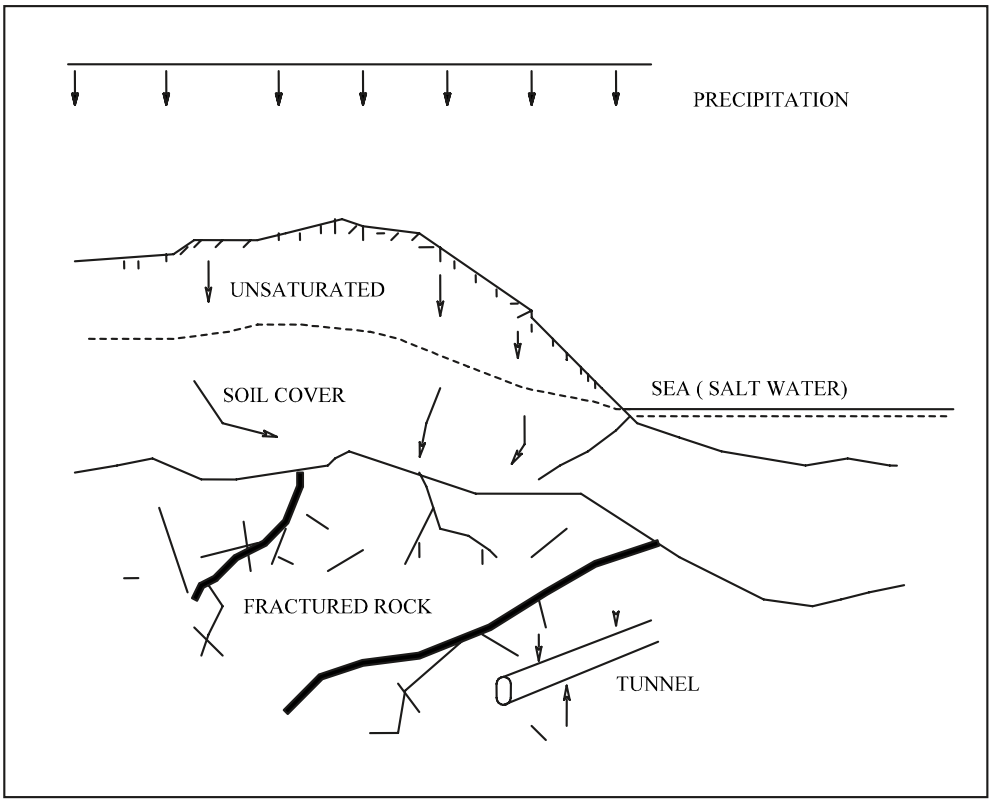
2.1 Situation in mind

As the name indicates DarcyTools is based on the concept of Darcian flow, i.e. the general momentum equations can be reduced to the Darcy equation. This class of flow includes flow in porous media and fractured rock, see Figure 2-1. In order to give a brief description of the physical processes involved, let us assume that it is of interest to analyse the origin of the water that leaks into the tunnel (Figure 2-1, top). Two possible sources are precipitation and sea water. To track the precipitation water, the simulation model must include descriptions of flow in the unsaturated zone, flow in the porous media and in the fractured rock. The groundwater table determines, largely, the pressure gradients in the porous media (and may influence conditions deep into the rock) and it is thus essential to calculate the groundwater table correctly. Sea water may be saline which introduces density effects, which in turn affect the pressure distribution. To handle this situation one needs to solve the coupled pressure-salinity problem and also consider dispersion processes. The complexity is hence greatly increased, when density effects come into play.

If a non uniform temperature distribution is present this will also affect the density. DarcyTools can handle also the temperature-salinity-pressure coupling.

If transport of a substance (salt, radionuclide, etc) is to be analyzed, it is necessary to consider processes on the mm scale, see Figure 2-1 (bottom). A number of processes, on a range of time and space scales, contribute to the dispersion of a substance as it travels with the flow. The processes on the mm scale are often very significant and DarcyTools includes a novel subgrid model, called FRAME, to deal with these processes.

Hopefully this brief introduction has indicated the scope of DarcyTools. For a complete account, see Report 1.



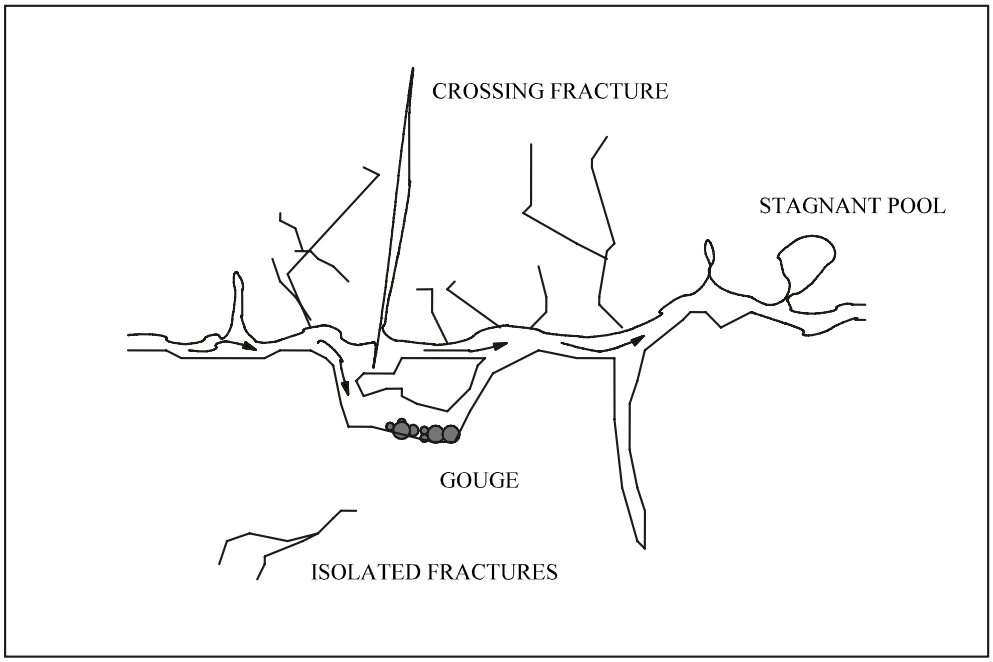


Figure 2-1. *Illustration of processes that can be analyzed with DarcyTools. The large scale view (top) and the mm scale view.*

2.2 Key features

Another type of characterization of the scope and content of DarcyTools can be obtained by listing the key features of the code:

- **Mathematical model.** DarcyTools is based on conservation laws (mass, heat, momentum and mass fractions) and state laws (density, porosity, etc.). The subgrid model utilizes the multi-rate diffusion concept and the fracture network (resolved and subgrid) is based on fractal scaling laws.
- **Unstructured grid.** DarcyTools V3.4 is based on a Cartesian unstructured grid, which offers great flexibility in description of geometries.
- **Continuum model.** Even if a fracture network forms the basis of the approach, DarcyTools should be classified as a continuum porous-medium (CPM) model.
- **Fractures and fracture network.** Fractures and fracture zones are idealized as conductive elements, to which properties (conductivity, porosity, flow wetted surface, etc.) are ascribed. Empirical laws are used for the determination of these properties. The fracture network is based on fractal scaling laws and statistical distributions (random in space, Fisher distribution for orientation, etc).
- **GEHYCO.** This is the algorithm, based on the intersecting volume concept (see Report 1), that transforms the fracture network (with properties of conductive elements) to grid cell properties.
- **FRAME.** Subgrid processes are parameterized as “diffusive exchange with immobile zones”. FRAME uses the multi-rate diffusion model and fractal scaling laws, to formulate a simple and effective subgrid model.
- **SOLVE.** When the continuum model is generated, effective CFD-methods are used to solve the resulting finite-volume equations. DarcyTools uses the MIGAL-solver, which is a multigrid solver with the capability to solve coupled problems (like pressure and salinity) in a fully coupled way. From version 3.4 DarcyTools is parallelized based on a SPMD (Single Program Multiple Data) approach.
- **PARTRACK.** This particle tracking algorithm is fully integrated with DarcyTools and uses the same basic concepts as FRAME. PARTRACK can handle Taylor dispersion, sorption and matrix diffusion simultaniously in large 3D grids (> 10^6 cells).
- **Verification and Validation.** A set of verification and validation studies is presented (see Report 2). This is considered to be an essential feature of the code.

A full description of features is given in Report 1.

3 How DarcyTools operates

3.1 Introduction

It was mentioned in Section 1.1 that the approach adopted in DarcyTools is “to first generate a fracture network and then represent the network as grid cell properties in the continuum model”. The adopted approach also governs the general structure and working of DarcyTools. The first step is to generate a grid and a fracture network. When grid cell properties have been generated the continuum problem is solved, as the second step. The final step is the postprocessing of the output.

3.2 General structure

DarcyTools is not one computer code, but a suite of “main codes” and a number of auxiliary ones. Figure 3-1 gives the relationship between the various components of DarcyTools. The figure also indicates the typical work process, as one normally “works from left to right in the diagram”. A first introduction to the various programs will now be given.

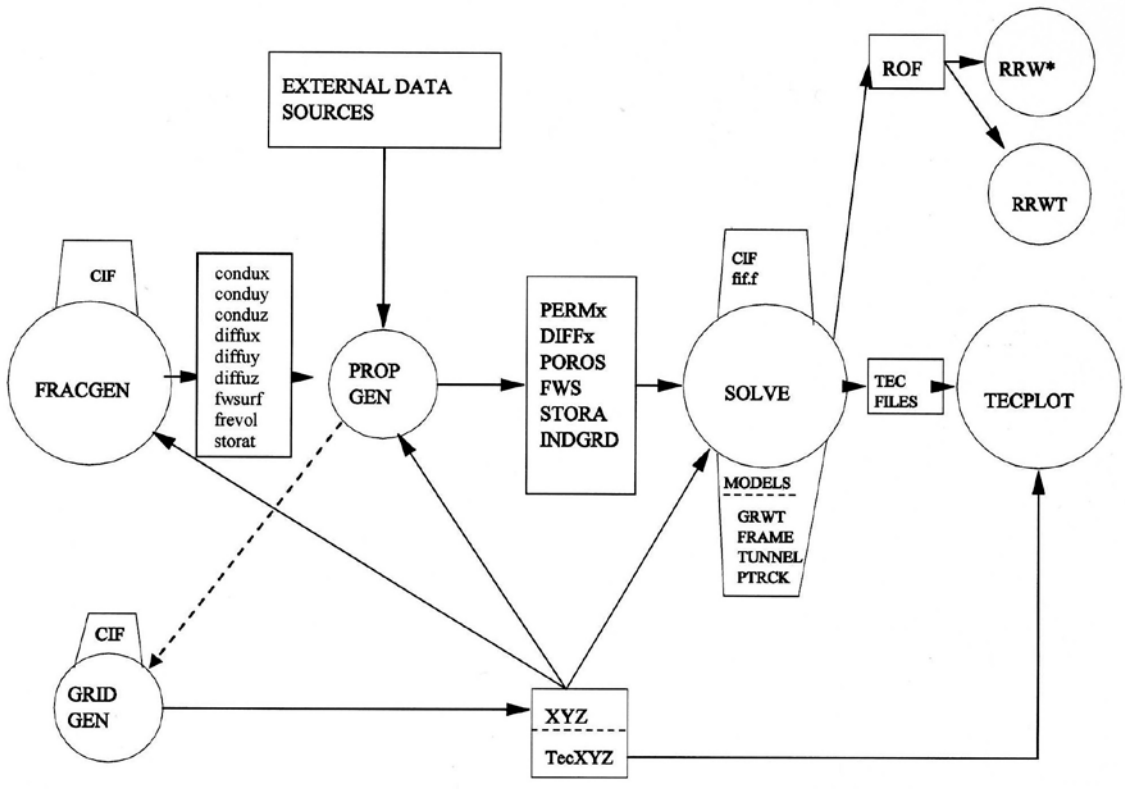


Figure 3-1. *How DarcyTools operates. Circles indicate a program, while squares indicate data files.*

GRIDGEN

As the name indicates this is the code that generates the grid coordinates. The default output file is called XYZ.

FRACGEN

This is the set of programs that generates the grid cell properties, i.e. conductivity, flow wetted surface, diffusivity and porosity fields. First deterministic (can be inhomogeneous) deformation zones and stochastic DFN:s are generated and, in a second step, their properties are translated into grid cell properties.

PROPGEN

This small auxiliary code is used to modify the properties generated by FRACGEN. This is useful if, as an example, the conductivity of the soil cover is to be established in a calibration process. The execution of FRACGEN may take hours in complex simulations, while PROPGEN will perform its task in a minute or two.

SOLVE

This is the code that solves the differential equations for pressure, salinity and other scalars. SOLVE hence contains subroutines for the formulation of the finite volume equations, boundary and source terms, etc. A key element in SOLVE is the solver of the linear equation system, MIGAL.

RRW and RRWT*

The Result Output File is called ROF. In ROF all data necessary for a restart are stored. Also intermediate data, for example a time series, can be stored, if so specified by the user. It is anticipated that a non-DarcyTools user may like to explore the simulation results stored in ROF, maybe using a postprocessor like Tecplot. The small program RRW* (Read ROF Write for some postprocessor) is intended to fulfil this need. RRWT is a small program that writes output files that can be viewed in Tecplot.

POSTPROC, Tecplot

As described above, (RRW*), DarcyTools prepares for the user to use the postprocessor he/she may prefer. However, Tecplot has been selected as the “standard” postprocessor for DarcyTools. From the input files a user can specify the content of Tecplot files, which are then generated by DarcyTools.

Cell removal and refinement

From version 3.3 it is possible to restart GRIDGEN and in the process refine the grid or remove cells. A file is generated in PROPGEN which is read by GRIDGEN. The dashed line in Figure 3-1 indicates this possibility.

3.3 Mathematical Model

DarcyTools represents fracture network flows using a continuum model in which the mass conservation equation is associated to several mass fraction transport equations for the salinity and/or particle mass concentrations, and to a heat transport equation.

Conservation of mass:

$$\frac{\partial \rho\theta}{\partial t} + \frac{\partial}{\partial x}(\rho u) + \frac{\partial}{\partial y}(\rho v) + \frac{\partial}{\partial z}(\rho w) = Q \tag{3-1}$$

where ρ is fluid density, θ porosity, u, v and w Darcy velocities and Q a source/sink term. The coordinate system is denoted x, y, z (space) and t (time).

Mass fraction transport equation:

$$\begin{aligned}\frac{\partial \rho\theta C}{\partial t} &+ \frac{\partial}{\partial x}\left(\rho u C - \rho\gamma D_x \frac{\partial C}{\partial x}\right) \\ &+ \frac{\partial}{\partial y}\left(\rho v C - \rho\gamma D_y \frac{\partial C}{\partial y}\right) \\ &+ \frac{\partial}{\partial z}\left(\rho w C - \rho\gamma D_z \frac{\partial C}{\partial z}\right) = QC + Q_C\end{aligned} \tag{3-2}$$

where C is transported mass fraction, D_x, D_y and D_z the normal terms of the diffusion-dispersion tensor and Q a source/sink term. The source term Q_c represents the exchange with immobile zones and is determined by the subgrid model FRAME. Note that the diffusion coefficients are the effective coefficients that include the porosity, see further explanation in connection with Equation (3-10) below.

Conservation of heat:

$$\begin{aligned}\frac{\partial\left(\rho\theta c_p + (1-\theta)c\right)T}{\partial t} &+ \frac{\partial}{\partial x}\left(\rho u c_p T - \lambda_x \frac{\partial T}{\partial x}\right) \\ &+ \frac{\partial}{\partial y}\left(\rho v c_p T - \lambda_y \frac{\partial T}{\partial y}\right) \\ &+ \frac{\partial}{\partial z}\left(\rho w c_p T - \lambda_z \frac{\partial T}{\partial z}\right) = Q c_p T + Q_T\end{aligned} \tag{3-3}$$

where λ_x, λ_y and λ_z are the normal terms of the equivalent (i.e. rock with fluid) thermal conductivity tensor, c is the rock thermal capacity and c_p the specific heat of the fluid and Q_T a source/sink term. We are hence only solving for one temperature, assuming thermal equilibrium between the rock and the water. Note that we have chosen to use c (Joule/m^3 °C) for the rock thermal capacity and c_p (Joule/kg °C) the specific heat of the fluid. The reason for this inconsistency is that this formulation does not require the rock density as an input variable.

The mass conservation equation is turned into a pressure equation under the well known Darcy's assumption:

$$\rho u = -\frac{K_x}{g}\frac{\partial P}{\partial x}$$

$$\rho v = -\frac{K_y}{g}\frac{\partial P}{\partial y} \tag{3-4}$$

$$\rho w = -\frac{K_z}{g}\frac{\partial P}{\partial z} - K_z(\rho - \rho_0)$$

where K_x, K_y and K_z are the local hydraulic conductivities in x, y and z direction, g the gravity acceleration, ρ_0 a reference fluid density and P the dynamic fluid pressure relative to the reference hydrostatic pressure.

$$P = p + \rho_0 g\, z \tag{3-5}$$

where p is the total pressure. The hydraulic conductivity, K, is related to the permeability, k, by the relation,

$$K = \frac{\rho g k}{\mu} \tag{3-6}$$

where μ is dynamic viscosity.

The fluid properties like the dynamic viscosity, μ, the density, ρ, and the specific heat, c_p, are given by state laws:

$$\mu = \mu_0\left[\,1 + a_1\left(T - T_\mu\right) + a_2\left(T - T_\mu\right)^2\,\right]^{n_\mu} \tag{3-7}$$

$$\rho = \rho_0\left[\,1 + \alpha_1 S + \alpha_2 S^2 - \beta_1\left(T - T_\rho\right) - \beta_2\left(T - T_\rho\right)^2\,\right] \tag{3-8}$$

$$c_p = c_{p0}\left(1 + b_1\, S + b_2\, S^2\right) \tag{3-9}$$

while the porosity, θ, and the compaction, γ, of the matrix are given the following dependencies:

$$\theta = \theta_0\, \gamma \tag{3-10}$$

$$\gamma = 1 + (\sigma/\theta_0)\,(P - P_0)/\rho g \tag{3-11}$$

In the above formulas S represents the salinity (salt mass fraction), θ_0 a reference porosity field given for a reference pressure field P_0 and σ the specific storativity field. n_μ, a_i, b_i, α_i, β_i, μ_0, ρ_0, c_{p0}, T_μ and T_ρ are constants.

In the advection/diffusion equation (3-2), it is common to write the diffusion coefficient as θD_{mol}, where D_{mol} is the molecular diffusion coefficient. In DarcyTools we choose to write the term as $\theta D_{mol} = \theta_0 \gamma D_{mol} = \gamma D$, where D is now the effective diffusion coefficient. The reason is that it is the effective diffusion coefficient that is specified for a conductive element and the GEHYCO-algorithm will hence deliver effective diffusion coefficients for cell walls. When a porous media case is simulated and the diffusion coefficients are specified, one thus needs to remember that it is the effective coefficients that should be given.

3.4 Finite volume equations and solver

CFD (Computational Fluid Dynamics) methods transform the differential equations into algebraic ones, which can be solved by a computer and a computer program. DarcyTools uses the so called finite volume method, which can be thought of as having three well-defined stages:

1) Discretize the computational domain into a number of cells, which fill entirely the domain.
2) Integrate each differential equation for each cell, to yield an algebraic equation.
3) Solve the resulting set of algebraic equations.

The differential equations were given in the previous section. After the integration, step 2 above, an algebraic equation of the following type results:

$$a_P\Phi_P = a_W\Phi_W + a_E\Phi_E + a_S\Phi_S + a_N\Phi_N + a_B\Phi_B + a_T\Phi_T + S_\phi \qquad (3\text{-}12)$$

where Φ denotes the variable in question, a coefficients and S_ϕ source terms. For further details see Report 1.

It is equations of type (3-12) that are solved by the solver MIGAL (see Report 1, Appendix A); in fact MIGAL can solve linked systems of this kind of equations, a feature that is used for the pressure-salinity coupling in the present set of equations.

From version 3.4 DarcyTools can utilize computers with multi-core processors, using the SPMD parallelization approach.

3.5 Grid

DarcyTools V3.4 is based on an unstructured Cartesian grid system, which allows great flexibility in local grid refinement. Another way to describe this grid system is to say that the grid attempts to resolve geometrical objects, which can be read in as CAD-files, with a user-controlled precision.

An illustration of the grid is given by Figure 3-2, which shows a vertical section and includes the following features: a surface topography, a rock surface, fractures zones and a tunnel. Some further details are given by the following points:

- The topography line is resolved by a fine grid and all cells above this line are removed.
- The rock surface is resolved by a somewhat coarser grid and all cells up to the topography line has the same size.
- Below the rock surface the cell size expands up to a specified limit.
- Deformation zones are resolved with a higher resolution.
- The tunnel is specified as a circular area for which we specify a certain resolution inside and on the border. In the near-field a higher resolution is found in a rectangular area.
- The white area above the tunnel represents an area where cells have been removed.

These points introduce some of the features of the grid system. One can, for example, specify that an object should have a certain resolution on the border, inside or outside of the object. It is further possible to remove cells; in the example all cells above the topography were removed but also some internal cells were removed, based on some criterion.

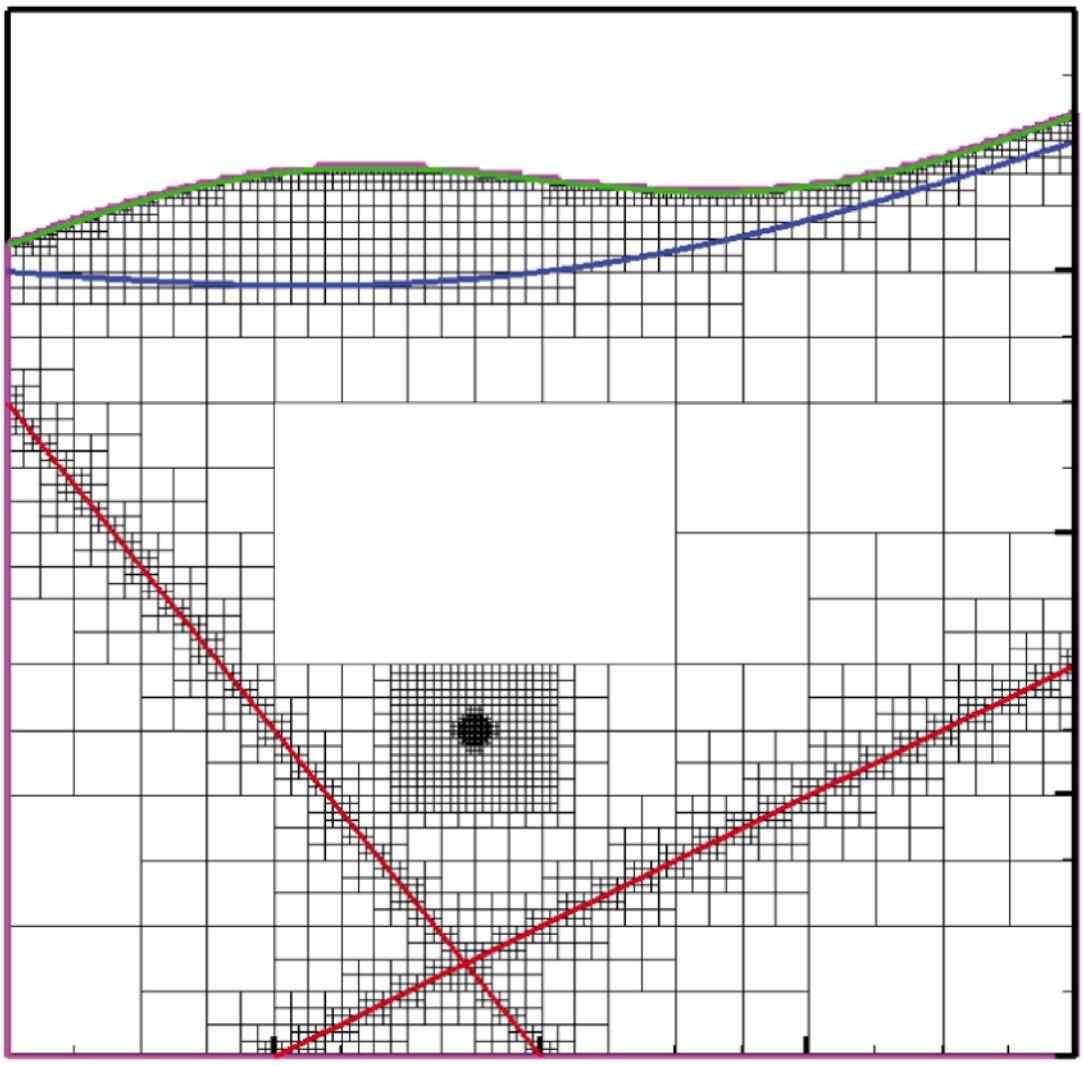


Figure 3-2. *The unstructured Cartesian grid. Green line represents topography, blue rock surface and red deformation zones.*

3.6 Hardware, OS, etc

DarcyTools is written in Fortran95. The following points give the currently favoured working environment:

- A PC/Workstation with as much power (processor, ram, disk space, graphics, etc) as possible. Both 32 and 64 bit computers are supported.
- Red Hat LINUX (enterprise) or Windows 7, as operating system.
- Intel Fortran Compiler.
- Intel MPI run time kit.
- Tecplot for postprocessing.

DarcyTools requires very little disk space (about 20MB), and calculations can be performed on a laptop. The concepts built into DarcyTools assume however a high resolution grid (more grid cells $\Rightarrow$ more accurate simulation) and real world application simulations can hence be quite demanding for the computer.

3.7 Directories, files and scripts

This section describes how to run DarcyTools and the recommended directory structure. It is assumed that the user has created this directory structure by running the installation on the delivery CD.

The reader is referred to the installation and configuration document for details about the directory structure. Presently, a directory “darcytools” is generated, which contains four subdirectories.

bin: contains runscripts and libraries. Examples:

RUNDTS xx	for running	SOLVE under Linux, where xx denotes the number of blocks in a parallel execution. For a sequential run xx is omitted. For a Windows platform a question about the number of blocks will appear.
RUNGGN	for running	GRIDGEN
RUNFGN	for running	FRACGEN
RUNOGN	for running	object generation from STL-files
RUNBGN	for running	Block Generation for Parallel execution.

project: contains user files. All files needed to start a new application, or project, are stored in this directory.

VerCase: contains verification cases; see Report 2 for a full description.

DemCase: contains demonstration cases; see Report 2 for a full description.

4 Compact Input File (CIF)

4.1 Introduction

DarcyTools implements a special command format for the Compact Input Files. This format is compatible with the XML format and aims to be flexible and explicit.

CIF files are now named with the xml extension (cif.xml). They start with the tag <cif> and finish with the tag </cif>. The comments and commands are inserted between these two tags.

Comments may contain several lines and always start with the four characters tag <!-- and end with the three characters tag -->.

Commands are made of a starting tag e.g. <xxx> followed by its corresponding ending tag </xxx>, where xxx is the command name. The command arguments are inserted between these two tags.

Arguments are made of a starting tag e.g. <yyy> followed by its corresponding ending tag </yyy>, where yyy is the argument name. The value of the argument is inserted between these two tags.

When presenting commands and arguments (section 4.3) the ending tags are omitted.

The xml extension of the CIF files facilitates the use of web browsers or xml editors for syntax checking. Note that the CIF format is case sensitive and that commands may be specified in any order. Most of the command arguments are optional. The notation below is adopted for the command description. As an example, optional arguments are denoted [1-] or [0+].

```
*
*  [1-] 0 or 1
*  [1=] 1 exactly (required argument)
*  [0+] 0 or more
*  [1+] 1 or more (at least one such argument is required)
*
```

Although the CIF commands can be given in any order, it is recommended that the structure below is adopted. The detailed command description, in section 4.3, follows this structure as well as the verification and demonstration cases.

```
<cif>

<!--============================================================
============== MAIN                            ================
==============                                  ================
============== Case:                            ================
============== Date:                            ================
============== Author :                         ================
============================================================-->

<!-- Physical models -->
<!-- *************** -->

<!-- Variables and initial cond. -->
<!-- *************************** -->

<!-- Boundary cond., Sources & Sinks -->
<!-- ****************************** -->

<!-- Special models -->
<!-- ************** -->

<!-- Equations -->
<!-- ********* -->

<!-- Solver -->
<!-- ****** -->

<!--============================================================
=============  INPUT-OUTPUT  ===================================
============================================================-->

<!-- Run % store -->
<!-- *********** -->

<!-- Screen -->
<!-- ****** -->

<!-- Tecplot -->
<!-- ******* -->

<!--============================================================
=============  GRID GENERATION  ================================
============================================================-->

<!-- ggn -->
<!-- *** -->

<!-- Zones -->
<!-- ***** -->

<!--============================================================
=============  FRACTURE NETWORK  ===============================
============================================================-->


<!--============================================================
=============  OBJECTS & LOCATIONS  ============================
============================================================-->


<!--============================================================
=============  FACILITIES  =====================================
============================================================-->


</cif>
```

4.2 Overview

The best way to understand how CIF-commands are used is probably by examples; a tutorial is part of the lecture notes presented at DarcyTools courses. Next a number of Verification cases can be studied in order to see a wide range of applications. During this process, the list of commands given in the next section should also be consulted, to get a deeper understanding of possibilities.

Still, it may be difficult to find the right command for a specific task. In order to somewhat simplify this task a table that gives an overview of the purpose and name of all commands is provided, see Table 4-1.

In Appendix D all commands with parameters are listed. It is recommended that this appendix is printed; it can then serve as a “Quick Reference”, when setting up a new problem.

Table 4-1. Overview of CIF-commands.

Objective, purpose	CIF-command	Category	Page
Assign universal constants	<cst>	Physical models	**18**
Fluid specific mass heat	<law_cp>	Physical models	**19**
Assign density	<law_rho>	Physical models	**20**
Assign fluid dynamic viscosity	<law_mu>	Physical models	**21**
Grouting properties	<mat>	Physical models	**22**
Initialize variables	<var>	Variables and initial cond.	**23**
Initial pressure, salinity and temperature	<spt>	Variables and initial cond.	**24**
Declaring flux variables	<flux>	Variables and initial cond.	**26**
Default boundary conditions at top boundary	<defbc>	Boundary cond. Sources & Sinks	**27**
General boundary and source and sink conditions	<bss>	Boundary cond. Sources & Sinks	**28**
Particle tracking	<partrack>	Special models	**29**
Multi-rate model	<mrpow>	Special models	**30**
Output of particle tracks	<tracks>	Special models	**31**
Groundwater table	<gwt>	Special models	**32**
Sub-grid model FRAME	<frame>	Special models	**33**
Tunnel model, grouting	<tunsec>	Special models	**34**
Detailed grouting	<grouting>	Special models	**35**
Permafrost, ice-model	<ice>	Special models	**36**
Predifined equations	<eqn>	Equations	**37**
Equation sets and solutions	<eqs>	Equations	**38**
Time discretisation	<time>	Solver	**39**
Equation loops	<loop>	Solver	**40**
Run titles, licence string	<run>	Run & Store, Input-Output	**41**
Restart conditions for solver	<slv>	Run & Store, Input-Output	**42**
Sequences for output	<seq>	Run & Store, Input-Output	**43**
Output file	<rof>	Run & Store, Input-Output	**44**
Format of restart file	<rst>	Run & Store, Input-Output	**45**
Cartesian output file	<cof>	Run & Store, Input-Output	**46**
Tunnel output file	<tof>	Run & Store, Input-Output	**47**
Indexfile for grid cells	<indg>	Run & Store, Input-Output	**48**
Screen output	<screen>	Screen Input-Output	**49**
Real time visualization	<hist>	Screen Input-Output	**50**
Output to Tecplot	<tecplot>	Tecplot Output	**51**
Grid generation, basics	<gridgen>	Grid Generation	**52**
Grid generation, parameters	<ggn>	Grid Generation	**53**
Grid generation, zones	<zone>	Grid Generation	**54**
Grid generation, restart	<ggr>	Grid Generation	**55**
Grid block generation	<blockgen>	Grid Generation	**56**
Fracture generation, basics	<fracgen>	Fracture Network	**59**
Random fracture sets	<ranf>	Fracture Network	**60**
Known fracture specification	<knwf>	Fracture Network	**64**
Fracture file commands	<fraccmds>	Fracture Network	**66**
Object generation	<obj>	Objects and Locations	**70**
Location generation	<loc>	Objects and Locations	**71**
Objects from stl-file	<ogn>	Facilities	**73**
Definition of stl-file	<stl>	Facilities	**74**
Distance file generation	<dgn>	Facilities	**75**

4.3 Commands


```
<cst>
  <g>     : default gravity acceleration   [1-] (default= 9.81)
```

This command sets the universal constants. Presently only the gravity constant is specified by the argument <g>.

Notes:

- The default value of the acceleration of gravity is 9.81 and does not normally need to be set. The access to the value may however be useful if a problem is scaled in some way.

```
<law_cp>
 <a0>       : default fluid specific mass heat [1-] (default= 4182.)
 <a1>       : linear salinity coefficient      [1-] (default= 0.13)
 <a2>       : quadratic salinity coefficient   [1-] (default=-0.0001)
 <store>    : store in array cp                [1-] (default = F)
```

This command sets the fluid specific mass heat state law parameters. The constants may be specified separately and represent the following law:

$C_p = a_0 (1 + a_1 S + a_2 S^2)$

where S is the salinity.

When <store> = T the property is stored in an array and may be modified in FIF and sent to Tecplot, for example.

Notes:

```
<law_rho>
 <a0>       : default fluid density              [1-] (default=1000)
 <t0>       : reference temperature              [1-] (default=0)
 <a1>       : linear salinity coefficient        [1-] (default=0.008)
 <a2>       : quadratic salinity coefficient     [1-] (default=0)
 <b1>       : linear temperature coefficient     [1-] (default=0)
 <b2>       : quadratic temperature coefficient  [1-] (default=0)
 <store>    : store in array density             [1-] (default = F)
```

This command sets the fluid density state law parameters. The constants may be specified separately and represent the following law:

$$\rho = a_0 (1 + a_1 S + a_2 S^2 - b_1 (T - T_0) - b_2 (T - T_0)^2)$$

where S is the salinity and T the temperature.

When <store> = T the property is stored in an array and may be modified in FIF and sent to Tecplot, for example.

Notes:

```
<law_mu>
 <a0>          : default dynamic viscosity          [1-] (default=1.78E-3)
 <t0>          : reference temperature              [1-] (default= 0)
 <a1>          : linear salinity coefficient        [1-] (default= 0)
 <a2>          : quadratic salinity coefficient     [1-] (default= 0)
 <b1>          : linear temperature coefficient     [1-] (default= 0)
 <b2>          : quadratic temperature coefficient  [1-] (default= 0)
 <n>           : exponent value                     [1-] (default=-1)
 <store>       : store in array viscosity           [1-] (default = F)
```

This command sets the fluid dynamic viscosity state law parameters. The constants may be specified separately and represent the following law:

$$\mu = a_0 (1 + a_1 S + a_2 S^2 + b_1 (T - T_0) + b_2 (T - T_0)^2)^n$$

where S is the salinity and T the temperature.

When <store> = T the property is stored in an array and may be modified in FIF and sent to Tecplot, for example.

Notes:

- It is left to the user to find the optimum constants that fit a certain case. Note also that a certain pressure effect may also need to be considered when chosing the constants.

```
<mat>
  <name>            : reference name           [1=]
  <density>         : density                  [1-] (default=law_rho%a0)
  <viscosity>       : dynamic viscosity        [1-] (default=law_mu%a0)
  <bingham_stress>  : Bingham yield stress     [1-] (default=0.0)
  <bingham_espilon> : Bingham regularisation   [1-] (default=1.E-4)
  <diffusivity>     : diffusivity coefficient  [1-] (default=0.0)
  <schmidt>         : Schmidt number           [1-] (default=0.7)
  <schmidt_turb>    : Turbulent Schmidt number [1-] (default=0.7)
  <gel>             : silicasol A, Tg, Vmax    [1-]
```

This command sets the grout properties such as density, viscosity, Bingham or gel constants. It is a subset of the <mat> command available for the pore scale and surface water solvers. In particular, DTS discards the diffusivity (specified by diffx, diffy and diffz variables), the Schimdt number and the Bingham regularisation parameters.

Notes:

- See Technical note about “Grouting model” available at the ProjectPlace.

```
<var>
 <name>       : reference name             [1=]
 <pos>        : cell position              [1-] (default='none')
 <nbsteps>    : number of steps (1-3)      [1-] (default=1)
 <nbvals>     : number of values (>0)      [1-] (default=1)
 <blank>      : blanking output value      [1-] (default=-1.0)
 <inifile>    : file name                  [1-]
 <ini>        : locname, val, vm1, vm2     [0+] (default=domain,0,0,0)
 <ini>        : locname, val, vm1          [0+]
 <ini>        : locname, val               [0+]
```

This command has two functionalities: 1/ specify initial values for predefined variables (listed below) and 2/ create a user-defined variable.

When the specified name is a predefined DarcyTools name, this command specifies the initial conditions of the variable. Otherwise, it defines a user variable for FIF use. The variable must have a unique name among variable names. The argument <nbsteps> specifies the number of arrays that the variable will contain. It can take the value 1, 2 or 3. When nbsteps=1 the sole 'val' array is allocated for the current time step value. When nbsteps=2 the array 'vm1' is allocated and automatically updated to contain the previous time step value. When nbsteps=3 the 'vm2' array is allocated and contains the value of time step n-2. The location of the variable is specified with the argument <pos> whose value is a string among 'cell', 'xface', 'yface', or 'zface'. When the location of the variable is set, the number of values in the variable <nbvals> is discarded and automatically set by the solver, according to the grid specification.

When <inifile> is present, the initial values of the variable are read in the specified file. When the variable is not initialized by reading a file, it can be initialized by <ini> arguments. The <ini> arguments take three forms, depending on whether also earlier time step values must be initialized. Using the <ini> arguments, the variables are uniformly initialized in the specified locations with the specified values. The locations are separately defined by commands <loc>. When a cell belongs to several overlapping locations the last <ini> setting of the command applies.

DarcyTools predefined variable names are:

pressure, salinity, temperature, darcy-u, darcy-v, darcy-w, poros, stora, pref, capa, permx, permy, permz, diffx, diffy, diffz, tcondx, tcondy, tcondz, pme, ssea, tsea, tland, frm_fws, gwt_fill, gwt_gh, pdens, npc, npo, ptk_fws, frevol, indg, cellmk, cp, tcond, grout, rho, mu, tif, ice_phi, ice_dphi, ice cp, ice_rho, ice_tcond.

Notes:

- The inifiles are in most cases generated by PROPGEN.
- The format of inifiles is given in Chapter 5.

```
<spt>
 <s0>          : salinity for z=0                [1-] (default=0)
 <t0>          : temperature for z=0             [1-] (default=0)
 <s1>          : salinity for z>0                [1-] (default=0)
 <t1>          : temperature for z>0             [1-] (default=0)
 <dwt>         : depth of water table (>0)       [1-] (default=0)
 <dsdz>        : salinity gradient (<0)          [1-] (default=0)
 <dtdz>        : temperature gradient (<0)       [1-] (default=0)
```

This command initializes the pressure, salinity and temperature fields assuming a vertical equilibrium given by:

$$\frac{\partial P}{\partial z} = (\rho_0 - \rho)g \qquad (4\text{-}1)$$

For the sake of efficiency the initialization procedure splits the domain into five zones. The zones 1, 2 and 3 are located under the land the zone 4 under the sea and the zone 5, which is not in the computational domain, represents the sea. Moreover, the zone 1 is located above the ground water table. Both zone 1 and 2 are also located above the sea level z=0. Finally the zone 3 is located under the sea level z=0.

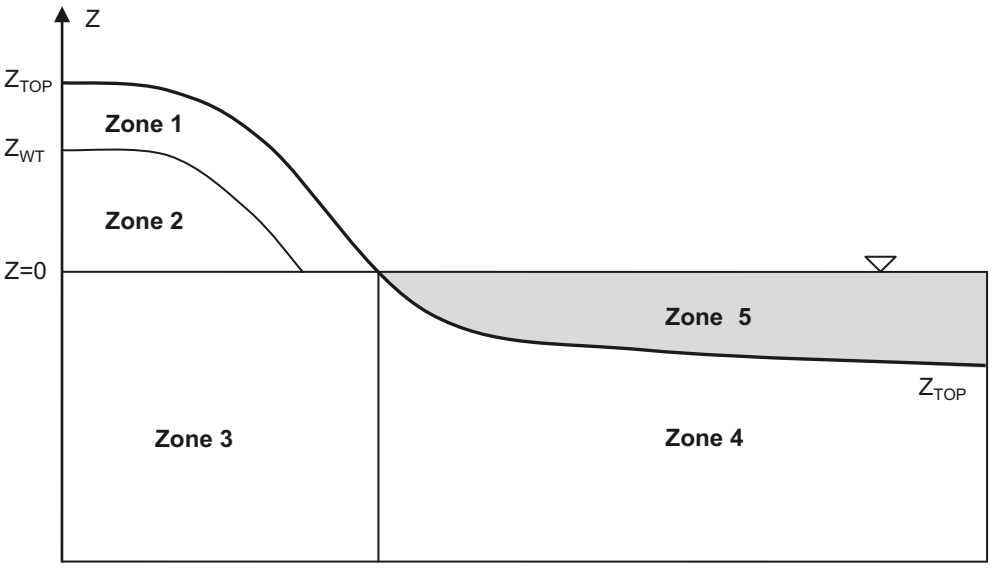


Figure 4-1. *SPT zone labeling.*

The initialization favours a smooth distribution in the bottom of the domain by setting a linear vertical profile for salinity and temperature and an equilibrium pressure in the zones 3 and 4. An approximated uniform distribution is assumed in zones 1 and 2.

Zone 1 & 2:
$$\begin{cases} S = S_1 \\ T = T_1 \\ P = \left(\rho_0 - \rho_{(S1,T1)}\right) g\,(z - z_{WT}) + \rho_0 g\, z_{WT} \end{cases} \quad (4\text{-}2)$$

Zone 3:
$$\begin{cases} S = S_0 + \dfrac{\partial S}{\partial z}\, z \\ T = T_0 + \dfrac{\partial T}{\partial z}\, z \\ P = \rho_{(S1,T1)}\, g\, z_{WT} - \rho_0 g\, f_0(z) \end{cases} \quad (4\text{-}3)$$

Zone 4:
$$\begin{cases} S = S_0 + \dfrac{\partial S}{\partial z}\,(z - z_{Top}) \\ T = T_0 + \dfrac{\partial T}{\partial z}\,(z - z_{Top}) \\ P = \left(\rho_0 - \rho_{(S0,T0)}\right) g\, z_{top} - \rho_0 g\, f_0(z - z_{Top}) \end{cases} \quad (4\text{-}4)$$

Zone 5:
$$\begin{cases} S = S_0 \\ T = T_0 \\ P = \left(\rho_0 - \rho_{(S0,T0)}\right) g\, z \end{cases} \quad (4\text{-}5)$$

with the function f_0, according to the density state law, such as:

$$\begin{aligned} f_0(x) = {} & x\left(a_1 S_0 + a_2 S_0^2 - b_1(T_0 - T_\rho) - b_2(T_0 - T_\rho)^2\right) + \\ & \frac{x^2}{2}\left((a_1 + 2a_2 S_0)\frac{\partial S}{\partial z} - (b_1 + 2b_2(T_0 - T_\rho))\frac{\partial T}{\partial z}\right) + \\ & \frac{x^3}{3}\left(a_2 \frac{\partial S}{\partial z}^2 - b_2 \frac{\partial T}{\partial z}^2\right) \end{aligned} \quad (4\text{-}6)$$

and where the groundwater table elevation is given by the depth d_{WT} (>0) and the formula (4-7) so that setting a very large value for d_{WT} (e.g. 1.E20) fixes the ground water table at sea level z=0.

$$z_{wt} = \max(0, z_{top} - d_{wt}) \quad (4\text{-}7)$$

z_{TOP} represents the elevation of the top boundary cell face center.

Notes:

```
<flux>
 <name>       : reference name                [1=]
 <loc>        : location name                 [1=]
 <faces>      : type boundary faces           [1-] (default='all')
```

This command declares three single value variables whose names are set by the argument <name> and the '+' and '-' suffixes. Each time the mass equation is solved, DarcyTools updates these variables by summing the mass fluxes through the faces of the location specified by the argument <loc>. Only the positive and negative mass fluxes are respectively summed for the variables 'name+' and 'name-'. The type of involved faces is set by the argument <faces>. 'all' means through all the faces of the location while 'east...low' means through the 'east...low' faces only.

When going 'west->east', the fluxes are positive on east faces and negative on west faces (i.e. positive when getting out of a box and negative when entering the box). When the face type is 'all' only 'cell-marker', 'patches', 'depth' and 'domain' are valid for location. When the face type is 'east' the same locations are allowed as well as the 'east-marker' locations (ditto for 'west...low').

Valid arguments for <faces>: east, west, south, north, high, low and all.

Notes:

- The command is useful when calculating the flux through a domain or the total inflow to a repository. In the first case <faces> is set to the face (cell-walls) in question, while the default value applies in the second case.

```
<defbc>
 <ssea>       : salinity of the sea                          [1-]
 <tsea>       : temperature of the sea                       [1-]
 <tland>      : temperature of the land                      [1-]
 <pme>        : precipitation-evaporation velocity on land   [1-]
```

This command sets and activates the default top boundary conditions. When setting <ssea>, <tsea>, <tland> and/or <pme> this command creates a new variable named 'ssea', 'tsea', 'tland' and/or 'pme'. These variables are one element long and contain the specified value namely the salinity of the sea, the temperature of the sea, the temperature of the land and the precipitation minus the evaporation velocity on land. Users may also create these variables with the command <var> instead of using the above arguments. In this case the sea and land condition will possibly vary in space. When varying in space, these variables will have to be allocated according to the sea and land array lengths. When 'ssea', 'tsea' or 'tland' are specified, the default top boundary condition fixes these values on the sea and land cells of the top boundary. When they are omitted the default top boundary condition freezes the initial salinity and temperature values on the sea and land cells of the top boundary. The sea and land locations are automatically built by the solver as being the most top cells whose highest face altitude is: strictly greater than zero for land and lower or equal to zero for sea. During the sea/land building process users may change the sea-land assignment by returning the appropriate value in the user function USRLAND.

The qsrc and qphi arrays produced by those default boundary conditions are overwritten by the <bss> commands where their respective locations overlay. They can be modified in the USRBSS routine, see Appendix A, p12.

Notes:

```
<bss>
 <name>       : reference name                [1=]
 <loc>        : location name                 [1=]
 <type>       : type source terms             [1-] (default='point')
 <qsrc>       : constant source coefficient   [1-] (default=0.0)
 <qphi>       : linear source coefficient     [1-] (default=0.0)
 <tmin>       : starting time                 [1-] (default=-oo)
 <tmax>       : ending time                   [1-] (default=+oo)
 <fixval>     : value to fix                  [1-]
 <fixflu>     : flux value to fix             [1-]
```

This command sets a boundary source-sink condition. A bss condition must be associated with a predefined or a user defined location using the <loc> argument. By default the <qsrc> and <qphi> values are strictly applied to all cells concerned by the specified location. Nevertheless, it is possible to automatically multiply the <qsrc> and <qphi> values by the cell volume, the cell x-cross area, the cell y-cross area or the cell z-cross area by setting the <type> argument to 'volume', 'xarea', 'yarea' or 'zarea'. When the <type> argument is set to 'freezing' the <qsrc> and <qphi> values are discarded and automatically set in order to freeze the variable to its previous time step value. Boundary source-sink conditions apply in the order they appear in the <eqn> command. Hence, when overlapping, only the last definition applies to the location intersection. The application of boundary source-sink conditions is restricted to the time window defined by the <tmin> and <tmax> arguments. The default time window value is such that the condition applies to all the time.

With the patch location and the 'volume', 'xarea', 'yarea' and 'zarea' boundary source-sink conditions, only the intersecting volumes and areas are applied when cells cross the patch boundary. DarcyTools predefines boundary source-sink conditions whose names are:

freez_east, freez_west, freez_north, freez_south, freez_high, freez_low.

The boundary source-sink condition freeze variables respectively on the east, west, north, south high and low predefined locations (all the time).

Two arguments have been added to the <bss> command in order to simplify its use. The <fixval> argument specifies the value to be set as Dirichlet condition, and the <fixflu> argument specifies the flux to be fixed. The <fixflu> argument can be associated to the <type> argument in order to specify the dimension of the flux. When <fixval> or <fixflu> are used the <qsrc> and qphi> values are automatically overwritten to fix the given value or the given flux.

Notes:

```
<partrack>
 <method>  : method (1 or 2)                          [1=]
 <seed>    : random generator seed                    [1-] (default=clock time)
 <traj>    : nbtrj,ntrjfr,lentrj,dltrj                [1-] (default=0,1,0,0.0)
 <steph>   : tsteph, wnstph(1-3)                      [1-] (default=1.E10,0.1,0.1,0.1)
 <nolond>  : no longitudinal dispersion               [1-] (default=T)
 <adsorp>  : adsorption                               [1-] (default=F)
 <reverse> : reverse flow moving                      [1-] (default=F)
 <sloc>    : (nb particles, loc name, tmin, tmax)     [0+]
 <eloc>    : loc name                                 [0+]
 <locfile> : file for reading sloc                    [1-]
```

PARTRACK has two ways (methods) of operation: 1/ streamline routing and 2/ the cell-jump method (see Report 1). The command sets and activates the particle tracking PARTRACK model according to the specified method. The starting particle locations and their respective number of released particles and time windows are specified by arguments <sloc> or by reading the file whose name is given by argument <locfile>. The file format (ascii or binary) is automatically detected during the reading process. The ending location where particles are captured is set by argument <eloc>. Both <sloc> and <eloc> refer to locations defined elsewhere using the command <loc>. When method 1 is selected, the argument <steph> specifies the velocity freezing time (during which the velocity is kept constant) and the x, y and z displacement limit for velocity freezing. The argument <nolond> switches off the longitudinal diffusion, <adsorp> switches on the adsorption and <seed> fixes the random generator seed. Beside those parameters, the <traj> argument sets the number of output trajectories, their distribution frequency among time steps (= 1, every time step) and their length in term of time steps. dltrj generates a trajectorie output as soon as the distance from the last position is greater than dltrj.

During the time step calculation the density of particles is regularly updated in the variable ‘pdens’ as well as the number of captured particles ‘npc’ and the number of captured or outputted particles ‘npo’. The trajectories and/or swarms of particles can be outputted to tecplot files using command <tecplot>. When the argument <reverse> is set true, the particles move upstream instead of down-stream.

Regarding the argument <adsorp>, see command <mrpow>.

Notes:

- The command <steph> determines how often the particle velocity should be updated., when using method = 1 (which is a traditional flow following tracking). The velocity update is either based on a time-criterion or a maximum displacement (expressed as fraction of cell size) in one of three coordinate directions.
- The porosity field will strongly affect the transport of particles. The porosity field is used to calculate the free space in each cell. However, if the frevol (free volume in cell) variable is defined by a variable statement, this array is used instead. This may be useful if small corrections to the transport times are needed.
- The variables npc, npo and pdens can usefully be visualized in hist-files.

```
<mrpow>
 <nalint>  : number of alpha interval (=naintt)     [1-] (default=0)
 <alphl>   : low alpha limit                        [1-] (default=0)
 <alphh>   : high alpha limit                       [1-] (default=0)
 <ak>      : late time slope of breakthrough curve  [1-] (default=0)
 <bettot>  : volume ratio value (immobile/mobile)   [1-] (default=0)
 <fxdift>  : uniform cross diffusion coefficient    [1-] (default=0)
 <retmob>  : retardation mobile                     [1-] (default=0)
 <nmobst>  : number of a/d states in mobile zone    [1-] (default=0)
 <naintn>  : number of a/d states in immobile zone  [1-] (default=0)
```

Associated to the PARTRACK model, this command activates and sets the multi-rate model parameters.

The reader is referred to Report 1 for details and a general background.

Notes:

- Presently the best source of information is the SKB report IPR-06-21.

```
<tracks>
 <file>     : file name for output     [1=]
 <binary>   : kind of output format    [1-] (default=T)
 <ievent>   : event to write in file   [0+] (from 1 to 4)
 <var>      : variable name            [0+]
```

Associated to the PARTRACK model, this command sets the name and the format of the file outputting the F-QUOTIENT and other variables along the particle tracks. By default all events are written in the file. However the argument <ievent> allows specifying the index of the only events that should be outputted. The specified variables must be cell centred.

In order to control how much is written to the file, the variable ievent has been introduced. It is possible to write only the initial values (1), or the information along the flow paths (2), or the conditions when a particle leaves the domain (3) or enters a catching box (4). The default is that all information is written to the output file.

Notes:

- See the description of the output file in Chapter 5.

```
<gwt>
 <relax>  : under-relaxation parameter                [1=]
 <facmin> : property reduction factor                 [1-] (default=0.001)
 <deltsy> : yield time constant for delayed response  [1-] (default=-1)
 <vlfrsy> : volume fraction of yield water            [1-] (default=0)
 <betasy> : immobile to mobile volume fraction        [1-] (default=0)
 <zmin>   : altitude above which gwt activates        [1-] (default=-∞)
```

This command activates and sets the parameters of the ground water table model, including its specific yield option.

The main feature of <gwt> is that the horizontal conductivities are reduced to a fraction <facmin> above the ground water table. Some under-relaxation of the movement of the water table, say relax = 0.3, is often required. It may also be useful to set a limit on the depth of a drawdown, for example due to a repository, by the parameter <zmin>.

Notes:

- The arguments <deltsy>, <vlfrsy> and <betasy> are part of the specific yield model. This model, see Report 1, is however not recommended for general use.

```
<frame>
 <var>     : variable name                       [1=]
 <nalint>  : number of alpha interval            [1=]
 <alphl>   : low alpha limit                     [1-] (default=1.E-9)
 <alphh>   : high alpha limit                    [1-] (default=0.05)
 <bettot>  : global volume ratio immobile/mobile [1-] (default=1.0)
 <frmk>    : FRAME minus late time slope         [1-] (default=1.5)
 <iniccv>  : ccv initialization from variable    [1-] (default=T)
 <alphd>   : single rate alpha diffusion         [1-] (default=-1.0)
```

This command activates and sets the parameters of the sub-grid FRAME models to be applied to the concentration equations. The argument <var> specifies the name of the variable (salinity or concentration) to which the model will apply.

The FRAME model is based on the multi-rate diffusion concept. A single-rate alternative can however be specified by the following set of parameters:

<alphd> (default = -1)

<bettot>

where alphd = D/h^2. The model considers a homogenous layer of thickness, h, with a diffusion coefficient, D, and a porosity, θ. The default value specifies the multi-rate option. When alphd > 0, alphl, alphh and frmk are ignored.

The reader is referred to Report 1 for details and a general background.

Notes:

- When activating this model one should be aware of the memory aspect. For each variable nalint values are stored in each cell.

```
<tunsec>
 <marker> : marker value                        [1=]
 <sink>   : mass withdrawal (kg/s)              [1-] (default=0.0)
 <skmax>  : maximun skin factor                 [1-] (default=1.E+10)
 <skmin>  : minimun skin factor                 [1-] (default=1.E-10)
 <relax>  : under relaxation parameter          [1-] (default=0.0)
 <skin>   : initial skin factor                 [1-] (default=0.001)
 <resat>  : resaturated status                  [1-] (default=F)
 <permax> : fixed wall permeability             [1-]
 <grout>  : laymin,laymax,pmin,pmax,pval        [0+]
```

The TUNNEL model handles the inflow to an open tunnel. It can consider a section with a given inflow or calculate the inflow based on a grouted rock.

This command activates the TUNNEL model by setting the definition of tunnel sections. A tunnel section is made of a collection of cells sharing the same <marker> value as defined during the grid generation.

Most applications of this model will use the <permax> parameter to specify the maximum permeability in the cells that are in contact with the tunnel walls. However, if detailed information about the grouting efficiency is available the <grout> parameter can be used. It specifies the permeability for grid layers laymin to laymax, provided the permeability is in the interval pmin to pmax. If so, the permeability is set to pval. The <resat> parameter removes the fixed atmospheric pressure condition inside the tunnel, when set to T.

The TUNNEL model may also use a specified <sink> for the section and, by iteration, find a suitable skinfaktor.

When <resat> is put to T the routine will not prescribe pressure inside the tunnel section.

Notes:

```
<grouting>
 <type>   : type of grout ('silicasol', 'bingham')    [1=]
```

This command activates the grouting model by specifying the type of grout in use.

This command automatically creates a `grout´material and the corresponding `grout´equation and equation set. The users are in charge of initializing the `grout´variable, defining properties with <mat> command and solving the `grout´equation by setting <eqs>grout</eqs> in a loop command. When the grout viscosity or density differs from the water viscosity and density, the `mass´and `grout´equations should be solved in the `sweep´loop.

Notes:

- See report on ProjectPlace called “Grouting model” for details.

```
<ice>
 <phi_fn>   : phi function ID                   [1=]
 <phi_tl>   : liquid temperature                [1-] (default=0.0)
 <phi_w>    : decay factor                      [1-] (default=1.0)
 <phi_max>  : scaling value                     [1-] (default=1.0)
 <alpha_fn> : alpha function ID                 [1-] (default=1)
 <alpha_a>  : phi exponent                      [1-] (default=5)
 <alpha_min>: minimum reduction factor          [1-] (default=1.E-5)
 <latent>   : latent heat of fusion             [1-] (default=3.33E+5)
 <rho_a0>   : default ice density               [1-] (default=917.)
 <rho_t0>   : reference temperature             [1-] (default=0.)
 <rho_b1>   : linear temperature coefficient    [1-] (default=0.)
 <rho_b2>   : quadratic temperature coefficient [1-] (default=0.)
 <cp_a0>    : default ice specific mass heat    [1-] (default=2110.)
 <cp_t0>    : reference temperature             [1-] (default=0.)
 <cp_b1>    : linear temperature coefficient    [1-] (default=0.)
 <cp_b2>    : quadratic temperature coefficient [1-] (default=0.)
 <tcond_a0> : default ice thermal conductivity  [1-] (default=2.14)
 <tcond_t0> : reference temperature             [1-] (default=0.)
 <tcond_b1> : linear temperature coefficient    [1-] (default=0.)
 <tcond_b2> : quadratic temperature coefficient [1-] (default=0.)
 <store>    : ice_xxx variable name to store    [0+]
```

This command activates the ice model.

Notes:

- See report on ProjectPlace called “Ice model” for details.

```
<eqn>
 <name>       : reference name                       [1=]
 <var>        : variable name                        [1-]
 <bss>        : bss name                             [0+]
 <nbgrad>     : number of evaluations for gradient [1-] (default = 3)
 <relin>      : linear relaxation factor           [1-] (default = 1.0)
```

This command defines the setting for DarcyTools predefined equations named: ‘mass’, ‘salt’ and ‘heat’. The mass equation solves the pressure, the salt equation solves the salinity and the heat equation solves the temperature. This command also defines user equations setting. Users mass concentration equations are created by using a new name in the <name> argument and by setting a user variable name in <var>. This command also specifies the boundary-source-sink conditions that will apply. These boundary source-sink conditions are defined elsewhere in the CIF by the command <bss>.

The parameter <relin> sets the relaxation factor to apply when updating the solution after MIGAL’s computation. The <relin> value set for the <eqs> equation applies instead of the equation value when smaller.

The parameter <nbgrad> sets the number of iterations used to evaluate gradients. The default value is recommended generally.

Notes:

```
<eqs>
 <name>       : reference name                   [1=]
 <relin>      : solution relaxation factor       [1-] (default=1)
 <resfac>     : rms ending reduction ratio       [1-] (default=0.1)
 <relax>      : smoother relaxation              [1-] (default=0.95)
 <alpha>      : agglomeration threshold          [1-] (default=0)
 <ratio>      : end coarsening ratio threshold   [1-] (default=1.5)
 <diag>       : artificial diagonal dominance    [1-] (default=0)
 <liter>      : maximum number of cycles         [1-] (default=2)
 <nbrelax>    : number of post-relaxations       [1-] (default=3)
 <ipreco>     : number of MG-cycles for GMRES    [1-] (default=1)
 <igmres>     : GMRES subspace dimension         [1-] (default=-1)
 <nbgrid>     : maximum number of grids          [1-] (default=20)
 <icycle>     : MG cycle type (0=V, 1=W)         [1-] (default=0)
 <nbprer>     : number of pre-relaxations        [1-] (default=1)
 <igms>       : smoother subspace dimension      [1-] (default=0)
 <ilink>      : kind of agglomeration            [1-] (default=2)
 <nbmax>      : maximum coarse cell size         [1-] (default=0)
 <nbmin>      : minimum coarse cell size         [1-] (default=0)
```

This command defines equation sets and the way they are solved. DarcyTools only solves equation sets. By default each predefined equation is associated to a predefined equation set having the same name: ‘mass’, ‘salt’ and ‘heat’. A forth predefined equation set whose name is ‘mass-salt’ is dedicated to the coupled solution of the mass equation with the salt equation. When users create their own equations, for instance to solve mass concentration, an equation set having the same name is automatically created. The main purpose of this command is to set the MIGAL parameters for solving the equation sets.

The parameter <relin> applies to every variable solved by the set of equations. When the <relin> parameter of the <eqn> command is set to a smaller values this value applies instead of the equation set value.

Notes:

- In case of convergence problems, a user may as a first step reduce <relin> and increase <liter>.
- As a second step <igmres> can be put to 4 and <ipreco> to 3.
- For parallelized cases <nbrelax> may be increased to speed up convergence.

```
<time>
 <order>      : order of time discretisation [1-] (default=0)
 <dt>         : time step                    [1-] (default=1.D+20)
 <t0>         : initial time                 [1-] (default=0)
 <zone1>      : n1, t1                       [1-] (default=0, t0)
 <zone2>      : n2, t2                       [1-] (default=0, t1)
```

This command specifies the time discretisation process. The argument <order> sets the order of the time scheme and can take the value 1 or 2. By default no time derivative is implemented in the equations (order=0). The initial time and the default time step are set with the arguments <t0> and <dt>. When an automatic time step generation is needed, this command may consider two zones before applying the default time step. The first zone <zone1> defines a constant time step between the initial time ‘t0’ and the time ‘t1’ by generating ‘n1’ time steps. The second zone <zone2> defines a hyperbolic tangent distribution from time ‘t1’ to time ‘t2’ by smoothly generating ‘n2’ time steps. After ‘t2’ the default time step is applied. None or only one of the zones may be set.

Notes:

- A variable time step can also be set in the user function USRDT.

```
<loop>
 <name>       : reference name                 [1=]
 <nbit>       : number of iteration            [1-] (default=0)
 <eqs>        : equation set to solve          [0+]
 <usrloop>    : user loop to run               [0+]
 <hist>       : history file to write          [0+]
 <rof>        : result output file to write    [0+]
 <cof>        : cartesian output file to write [0+]
 <tof>        : tunnel output file to write    [0+]
 <tecplot>    : tecplot file to write          [0+]
```

This command defines the DarcyTools algorithm i.e. the equation to be solved the numbers of time steps or sweeps, the hist or tecplot outputs and when these actions are taken. By default DarcyTools defines four loops named: ‘main’, ‘ini’, ‘step’ and ‘sweep’. The ‘main’ loop has only one iteration <nbit>. It calls the ‘ini’ loop first and then the ‘step’ loop. The ‘ini’ loop has only one iteration and calls the USRINI routine. The ‘step’ loop has a single iteration by default and calls the USRDT routine followed by the ‘sweep’ loop. The sweep loop (for iterations on the step) has no iteration by default. For solving an equation set inside a loop, users must specify its name with the argument <eqs>. For writing a tecplot file, a history file, a result output file, a Cartesian file or a tunnel output file, their name must be specified with the argument <tecplot>, <hist>, <rof>, <cof> or <tof>. This command also allows users to create their own loops by setting a new name in the argument <name> and to insert those loops in other loops with the argument <usrloop>.

Actions are executed following their order in the <loop> command. The resulting algorithm can be checked in the ‘solver.log’ file or on screen when setting the <screen> command for it.

```
<!-- loops -->

<loop>
 <name>    ini      </name>
 <tecplot> caseA4   </tecplot>
</loop>

<loop>
 <name>    step     </name>
 <nbit>    10       </nbit>
 <eqs>     mass     </eqs>
 <eqs>     salt     </eqs>
 <hist>    hist1    </hist>
 <hist>    hist2    </hist>
 <tecplot> caseA4   </tecplot>
</loop>
```

```
-------------------------------------------
|               ALGORITHM                 |
-------------------------------------------

 do main=1,1
  do ini=1,1
   write tec:caseA4
  enddo
  do step=1,10
   solve eqs:mass
   solve eqs:salt
   write hist:hist1
   write hist:hist2
   write tec:caseA4
  enddo
 enddo
```

Example of a loop CIF setting and its resulting algorithm.

Notes:

```
<run>
 <title>      : title of the run          [1-] (default='')
 <licence>    : licence string            [1-] (default='')
 <gridfile>   : grid file name            [1-] (default='xyz')
 <nbthreads>  : number of threads to use [1-] (default = 0)
```

This command sets the title of the run, the licence string and the grid file name. The title is used as default title for the Tecplot output files while the grid file name is used for the grid generation outputs and solver inputs. When the licence string is omitted DarcyTools uses the content of the "mgl.lic" file.

The <nbthreads> parameter specifies how many cores (assuming a multi core processor) that the run will use.

Notes:

```
<slv>
 <restart>    : restart from rst file       [1-] (default=F)
 <reset>      : reset time and step to 0    [1-] (default=F)
```

This command sets the restart conditions for the solver. When <restart> is set to true (T), the computation is restarted by reading the file ‘rstslv’. During a restart, the initial time and the step counter can be reset to zero by setting the argument <reset> to true.

All variables needed for a correct restart are stored in the rst file.

Notes:

- It is possible to restart the solver but not rstptk (for PARTRACK) or rsttun (for Tunnel) or rstgwt (for Ground Water Table) or rstfrm (for FRAME) by simply removing the relevant restart file from the working directory.
- When rsttun or rstgwt are used also permeability files are restarted, as these are modified in these models.

```
<seq>
 <name>       : sequence name                [1=]
 <step>       : step value                   [0+]
 <time>       : time value                   [0+]
```

This command sets a time sequence for outputs. A time sequence is a collection of steps (index value of the time steps) and time values. When a sequence is specified in an output command such as <tecplot>, <rof>… the output writing occurs only for the time steps whose values are present in the sequence or when a ‘time’ value exists in the sequence such that $t < time \leq t+dt$.

Notes:

```
<rof>
  <name>       : reference name             [1=]
  <file>       : file name                  [1-] (default='name.rof')
  <binary>     : kind of output format      [1-] (default= T)
  <double>     : 8 bytes reals outputs      [1-] (default= F)
  <seq>        : output time sequence       [1-]
  <var>        : add a variable to output   [0+]
```

This command creates an output file for storing variable values during the simulation. To be stored a variable must be declared by argument <var> and the reference <name> of the ROF file must be inserted in a <loop> command. The variables will be stored according to the output sequence given by argument <seq> when specified (every loop iteration otherwise). The file format and its name can be specified using arguments <binary> and <file>. When the variable values need to be stored with 8 bytes values, the <double> argument must be set .true.

Notes:

- See Chapter 5 for file format.

```
<rst>
 <binary>     : kind of output format      [1-] (default= T)
 <var>        : add a variable to output   [0+]
```

This command specifies the format of the restart file ‘rstslv’. The restart file is a ROF file used by the solver to store all the variables needed for a safe restart. When users modify some variables like properties during the process and need them for a restart, these variables can be added to the ‘rstslv’ file using the argument <var>. The specified variables will be reloaded in the restart process.

Notes:

```
<cof>
 <name>       : reference name            [1=]
 <var>        : add a variable to output  [1=]
 <nodes>      : nx, ny, nz                [1=]
 <xspan>      : xmin, xmax                [1-] (default=domain xsize)
 <yspan>      : ymin, ymax                [1-] (default=domain ysize)
 <zspan>      : zmin, zmax                [1-] (default=domain zsize)
 <file>       : file name                 [1-] (default='name.cof')
 <binary>     : kind of output format     [1-] (default=T)
 <seq>        : output sequence name      [1-]
```

This command creates an output file associated to a variable for storing its current values during the simulation. The outputted values are located on nodes of a uniformly spaced Cartesian grid. They are given by the center value of the cells containing the Cartesian grid nodes. The BLANK variable attribute (see var command) is outputted for the nodes outside the variable domain definition. To be stored a variable must be declared by the argument <var> and the reference <name> of the COF file must be inserted in a <loop> command. The variables will be stored according to the output sequence given by argument <seq> when specified (every loop iteration otherwise). The file format and its name can be specified using arguments <file> and <binary>. Finally the Cartesian output grid is set by its spans and its number of nodes in each direction. When a span argument is omitted the effective domain size is automatically set.

Notes:

- See Chapter 5 for file format.

```
<tof>
 <name>       : reference name             [1=]
 <file>       : file name                  [1-] (default='name.tof')
 <seq>        : output sequence name       [1-]
 <binary>     : kind of output format      [1-] (default= T)
 <var>        : add a variable to output   [0+]
 <tunsec>     : add a section marker       [0+] (default= 0)
```

This command creates an output file for storing variable values from tunnel walls during the simulation. To be stored a variable must be declared by argument <var> and the reference <name> of the TOF file must be inserted in a <loop> command. The variables will be stored according to the output sequence given by argument <seq> when specified (every loop iteration otherwise). The file format and its name can be specified using arguments <binary> and <file>. When no specific tunnel section is set by specifying a marker value, DarcyTools will output all tunnel sections. As soon as at least one tunnel section is specified, only the given sections are outputted.

Notes:

- See Chapter 5 for file format.

```
<indg>
 <infile>    : input file name    [1=]
```

This command sets the name of the file containing the INDGRD array. The type (ASCII, binary) of the file is automatically detected during the reading process.

Notes:

- Operational, but not yet utilized.

```
<screen>
 <error>     : errors                     [1-] (default=T)
 <head>      : header                     [1-] (default=T)
 <cif>       : cif commands               [1-] (default=F)
 <algo>      : solver algorithm checkup   [1-] (default=T)
 <checkup>   : checkup                    [1-] (default=F)
 <init>      : initialization progress    [1-] (default=F)
 <ggn>       : grid generation progress   [1-] (default=T)
 <stats>     : grid statistics            [1-] (default=T)
 <slv>       : solver progress            [1-] (default=T)
 <end>       : ending progress            [1-] (default=T)
 <all>       : apply to all triggers      [1-] (default=F)
```

This command controls the screen outputs (in the command window) during the programs execution. All the arguments are Boolean. T is for setting on and F for switching off printings.

When <all> is set T or F its value applies to every parameter.

Information not printed on screen is still written in the log files.

Notes:

```
<hist>
 <name>       : reference name                   [1=]
 <var>        : add a variable to output         [1+]
 <title>      : history graph title              [1-]
 <profile>    : associate the graph to a profile [1-]
 <spot>       : associate the graph to a spot    [1-]
 <cut>        : plane or plate loc name          [1-]
 <palette>    : valmin, valmax, nbcolors         [1-] (default=0,0,256)
 <showgrid>   : true for showing grid on cut     [1-] (default=F)
 <expand>     : h/w factor in cut views          [1-] (default=1.0)
```

This command defines a new graph (history output) for the DarcyTools real time visualization. The graph itself is not activated until it is inserted into a loop command. Its name must be unique among hist names. At least one variable name must be specified. The default graph output represents the residual of the field variables or the value of single element variables. However this setting may be changed for a profile or a spot value output by using the arguments <profile> or <spot>. The names set in these arguments are the reference names of the profile or spot objects elsewhere defined in the CIF file. The <cut> argument refers to a plane or a cut location defined elsewhere by a command <loc>. It creates a two dimensional contour plot of the specified variable. Associated with the <cut> argument, the <showgrid> and <palette> arguments respectively set the grid output and fix the palette. In the latter case when valmin=valmax the minimum and maximum values are automatically computed when updating the output window. Also associated with the <cut> argument, <expand> specifies the dilatation factor in the vertical screen direction. A negative or null value indicates DarcyTools to compute the appropriate factor for a nice canvas filling. The computed value can be retrieved in the solve.log file. The default title of the graph can be specified with the <title> argument.

Notes:

- The hist files are formatted and may, after some editing, be imported to Tecplot and used for xy-plots.

```
<tecplot>
 <name>  : reference name                                    [1=]
 <file>  : file name                                         [1-] (default='name.plt')
 <title> : tecplot title                                     [1-] (default=run_title)
 <seq>   : output sequence name                              [1-]
 <loc>   : add a location zone                               [0+]
 <obj>   : add a tecplot zone made of the given object       [0+]
 <traj>  : add a tecplot zone made of the given trajectory   [0+]
 <swarm> : add swarm of all trajectory particles             [0+]
 <var>   : add a variable to the output                      [0+]
 <action>: output variable action (varname, action, value)   [0+]
```

This command defines a Tecplot output file and its content. The default file name is made of the (unique) reference name followed by a '.plt' extension. The default title string is the <run> command title. Beside the grid coordinates, Tecplot zones may contain variables values. The names of variables are specified by the argument <var>. When a modified variable value is to be outputted the argument <action> set the action to proceed. An action argument is made of the variable name and the action name. This action name is '+' for adding a value, '*' for multiplication by a value, 'log10' for outputting the logarithm base 10 of the variable and 'ln' for outputting the Neperian logarithm of the variable. For the '+' and '*' action a value must be added to the argument. This value may be omitted in the logarithm cases. When not concerning particles, Tecplot zones must contain at least one of the location specified by the <loc> or <obj> arguments. The allowed <loc> arguments are domain and plane locations. The <obj> arguments refer to objects defined elsewhere in the CIF file. The <traj> argument specifies the index of the particle whose trajectory is to be outputted. When a zero index value is set, all trajectories are outputted. The <swarm> argument switches on the output of trajectories in a 'swarm' fashion, i.e. one zone per time step. Tecplot outputs are always binary files for Tecplot version 10.

To output indg values, the name specified in argument <var> must have the following pattern: 'indgxy' where x represents the desired value and y the position for this value.

The outputs will be proceeded according to the output sequence given by argument <seq> when specified (every loop iteration otherwise).

Notes:

```
<gridgen>
 <span>      : x, y, z domain span              [1-] (default=1,1,1)
 <origin>    : x, y and z origin coordinates    [1-] (default=0,0,0)
 <binary>    : grid output file format          [1-] (default=T)
 <ggn>       : ggn name                         [1+]
 <tecplot>   : tecplot name                     [0+]
 <cluster>   : zone name                        [0+]
 <ggr>       : grid generation restart name     [0+]
```

This command sets the way the program GRIDGEN will build the grid. First, users should fix the domain sizes and origin location by using respectively the arguments and <origin>. Both argument values are made of three real numbers relative to the x, y and z directions. Then they should specify one or several grid generators with the argument <ggn>. GRIDGEN will successively call the grid generators in the order of their declaration. Grid generators are elsewhere defined by the command <ggn>.

The <cluster> argument specifies the clusters of cells that GRIDGEN will keep in the grid. A cluster of cell gathers the cells which can be flooded, i.e. that are connected, from the cells of the specified zone. When no <cluster> argument is used all cells are kept. The format of the output grid file, whose name is set by command <run>, can be specified by using the argument <binary>. Finally the program GRIDGEN may output one or several Tecplot files with the <tecplot> argument. The content of those files is specified by the corresponding <tecplot> commands.

When launched GRIDGEN starts by looking for the grid generation restart arguments <ggr> in the <gridgen> command. When one or several <ggr> arguments exists only the last one is considered and the <ggn>, <tecplot> and <cluster> arguments of the <gridgen> command are ignored. Instead of them GRIDGEN uses the <ggn>, <tecplot> and <cluster> arguments of the <ggr> command to generate the new grid.

Notes:

- Further information about GRIDGEN can be found at the ProjectPlace.
- See Chapter 5 for file format used for a grid generation restart.

```
<ggn>
 <name>      : ggn name                              [1=]
 <nbcells>   : maximum number of cell                [1-] (default=10000000)
 <maxlevel>  : maximum level                         [1-] (default=32)
 <isotropy>  : isotropic cutting allowance           [1-] (default=T)
 <c221>      : cross 221 rule application            [1-] (default=T)
 <l221>      : longitudinal 221 rule application     [1-] (default=F)
 <blur>      : blurring factor                       [1-] (default=0)
 <xcut>      : x direction cutting allowance         [1-] (default=T)
 <ycut>      : y direction cutting allowance         [1-] (default=T)
 <zcut>      : z direction cutting allowance         [1-] (default=T)
 <dxmax>     : maximum cell size in x direction      [1-] (default=-1)
 <dymax>     : maximum cell size in y direction      [1-] (default=-1)
 <dzmax>     : maximum cell size in z direction      [1-] (default=-1)
 <zone>      : zone name                             [0+]
```

This command sets the grid generator parameters. When 2D or 1D cases are of interest, users can disable the x, y and/or z cell cutting by using the logical <xcut>, <ycut> and <zcut> arguments. The Boolean arguments <c221>, <l221> respectively set the cross and longitudinal 221 rule i.e. the fact that two adjacent cells have always a cross (longitudinal) size ratio at most 2:1. <blur> specifies the default blurring factor for the domain. <isotropy> disables the isotropic refinement in the transformed space, i.e. the fact that refining a cell in one direction automatically refines it in all directions. The <nbcells>, <maxlevel>, <dxmax>, <dymax>, and <dzmax> arguments respectively specify the maximum desired number of cells, the maximum refinement level and the default maximum cell sizes in the x, y and z direction. They are ending conditions. When specified, the grid generator stops as soon as the number of cells exceeds nbcells or as soon as the smallest cell reaches the maximum level or as soon as the cell size criterion is reached everywhere. The <zone> arguments define different local refinement, removal and/or marking criteria. When a cell belongs to several overlapping zones the criteria of the first <zone> argument applies.

Notes:

- Further information (figures, details, etc) at the ProjectPlace.

```
<zone>
 <name>      : zone name                          [1=]
 <mask>      : (maskid, object name)              [1+]
 <marker>    : cell marker (≤0 for removal)       [1-] (default= 1)
 <dxmax>     : maximum cell size in x-direction   [1-] (default=-1)
 <dymax>     : maximum cell size in y-direction   [1-] (default=-1)
 <dzmax>     : maximum cell size in z-direction   [1-] (default=-1)
 <blur>      : blurring factor                    [1-] (default=0)
 <operand>   : mask operand (or, and)             [1-] (default=or)
```

A zone defines a part of the domain (a group of cells) that is to be modified (mesh refinement and/or cell removal).

This command defines zones for grid generation and/or boundary conditions setting. After specifying a unique name (among zone names) users must give at least one mask. A mask is defined by a mask ID and an object name. A mask ID is a string value among: “border”, “east”, “west”, “north”, “south”, “high”, “low”, “inside” and “outside”. The “border” value selects all cells cutting the boundary of the specified object. The cardinal values “east, west...” select all cells not cutting the boundary of an open surface object but having a center point located at the “east, west...” of it. The “inside, outside” values select all cells not cutting the boundary of a closed surface object but having a center point located inside or outside of it. When several masks are used, a cell belongs to a zone as soon as it belongs to one of the masks with the default ‘or’ operand. When the ‘and’ operand is selected a cell must belong to every mask to belong to the zone. Zones have three functions. The first one is to mark cells for preparing solvers actions. The second is to specify the grid refinement criteria. The third one is to remove useless cells from the grid. To mark cells, users will specify the <marker> argument integer value. When this value is negative or null the grid generator removes the cells and set the value in the high or low face attribute of the cells neighbouring this zone. The three real arguments <dxmax>, <dymax> and <dzmax> specify the maximum cell sizes (in x, y and z direction) allowed in the zone. The grid generator keeps refining the cells whose size is larger that the specified value, unless another ending criterion is encountered. The default size values are -1 which means that the maximum size allowed is the domain size itself (no refinement is performed). <blur> specifies the default blurring factor to apply when cutting cells of the zone.

For removing zones, the cell size criteria applies to the zone border in order to specify the precision at which the boundary will be handled. Because of the solver conventions, removing cells by setting marker = 0 should be reserved for the external boundary of the domain.

Notes:

- Further information at the ProjectPlace.

```
<ggr>
 <name>     : ggr name                                 [1=]
 <ggn>      : ggn name                                 [1+]
 <tecplot>  : tecplot name                             [0+]
 <cluster>  : zone name                                [0+]
 <rfile>    : refinement-removal criteria file name [1-]
```

This command offers more flexible grid modifications, based on the cell properties of an existing grid. For example, cells without fractures intersection can be identified in PROPGEN and be marked by a negative integer in an exported RRC file. In a second step, these cells can be deleted by the <ggr> command. (see Figure 3-1).

This command sets the grid generation restart parameters. The <ggn>, <tecplot> and <cluster> arguments are similar to the equivalent <gridgen> arguments and the specified name must be unique among ggr names. When specified, <rfile> represents the name and the path of the RRC file to read and apply for the restart. During a restart, cells are firstly removed according to negative or null markers of the RRC file. Then the maximum cell size of the RRC file are applied as default values for each <ggn> command when more restrictive than the dxmax, dymax and dzmax uniform values of these commands. Finally cells are marked according to the RRC file values before zone marking. In particular, during the refinement process, children cells are marked according to the specified RRC value of the parent cells.

Notes:

- See Chapter 5 for file format.

```
<blockgen>
 <nblocks>    : number of blocks                  [1=]
 <load>       : block id, load in percent         [0+]
 <partfile>   : partition file name               [1-] (default='')
 <tecplot>    : tecplot name                      [1-] (default='')
 <binary>     : grid output file format           [1-] (default=T)
 <gridpart>   : grid partitionning                [1-] (default=T)
 <partition>  : file to partition                 [0+] (default='')
 <merge>      : file to merge (no #xxx suffix)    [0+] (default='')
```

The <blockgen> command is considered only by the new BlockGeN program (BGN). The first argument <nblocks> sets the number of blocks for grid file partitioning and for file merging. The <gridpart> argument then specifies whether BGN will generate block grids from the original domain or will consider that the block grids already exist and simply have to be read. The <binary> argument fixes the type of output file format. When set to true, output files are formatted. When set to false, output files are unformatted. The <tecplot> argument refers to a <tecplot> command name for outputting grids similarly to the <gridgen> command. The <load> argument is for grid partitioning only and fixes the percentage of the total number of domain cells that a block must contain. This argument is ignored when the <partfile> argument exists and sets the file name of a user defined partition. Finally the <partition> and <merge> arguments can be used to specify names of files to be partitioned or merged according to the grid partitioning.

Every input data file related to the original grid will have to be partitioned with BGN before running parallel DarcyTools. For this, the usual procedure consists in using BGN in a first step for partitioning the grid. Then, in a second step, the <gridpart> parameter is set to false in the <blockgen> command and BGN is rerun with the original (i.e. associated to the sequential grid) data files names specified in the <partition> arguments. In this step, BGN produces block files with the same names suffixed according to the block ID. These new files are signed with the block grid signature to prevent errors during the file loading step of the solver. When input data files are ready at grid partitioning step, both jobs can be done simultaneously by keeping <gridpart> as true. However, because the run signature is regenerated during block building, setting <gridpart> to false is necessary to keep already partitioned input data files valid.

Files that need to be partitioned are: Variables initialization files, Location files and INDG files. The Cartesian input files do not because they do not refer to the grid and are valid for any block.

Output files suffixed according to the block ID can be post processed by directly using the grid to the associated block. However it is also possible to merge them into a single file whose values and indexes refer to the original grid. For this, one can run BGN with the merged files names specified in <merge> arguments of the <blockgen> command. In such a case, BGN automatically switches the <gridpart> argument to false. The file name of the merged files must coincide with the names declared in the CIF command for setting the output files e.g. “myfile”. BGN will gather files whose name is the original file name suffixed by block IDs e.g. myfile#1, myfile#2, myfile#3... The file signature is checked and duplicated into the merged file in order to indicate that it results from a parallel run.

Files that can be merged are: Result output files, Tunnel files and Track files.

Notes:

Fracture Network Overview

This section covers CIF commands used in execution of the module FRACGEN (see Fig 3-1). The FRACGEN module has four functions: 1) generation of stochastic DFNs, 2) removing isolated fractures, 3) generation of inhomogeneous “known fractures”, and 4) translating hydraulic properties from DFNs (including “known fractures”, e.g., deformation zones) onto the computational grid.

To be suitable for a FRACGEN program run, the cif.xml file must contain one <fracgen> command and one <fraccmds> command. It should normally also contain one or more <knwf> or <ranf> commands, although known and random fracture data files specified by <knwfile> and <ranfile> arguments in the <fracgen> command will have the same effect. If objects are referred to in the <ranf> commands or the random fracture data files, the cif.xml file must also contain <obj> commands specifying the objects in question. Finally, there must be a <run> command, whose <gridfile> argument shall specify the name of a file with the unstructured computational grid to be used during the run.

A FRACGEN program run consists of the following main steps. The cif.xml file is read in. Objects specified by <obj> commands are constructed. Known and random fracture data files specified by <knwfile> and <ranfile> arguments in the <fracgen> command are read in. The unstructured computational grid specified by the <gridfile> argument of the <run> command is read in. Fracture commands corresponding to arguments in the <fraccmds> command are executed in sequence, which may include reading and writing of fracture files and writing of cell property files. During the run, log information will be written to the computer screen and the file fracgen.log.

A “main random number sequence” is initialized at the beginning of each FRACGEN program run, see the <seed> argument of the <fracgen> command. Normally, whenever a random number is needed during the run, it will be taken from this sequence. There are however two exceptions to this rule, which will be explained in the next two paragraphs.

Consider the generation of random fractures corresponding to a random fracture set. (The set must have been defined by a <ranf> command or a data record in a random fracture data file, and the generation occurs when a <genran> fracture command is executed.) The random numbers necessary for the generation are normally taken from the main random number sequence. However, if a “special random number seed” has been specified for the random fracture set (see the <seed> argument of the <ranf> command and Section 5), the seed will be used to initialize a “special random number sequence”, and the random numbers will instead be taken from this sequence. A new special random number sequence will be initialized each time fractures corresponding to the set are generated. As these sequences are initialized with the same seed, they will be identical, and exactly the same fractures will be generated each time. When a special random number seed has not been specified for a random fracture set and fractures corresponding to the set are generated more than once during the same program run, the generated fractures will normally be different each time.

Nearly the same holds for an inhomogeneous known fracture. There is however a difference. Random numbers are not needed during the generation of an inhomogeneous known fracture (with the <genknw> fracture command), but they are needed when the fracture is split into pieces in connection with cell property calculation (with the <cellprop> fracture command). If a special random number seed has been specified for the fracture, it will be used to initialize a special random number sequence at cell property calculation, not at fracture generation. (To make this possible, the seed is stored with the generated fracture in the fracture list.)

One can imagine that a user could want to generate fractures corresponding to a few different sets of <knwf> commands, or a few different sets of <ranf> commands, and put the generated fractures in one fracture list for each set. Unfortunately, this is not possible. The same effect can however be obtained by making several FRACGEN program runs, storing the generated fractures in fracture files between runs. Each run should be made with a different cif.xml file. (However, this multiple-run workaround could give a problem with the main random number sequences used in the different runs not being stochastically independent.)

```
<fracgen>
 <seed>    : random number seed [0, 1]     [1-] (default = time set)
 <wkdivh>  : wkdivh1, wkdivh2, wkdivh3     [1-] (default = 1500)
 <knwfile> : known fractures file name     [0+]
 <ranfile> : random fractures file name    [0+]
```

The <fracgen> command should occur exactly once in a cif.xml file used at a FRACGEN program run. A few different kinds of input data to FRACGEN can be specified using the arguments of this command.

The optional <seed> argument is used to specify FRACGEN's ``main random number seed'', and the value should belong to the interval [0,1]. If the argument is present in the command, the seed is used to initialize FRACGEN's main random number sequence. If the argument is absent, FRACGEN uses the computer's real time clock instead to initialize the sequence (unless the clock turns out to be inaccessible to the program, in which case the program halts with an error message). The main random number sequence is available throughout the program run to provide random numbers when such are needed in various calculations. For more information about random number issues, see the Fracture Network Overview above.

When fractures are sorted with respect to isolation, the program uses a so-called binary grid to speed up the sorting. The three values wkdivh1, wkdivh2 and wkdivh3 of the optional <wkdivh> argument affect the construction of the grid. The values are specified in m, and a default value of 1,500 m is used for all three values if the argument is absent. See the <sortisol> argument of the <fraccmds> command for more information about fracture sorting with respect to isolation.

Known fractures are normally defined using <knwf> commands. They can also be defined using ``known fracture data files''. Each <knwfile> argument specifies the name of a known fracture data file. Each data record in such a file defines a known fracture using the same kinds of data as in a <knwf> command. If both <knwf> commands and <knwfile> arguments are present, all the data will be used. The format of known fracture data files is described in Chapter 5.

In the same way, random fracture sets can be defined using both <ranf> commands and ``random fracture data files'' specified with <ranfile> arguments. The format of random fracture data files is described in Chapter 5.

Notes:

- It is recommended that *wkdivh*$*$ is two to ten times the smallest fracture considered. A sensitivity study may give the optimum value. The three values refer to coordinate directions.

```
<ranf>
 <lref>       : power law reference length           [1=]
 <expid>      : power law exponent                   [1=]
 <idref>      : power law density intensity factor   [1=]
 <sizes>      : fracture length interval limits      [1=]
 <thknes>     : thickness of fractures               [1=]
 <lambda>     : lambda1, lambda2, lambda3            [1=]
 <obj>        : name of object                       [1-]
 <frevol>     : porosity of the fractures            [1=]
 <fwsurf>     : flow wetted surface                  [1=]
 <storat>     : specific storativity                 [1=]
 <tconst>     : tconsta, tconstb, tconstc, tconstd   [1=]
 <diffus>     : diffusion coefficient                [1=]
 <seed>       : random number seed                   [1-]
```

The <ranf> command may occur zero or more times in the cif.xml file. Each <ranf> command defines a random fracture set. To generate fractures corresponding to the <ranf> commands, the <genran> argument of the <fraccmds> command should be used.

The generation of random fractures corresponding to a random fracture set is done as follows. First the number of fractures, their edge lengths and the locations of their centres are calculated using a mostly random method. Then the orientation of each fracture is calculated using a random method. Finally physical properties are assigned to each fracture using a partly random method.

Geometrically, a random fracture is a rectangular parallelepiped, two of whose three edge lengths are equal. We refer to the two equal edge lengths as the “length” l of the fracture, and to the third edge length as the “thickness” b. The “centre” of the fracture is the geometrical central point of the parallelepiped, and the “axis” is the straight line through the centre which is parallel to the “thickness” edges.

All fractures in the fracture set have the same thickness b, which is specified in m by the <thknes> argument. The lengths l are restricted to the interval $[l_{min}, l_{max}]$, whose endpoints are specified in m in the <sizes> argument. The centre locations are restricted to a region of space called the “generation domain”, which will be described below. The lengths and centre locations of the fractures are calculated in a random process, where the number of fractures comes out as a by-product. The process also has the parameters l_{ref}, p and I_{ref}, which are specified by the arguments <lref>, <expid> and <idref>. l_{ref} is specified in m, p is dimensionless, and I_{ref} is specified in m^{-3}. The process has the following stochastic properties. Consider an infinitesimal length interval $[l, l + dl]$ belonging to the interval $[l_{min}, l_{max}]$. Also consider an infinitesimal volume element belonging to the generation domain, and let dV be the volume of the element. Let dN denote the number of generated fractures whose lengths belong to $[l, l + dl]$ and whose centres belong to the volume element. The expected value of dN will then be given by the following power law:

$$E[dN] = I_{ref}\left(\frac{l}{l_{ref}}\right)^{p-1}\frac{dl}{l_{ref}}dV \qquad (4\text{-}8)$$

The numbers of fractures generated for disjunct pairs of infinitesimal length intervals and volume elements are stochastically independent. (Two pairs are considered disjunct if the length intervals of the pairs are disjunct or the volume elements of the pairs are disjunct.)

From these properties it is possible to conclude that the total number N of fractures generated for the entire length interval $[l_{\min}, l_{\max}]$ and the entire generation domain will have a Poisson distribution with expected value

$$E[N] = \begin{cases} \dfrac{I_{ref}}{p}\left[\left(\dfrac{l_{\max}}{l_{ref}}\right)^p - \left(\dfrac{l_{\min}}{l_{ref}}\right)^p\right]V, & p \neq 0, \\ I_{ref}\left(\ln\dfrac{l_{\max}}{l_{\min}}\right)V, & p = 0, \end{cases} \tag{4-9}$$

where V is the volume of the generation domain.

The four numbers p, $l_{\min}$ $l_{\max}$, and l_{ref} are input parameters to the part of the FRACGEN program which does the power law calculations. The basic requirements on them are that $0 < l_{ref}$ and $0 < l_{\min} \leq l_{\max}$, while p may be any real number. However, assuming that the combination of four numbers meets the basic requirements, it is still possible that the program cannot handle the combination for numerical reasons. In this case the program will halt with an error message, otherwise it will do the power law calculations with fairly good numerical precision. The exact rules for which combinations will pass will not be given here, but l_{ref} can with advantage be chosen not too far from $l_{\min}$ and $l_{\max}$.

l_{ref} may seem to be an unnecessary constant, as the factor l_{ref}^{-p} could have been included in I_{ref}. It is however used for convenience and for numerical reasons.

When the optional <obj> argument is absent in the command, we want to model a situation where fractures are distributed over the entire three-dimensional space. However, to generate all these fractures would be both impossible, as it would require infinite amounts of computer time and memory, and unnecessary, as fractures which do not intersect any cells of the unstructured computational grid will not affect cell property calculation. (Regarding the grid, see Fracture Network Overview. Regarding cell property calculation, see the <cellprop> argument of the <fraccmds> command.) Fracture generation is therefore restricted so that only fractures whose centres belong to a finite generation domain are generated. Let the "grid enclosure" be the smallest isothetic (i.e. aligned with the coordinate axes) parallelepiped which contains all grid cells (not counting cells which were removed at grid generation). All parts of a fracture are located within the distance $\sqrt{2l_{\max}^2 + b^2}/2$ from the centre of the fracture. Let the "extended grid enclosure" be the smallest isothetic parallelepiped which contains all points which are located within this distance from

the grid enclosure. A fracture whose centre is located outside the extended grid enclosure will not intersect any grid cell. As the generation domain we therefore choose the extended grid enclosure.

When the <obj> argument is present, we want to model a situation where fractures are distributed only within the object whose name is specified in the argument. (Objects can be defined using the <obj> command.) More precisely, the centres of the fractures are required to belong to the object. It will not be described here exactly how the generation domain is chosen in this case. However, the generation domain will be a finite region of space which contains the intersection of the object and the extended grid enclosure. Generated fractures whose centres turn out to be located outside the object are discarded.

When the edge lengths and centre location of a fracture have been calculated, it is time to calculate its orientation. The orientation distribution is controlled by the vector parameter $\boldsymbol{\lambda} = (\lambda_x, \lambda_y, \lambda_z)$. The components of $\boldsymbol{\lambda}$ are dimensionless and are specified in the <lambda> argument.

When $|\boldsymbol{\lambda}| = 0$ the axis of the fracture will have no preferred orientation, but when $|\boldsymbol{\lambda}|$ is large, the axis will be nearly aligned with $\boldsymbol{\lambda}$ with high probability. More specifically, the axis is required to be parallel to a random unit vector **V**. This vector is calculated using a Fisher distribution defined by the following conditions. Unit vectors can be viewed as points on a unit sphere. Let **v** be an arbitrary unit vector, and let *dS* be the area of an infinitesimal surface element surrounding **v** on the sphere. The probability that **V** will belong to the surface element is

$$\begin{cases} \dfrac{|\boldsymbol{\lambda}|}{4\pi \sinh|\boldsymbol{\lambda}|} e^{\boldsymbol{\lambda}\cdot\mathbf{v}}\, dS, & |\boldsymbol{\lambda}| > 0, \\ \dfrac{1}{4\pi} dS, & |\boldsymbol{\lambda}| = 0. \end{cases} \tag{4-10}$$

The rotational orientation of the fracture around its axis is chosen using a uniform random distribution, so that all angles are equally probable, loosely speaking.

Finally, physical properties are assigned to the fracture. Four of the properties are specified directly by the arguments <frevol>, <fwsurf>, <storat> and <diffus>. <frevol> is the porosity in the interval [0,1], where 0 and 1 mean 0% and 100% fluid content, respectively. <fwsurf> is the flow-wetted area per unit volume in m^{-1}. <storat> is the specific storativity in m^{-1}. <diffus> is the effective diffusion coefficient in m^2 /s. The hydraulic conductivity of the fracture is calculated as the transmissivity *T* divided by the thickness *b* of the fracture. *T* is in turn calculated using the formula

$$T = \min\left[a_T \cdot 10^{d_T U} \cdot \left(\frac{l}{100\text{m}} \right)^{b_T}, c_T \right], \tag{4-11}$$

where *U* is a random number with a uniform distribution in the interval [–0.5, 0.5] and a_T, b_T, c_T and d_T are constants. The four constants are specified in the <tconst> argument. a_T and c_T are specified in m^2 /s, while b_T and d_T are dimensionless. A random fracture is always homogeneous regarding its physical properties.

The optional <seed> argument is used to specify a special random number seed for the fracture set, and the value should belong to the interval [0,1]. If the argument is present in the command, the seed is used to initialize a special random number sequence, and the random numbers necessary to generate fractures corresponding to the fracture set will be taken from this sequence. If the argument is absent, the main random number sequence will be used instead. For more information about the random number issues, see Fracture Network Overview.

Notes:

- If several random fracture sets are specified and the <seed> argument is used, it may be necessary to give different <seed> argument to different sets. If not, fractures from different sets may get the same fracture centre positions (which is normally not intended).

```
<knwf>
 <triang> : x1, y1, z1, x2, y2, z2, x3, y3, z3 [1=]
 <thknes> : thickness                          [1=]
 <frevol> : porosity                           [1=]
 <fwsurf> : flow-wetted area                   [1=]
 <storat> : specific storativity               [1=]
 <conduc> : hydraulic conductivity             [1=]
 <diffus> : effective diffusion coefficient    [1=]
 <inhom>  : true for an inhomogeneous fracture [1-] (default = F)
 <vkir>   : u, v, w                            [1-]
 <lewir>  : length, width                      [1-]
 <rcorr>  : correlation length                 [1-]
 <fsdev>  : <frevol> standard deviation        [1-]
 <fwsdev> : <fwsurf> standard deviation        [1-]
 <ssdev>  : <storat> standard deviation        [1-]
 <csdev>  : <conduc> standard deviation        [1-]
 <dsdev>  : <diffus> standard deviation        [1-]
 <seed>   : special random number seed         [1-]
```

This command allows generating stochastic, inhomogeneous properties of fractures with known (i.e., deterministic) geometry.

The <knwf> command may occur zero or more times in the cif.xml file. Each <knwf> command defines a known fracture. To generate fractures defined by the <knwf> commands, the <genknw> argument of the <fraccmds> command should be used.

The geometrical properties of the fracture are specified using a triangle and a thickness. A point in space is regarded to belong to the fracture if its distance to the plane of the triangle is less than half the thickness and if its orthogonal projection onto the plane belongs to the triangle. The coordinates of the three vertices of the triangle (x_1, y_1, z_1), (x_2, y_2, z_2) and (x_3, y_3, z_3) are specified in m using the nine values of the <triang> argument. The thickness is specified in m using the value of the <thknes> argument.

Each of the five arguments <frevol>, <fwsurf>, <storat>, <conduc> and <diffus> has one value, which specifies a physical property of the fracture. <frevol> is the porosity in the interval [0,1], where 0 and 1 mean 0% and 100% fluid content, respectively. <fwsurf> is the flow-wetted area per unit volume in m^{-1}. <storat> is the specific storativity in m^{-1}. <conduc> is the hydraulic conductivity in m/s. <diffus> is the effective diffusion coefficient in m^2 /s.

A fracture can be homogeneous or inhomogeneous regarding its physical properties. This is specified with the optional argument <inhom>, whose value can be F (default) for a homogeneous or T for an inhomogeneous fracture. For an inhomogeneous fracture the arguments <vkir>, <lewir>, <rcorr>, <fsdev>, <fwsdev>, <ssdev>, <csdev> and <dsdev> are mandatory and <seed> is optional, while for a homogeneous fracture these arguments should all be absent.

For a homogeneous fracture, the physical properties are homogeneous within the fracture, and they are specified directly by the arguments <frevol>, <fwsurf>, <storat>, <conduc> and <diffus>. A homogeneous fracture is not random in any way.

An inhomogeneous fracture is split into a number of pieces. The physical properties are homogeneous within each piece but vary between pieces.

The splitting into pieces is controlled by the vector whose coordinates (u, v, w) are specified in the <vkir> argument, and by the length and width specified in m in the <lewir> argument. A rectangular grid is drawn in the plane of the triangle. The rectangles of the grid are congruent, all having the same length and the same width as specified by the <lewir> argument. The grid is oriented so that the "length" sides of the rectangles are parallel to the orthogonal projection of the vector (u, v, w) onto the plane. The translational location of the grid in the plane is chosen using a two-dimensional uniform random distribution, giving all locations equal probability, loosely speaking. The grid splits the triangle into pieces. The fracture is split into corresponding pieces, so that if the fracture is projected orthogonally onto the plane of the triangle, each fracture piece will be mapped onto the corresponding triangle piece.

Let us number the rectangles of the grid with pairs of integer indices (r, k). r is the index in the "length" direction and k the index in the "width" direction. Let us number the fracture pieces correspondingly, so that each piece has the same index pair as the corresponding rectangle. To each fracture piece (r, k) a random number H_{rk} is assigned. The H_{rk} have a multivariate normal distribution which approximately satisfies the following conditions:

$$\begin{aligned} E\left[H_{rk}\right] &= 0 \\ Var\left(H_{rk}\right) &= 1 \\ Cov\left(H_{r_1k_1}, H_{r_2k_2}\right) &= \exp\left(-\frac{\left(r_2-r_1\right)^2+\left(k_2-k_1\right)^2}{2a^2}\right) \end{aligned} \tag{4-12}$$

a is a correlation length, which controls at how long distances the random numbers H_{rk} are correlated. For more information about the multivariate normal distribution, see Appendix C in Report 1. a (dimensionless) is specified by the argument <rcorr>, and it is required to belong to the interval [0.01,100] for numerical reasons. a should be multiplied by the rectangle length and width, respectively, to obtain the correlation lengths in m in the two directions.

The porosity of a fracture piece is calculated as the product of the <frevol> value and the exponential of the product of the <fsdev> value and the random number H_{rk} of the piece. The porosity will thus have a lognormal distribution, and the standard deviation of the natural logarithm of the porosity will be given by the <fsdev> value. The four other physical properties are calculated in the same way, the standard deviations being specified by the <fwsdev>, <ssdev>, <csdev> and <dsdev> arguments. The same random number is used for all five properties within each piece.

The <seed> argument is used to specify a special random number seed for the inhomogeneous fracture, and the value should belong to the interval [0,1]. If the argument is present in the command, the seed is used to initialize a special random number sequence, and the random numbers necessary to split the fracture into pieces and calculate their properties will be taken from this sequence. If the argument is absent, the main random number sequence will be used instead. For more information about the random number issues, see Fracture Network Overview.

Notes:

```
<fraccmds>
 <clear>     : lf                          [0+]
 <move>      : lffrom, lfto                [0+]
 <assign>    : lffrom, lfto                [0+]
 <readfile>  : filef, lf                   [0+]
 <writefile> : lf, filef                   [0+]
 <genknw>    : lf                          [0+]
 <genran>    : lf                          [0+]
 <sortisol>  : lffix, lfind, lfsav, lfdsc [0+]
 <cellprop>  : lf, filesuffix              [0+]
```

The <fraccmds> command should occur exactly once in a cif.xml file used at a FRACGEN program run.

Starting with DarcyTools version 3.0, FRACGEN has a new structure which makes it more flexible to use. With the new <fraccmds> command the user can (and must) now control which computational steps shall be taken and in which order. Each argument of the <fraccmds> command is a ‘’fracture command’’ which triggers the execution of a computational step. (Note that there are two different kinds of ‘’commands’’ here, cif.xml ‘’commands’’ and “fracture commands”.) The values specified with the argument are the ‘’parameters’’ of the fracture command. The fracture commands are executed in the order in which they occur as arguments in the <fraccmds> command. (This makes the <fraccmds> command different from many other commands, where the order among the arguments is not important.) The <fraccmds> command with its arguments can be seen as a computer program written in a very rudimentary and special-purpose programming language.

All the fracture commands operate on “fracture lists”. For example, the <genran> fracture command generates random fractures, and they are stored in a fracture list. Each fracture list has a name. If a fracture command operates on a fracture list whose name has not occurred in previous fracture commands, a new empty fracture list with this name is immediately created. The same fracture list may not occur as a parameter more than once in the same fracture command (e.g., “<move> keptfracs keptfracs < / move>” is not permitted). For each fracture in a fracture list, information is stored about its geometrical and physical properties.

Fracture lists can also be written to and read from “fracture files”. The format of fracture files is described in Section 5.

Here follow descriptions of all the available fracture commands. Fracture command parameters which begin with “lf” here are names of fracture lists, and parameters called “filef” are fracture file names.

<clear> lf </clear>: Clear fracture list lf, i.e. delete the fractures in the list, leaving it empty.

<move> lffrom lfto </move>: Move the fractures in lffrom to lfto. Afterwards, lffrom will be empty, and lfto will contain the fractures which were originally in lfto followed by those originally in lffrom.

<assign> lffrom lfto </assign>: First clear lfto. Then copy all fractures in lffrom to lfto, leaving lffrom unchanged.

<readfile> filef lf </readfile>: First clear lf. Then read the fractures from fracture file filef and store them in lf.

<writefile> lf filef </writefile>: Write the fractures in lf to filef, leaving lf unchanged.

<genknw> lf </genknw>: First clear lf. Then generate known fractures and store them in lf. A fracture is generated for each occurring <knwf> command. A fracture is also generated for each data record in each known fracture data file specified by a <knwfile> argument in the <fracgen> command. The work done by <genknw> to generate an inhomogeneous known fracture does not include splitting the fracture into pieces. (The pieces are not needed until cell properties are to be calculated, and the splitting is therefore done by the <cellprop> fracture command.) However, information about how to do the splitting is stored with the fracture in the fracture list. (See Fracture Network Overview about random number issues.)

<genran> lf </genran>: First clear lf. Then generate random fractures and store them in lf. Fractures are generated for each occurring <ranf> command. Fractures are also generated for each data record in each random fracture data file specified by a <ranfile> argument in the <fracgen> command. (See Fracture Network Overview about random number issues.)

<sortisol> lffix lfind lfsav lfdsc </sortisol>: First clear lfsav and lfdsc. Then sort the fractures in lfind with respect to isolation, moving the isolated fractures to lfdsc and the others to lfsav. Afterwards lffix will be unchanged and lfind empty. For details about the sorting, see below.

<cellprop> lf filesuffix </cellprop>: Calculate physical properties of the grid cells and staggered cells in the unstructured computational grid, corresponding to the physical properties of the fractures in lf, and write the calculated properties to nine ‘’cell property files’’ whose names will end in filesuffix. Each inhomogeneous known fracture in lf will first be split into pieces. lf will be unchanged afterwards. For details, see below.

The following three paragraphs give details about fracture sorting.

The fracture sorting is done using the following three-phase method. During the first phase, each fracture in lfind (indeterminate fractures) is examined with regard to its location in relation to the grid enclosure. (The grid enclosure is defined in the section about the <ranf> command.) If the fracture is located entirely outside the grid enclosure, it is moved to lfdsc (discarded fractures). If it intersects the boundary of the grid enclosure, it is moved to lfsav (saved fractures). If it is located entirely inside the grid enclosure, it remains in lfind. During the second phase, each fracture in lffix (fixed fractures) and lfsav, including any which have been moved from lfind to lfsav earlier during the phase, is examined. If the fracture intersects any of the fractures in lfind, these are moved to lfsav. During the third phase, all fractures remaining in lfind are moved to lfdsc (as they are regarded as isolated).

Intersection detection is not always exact with the algorithms used. Fractures entirely outside the grid enclosure can sometimes be classified as intersecting the boundary of the grid enclosure. Two non-intersecting fractures can sometimes be classified as intersecting. Also consider the special role

played by the grid enclosure and its boundary during the first phase. It might have been a more logical choice to use the union of the grid cells and its boundary instead.

To speed up the second phase, the program organizes the fractures that remain in lfind after the first phase in a “binary grid” (which is separate from the unstructured computational grid). (The following description of the binary grid is simplified and incomplete.) The binary grid is a subdivision of the grid enclosure (which is admittedly related to the unstructured computational grid) into “sections”, which are isothetic parallelepipeds, all congruent to each other. The number of sections in the x direction is the smallest power of 2 such that the edge length of the sections in the x direction does not exceed the constant wxkdivh (see the <wkdivh> argument of the <fracgen> command). The corresponding holds for the y and z directions, with constants wykdivh and wzkdivh. The choice of the three constants will not affect which fractures in lfind are moved to lfsav during the second phase, but it will affect the efficiency of the phase. The user is encouraged to try out different constant values to minimize execution time. One guideline, which might have exceptions, is that it is probably not a good idea to choose so small values that the sections will be smaller than the smallest fractures in lfind or more numerous than the fractures in lfind.

The following seven paragraphs give details about cell property calculation.

As was said above, each inhomogeneous known fracture in fracture list lf is first split into pieces. The fracture list is thus transformed into a set of ‘’conductivity regions’’, each of which is either a piece of an inhomogeneous known fracture or an entire homogeneous known fracture or random fracture. The set of conductivity regions has the advantage that each conductivity region is homogeneous regarding its physical properties. (For random number issues in connection with splitting inhomogeneous known fractures into pieces, see Fracture Network Overview).

The unstructured computational grid has a number of grid cells and a number of faces. Each grid cell is an isothetic parallelepiped. Each face is an isothetic rectangle which is perpendicular to one of the coordinate axes. A face can therefore be classified as an *x* face, a *y* face or a *z* face.

A grid cell and a face can be associated, which means that the face is part of the cell's surface. A face can be associated with up to two grid cells, e.g. an *x* face may be associated with one cell located on its low *x* side and another on its high *x* side. As a big cell may be adjacent to a number of smaller cells, a grid cell may be associated with more than six faces.

For each face in the grid there is a so-called staggered cell, which is an isothetic parallelepiped composed of up to two parts, one part for each grid cell associated with the face. Each part is itself an isothetic parallelepiped. In the coordinate direction perpendicular to the face, the part extends from the face and halfway to the opposite side of the grid cell. In the two other coordinate directions, the part has the same extensions as the face.

A grid cell has three different physical properties, namely the porosity, the flow-wetted area and the specific storativity. The porosity of a grid cell is calculated as the sum of contributions from those conductivity regions which intersect the cell. The contribution to the porosity of a grid cell from an intersecting conductivity region is equal to the porosity of the conductivity region multiplied by the volume of the intersection. The flow-wetted area and the specific storativity of the grid cell are calculated analogously. Assuming that ".dat" has been chosen as the file name suffix, the porosities, flow-wetted areas and specific storativities of all grid cells will be written to the cell property files frevol.dat, fwsurf.dat and storat.dat, respectively.

A staggered cell has two different physical properties, namely the hydraulic conductivity and the effective diffusion coefficient. The hydraulic conductivity of a staggered cell is calculated as the sum of contributions from those conductivity regions which intersect the staggered cell. The contribution to the hydraulic conductivity of a staggered cell from an intersecting conductivity region is equal to the hydraulic conductivity of the conductivity region multiplied by the volume of the intersection. The effective diffusion coefficient of the staggered cell is calculated analogously. Before being written to the cell property files, the physical properties of each staggered cell are divided by the volume of the staggered cell. (The physical properties of the grid cells are *not* divided in this way.) Assuming that ‘’.dat’’ has been chosen as the file name suffix, the hydraulic conductivities and the effective diffusion coefficients of the staggered cells of all the *x* faces will be written to the cell property files condux.dat and diffux.dat, respectively. Analogously, the corresponding properties of the staggered cells of the *y* and *z* faces will be written to the files conduy.dat, diffuy.dat, conduz.dat and diffuz.dat.

The format of cell property files is described in Section 5.

Finally, here is an example of a typical and simple <fraccmds> section which could occur in a cif.xml file:

```
<fraccmds>
 <genknw> keptfracs </genknw>
 <genran> randomfracs </genran>
 <sortisol> keptfracs randomfracs savedfracs discardedfracs </sortisol>
 <writefile> savedfracs savedfracs.dat </writefile>
 <move> savedfracs keptfracs </move>
 <cellprop> keptfracs .dat </cellprop>
</fraccmds>
```

Notes:

- See Chapter 5 for file format.
- In commands <knwf> and <ranf> the product of <thknes> and <frevol> gives the aperture.

```
<obj>
 <name>       : reference name                          [1=]
 <file>       : file name                               [1-]
 <refine>     : refinement parameter      (>0.5)        [1-]
 <brick>      : xmin, xmax, ymin, ymax, zmin, zmax      [1-]
 <cylinder>   : x1, y1, z1, x2, y2, z2, r, nbs          [1-]
 <sphere>     : x1, y1, z1, r, nbs                      [1-]
 <plate>      : x1, y1, z1, x2, y2, z2, x3, y3, z3      [1-]
 <line>       : x1, y1, z1, x2, y2, z2                  [1-]
 <spot>       : x1, y1, z1                              [1-]
 <user>       : eltsize, nbelts, nbnodes                [1-]
 <move>       : dx, dy, dz                              [1-]
 <dilat>      : dilx, dily, dilz                        [1-]
 <rot>        : degx, degy, degz                        [1-]
```

This command defines the objects for grid generation, fracture generation and outputs. The name of the object is specified with <name> and must be unique among object names. When the argument <file> is used, the object definition is read from the given file. Otherwise, the object type must be specified by one of the <brick>, <cylinder>, <sphere>, <plate>, <line>, <spot> or <user> arguments. <brick> builds a parallelepiped object aligned with the domain axes. <cylinder> builds a cylinder defined by its axes from point 1 to point 2, by its radius r and a number of sectors per half perimeter nbs. <sphere> builds a sphere centered on point 1 with a radius r and a number of sectors per half perimeter nbs. <plate> builds a plate whose parallelogram is defined by three points and such that the forth corner is given by 24 parallel with 13. <line> builds a line segment from point 1 to point 2 and <spot> a point object. Finally, <user> allocates a dummy object whose node coordinates and connectivity must be specified by programming the function USROBJ in the FIF file. In this case, the number of nodes per element, the dimensions of the elements and nodes arrays must be specified.

Predefined objects or object read in from files may be modified by arguments <move>, <rot> or <dilat>. The rotations are anti-clockwise looking along the direction specified by the DEGX, DEGY and DEGZ degree angles. The center of rotation is the minimum corner of the bounding box of the object. The rotations are cumulative so that DEGY is applied to the z-rotated object and DEGX to the yz-rotated object. The rotation is applied after the call to USROBJ, after the dilatations and before moving the object. When an object is used by the solver, it may be refined for output precision. In that case, the argument <refine> specifies how large the object elements should be compared to the size of the grid cells they intersect. The type (ASCII, binary) of the object files is automatically detected during the reading process.

Notes:

- See Chapter 5 for file format.

```
<loc>
 <name>       : reference name                [1=]
 <marker>     : marker type and value         [1-] (default='cell',-1)
 <patch>      : xmin, xmax, ..., zmax         [1-] (default=0,0,0,0,0,0)
 <xplane>     : x                             [1-]
 <yplane>     : y                             [1-]
 <zplane>     : z                             [1-]
 <xplate>     : x, ymin, ymax, zmin, zmax     [1-]
 <yplate>     : y, xmin, xmax, zmin, zmax     [1-]
 <zplate>     : z, xmin, xmax, ymin, ymax     [1-]
 <xline>      : y, z                          [1-]
 <yline>      : x, z                          [1-]
 <zline>      : x, y                          [1-]
 <xseg>       : y, z, xmin, xmax              [1-]
 <yseg>       : x, z, ymin, ymax              [1-]
 <zseg>       : x, y, zmin, zmax              [1-]
 <spot>       : x, y, z                       [1-]
 <dplane>     : depth                         [1-]
 <dplate>     : d, xmin, xmax, ymin, ymax     [1-]
 <file>       : filename                      [1-]
 <not>        : locname                       [0+]
```

This command defines a location, i.e. a collection of all cells of the domain sharing the same property. After giving a unique name among loc names, users must define one of the two possible properties. The <patch> property gathers all cells intersecting with a rectangular box given by its minimum and maximum coordinates. Patches may be parallelepipeds, planes, plates, lines, segments or spots according on the min and max values set in each direction. The <xplane>...<spot> arguments are facilities for setting patches. Depending on its type: 'cell', 'east,' 'west', 'north', 'south', 'high' or 'low', the <marker> property gathers cells having the given marker value, or having an east, west, north, south, high, low neighbouring cell with the given marker value. At boundaries the east...low setting specifies the marker value of the removed zones according to the grid generator conventions. The <dplane> and <dplate> arguments specify locations similar to <zplane> and <zplate> but concern the depth instead of the z-coordinate. These locations gather cells intersecting a surface located at 'depth' under the ground surface. Finally, when the <file> argument is set, the location is built from the indexes read in the specified file.

DarcyTools predefined location names are:

domain, sea, land, tunwall, east, west, north, south, high, low, TSxxx, TSGxxx.

The sea and land locations are automatically computed during the initialization process. All cells of the top of the domain contribute to the sea/land definition. By default the land cells have a higher z-face with a strictly positive altitude, but the indexes of the top cells are also provided to users (USRLAND) for having the final decision about their land or sea belonging. The east...low locations are east...low marker type locations associated to the null marker value. They usually represent the domain boundaries. The tunwall location contains any tunnel section cell intersecting a tunnel border else than a tunnel section. The TSxxx locations gather the cells of the tunnel sections having xxx as marker value. The TSGxxx locations gather cells of the layers affected by grouting.

A location can be defined as being the complement in space of one or several locations. For this users may use the parameter <not>. Then the new location gathers all cells not being in the specified locations.

Notes:

- See Chapter 5 for file format.
- The two commands <loc> and <obj> can sometimes be used for the same purpose. However, the basic distinction is that <obj> is a geometry, while <loc> is a collection of cells.

```
<ogn>
 <stl>     : stl name                          [1+]
 <outfile> : object output file name           [1-]
 <binary>  : output file format                [1-] (default=T)
 <fusion>  : tolerance for fusing vertexes     [1-] (default=0.0)
 <tecplot> : tecplot output file name          [1-]
 <obj>     : input obj name                    [1-]
 <scale>   : roughness scale                   [1-] (default=0.0)
 <hurst>   : fractal Hurst exponent            [1-] (default=0.9)
 <level>   : refinement level                  [1-] (default=0)
 <seed>    : random generator seed             [1-] (default=0.0)
 <inidev>  : initial random deviation          [1-] (default=.false.)
```

This command concerns only the program OBJGEN. The program reads STL files (Stereolithographic CAD format) and translates them into an object file (DarcyTools format) ready for the grid generation process. The STL files and their coordinate transformation are separately defined by command <stl>. If an output file is specified, a single object is combined from all the facets of the STL file (after coordinate transformation, see command <STL>, below) and exported. When omitted only the statistics of each STL file after coordinate transformation is printed. To satisfy the objects convention, the OBJGEN program fuses multi-defined vertexes (e.g. vertexes belonging to several triangles) as soon as their three coordinates are such that |v1-v2|≤fusion. Finally when specified, the Tecplot output file automatically contains the object ‘stl-object’ after its coordinate transformation.

The program OBJGEN can also produce rough objects by refining the object given by argument <obj> or generated from the <stl> file. The object roughness is generated by a self affine procedure during successive refinement. The nodes of the object are moved in the direction normal to the surface according to the specified <scale> and <hurst> exponent. When needed the nodes of the initial object can also be deviated before refining when <inidev> is set true.

Notes:

- When several object files are to be generated they must be generated one at a time, as only one <ogn>… </ogn> command is considered.

```
<stl>
 <name>    : reference name                    [1=]
 <file>    : input file name                   [1-] (default='name'.stl)
 <pos>     : dx, dy, dz displacements          [1-] (default=0,0,0)
 <rot>     : degx, degy, degz rotation angles  [1-] (default=0,0,0)
```

This command defines a STL file (Stereolithographic CAD format) and its coordinate transformation for the program OBJGEN. The rotation <rot> is a rotation around the minimum point of the bounding STL object. The angles of rotations degx, degy and degz are anticlockwise in degrees. They are applied around the fixed x-axis, y-axis and z-axis in this order. A translation of the object is applied according to the dx, dy and dz displacements.

The name of the STL file is preformatted by the reference name followed by the '.stl' extension. The format (binary or ASCII) of STL files is automatically detected by the reading process.

Notes:

```
<dgn>
 <obj>      : object name                          [1+]
 <binary>   : output files format                  [1-] (default=T)
 <method>   : distance computational method        [1-] (default=1)
 <distfile> : output var file name                 [1-]
 <locfile>  : dmin, dmax, output loc file name     [1-]
 <objfile>  : dmin, dmax, output obj file name     [1-]
 <tecplot>  : tecplot name                         [1-]
 <nbsteps>  : number of iterations for method 2    [1-] (default =5)
```

The <dgn> command concerns only the program DISTGEN. This program reads object files and computes the distance to objects. The resulting distance may be stored in a variable file and/or may be used to create locations of cells having their distance greater or equal to dmin and strictly lower than dmax. The created locations may be outputted as location files or object files. In the latter case the object is built from the boundary faces of the location.

The <obj> argument specifies the object names elsewhere defined by an <obj> command. One or several objects may be used by the program DISTGEN. When several objects are involved, the distance refers to the nearest object. The argument <binary> specifies whether the output files will be formatted (false) or not (true).

The <method> argument specifies the method that DISTGEN will use for computing the distance: 1 means the exact brute force method where the distance is exact for each cell centre, 2 means the Poisson method where the distance is approximated using the resolution of a Poisson equation and 3 means the transient method where the distance is approximated by solving a transient heat equation and by tracking the temperature front progression. Methods 2 and 3 use MIGAL for solving the partial differential equation they involve and are of no interest when a poor convergence make them more CPU consuming than method 1. For setting the MIGAL parameters DISTGEN uses an equation set <eqs> named ‘dist’.

The parameter <distfile> specifies the name of a variable file for saving the distance when needed. Then this distance will be accessible from the solver by using the <var><inifile> command. The parameter <locfile> specifies the name of a location file for saving the indexes of the cells whose distance value is such as dmin≤d<dmax. The parameter <objfile> specifies the name of an object file for saving the object made of the boundary of the location containing cells such as dmin≤d<dmax.

The argument <tecplot> specifies a Tecplot name elsewhere defined by a command <tecplot>. The program DISTGEN produces a variable named ‘distance’ as well as locations named ‘dist_loc*’ and objects named ‘dist_obj*’. The character ‘*’ represents the id of the associated locfile and objfile starting from 1 and increasing in the order of <locfile> and <objfile> declarations. The names ‘distance’, ‘dist_loc*’ and ‘dist_obj’ can be used in theTecplot associated command for respectively referring a variable, a location and an object.

Notes:

5 File Formats

Beside Tecplot files whose format has been fixed to the Tecplot binary format, DarcyTools involves several input-output files such as the grid, the object or the variable files. No specific extension is used for those files. They may be ASCII or binary files.

Known fracture data files

A known fracture data file is an unformatted sequential file, and it consists of a header record followed by a number of data records.

The header record is written by the following statement:

```
write (iunit) '#KF#V310'
```

Each data record defines a known fracture, with the geometry of a triangle and with properties. A known fracture can also be defined using a <knwf> command in a CIF file, but the same kinds of data are used in both cases. A data record is written by the following statement:

```
 write (iunit) x1, y1, z1, x2, y2, z2, x3, y3, z3, thknes,
&frevol, fwsurf, storat, conduc, diffus,
&inhom,
&u, v, w, length, width, rcorr,
&fsdev, fwsdev, ssdev, csdev, dsdev, seed
```

All variables written are real except inhom, which is logical. The first 15 real variables and inhom must always have valid values. When inhom is false, the fracture is homogeneous, and when inhom is true, it is inhomogeneous. For a homogeneous fracture the last 12 real variables do not apply, but they must nevertheless have defined values when the record is written. What values are chosen is not important, e.g. all 12 variables can be set to 0.0. For an inhomogeneous fracture the last 12 real variables must have valid values. seed shall then either be a special random number seed in the interval [0,1], or –1.0 indicating that no special random number seed is given.

Random fracture data files

A random fracture data file is an unformatted sequential file, and it consists of a header record followed by a number of data records.

The header record is written by the following statement:

```
write (iunit) '#RF#V310'
```

Each data record defines a random fracture set. A random fracture set can also be defined using a <ranf> command in a CIF file, but the same kinds of data are used in both cases. A data record is written by the following statement:

```
 write (iunit) lref, expid, idref, lmin, lmax, thknes,
&lambdax, lambday, lambdaz, obj,
&frevol, fwsurf, storat, at, bt, ct, dt, diffus, seed
```

All variables written are real except obj, which is character(32). All variables must have valid values. If an object is used, obj shall be the name of the object padded by trailing blanks, otherwise obj shall be all blanks. seed shall either be a special random number seed in the interval [0,1], or –1.0 indicating that no special random number seed is given.

Generated fracture files

A generated fracture file is an unformatted sequential file, and it consists of a header record followed by a number of data records.

The header record is written by the following statement:

```
write (iunit) '#FR#V310'
```

Each data record specifies a known or random fracture. A data record for a homogeneous known fracture or a random fracture is written by the following statement (specifying a parallelepiped with 8 nodes and properties):

```
 write (iunit) ns, nv, .false., iset,
&(vxks(is), vyks(is), vzks(is), sps(is), is = 1, ns),
&(pxkv(iv), pykv(iv), pzkv(iv), iv = 1, nv),
&qxkl, qxkh, qykl, qykh, qzkl, qzkh,
&frevol, fwsurf, storat, conduc, diffus
```

A data record for an inhomogeneous known fracture (see command <knwf>) is written by the following statement:

```
 write (iunit) ns, nv, .true., iset,
&(vxks(is), vyks(is), vzks(is), sps(is), is = 1, ns),
&(pxkv(iv), pykv(iv), pzkv(iv), iv = 1, nv),
&qxkl, qxkh, qykl, qykh, qzkl, qzkh,
&def1, def2, def3, def4, def5,
&x1, y1, z1, x2, y2, z2, x3, y3, z3, thknes,
&frevol, fwsurf, storat, conduc, diffus,
&.true.,
&u, v, w, length, width, rcorr,
&fsdev, fwsdev, ssdev, csdev, dsdev, seed
```

All values written are real, except the integer values ns, nv and iset, and the values explicitly specified as .false. or .true..

The first three values written, i.e. ns, nv and the .false. or .true. value, specify the layout of the remainder of the record. The third value indicates whether the record is of the first or second kind.

iset indicates which ``fracture set'' the fracture belongs to. The present version of the FRACGEN program numbers random fracture sets 1, 2, 3, ..., and when it generates random fractures corresponding to a random fracture set, each fracture's iset value is set to the number of the random fracture set. When the program generates a known fracture, its iset value is set to 0. The iset values are however never used by the present version of the program, except that they occur in the log output during random fracture generation and that they are written to and read from fracture files. In any case, iset must be defined when a data record is written.

For each is = 1, 2, ..., ns, the four values vxks(is), vyks(is), vzks(is) and sps(is) specify a directed plane. A point (pxk, pyk, pzk) is regarded to be located on the inside of the plane if vxks(is) pxk + vyks(is) pyk + vzks(is) pzk < sps(is). Geometrically, the fracture is the region of space which is located on the insides of all the ns planes. (It could be reasonable to think that there should be as

many directed planes as faces of a fracture. However, when the FRACGEN program generates a fracture, it may also include redundant directed planes for performance reasons.)

(pxkv(iv), pykv(iv), pzkv(iv)), iv = 1, 2, ..., nv, are the vertices of the fracture.

qxkl, qxkh, qykl, qykh, qzkl and qzkh specify the smallest isothetic parallelepiped which contains the fracture. A point (pxk, pyk, pzk) belongs to the parallelepiped if qxkl < pxk < qxkh, qykl < pyk < qykh, and qzkl < pzk < qzkh.

In a record of the first kind, frevol, fwsurf, storat, conduc and diffus are the porosity, flow-wetted area, specific storativity, hydraulic conductivity and effective diffusion coefficient of the fracture, respectively. In a record of the second kind, def1, def2, def3, def4 and def5 are arbitrary defined values which will not be used. The reason for the difference is that a record of the second kind specifies an inhomogeneous known fracture, whose physical properties vary over the fracture.

The remaining values in a record of the second kind are the known fracture data from which the inhomogeneous known fracture was generated. If the data were read in from a record in a known fracture data file, the last part of the record in the fracture file is identical to the record in the known fracture data file. If the data were instead read in from a <knwf> command in a CIF file, the last part of the record in the fracture file still has the same format as a record in a known fracture data file.

Cell property files

A cell property file is an unformatted sequential file, and it consists of a number of data records. (There is no header record.) Each data record has one real value, which specifies a physical property of a grid cell or staggered cell in the unstructured computational grid. When the <cellprop> fracture command is executed by the FRACGEN program (see the <cellprop> argument of the <fraccmds> command), nine cell property files are written. Assuming that ".dat" has been chosen as the file name suffix, the files will be frevol.dat, fwsurf.dat, storat.dat, condux.dat, diffux.dat, conduy.dat, diffuy.dat, conduz.dat and diffuz.dat.

Each of the files frevol.dat, fwsurf.dat and storat.dat is used to store one physical property of all grid cells. Each file therefore has one record per grid cell. The values are written in grid cell order.

Each of the files condux.dat and diffux.dat is used to store one physical property of the staggered cells of all *x* faces in the grid. Each file therefore has one record per *x* face. The values are written in *x* face order.

The corresponding holds for conduy.dat and diffuy.dat and the *y* faces, and for conduz.dat and diffuz.dat and the *z* faces.

Grid files

Grid files always start with the #GD#V304 file identifier. The first four characters indicate an unstructured Cartesian grid file while the number following the V character specifies the format version. The grid data follow this record so that ASCII files are written by:

```
  write(iunit,*) '#GD#V304'
  write(iunit,*) nbcells,     nbxfaces,
&                nbyfaces,    nbzfaces,
&                nbcgtnodes, nbmark,
&                nbcutcells,  nbxcutfaces,
&                nbycutfaces, nbzcutfaces,
&                nbpolcells,  nbxpolfaces,
&                nbypolfaces, nbzpolfaces,
&                nbtpolfaces, nbpolys,
&                nbpts
  write(iunit,*) lvmax
  write(iunit,*) dxmin, dymin, dzmin
  write(iunit,*) xor, yor, zor
  do i=1,nbcells+nbcutcells
   write(iunit,*) cells(i)%ix, cells(i)%iy, cells(i)%iz,
 &                cells(i)%lx, cells(i)%ly, cells(i)%lz,
 &                cells(i)%mk
  enddo
  do i=1,nbxfaces+nbxcutfaces
   write(iunit,*) xfaces(i)%high, xfaces(i)%low
  enddo
  do i=1,nbyfaces+nbycutfaces
   write(iunit,*) yfaces(i)%high, yfaces(i)%low
  enddo
  do i=1,nbzfaces+nbzcutfaces
   write(iunit,*) zfaces(i)%high, zfaces(i)%low
  enddo
  do i=1,nbcgtnodes
   write(iunit,*) cgt%nodes(i)%right, cgt%nodes(i)%left,
 &                cgt%nodes(i)%cut
  enddo
  do i=1,nbmark
   write(iunit,*) cells(i)%mk
  enddo
```

and binary (unformatted) files by:

```
      write(iunit) '#GD#V304'
      write(iunit) nbcells,     nbxfaces,
     &               nbyfaces,    nbzfaces,
     &             nbcgtnodes, nbmark,
     &             nbcutcells,  nbxcutfaces,
     &             nbycutfaces, nbzcutfaces,
     &             nbpolcells,  nbxpolfaces,
     &             nbypolfaces, nbzpolfaces,
     &             nbtpolfaces, nbpolys,
     &             nbpts
      write(iunit) lvmax
      write(iunit) dxmin, dymin, dzmin
      write(iunit) xor, yor, zor
      do i=1,nbcells+nbcutcells
       write(iunit) cells(i)%ix, cells(i)%iy, cells(i)%iz,
     &              cells(i)%lx, cells(i)%ly, cells(i)%lz,
     &              cells(i)%mk
      enddo
      do i=1,nbxfaces+nbxcutfaces
       write(iunit) xfaces(i)%high, xfaces(i)%low
      enddo
      do i=1,nbyfaces+nbycutfaces
       write(iunit) yfaces(i)%high, yfaces(i)%low
      enddo
      do i=1,nbzfaces+nbzcutfaces
       write(iunit) zfaces(i)%high, zfaces(i)%low
      enddo
      do i=1,nbcgtnodes
       write(iunit) cgt%nodes(i)%right, cgt%nodes(i)%left,
     &               cgt%nodes(i)%cut
      enddo
      do i=1,nbmark
       write(iunit) cells(i)%mk
      enddo
```

The parameters of the grid file are not defined or explained in the present context. The reason for this is that a user does not need to read, write or modify this file.

Object files

Object files always start with the #OB#V301 file identifier. The first four characters indicate an object file while the number following the V character specifies the format version. The objects data follows this record so that ASCII files must be written with:

```
      write(iunit,*) '#OB#V301'
      write(iunit,*) nsz, nbn, nbe
      write(iunit,*) ((nodes(i,j),i=1,3  ),j=1,nbn)
      write(iunit,*) ((elts (i,j),i=1,nsz),j=1,nbe)
```

and binary (unformatted) files with:

```
      write(iunit) '#OB#V301'
      write(iunit) nsz, nbn, nbe
      write(iunit) ((nodes(i,j),i=1,3  ),j=1,nbn)
      write(iunit) ((elts (i,j),i=1,nsz),j=1,nbe)
```

Where ‘nsz’, ‘nbn’ and ‘nbe’ are three 4 bytes integers representing respectively the number of nodes per elements, the number of nodes and the number of elements. Nodes is a 4 bytes real array with nodes(1,i), nodes(2,i) and nodes(3,i) representing respectively the x, y and z coordinates of the node (i). Finally, ‘elts’ is a 4 bytes integer array containing the indices of the nodes forming each element. Since elements can only be points, segments or triangles, ‘nsz’ must be 1, 2 or 3.

Variable files

Variable files always start with the #V5#V300 file identifier. The first four characters indicate a variable file while the number following the V character specifies the format version. The variable data follow this record so that ASCII files must be written with:

```
write(iunit,*) '#V5#V300'
write(iunit,*) name, varpos
write(iunit,*) nbvals, nbsteps, nbindex
write(iunit,*) (val(i),i=1,nbvals)
if(nbsteps.ge.2) write(iunit,*) (vm1(i),i=1,nbvals)
if(nbsteps.ge.3) write(iunit,*) (vm2(i),i=1,nbvals)
if(nbindex.gt.0) write(iunit,*) (index(i),i=1,nbindex)
```

and binary (unformatted) files with:

```
write(iunit) '#V5#V300'
write(iunit) name, varpos
write(iunit) nbvals, nbsteps, nbindex
write(iunit) (val(i),i=1,nbvals)
if(nbsteps.ge.2) write(iunit) (vm1(i),i=1,nbvals)
if(nbsteps.ge.3) write(iunit) (vm2(i),i=1,nbvals)
if(nbindex.gt.0) write(iunit) (index(i),i=1,nbindex)
```

Where ‘name’ and ‘varpos’ are two 32 characters long strings representing the name of the variable and its position on the grid (see command <var>. Valid ‘varpos’ values are: ‘none’, ‘cell’, ‘xface’, ‘yface’, ‘zface’ and ‘face’. The third line is made of three 4 bytes integers ‘nbvals’, ‘nbsteps’, and ‘nbindex’ representing respectively the number of values, the number of time steps and the number of location indexes when the variable is located. Val, vm1 and vm2 are three 4 bytes real arrays for the initial variable value, the step n-1 and step n-2 values. Finally ‘index’ is an array of 4 bytes integers setting the cell indexes of located variables. The format #V5#V400 is also allowed. It is identical to V300 but real values are 8 bytes values instead of 4 bytes values.

Cartesian files

These files are used for input or output of data that are given on a Cartesian grid of uniform cell sizes.

Cartesian files always start with the #CV#V300 file identifier. The first four characters indicate a Cartesian variable file while the number following the V character specifies the format version. The variable data follow this record so that ASCII files must be read with:

```
read(iunit,*) header
read (iunit,*) name
read (iunit,*) xmin, xmax, nx
read (iunit,*) ymin, ymax, ny
read (iunit,*) zmin, zmax, nz
read (iunit,*) blank
read (iunit,*) time,(((val(i,j,k),i=1,nx),j=1,ny),k=1,nz)
.
.
.
read (iunit,*) time,(((val(i,j,k),i=1,nx),j=1,ny),k=1,nz)
```

and binary (unformatted) files with:

```
read (iunit) header
read (iunit) name
read (iunit) xmin, xmax, nx
read (iunit) ymin, ymax, ny
read (iunit) zmin, zmax, nz
read (iunit) blank
read (iunit) time,(((val(i,j,k),i=1,nx),j=1,ny),k=1,nz)
.
.
.
read (iunit) time,(((val(i,j,k),i=1,nx),j=1,ny),k=1,nz)
```

Where ‘name’ is a 32 characters long string and where xmin, xmax, ymin, ymax, zmin, zmax and blank are 4 bytes reals representing respectively the span of the Cartesian grid and the default value of the variable outside its domain definition. nx, ny and nz are 4 bytes integers representing the number of nodes in each direction. time is a 8 bytes real value representing the simulation time of the record and val is a 4 bytes real array given the grid node variable values. When Cartesian files are used to initialize variables DarcyTools discards the blanking and time values. Then the first val record stands for VAL and the following records, when existing, for VM1 and VM2. During the output process only VAL is written into the COF file.

Tunnel files

Tunnels files always start with the #TO#V300 file identifier. The first four characters indicate a Tunnel file while the number following the V character specifies the format version. The variable data follow this record so that ASCII files must be read with:

```
read(iunit,*) header
read(iunit,*) it, time, nbtunsecs, nbvars
do iv=1,nbvars
 read(iunit,*) varname
enddo
do its=1,nbtunsecs
 read(iunit,*) marker, rhovin, nbc
 do ic=1,nbc
  read(iunit,*) icell, (var(k),k=1,nbvars)
 enddo
enddo
 .
 .
 .
read(iunit,*) it, time, nbtunsecs, nbvars
do iv=1,nbvars
 read(iunit,*) varname
enddo
do its=1,nbtunsecs
 read(iunit,*) marker, rhovin, nbc
 do ic=1,nbc
  read(iunit,*) icell, (var(k),k=1,nbvars)
 enddo
enddo
```

and binary (unformatted) files with:

```
read(iunit) header

read(iunit) it, time, nbtunsecs, nbvars
do iv=1,nbvars
 read(iunit) varname
enddo
do its=1,nbtunsecs
 read(iunit) marker, rhovin, nbc
 do ic=1,nbc
  read(iunit) icell, (var(k),k=1,nbvars)
 enddo
enddo
 .
 .
 .
read(iunit) it, time, nbtunsecs, nbvars
do iv=1,nbvars
 read(iunit) varname
enddo
do its=1,nbtunsecs
 read(iunit) marker, rhovin, nbc
 do ic=1,nbc
  read(iunit) icell, (var(k),k=1,nbvars)
 enddo
enddo
```

Where ‘it’ is a 4 bytes integer representing the iteration index of the output. ‘time’ is a 8 bits real representing the current time value of the step. ‘nbtunsec’ and ‘nbvars’ are two 4 bytes integers representing the number of tunnel sections and the number of variables in the records. The variables names ‘varname’ follow as 32 characters long strings. Fore each tunnel section, ‘marker’ is the 4 bytes integer representing the associated marker value, ‘rhovin’ is a 4 bytes real value representing the mass input flow for entire section and ‘nbc’ a 4 bytes integer giving the number of wall cells in the section. For each wall cell of the section ‘icell’ represents the 4 bytes index in the grid and var(k) represent the ‘nbvars’ 4 bytes reals for the cell values of the given variables. The four first variables are always the x, y and z coordinates of the center of the wall cells, followed by the area to apply in order to get the mass flux from the mass flux density ‘tif’.

Result output files

Result output files always start with the #R5#V300 file identifier. The first four characters indicate a ROF file while the number following the V character specifies the format version. Then follows a header made of three strings representing the DarcyTools solver version, the date of run and the title. Several variable records follow the header. Each record starts with the step index, the internal loop index, the time, the time step, the previous time step and the number of stored variables. Then the variables appear with a format similar to the variable files. ASCII files must be read with:

```
read(iunit,*) header
read(iunit,*) version
read(iunit,*) date
read(iunit,*) title

read(iunit,*) step,it,t,dt,dtm1,nbvars
do iv=1,nbvars
 read(iunit,*) name, varpos
 read(iunit,*) nbvals, nbsteps, nbindex
 read(iunit,*) (val(i),i=1,nbvals)
 if(nbsteps.ge.2) read(iunit,*) (vm1(i),i=1,nbvals)
 if(nbsteps.ge.3) read(iunit,*) (vm2(i),i=1,nbvals)
 if(nbindex.gt.0) read(iunit,*) (index(i),i=1,nbindex)
enddo
.
.
```

and binary (unformatted) files with:

```
read(iunit) header
read(iunit) version
read(iunit) date
read(iunit) title

read(iunit) step,it,t,dt,dtm1,nbvars
do iv=1,nbvars
 read(iunit) name, varpos
 read(iunit) nbvals, nbsteps, nbindex
 read(iunit) (val(i),i=1,nbvals)
 if(nbsteps.ge.2) read(iunit) (vm1(i),i=1,nbvals)
 if(nbsteps.ge.3) read(iunit) (vm2(i),i=1,nbvals)
 if(nbindex.gt.0) read(iunit) (index(i),i=1,nbindex)
enddo
.
.
.
```

‘Version’, ‘date’ and ‘title’ are respectively 10, 19 and 64 characters long while ‘step’, ‘it’ and ‘nbvars’ are 4 bytes integers and ‘t’, ‘dt’ and ‘dtm1’ are 8 bytes reals. The format #R5#V400 is also used. It is identical to V300 but real values are 8 bytes values instead of 4 bytes values.

Starting location files

Starting location files contain patches definition for PARTRACK starting location definitions. They always start with the #PL#V300 file identifier. The first four characters indicate the type of data while the number following the V character specifies the format version. Several starting location definition records follow the header. Each record contains the number of particles, the x, y, z and time extents. ASCII files must be written with:

```
write(iunit,*) '#PL#V300'
write(iunit,*) nps, xmin, xmax, ymin, ymax, zmin ,zmax, tmin, tmax
.
.
.
```

and binary (unformatted) files with:

```
write(iunit) '#PL#V300'
write(iunit) nps, xmin, xmax, ymin, ymax, zmin ,zmax, tmin, tmax
.
.
.
```

'nps' is a 4 bytes integer while 'xmin', 'xmax', 'ymin', 'ymax', 'zmin' and 'zmax' are 4 bytes reals and 'tmin', 'tmax' are 8 bytes reals.

Indg files

Indg files contain encrypted 4 bytes integer values associated to each cell of a given grid. They always start with the #IG#V300 file identifier. The first four characters indicate the type of data while the number following the V character specifies the format version. ASCII files must be written with:

```
write(iunit,*) '#IG#V300'
write(iunit,*) (indg(i),i=1,nbcells)
```

and binary (unformatted) files with:

```
write(iunit) '#IG#V300'
write(iunit) (indg(i),i=1,nbcells)
```

Tracks files

Tracks files contain some particle history data computed during the PARTRACK integration. They always start with the #TK#V301 file identifier. The first four characters indicate the type of data while the number following the V character specifies the format version. Following the file identifier, a header is made of the number of starting location, the locations definition according to the starting location file format, the number of variables (four bytes integer) and the variables names (32 characters long). ASCII files must be read with:

```
read(iunit,*) header

read(iunit,*) nlocs
do i=1,nlocs
 read(iunit,*) nps, xmin, xmax, ymin, ymax, zmin ,zmax, tmin, tmax
enddo

read(iunit,*) nvars
do i=1,nvars
 read(iunit,*) varname(i)
enddo
```

```
read(iunit,*) ievent
if(ievent.eq.1) then
 read(iunit,*) nop, iloc, t, x, y, z, u, v, w
else
 read(iunit,*) nop, t, x, y, z, dist, fquo
endif
do i=1,nvars
 read(iunit,*) var
enddo
.
.
.
```

and binary (unformatted) files with:

```
read(iunit) header

read(iunit) nlocs
do i=1,nlocs
 read(iunit) nps, xmin, xmax, ymin, ymax, zmin ,zmax, tmin, tmax
enddo

read(iunit) nvars
do i=1,nvars
 read(iunit) varname(i)
enddo

read(iunit) ievent
if(ievent.eq.1) then
 read(iunit) nop, iloc, t, x, y, z, u, v, w
else
 read(iunit,*) nop, t, x, y, z, dist, fquo
endif
do i=1,nvars
 read(iunit) var
enddo
.
.
.
```

Two kinds of records follow the header. The birth records start with a four bytes integer, ievent, whose value is always 1 and continue with two four bytes integers (the particle id and its starting location index) and seven four bytes reals (the time, the x, y, z coordinates and u, v, w Darcy velocity components). The life records always start with a four bytes integer, ievent, whose value is 2, 3 or 4 and continue with one four bytes integer (particle id) and six four bytes reals (the time, the x, y, z coordinates, the travel distance and the F quotient). Both type of record (birth or life) is followed by nvars records made of the variable value in the cell containing the particle (four bytes real). The type 2 is dedicated to moving particles. The type 3 indicates that the particle is leaving the domain. The type 4 indicates that the particle enters a catching box. Since the births of particles are spread in time, the records of type 1 and 2-3-4 may be interlaced. However a type 1 record always precedes any type 2-3-4 record for a given particle ‘nop’.

Location files

Location files contain a list of cell indexes for creating locations in the solver. They always start with the #LO#V300 file identifier. The first four characters indicate the type of data while the number following the V character specifies the format version. Following the file identifier, a header is made of the kind of indexes (four bytes integer always 1 at the moment) followed by the (four byte integer) number of (four bytes integers) indexes in the file. ASCII files must be read with:

```
read(iunit,*) header
read(iunit,*) ikind
read(iunit,*) nbind
read(iunit,*) (indx(i),i=1,nbind)
```

and binary (unformatted) files with:

```
read(iunit) header
read(iunit) ikind
read(iunit) nbind
read(iunit) (indx(i),i=1,nbind)
```

Licence files

The licence files are always named “mgl.lic” and must be placed in the DT30 directory. They are always ASCII files whose blank lines and lines starting with the ‘*’ character are ignored. A licence file must contain a line with the licence string. DarcyTools starts by trying to load the licence string contained in the mgl.lic file. If it succeeds the licence string becomes the default licence string which can be overwritten by the <licence> argument of the <run> command.

```
* ------------------------------
* Key serial number = 1549002311
* Expiration date   = 31-01-2006
* Concurrent runs   = enabled
* ------------------------------
Z9bjbacfnu8I3hzNc8Lt
```

Partition files

A partition file always starts with the #BG#V300 file identifier. The following line sets a 4 bytes integer ‘nbcells’ which represents the length of the 4 bytes integers array ‘iblock’. Data can be written as follows for ASCII files:

```
write(iunit,*) #BG#V300
write(iunit,*) nbcells
write(iunit,*) (iblock(i),i=1,nbcells)
```

and binary (unformatted) files with:

```
write(iunit) #BG”V300
write(iunit) nbcells
write(iunit) (iblock(i),i=1,nbcells)
```

Refine- Removal Criteria file

The refine-removal criteria file is defining conditions for mesh refinement and cell removal in a grid restart and is normally written in PROPGEN. The following format should be used for ASCII files:

```
write (iunit,*)  ‘#RR#V300’
write (iunit,*)   nbm,nbx,nby,nbz
write (iunit,*)  (marker(i), i=1,nbm)
write (iunit,*)  (dxmax(i),  i=1,nbx)
write (iunit,*)  (dymax(i),  i=1,nby)
write (iunit,*)  (dzmax(i),  i=1,nbz)
```

and binary (unformatted) files with:

```
write (iunit)  ‘#RR#V300’
write (iunit)   nbm,nbx,nby,nbz
write (iunit)  (marker(i), i=1,nbm)
write (iunit)  (dxmax(i),  i=1,nbx)
write (iunit)  (dymax(i),  i=1,nby)
write (iunit)  (dzmax(i),  i=1,nbz)
```

Where nbx, nby, nbz and nbm are 4 bytes integers respectively representing the number of dxmax, dymax dzmax and marker values. They have to be 0 or equal to the number of cells of the grid to modify.

Where the marker values are negative or null cells are removed and the corresponding dxmax, dymax and dzmax values are ignored.

Where the marker values are strictly positive, cells and their children (created inside) will be marked with the given value before the zone markers apply and the dxmax, dymax and dzmax values will be used as default maximum cell size values for each ggn of the ggr command.

6 Property Generation (PROPGEN)

The property generation file, shown in Figure 6-1, is only an example. It is expected that this program will be further developed, as it should be the place where external data (RVS, SICADA, GIS, etc) is introduced and allowed to influence the property fields.

As indicated in Figure 3-1, PROPGEN may also be used to generate a file that governs the refinement and removal of cells.

```
      PROGRAM PROPGEN
C
C-----This program generates property files for SOLVE.
C
C    --It reads the grid coordinates from XYZ
C    --It reads Fracture properties from GEHYCO
C    --It reads data from external data sources.
C    --It generates the property files for SOLVE
C
      use M_UTIL

      character(len=32) name, varpos
      real, dimension(:), pointer :: poros, frevol, condx, condy,condz

C   Read Grid coordinates from file  "XYZ"
C   ======================================

      WRITE(6,*) 'Read in grid'

      call UTIL_MSG_UNIT(6)        !declare screen for error printout
      call UTIL_LOAD_GRID('xyz')   !read grid in 'xyz' file

      nbx = GET_NBXFACES() !get number of x-faces
      nby = GET_NBYFACES() !get number of y-faces
      nbz = GET_NBZFACES() !get number of z-faces
      nbc = GET_NBCELLS()  !get number of cells


C   Allocate arrays
C   ===============

      allocate(poros(nbc))
      allocate(frevol(nbc))
      allocate(condx(nbx))
      allocate(condy(nby))
      allocate(condz(nbz))
```

Figure 6-1. *Example of PROPGEN.*

```
C  Read in fracture properties from file "xxxxx.dat"
C  ================================================
C
C Note: -frevol is free volume in cell, not porosity

      WRITE(6,*) 'Read in Fracture properties'

      open (unit=61, file='frevol.dat', form='UNFORMATTED')
      do i=1,nbc
       read(61) frevol(i)
      enddo
      close(61)

      open (unit=61, file='condux.dat', form='UNFORMATTED')
      do i=1,nbx
       read(61) condx(i)
      enddo
      close(61)

      open (unit=61, file='conduy.dat', form='UNFORMATTED')
      do i=1,nby
       read(61) condy(i)
      enddo
      close(61)

      open (unit=61, file='conduz.dat', form='UNFORMATTED')
      do i=1,nbz
       read(61) condz(i)
      enddo
      close(61)


C --Read in data from external sources
C   ==================================
      WRITE(6,*) 'Read in data from external sources'
```

Figure 6-1. *Cont.*

```
C  --Modify properties and prepare arrays for SOLVE
C   ================================================
C

C-- Generate porosity, add min values and convert to permeabilities

      cnvert= 2.0387E-7
      permin= 1.E-20
      pormin= 1.E-5

      do i=1,nbc
       vol = GET_CELL_VOL(i)
       poros(i) = frevol(i)/vol + pormin
      enddo

      do i=1,nbx
       condx(i) = condx(i)*cnvert + permin
      enddo

      do i=1,nby
       condy(i) = condy(i)*cnvert + permin
      enddo

      do i=1,nbz
       condz(i) = condz(i)*cnvert + permin
      enddo


C--write property arrays to be read by SOLVE
C============================================

C
      WRITE(6,*) 'Generated arrays for DTS'

      open (unit=61, file='PERMX', form='UNFORMATTED')
       name    = 'permx'
       varpos  = 'xface'
       write(61) '#V5#V300'
       write(61) name, varpos
       write(61) nbx, 1, 0
       write(61) (condx(i),i=1,nbx)
      close(61)

      open (unit=61, file='PERMY', form='UNFORMATTED')
       name    = 'permy'
       varpos  = 'yface'
       write(61) '#V5#V300'
       write(61) name, varpos
       write(61) nby, 1, 0
       write(61) (condy(i),i=1,nby)
      close(61)
```

Figure 6-1. *Cont.*

```
      open (unit=61, file='PERMZ', form='UNFORMATTED')
       name    = 'permz'
       varpos  = 'zface'
       write(61) '#V5#V300'
       write(61) name, varpos
       write(61) nbz, 1, 0
       write(61) (condz(i),i=1,nbz)
      close(61)

      open (unit=61, file='PORO', form='UNFORMATTED')
       name    = 'poros'
       varpos  = 'cell'
       write(61) '#V5#V300'
       write(61) name, varpos
       write(61) nbc, 1, 0
       write(61) (poros(i),i=1,nbc)
      close(61)

C--Free memory
C=============

      call UTIL_FREE()
      deallocate(poros)
      deallocate(frevol)
      deallocate(condx)
      deallocate(condy)
      deallocate(condz)

      end
```

Figure 6-1. *Cont.*

7 Fortran Input File (FIF)

7.1 Why needed?

The CIF-commands have been designed to handle most “standard” applications. It is however easy to find situations where it is impossible to handle the problem specification by this method:

– Transient source terms, that may be based on measured data (for example pumping in a borehole).

– Fluid properties that are a function of calculated variables (for example pressure effects on viscosity).

– Boundary conditions that need to be specified as a function of space and time coordinates.

DarcyTools provides a Fortran input file (FIF), where more advanced problem specifications can be formulated.

7.2 How to use

First of all a word of warning may be in place. The FIF offers great flexibility, but it also requires more skill and caution of the user. It is up to the user to check that the coding is correct.

The FIF is visited after the CIF-commands have been interpreted and hence overwrites these. Some parts are also visited every time step or sweep, which needs to be considered when, for example, transient conditions are to be specified.

Probably the best way to learn how to use FIF is by way of examples; one such example will be given later in this section together with explaining comments. A few general comments may however be useful:

– The main difficulty is to “navigate” in the unstructured grid.

– Parallelization introduces further complexity as the domain is split into blocks.

– To help the user, a number of “service routines” have been created. These will, for example, help the user to find a variable or modify some fluid property.

– When FIF has been modified, it is of course required that it is recompiled and a new executable is created. (MYDTS).

Utility routines are described in the Appendices.

An empty FIF is given in Figure 7-1. As can be seen it consists of a number of subroutines and the user’s task is to program one or several of these.

```
*=====================================================================*
*=====================================================================*
*=====================================================================*
*===========                                                =========*
*===========                                                =========*
*===========                        FIF                     =========*
*===========                                                =========*
*===========                FORTRAN INPUT FILE              =========*
*===========                                                =========*
*===========                                                =========*
*=====================================================================*
*=====================================================================*
*=====================================================================*
*
      subroutine usrobj(name, nsz, nbe, nbn, nodes, elts)

      character(len=*),            intent(in)    :: name
      integer,                     intent(in)    :: nsz
      integer,                     intent(in)    :: nbe
      integer,                     intent(in)    :: nbn
      real,    dimension(3,nbn),   intent(inout) :: nodes
      integer, dimension(nsz,nbe), intent(inout) :: elts

      end subroutine
*
*=====================================================================*
*
      subroutine usrini()
      end subroutine
*
*=====================================================================*
*
      subroutine usrdt(dt)

      real*8, intent(inout) :: dt

      end subroutine
*
*=====================================================================*
*
      subroutine usrprop(eqname)

      character(len=*), intent(in) :: eqname

      end subroutine
*
*=====================================================================*
*
      subroutine usrbss(eqname, qsrc, qphi, ndim)

      character(len=*),      intent(in)    :: eqname
      real, dimension(ndim), intent(inout) :: qsrc
      real, dimension(ndim), intent(inout) :: qphi

      end subroutine
```

Figure 7-1. *An empty FIF.*

```
*======================================================================*
*
      subroutine usrslv(eqsname)

      character(len=*), intent(in) :: eqsname

      end subroutine
*
*======================================================================*
*
      subroutine usrout(eqsname)

      character(len=*), intent(in) :: eqsname

      end subroutine
*
*======================================================================*
*
      subroutine usrland(icell, island)

      integer, intent(in)    :: icell
      logical, intent(inout) :: island

      end subroutine
*
*======================================================================*
*
      subroutine usrstop(id, message)

      integer, intent(in) :: id
      character(len=*), intent(in) :: message

      end subroutine
*
*======================================================================*
*
      subroutine usrptk(nop, x, y, z, birth)

      integer, intent(in)    :: nop
      real,    intent(inout) :: x
      real,    intent(inout) :: y
      real,    intent(inout) :: z
      real(8), intent(inout) :: birth

      end subroutine
*
*======================================================================*
*
      subroutine usrchkp(chkpname)

      character(len=*), intent(in) :: chkpname

      end subroutine
*
*======================================================================*
```

Figure 7-1. *Cont.*

Example of use

```
*
*=====================================================================*
*
      subroutine usrbss(eqname, qsrc, qphi, ndim)

      use M_UTIL


      character(len=*),      intent(in)    :: eqname
      real, dimension(ndim), intent(inout) :: qsrc
      real, dimension(ndim), intent(inout) :: qphi
      integer, pointer :: indx(:)

      if(eqname .eq. 'mass') then

*.....get indexes of top cells whose minimum z-coordinate is lower or
*      equal to zero. Note that the function GET_LAND_INDX is for
*      altitude strictly greater than zero.

       indx => GET_SEA_INDX()

       dist1 = 2.E4
       dist2 = dist1 / 4
       dist3 = dist1 / 8

       do i=1,size(indx)

        ic = indx(i)

*.......compute the x coordinate of the center of cell IC

        xcc = GET_CELL_X0(ic) + 0.5 * (GET_CELL_X1(ic)-GET_CELL_X0(ic))

*.......compute the sources terms

        val1 = 400*1000*9.81*xcc/dist1
        val2 = 50*1000*9.81*sin(2*3.141592*xcc/dist2)

        qsrc(ic) = 1.E20 * (val1 + val2)
        qphi(ic) = 1.E20

       enddo

      endif

      end subroutine
*
*=====================================================================*
```

Figure 7-2. *Example of coding in FIF. For an explanation of the situation considered, see verification case B1.*

8 A Generic Äspö Model

8.1 Introduction

In Report 1 results from this case were presented and discussed. It was also noted that the case is generic, but share many features of the Äspö site.

Here the emphasis is on the set-up of the case in DarcyTools; the CIF and FIF files, and other programs, will be presented and discussed.

In order to make the presentation self-contained, the basic features of the case will first be repeated.

8.2 The situation studied

The situation studied is outlined in Figure 8-1. It is a coastal site, with sea water of a brackish nature (salinity of 1%). We assume a certain precipitation on land and we hence have a density stratification to take into account. Two hills give a certain topography on land. A tunnel, with a certain inflow, will be placed below the small island in the experimental volume shown in the figure. The focus of the analysis will be on the effects of the tunnel. The situation has a clear resemblance with the Äspö region. The fracture system will however be much simpler in this demo, as compared to the detailed information available for Äspö HRL. Four major fracture zones, shown in Figure 8-1, are assumed to represent the deterministic system. Random fractures will be generated to build a working fracture network.

A summary of the problem specification is given in Table 8-1. It is not the intention to give a complete account of the input data; this is considered to be outside the scope (the specification of the fracture network would be lengthy, for example). A few comments may be needed as a complement to the key features in the table (see also Figure 8-1):

- Domains. The site model covers the whole domain, while the laboratory model is located below the island. The first of these two grids follows the topography, while the second grid is a cartesian box. The experimental model is placed in the laboratory model, enclosing the tunnel area.
- Properties. Porosity is specified for each fracture and deformation zone. The diffusion coefficient is given a value of ten times the value for molecular diffusion for salt; this process is hence insignificant. Transmissivities, orientations, etc for the random fractures are set according to the values found appropriate for Äspö HRL.
- Random fractures. When the fracture network for the site domain is generated, random fracture in the interval 40 → 1,000 metres are generated. Random fractures in the interval 10–40 metres are then added for the laboratory volume. For the experimental volume random fractures in the interval 2 – 10 metres are added.

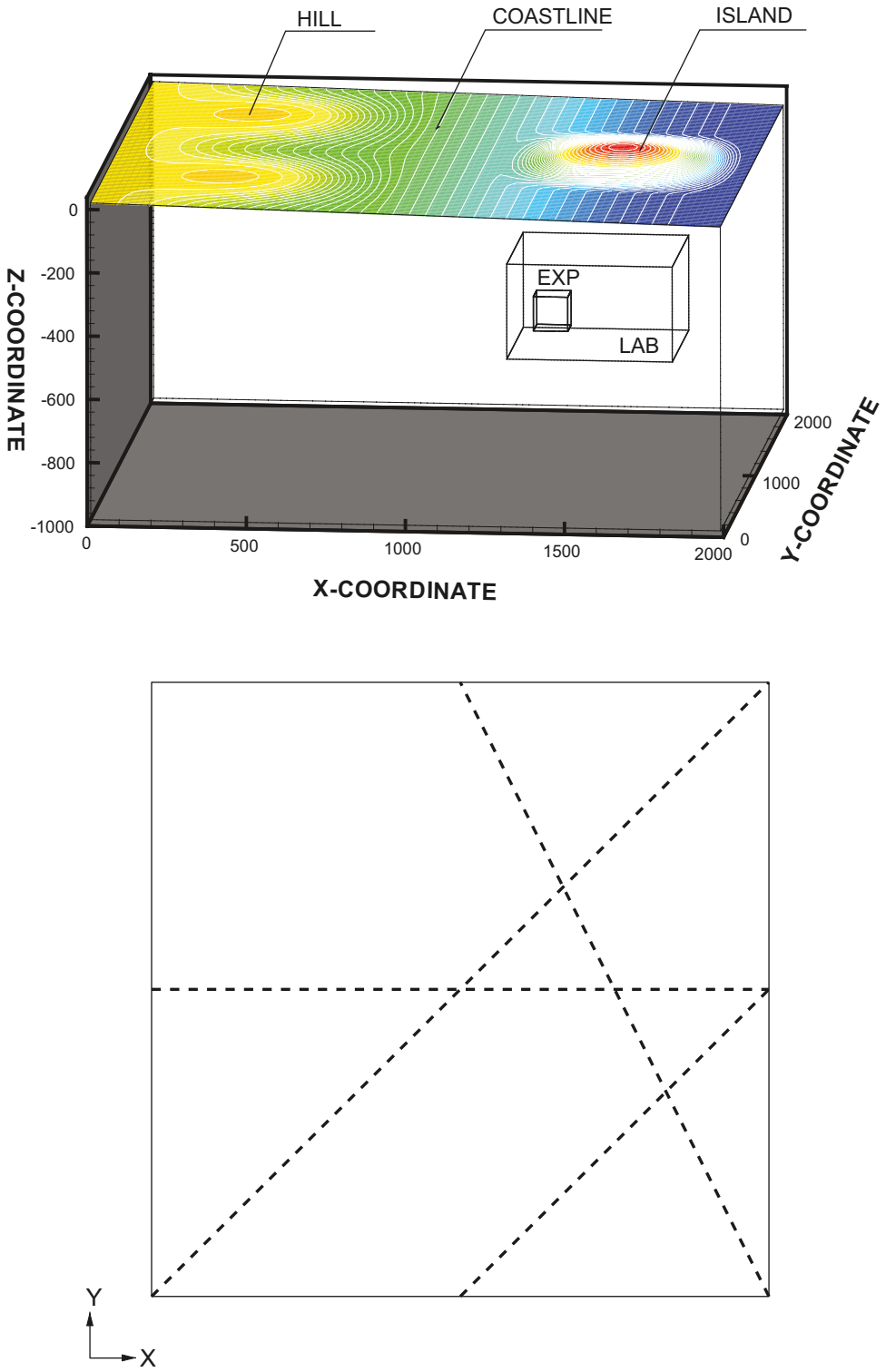


Figure 8-1. *Situation considered (top) and a horizontal view of the deterministic deformation zones.*

Table 8-1. Summary of problem specification.

Domains and grids	Site: 2 x 2 x 1km^3 Δ =32m Laboratory: 500 x 500 x 400m^3 Δ = 16m Experimental: 200 x 100 x 100m^3 Δ = 8m Tunnel: Δ = 2m
Properties	– Deterministic zones according to Figure 8-1, Transmissivity = 10^{-5} m^2/s – Random fractures Site: l = 40 → 1,000 m Laboratory: l = 10 → 40 m Experimental: l = 2 → 10 m – Diffusion coefficients: 10^{-8} m^2/s, constant
Boundary conditions	Top: Precipitation on land , 50 mm/year Pressure and salinity fixed below sea Vertical boundaries: Zero flux at x = 0 boundary. Prescribed pressure and salinity on other vertical boundaries. Fixed salinity at bottom boundary. Tunnel : Grouting with a maximum conductivity of 10^{-8} m/s. Groundwater table: As part of the simulation the groundwater table is calculated.

8.3 The set-up

In this section the files that make up the case will be presented and commented. Only some of the settings will however be commented, as most settings are “standard” and are discussed in the previous chapters of this report.

CIF

```
<cif>

<!--============================================================
============== MAIN                                 ================
==============                                     ================
============== Case:    Generic Aspo model          ================
============== Date:    2010-11-18                  ================
============== Author : Urban Svensson              ================
============================================================-->

<!-- Physical models -->
<!-- *************** -->

<law_mu><a0>  2.E-3   </a0></law_mu>

<!-- Variables and initial cond. -->
<!-- *************************** -->

<var>
 <name>    permx  </name>
 <inifile> PERMX  </inifile>
</var>

<var>
 <name>    permy  </name>
 <inifile> PERMY  </inifile>
</var>

<var>
 <name>    permz  </name>
 <inifile> PERMZ  </inifile>
</var>

<var>
 <name>    diffx          </name>
 <ini>     domain 1.E-8   </ini>
</var>

<var>
 <name>    diffy          </name>
 <ini>     domain 1.E-8   </ini>
</var>

<var>
 <name>    diffz          </name>
 <ini>     domain 1.E-8   </ini>
</var>

<var>
 <name>    poros          </name>
 <inifile> PORO           </inifile>
</var>

<flux>
 <name>  fluxtun     </name> <!-- Inflow to tunnel -->
 <loc>   fluxloc     </loc>
</flux>
```

```
<spt>
 <s0>       1.         </s0>
 <s1>       0.0        </s1>
 <dsdz>    -2.E-3      </dsdz>
 <dwt>      2.         </dwt>
</spt>

<!-- Boundary cond., Sources & Sinks -->
<!-- ****************************** -->

<defbc>
 <ssea>   1.         </ssea>
 <pme>    1.6E-9     </pme> <!--  1.6E-9 = 50 mm/year -->
</defbc>

<!-- Special models -->
<!-- ************** -->

<gwt>
 <relax>   0.01       </relax>
</gwt>


<tunsec>
 <marker>   2          </marker>
 <permax>   2.E-16     </permax> <!-- Grouting  -->
</tunsec>

<!-- Equations -->
<!-- ********* -->

<eqn>
 <name>   mass           </name>
 <bss>    freez_east     </bss>
 <bss>    freez_south    </bss>
 <bss>    freez_north    </bss>
</eqn>

<eqn>
 <name>   salt           </name>
 <bss>    freez_east     </bss>
 <bss>    freez_south    </bss>
 <bss>    freez_north    </bss>
 <bss>    freez_low      </bss>
</eqn>

<eqs>
 <name>   mass-salt     </name>
 <relin>  0.4           </relin>
 <liter>  8             </liter>
 <resfac> 0.001         </resfac>
 <ipreco> 3             </ipreco>
 <igmres> 4             </igmres>
</eqs>

<!-- Solver -->
<!-- ****** -->

<time>
 <order>  1          </order>
 <dt>     43200.    </dt>      <!-- 43200 = 12 hours -->
</time>
```

```
<loop>
 <name>    step       </name>
 <nbit>    3000       </nbit>
 <eqs>     mass-salt  </eqs>
 <hist>    hist1      </hist>
 <hist>    hist2      </hist>
 <hist>    hist3      </hist>
 <hist>    hist4      </hist>
 <hist>    hist5      </hist>
</loop>

<loop>
 <name>    main     </name>
 <tecplot> ASPO     </tecplot>
</loop>

<!--============================================================
=============  INPUT-OUTPUT  ===================================
============================================================-->

<!-- Run % store -->
<!-- *********** -->

<run>
 <title>   'GENERIC ASPO MODEL'   </title>
</run>

<slv>
 <restart>  T          </restart>
</slv>

<!-- Screen -->
<!-- ****** -->

<hist><name>    hist1     </name>
      <var>     pressure  </var>
      <var>     fluxtun   </var>
</hist>

<hist><name>    hist2     </name>
      <var>     pressure  </var>
      <var>     salinity  </var>
      <spot>    pcenter   </spot></hist>

<hist><name>    hist3     </name>
      <var>     pressure  </var>
      <var>     salinity  </var>
      <var>     darcy-w   </var>
      <profile> vertical  </profile></hist>

<hist><name>    hist4      </name>
      <var>     pressure   </var>
      <var>     permx      </var>
      <var>     darcy-u    </var>
      <profile> horisontal </profile></hist>

<hist><name>    hist5       </name>
      <var>     salinity    </var>
      <cut>    'plane y'    </cut></hist>
```

```
<!-- Tecplot -->
<!-- ******* -->

<tecplot>
 <name>    ASPO         </name>
 <loc>    'plane x'     </loc>
 <loc>    'plane z450'  </loc>
 <loc>    'plane y'     </loc>
 <loc>    'land'        </loc>
 <loc>    'plane y1500' </loc>
 <var>     pressure     </var>
 <var>     salinity     </var>
 <var>     darcy-u      </var>
 <var>     darcy-v      </var>
 <var>     darcy-w      </var>
 <var>     gwt_fill     </var>
 <var>     gwt_gh       </var>
 <var>     permx        </var>
 <var>     permy        </var>
 <var>     permz        </var>
 <var>     poros        </var>
 <obj>     tunnel       </obj>
 <obj>     lab          </obj>
 <obj>     exp          </obj>
</tecplot>

<tecplot>
 <name>   grid      </name>
 <loc>   'plane y'  </loc>
 <loc>   'plane x'  </loc>
 <loc>   'plane zH' </loc>
 <obj>    top       </obj>
 <obj>    tunnel    </obj>
 <loc>    land      </loc>
</tecplot>

<!--============================================================
=============  GRID GENERATION  ==============================
============================================================-->

<!-- ggn -->
<!-- *** -->

<gridgen>
 <ggn>       gg1                 </ggn>
 <span>      2048. 2048. 2048.   </span>
 <origin>    0.    0.   -1000.   </origin>
 <tecplot>   grid                </tecplot>
</gridgen>

<ggn>
 <name>      gg1                 </name>
 <isotropy>  F                   </isotropy>
 <dxmax>     32.                 </dxmax>
 <dymax>     32.                 </dymax>
 <dzmax>     32.                 </dzmax>
 <zone>      ztop                </zone>
 <zone>      zhightop            </zone>
 <zone>      ztun                </zone>
 <zone>      zexp                </zone>
 <zone>      zlab                </zone>
 <zone>      zoff                </zone>
</ggn>
```

```
<zone>
 <name>   ztop                    </name>
 <marker> 5                      </marker>
 <dzmax>  1.                     </dzmax>
 <dxmax>  8.                     </dxmax>
 <dymax>  8.                     </dymax>
 <mask>   border top             </mask>
</zone>

<zone>
 <name>    zhightop              </name>
 <marker> -2                     </marker>
 <dzmax>  1.                     </dzmax>
 <dxmax>  8.                     </dxmax>
 <dymax>  8.                     </dymax>
 <mask>    high top              </mask>
</zone>

<zone>
 <name>   ztun                   </name>
 <marker> 2                      </marker>
 <dzmax>  2.                     </dzmax>
 <dxmax>  2.                     </dxmax>
 <dymax>  2.                     </dymax>
 <mask>   inside tunnel          </mask>
</zone>

<zone>
 <name>   zexp                   </name>
 <marker> 3                      </marker>
 <dzmax>  8.                     </dzmax>
 <dxmax>  8.                     </dxmax>
 <dymax>  8.                     </dymax>
 <mask>   inside exp             </mask>
</zone>

<zone>
 <name>   zlab                   </name>
 <marker> 4                      </marker>
 <dzmax>  16.                    </dzmax>
 <dxmax>  16.                    </dxmax>
 <dymax>  16.                    </dymax>
 <mask>   inside lab             </mask>
</zone>


<zone>
 <name>   zoff                   </name>
 <marker> 0                      </marker>
 <mask>   outside off            </mask>
</zone>
```

```
<!--============================================================
=============  FRACTURE NETWORK  ==============================
============================================================-->

<fracgen>
</fracgen>

<knwf>
 <thknes>  1.0       </thknes>
 <frevol>  0.001     </frevol>
 <fwsurf>  1.        </fwsurf>
 <storat>  1.E-6     </storat>
 <conduc>  1.E-5     </conduc>
 <diffus>  1.E-10    </diffus>
 <triang>  0. 1000. 100. 2000. 1000. 100. 1000. 1000. -10000. </triang>
</knwf>

<knwf>
 <thknes>  1.0       </thknes>
 <frevol>  0.001     </frevol>
 <fwsurf>  1.        </fwsurf>
 <storat>  1.E-6     </storat>
 <conduc>  1.E-5     </conduc>
 <diffus>  1.E-10    </diffus>
 <triang>  0. 0. 100. 2000. 2000. 100. 1000. 1000. -10000. </triang>
</knwf>

<knwf>
 <thknes>  1.0       </thknes>
 <frevol>  0.001     </frevol>
 <fwsurf>  1.        </fwsurf>
 <storat>  1.E-6     </storat>
 <conduc>  1.E-5     </conduc>
 <diffus>  1.E-10    </diffus>
 <triang>  1000. 0. 100. 2000. 1000. 100. 1500. 500. -10000. </triang>
</knwf>

<knwf>
 <thknes>  1.0       </thknes>
 <frevol>  0.001     </frevol>
 <fwsurf>  1.        </fwsurf>
 <storat>  1.E-6     </storat>
 <conduc>  1.E-5     </conduc>
 <diffus>  1.E-10    </diffus>
 <triang>  1000. 2000. 100. 2000. 0. 100. 1500. 1000. -10000. </triang>
</knwf>

<ranf>
 <sizes>   40. 1000.          </sizes>
 <thknes>  3.                 </thknes>
 <frevol>  1.E-3              </frevol>
 <storat>  1.E-6              </storat>
 <diffus>  1.E-10             </diffus>
 <fwsurf>  1.                 </fwsurf>
 <tconst>  1.E-5 0.0 1.E-5 0. </tconst>
 <lref>    1.                 </lref>
 <expid>  -2.6                </expid>
 <idref>   0.2                </idref>
 <lambda>  0. 0. 0.           </lambda>
</ranf>
```

```
<ranf>
 <sizes>   10. 40.            </sizes>
 <thknes>  1.                 </thknes>
 <frevol>  1.E-3              </frevol>
 <storat>  1.E-6              </storat>
 <diffus>  1.E-10             </diffus>
 <fwsurf>  1.                 </fwsurf>
 <tconst>  1.E-5 0.0 1.E-5 0. </tconst>
 <lref>    1.                 </lref>
 <expid>  -2.6                </expid>
 <idref>   0.2                </idref>
 <lambda>  0. 0. 0.           </lambda>
 <obj>     lab                </obj>
</ranf>

<ranf>
 <sizes>   2. 10.             </sizes>
 <thknes>  .1                 </thknes>
 <frevol>  1.E-3              </frevol>
 <storat>  1.E-6              </storat>
 <diffus>  1.E-10             </diffus>
 <fwsurf>  1.                 </fwsurf>
 <tconst>  1.E-5 0.0 1.E-5 0. </tconst>
 <lref>    1.                 </lref>
 <expid>  -2.6                </expid>
 <idref>   0.2                </idref>
 <lambda>  0. 0. 0.           </lambda>
 <obj>     exp                </obj>
</ranf>

<fraccmds>
 <genknw>    keptfracs                                          </genknw>
 <genran>    randomfracs                                        </genran>
 <sortisol>  keptfracs randomfracs savedfracs discardedfracs </sortisol>
 <writefile> savedfracs savedfracs.dat                          </writefile>
 <move>      savedfracs keptfracs                               </move>
 <cellprop>  keptfracs .dat                                     </cellprop>
</fraccmds>

<!--============================================================
=============  OBJECTS & LOCATIONS  ============================
============================================================-->

<obj>
 <name>     tunnel                                          </name>
 <cylinder> 1450. 1000. -450. 1550. 1000. -450  5. 10    </cylinder>
</obj>

<obj>
 <name>  off                                                </name>
 <brick> 00. 2000. 00. 2000. -1000  100.                    </brick>
</obj>

<obj>
 <name>   lab                                               </name>
 <brick> 1200. 1700. 750. 1250. -600.  -200.                </brick>
</obj>

<obj>
 <name>   exp                                               </name>
 <brick> 1400. 1600. 950. 1050. -500.  -400.                </brick>
</obj>
```

```
<obj>
  <name>   top                                                  </name>
  <file>   top.dat                                              </file>
</obj>

!-- spots -->
<loc>
 <name>    'pcenter'              </name>
 <spot>     1000. 1000.  -500.    </spot>
</loc>

<!-- profiles -->
<loc>
 <name>   'vertical'             </name>
 <zline>   500. 500.             </zline>
</loc>

<loc>
 <name>   'horisontal'           </name>
 <xline>   501.0 0.              </xline>
</loc>

<!-- planes -->
<loc>
 <name>   'plane y'              </name>
 <yplane>  1000.0                </yplane>
</loc>
<loc>
 <name>   'plane x'              </name>
 <xplane>  1500.                 </xplane>
</loc>

<loc>
 <name>   'plane zH'             </name>
 <zplane>  0.                    </zplane>
</loc>

<loc>
 <name>   'plane y1500'          </name>
 <yplane>  1500.0                </yplane>
</loc>

<loc>
 <name>   'plane z450'           </name>
 <zplane>  -450.                 </zplane>
</loc>

<loc>
 <name>     fluxloc                            </name>
 <patch>   1400. 1600. 900. 1100. -500. -450. </patch>
</loc>

<!--============================================
=============  FACILITIES  =====================
============================================-->

</cif>
```

PROPGEN

```
             PROGRAM PROPGEN
C
C-----This program generates property files for SOLVE.
C
C    --It reads the grid coordinates from XYZ
C    --It reads Fracture properties from GEHYCO
C    --It reads data from external data sources.
C    --It generates the property files for SOLVE
C
      use M_UTIL

      character(len=32) name, varpos
      real, dimension(:), pointer :: poros, frevol, condx, condy,condz

C   Read Grid coordinates from file  "XYZ"
C   ======================================

      WRITE(6,*) 'Read in grid'

      call UTIL_MSG_UNIT(6)        !declare screen for error printout
      call UTIL_LOAD_GRID('xyz')   !read grid in 'xyz' file

      nbx = GET_NBXFACES() !get number of x-faces
      nby = GET_NBYFACES() !get number of y-faces
      nbz = GET_NBZFACES() !get number of z-faces
      nbc = GET_NBCELLS()  !get number of cells

C   Allocate arrays
C   ===============

      allocate(poros(nbc))
      allocate(frevol(nbc))
      allocate(condx(nbx))
      allocate(condy(nby))
      allocate(condz(nbz))

C  Read in fracture properties from file "xxxxx.dat"
C  ================================================
C
C Note: -frevol is free volume in cell, not porosity

      WRITE(6,*) 'Read in Fracture properties'

      open (unit=61, file='frevol.dat', form='UNFORMATTED')
      do i=1,nbc
       read(61) frevol(i)
      enddo
      close(61)

      open (unit=61, file='condux.dat', form='UNFORMATTED')
      do i=1,nbx
       read(61) condx(i)
      enddo
      close(61)

      open (unit=61, file='conduy.dat', form='UNFORMATTED')
      do i=1,nby
       read(61) condy(i)
      enddo
      close(61)
```

```
      open (unit=61, file='conduz.dat', form='UNFORMATTED')
      do i=1,nbz
       read(61) condz(i)
      enddo
      close(61)


C --Read in data from external sources
C   ==================================
      WRITE(6,*) 'Read in data from external sources'



C  --Modify properties and prepare arrays for SOLVE
C   ============================================
C

C-- Generate porosity, add min values and convert to permeabilities

      cnvert= 2.0387E-7
      permin= 1.E-20
      pormin= 1.E-5

      do i=1,nbc
       vol = GET_CELL_VOL(i)
       poros(i) = frevol(i)/vol + pormin
      enddo

      do i=1,nbx
       condx(i) = condx(i)*cnvert + permin
      enddo

      do i=1,nby
       condy(i) = condy(i)*cnvert + permin
      enddo

      do i=1,nbz
       condz(i) = condz(i)*cnvert + permin
      enddo


C--write property arrays to be read by SOLVE
C============================================

C
      WRITE(6,*) 'Generated arrays for DTS'

      open (unit=61, file='PERMX', form='UNFORMATTED')
       name    = 'permx'
       varpos  = 'xface'
       write(61) '#V5#V300'
       write(61) name, varpos
       write(61) nbx, 1, 0
       write(61) (condx(i),i=1,nbx)
      close(61)

      open (unit=61, file='PERMY', form='UNFORMATTED')
       name    = 'permy'
       varpos  = 'yface'
       write(61) '#V5#V300'
       write(61) name, varpos
       write(61) nby, 1, 0
       write(61) (condy(i),i=1,nby)
      close(61)
```

```
      open (unit=61, file='PERMZ', form='UNFORMATTED')
       name    = 'permz'
       varpos  = 'zface'
       write(61) '#V5#V300'
       write(61) name, varpos
       write(61) nbz, 1, 0
       write(61) (condz(i),i=1,nbz)
      close(61)

      open (unit=61, file='PORO', form='UNFORMATTED')
       name    = 'poros'
       varpos  = 'cell'
       write(61) '#V5#V300'
       write(61) name, varpos
       write(61) nbc, 1, 0
       write(61) (poros(i),i=1,nbc)
      close(61)

C--Free memory
C=============

      call UTIL_FREE()
      deallocate(poros)
      deallocate(frevol)
      deallocate(condx)
      deallocate(condy)
      deallocate(condz)

      end
```

objtop.f

```
*=====================================================================*
*=====================================================================*
*=====================================================================*
*===========                                               ===========*
*===========                                               ===========*
*===========                     OBJDEMO                   ===========*
*===========                                               ===========*
*===========                                  2010-11-19   ===========*
*===========                                               ===========*
*=====================================================================*
*=====================================================================*
*=====================================================================*
*                                                                     *
*     This program builds the top.dat object for the Generic Case     *
*                                                                     *
*=====================================================================*
*


      program objdemo

      parameter(ni=101, nj=101)
      real x(ni+1),y(nj+1),z(ni+1,nj+1)

      open(unit=11,file='top.dat')
       write(11,*) '#OB#V300'
       call top(x,y,z,ni,nj)
       call saveobj(11,x,y,z,ni,nj,'top')
      close(11)

      end
*
*=====================================================================*
*
      subroutine saveobj(iunit,x,y,z,ni,nj,name)

      real x(ni+1),y(nj+1),z(ni+1,nj+1)
      character(len=*) name

       real,   pointer :: nodes(:,:)
       real,   pointer :: elts(:,:)

       allocate(nodes(3,(ni+1)*(nj+1)))
       allocate(elts(3,2*ni*nj))


      nsz = 3
      nbn = (ni+1)*(nj+1)
      nbe = 2*ni*nj


      k = 0
      do i=1,ni+1
      do j=1,nj+1
       k = k+1
       nodes(1,k) = x(i)
       nodes(2,k) = y(j)
       nodes(3,k) = z(i,j)
      enddo
      enddo
```

```
      k = 0
      do i=1,ni
      do j=1,nj
       n = j + (i-1)*(nj+1)
       k = k + 1
       elts(1,k) = n
       elts(2,k) = n+nj+1
       elts(3,k) = n+nj+2
       k = k + 1
       elts(1,k) = n
       elts(2,k) = n+nj+2
       elts(3,k) = n+1
      enddo
      enddo

      write(iunit,*) nsz, nbn, nbe
      write(iunit,*) ((nodes(i,j),i=1,3),j=1,nbn)
      write(iunit,*) ((elts (i,j),i=1,3),j=1,nbe)

      deallocate(nodes)
      deallocate(elts)

      end subroutine
*
*=====================================================================*
*
      subroutine top(x,y,z,ni,nj)
      real x(ni+1),y(nj+1),z(ni+1,nj+1)

      XMIN=0.
      YMIN=0.
      DX=20.
      DY=20.

      x(1)=XMIN
      do 10 i=2,ni+1
       x(i)=x(i-1)+DX
 10   continue

      y(1)=YMIN
      do 20 j=2,nj+1
       y(j)=y(j-1)+DY
 20   continue

      do 40 j=1,nj+1
      do 50 i=1,ni+1
       slope=20.-40.*x(i)/x(ni+1)
       PI=3.14159
       PId2=PI/2.
       hhill=10.
       hisla=50.
       if(i.le.50) then
```

```
c---two hills
       if(j.le.50) then
       SN=0.5*(sin(-PId2+PI*2.*y(j)/y(50))+1.)
       EW=0.5*(sin(-PId2+PI*2.*x(i)/x(50))+1.)
       htop=hhill*SN*EW
        slope=slope+htop
       else
       SN=0.5*(sin(-PId2+PI*2.*y(j-50)/y(51))+1.)
       EW=0.5*(sin(-PId2+PI*2.*x(i)/x(50))+1.)
       htop=hhill*SN*EW
        slope=slope+htop
       endif
c---island
       else
       if(j.gt.25.and.j.le.75.and.i.gt.60) then
       SN=0.5*(sin(-PId2+PI*2.*y(j-25)/y(50))+1.)
       EW=0.5*(sin(-PId2+PI*2.*x(i-60)/x(41))+1.)
       htop=hisla*SN*EW
       slope=slope+htop
       endif
       endif

       z(i,j)=slope
 50    continue
 40    continue

      end subroutine
```

9 References

SKB's (Svensk Kärnbränslehantering AB) publications can be found at www.skb.se/publications.

Ferry M, 2002. MIGAL. [Online]. Available at: http://www.mfrdc.com/migal.htm.

Jackson C P, Andrew R H, Todman S, 2000. Self-consistency of heterogeneous continuum porous medium representation of a fractured medium. Water Resources Research, 36, pp 189–202.

Sahimi M, 1995. Flow and transport in porous media and fractured rock: from classical methods to modern approaches. Weinheim: VCH.

Spalding D B, 1981. A general purpose computer program for multi-dimensional one- and two-phase flow. Mathematics and Computers in Simulation, 8, pp 267–276. (See also http://www.cham.co.uk/.)

Svensson U, 2010. DarcyTools, Version 3.4. Verification, validation and demonstration. SKB R-10-71, Svensk Kärnbränslehantering AB.

Svensson U, Ferry M, Kuylenstierna H-O, 2010. DarcyTools, Version 3.4. Concepts, methods and equations. SKB R-07-38, Svensk Kärnbränslehantering AB.

10 Notation

Most variable names and parameters are defined in connection with the description of the basic equations and the CIF commands. It should also be pointed out that DarcyTools uses SI-units.

The following is a compilation of the main variables.

Variable	Meaning	Unit
a	Power law exponent	–
b	Fracture thickness	m
C	Concentration	varies
c_p	Specific heat of fluid	Joule/kg °C
c	Rock thermal capacity	Joule/m^3 °C
D	Diffusion coefficient	m^2/s
e_T	Aperture	m
FWS	Flow wetted surface	m^2/m^3
g	Gravitational constant	m^2/s
I	Intensity, eqn (4-14)	number/m^3
k	Permeability	m^2
k	Late time slope in BTC	–
K	Hydraulic conductivity	m/s
l	Length	m
n	Fracture intensity, eqn (4-14)	number/m^3
P	Dynamic fluid pressure	Pa
p	Total fluid pressure	Pa
Q	Source term	varies
S	Salinity	%
T	Transmissivity	m^2/s
T	Temperature	°C
t	Time	s
u	Darcyflux	m/s
v	Darcy flux	m/s
w	Darcy flux	m/s
x	Space coordinate	m
y	Space coordinate	m
z	Space coordinate	m
α	Mass transfer coefficient	s^{-1}
β	Volume ratio	–
γ	Compaction coefficient	–
Δ	Cell size	m
θ	Porosity	–
λ	Thermal conductivity	Joule/m °Cs
λ	Fracture orientation, when a vector	–
μ	Fluid dynamic viscosity	kg/ms
ρ	Density	kg/m^3
σ	Specific storativity	m^{-1}

Appendix A

User Subroutines

Contents

Introduction **120**

M_UTIL module **120**
Domain functions 121
Cells functions 121
X-Faces functions 122
Y-Faces functions 122
Z-Faces functions 123
Variables functions 124
Locations functions 124
Runs functions 124
Grid functions 125
Miscellaneous functions 126

Fortran Input file **127**
USROBJ 127
USRLAND 127
USRINI 128
USRDT 128
USRPROP 128
USRBSS 128
USRSLV 129
USROUT 129
USRSTOP 129

MIGAL functions **129**

State laws functions **130**

M_BCI module **131**
BCI_MYRANK() 131
BCI_MASTER() 131
BCI_NBPROCS() 131
BCI_GATHER(val, nbvals) 131
BCI_BCAST(val) 132
BCI_MIN(val, domval, blkval) 132
BCI_MAX(val, domval, blkval) 132
BCI_MEAN(val, domval, blkval) 132
BCI_SUM(val, domval, blkval) 132
BCI_SUFFIX(string) 132
BCI_HALO_ISIN(nbcells) 132

Introduction

In order to provide users with programming capabilities the DarcyTools solver and grid generator automatically calls template subroutines named user's routines. Those routines are gathered in the Fortran Input File (FIF) of the User-Section and are initially empty. Depending on the case to run, users will be able to program one or several routines. A special interface module M_UTIL is provided to ease this programming.

M_UTIL module

The module M_UTIL is a user interface programming facility which gathers many functions dedicated to pointers or values retrieving such as cell coordinates, faces areas, variables arrays… The use of those functions is highly recommended for safely programming the Fortran input file FIF or the property generation program PROPGEN.

```
subroutine usrini()

use M_UTIL
real, dimension(:), pointer :: permx, permy

nbx = GET_NBXFACES()
nby = GET_NBYFACES()

permy => GET_VAR_VAL('permy')
permx => GET_VAR_VAL('permx')

do i=1,nbx
 ih = GET_XFACE_HIGH(i)
 il = GET_XFACE_LOW(i)
 if(ih.gt.0 .and. il.gt.0) then
  mkh = GET_CELL_MK(ih)
  mkl = GET_CELL_MK(il)
  if(mkh.eq.2 .and. mkl.eq.2) permx(i) = 2.E-12
 endif
enddo

do i=1,nby
 ih = GET_YFACE_HIGH(i)
 il = GET_YFACE_LOW(i)
 if(ih.gt.0 .and. il.gt.0) then
  mkh = GET_CELL_MK(ih)
  mkl = GET_CELL_MK(il)
  if(mkh.eq.2 .and. mkl.eq.2) permy(i) = 2.E-12
 endif
enddo

end subroutine
```

Figure A-1. *Programming example using M_UTIL module.*

Domain functions

Some of the M_UTIL module functions are dedicated to the domain geometry:

`real function GET_GRID_XOR()`
Returns the x-coordinate of the origin of the domain.

`real function GET_GRID_YOR()`
Returns the y-coordinate of the origin of the domain.

`real function GET_GRID_ZOR()`
Returns the z-coordinate of the origin of the domain.

`real function GET_GRID_XSPAN()`
Returns the x-span of the domain.

`real function GET_GRID_YSPAN()`
Returns the y-span of the domain.

`real function GET_GRID_ZSPAN()`
Returns the z-span of the domain.

Cells functions

Some of the M_UTIL module functions are dedicated to the grid cell geometry:

`integer function GET_NBCELLS()`
Returns the number of cells in the grid

`real function GET_CELL_X0(icell)`
Returns the minimum x-coordinate of the cell whose index is ICELL

`real function GET_CELL_Y0(icell)`
Returns the minimum y-coordinate of the cell whose index is ICELL

`real function GET_CELL_Z0(icell)`
Returns the minimum z-coordinate of the cell whose index is ICELL

`real function GET_CELL_DX(icell)`
Returns the x-size of the cell whose index is ICELL

`real function GET_CELL_DY(icell)`
Returns the y-size of the cell whose index is ICELL

`real function GET_CELL_DZ(icell)`
Returns the z-size of the cell whose index is ICELL

`integer function GET_CELL_MK(icell)`
Returns the marker value of the cell whose index is ICELL

`real function GET_CELL_VOL(icell)`
Returns the volume of the cell whose index is ICELL

`subroutine GET_CELL(icell,x0,y0,z0,mk,x1,y1,z1)`
Returns in one call the cell location, size and marker values of the cell whose index is ICELL

X-Faces functions

Some of the M_UTIL module functions are dedicated to the x-faces geometry. Caution: When the low or high value of a face is negative or null, the sizes and positions of the face may be wrong if the grid has not been generated with the single boundary face forcing option. In that case the returned values correspond to the size and positions of the neighbour cell.

`integer function GET_NBXFACES()`

Returns the number of x-faces in the grid

`real function GET_XFACE_AREA(iface)`

Returns the area of the x-face whose index is IFACE

`integer function GET_XFACE_HIGH(iface)`

Returns, for the x-face whose index is IFACE, the index of the cell located on its high side. A null value indicates that the face is located on the domain boundary. A strictly negative value means that the face is located on a removed zone boundary whose marker corresponds to the value.

`integer function GET_XFACE_LOW(iface)`

Returns, for the x-face whose index is IFACE, the index of the cell located on its low side. A null value indicates that the face is located on the domain boundary. A strictly negative value means that the face is located on a removed zone boundary whose marker corresponds to the value.

`real function GET_XFACE_X0(iface)`

Returns the x-coordinate of the x-face whose index is IFACE

`real function GET_XFACE_Y0(iface)`

Returns the minimum y-coordinate of the x-face whose index is IFACE

`real function GET_XFACE_Z0(iface)`

Returns the minimum z-coordinate of the x-face whose index is IFACE

`real function GET_XFACE_DY(iface)`

Returns the y-size of the x-face whose index is IFACE

`real function GET_XFACE_DZ(iface)`

Returns the z-size of the x-face whose index is IFACE

`subroutine GET_XFACE(iface,x0,y0,z0,ih,il,y1,z1)`

Returns in one call the face location, size and cell indexes values of the x-face whose index is IFACE.

Y-Faces functions

Some of the M_UTIL module functions are dedicated to the y-faces geometry. Caution: When the low or high value of a face is negative or null, the sizes and positions of the face may be wrong if the grid has not been generated with the single boundary face forcing option. In that case the returned values correspond to the size and positions of the neighbour cell.

`integer function GET_NBYFACES()`

Returns the number of y-faces in the grid

`real function GET_YFACE_AREA(iface)`

Returns the area of the y-face whose index is IFACE

`integer function GET_YFACE_HIGH(iface)`

Returns, for the y-face whose index is IFACE, the index of the cell located on its high side. A null value indicates that the face is located on the domain boundary. A strictly negative value means that the face is located on a removed zone boundary whose marker corresponds to the value.

integer function GET_YFACE_LOW(iface)

Returns, for the y-face whose index is IFACE, the index of the cell located on its low side. A null value indicates that the face is located on the domain boundary. A strictly negative value means that the face is located on a removed zone boundary whose marker corresponds to the value.

real function GET_YFACE_X0(iface)

Returns the minimum x-coordinate of the y-face whose index is IFACE

real function GET_YFACE_Y0(iface)

Returns the y-coordinate of the y-face whose index is IFACE

real function GET_YFACE_Z0(iface)

Returns the minimum z-coordinate of the y-face whose index is IFACE

real function GET_YFACE_DX(iface)

Returns the x-size of the y-face whose index is IFACE

real function GET_XFACE_DZ(iface)

Returns the z-size of the y-face whose index is IFACE

subroutine GET_YFACE(iface,x0,y0,z0,ih,il,x1,z1)

Returns in one call the face location, size and cell indexes values of the y-face whose index is IFACE.

Z-Faces functions

Some of the M_UTIL module functions are dedicated to the z-faces geometry. Caution: When the low or high value of a face is negative or null, the sizes and positions of the face may be wrong if the grid has not been generated with the single boundary face forcing option. In that case the returned values correspond to the size and positions of the neighbour cell.

integer function GET_NBZFACES()

Returns the number of z-faces in the grid

real function GET_ZFACE_AREA(iface)

Returns the area of the z-face whose index is IFACE

integer function GET_ZFACE_HIGH(iface)

Returns, for the z-face whose index is IFACE, the index of the cell located on its high side. A null value indicates that the face is located on the domain boundary. A strictly negative value means that the face is located on a removed zone boundary whose marker corresponds to the value.

integer function GET_ZFACE_LOW(iface)

Returns, for the z-face whose index is IFACE, the index of the cell located on its low side. A null value indicates that the face is located on the domain boundary. A strictly negative value means that the face is located on a removed zone boundary whose marker corresponds to the value.

real function GET_ZFACE_X0(iface)

Returns the minimum x-coordinate of the z-face whose index is IFACE

real function GET_ZFACE_Y0(iface)

Returns the minimum y-coordinate of the z-face whose index is IFACE

real function GET_ZFACE_Z0(iface)

Returns the minimum z-coordinate of the z-face whose index is IFACE

real function GET_ZFACE_DX(iface)

Returns the x-size of the z-face whose index is IFACE

real function GET_ZFACE_DY(iface)

Returns the y-size of the z-face whose index is IFACE

subroutine GET_ZFACE(iface,x0,y0,z0,ih,il,x1,y1)

Returns in one call the face location, size and cell indexes values of the z-face whose index is IFACE.

Variables functions

Some of the M_UTIL module functions are dedicated to the variables:

function GET_CST_G()

Returns the value of the gravity acceleration.

function GET_VAR_VAL(varname)

Returns a pointer on the real array containing the current values of the variable named ‘varname’. The returned pointer is null when the specified variable does not exist.

function GET_VAR_VM1(varname)

Returns a pointer on the real array containing the previous time step values of the variable named ‘varname’. The returned pointer is null when the specified variable does not exist. This function is equivalent to GET_VAR_VAL when the time scheme order is lower than 1.

function GET_VAR_VM2(varname)

Returns a pointer on the real array containing the antepenultimate time step values of the variable named ‘varname’. The returned pointer is null when the specified variable does not exist. This function is equivalent to GET_VAR_VM1 when the time scheme order is lower than 2.

Locations functions

Some of the M_UTIL module functions are dedicated to locations:

function GET_SEA_INDX()

Returns a pointer on the integer array containing the cell indexes of the location named ‘sea’ i.e. the top cells whose minimum z-coordinate is lower or equal to zero. The ‘land’ and ‘sea’ locations may be user defined in routine USRLAND.

function GET_LAND_INDX()

Returns a pointer on the integer array containing the cell indexes of the location named ‘sea’ i.e. the top cells whose minimum z-coordinate is strictly higher than zero. The ‘land’ and ‘sea’ locations may be user defined in routine USRLAND.

Runs functions

Some of the M_UTIL module functions are dedicated to the run parameters:

real*8 function GET_TIME()

Returns the current time value.

real*8 function GET_DT()

Returns the current time step value.

real*8 function GET_DTM1()

Returns the previous time step value.

integer GET_STEP()

Returns the current step index.

integer GET_SWEEP()

Returns the current sweep index.

Grid functions

For programming outside the Fortran Input File (e.g. for property generation program) users may need to read, write and/or manipulate the grid. For this purpose using module M_UTIL allows grid loading from files but also a cell to face connectivity approach, a cell removal and a grid writing facility. The main functionalities of the module are:

subroutine UTIL_MSG_UNIT(iunit)

Specifies an open logical unit where to output error messages. By default no message is printed out.

subroutine UTIL_LOAD_GRID(filename)

Reads the grid from the specified file and prepare cell to face connectivity. This routine allocates memory that should be freed using HOF_FREE.

subroutine UTIL_FREE()

Deletes the grid when loaded and frees the associated memory.

integer function UTIL_NBFACES_CELL(icell)

Returns the number of faces of the cell whose index is ICELL.

subroutine UTIL_FACE_CELL(icell, lface, ikind, ipos, iface)

Given a cell index ICELL, and the index of the face LFACE in the list of faces of the given cell, this routine returns in IKIND the face orientation (1,2,3 for x,y,z), in IPOS the face position (0 for low, 1 for high) and in IFACE the index of the face. One should note that a face is uniquely defined only by (IKIND, IFACE).

subroutine UTIL_CLEAN_GRID()

This routine removes cells whose marker is negative or null. The grid is resized accordingly and the cell to face connectivity is rebuilt. This routine will fail if creating non single boundary face cells.

function UTIL_CLEAN_CVAL(val)

This function removes and reorders the values of the cell array VAL accordingly to the negative of null cell markers. It returns the number of valid values in the VAL array. It should be called before UTIL_CLEAN_GRID since this routine remove the negative or null marked cells.

function UTIL_CLEAN_XVAL(val)

This function removes and reorders the values of the x-face array VAL accordingly to the negative of null cell markers. It returns the number of valid values in the VAL array. It should be called before UTIL_CLEAN_GRID since this routine remove the negative or null marked cells.

function UTIL_CLEAN_YVAL(val)

This function removes and reorders the values of the y-face array VAL accordingly to the negative of null cell markers. It returns the number of valid values in the VAL array. It should be called before UTIL_CLEAN_GRID since this routine remove the negative or null marked cells.

function UTIL_CLEAN_ZVAL(val)

This function removes and reorders the values of the z-face array VAL accordingly to the negative of null cell markers. It returns the number of valid values in the VAL array. It should be called before UTIL_CLEAN_GRID since this routine remove the negative or null marked cells.

subroutine UTIL_SAVE_GRID(filename, binary)

This routine writes the grid in the file named “filename”. When the optional logical parameter BINARY is omitted or set to TRUE the output file is an unformatted binary file. Otherwise the output file is a formatted ASCII file.

Miscellaneous functions

The other functions of the M_UTIL module are:

integer function GET_FREE_UNIT()

Returns a free logical unit for opening files.

character(len=4) function GET_FILE_TYPE(fname)

Returns a four character long string containing the type of the file whose name is given by fname. The returned value is valid only for DarcyTools input/output files.

integer function GET_FILE_VERSION(fname)

Returns the file format version of the file whose name is given by fname. The returned value is valid only for DarcyTools input/output files.

logical function GET_FILE_BINARY(fname)

Returns true when the file given by the name fname was written with an unformatted format. The returned value is valid only for DarcyTools input/output files.

function GET_DEPTH()

This function allocates a real array “nbcells” long and fills it with the vertical distance between each cell center and the top of the grid. The returned value is the pointer on the allocated array. Users are responsible for freeing the memory associated to this array. A good practice follows:

```
use M_UTIL
real, dimension(:), pointer :: depth

depth => GET_DEPTH()
if(associated(depth)) then
 ...
 deallocate(depth)
endif
```

Figure A-2. *Programming example for function GET_DEPTH.*

subroutine SET_CELL_MK(mk,icell)

Sets ‘mk’ as marker value for the cell whose index is ‘icell’. It should be used before calling UTIL_CLEAN_GRID for cell removal purpose.

function GET_ROF_VAL(fname, vname, istep)

This function reads the rof file given by its name fname and looks for the val array of the variable vname. When a record exists for the time step istep this function allocates a real array “nbvals” long, fills it with read values and returns its pointer. When no val array is found for the variable and the given step index, a null pointer is returned. Users are responsible for freeing the memory associated to this array. A good practice follows:

```
use M_UTIL
real, dimension(:), pointer :: val

val => GET_ROF_VAL(fname, vname, istep)
if(associated(val)) then
 nbvals = size(val)
 ...
 deallocate(val)
endif
```

Figure A-3. *Programming example for function GET_ROF_VAL.*

function GET_ROF_VM1(fname, vname, istep)

This function reads the rof file given by its name fname and looks for the vm1 array of the variable vname. When a record exists for the time step istep this function allocates a real array “nbvals” long, fills it with read values and returns its pointer. When no vm1 array is found for the variable and the given step index, a null pointer is returned. Users are responsible for freeing the memory associated to this array.

function GET_ROF_VM2(fname, vname, istep)

This function reads the rof file given by its name fname and looks for the vm2 array of the variable vname. When a record exists for the time step istep this function allocates a real array “nbvals” long, fills it with read values and returns its pointer. When no vm2 array is found for the variable and the given step index, a null pointer is returned. Users are responsible for freeing the memory associated to this array.

Fortran Input file

The user’s routines of the Fortran Input File (FIF) form a set of template routines which are automatically called by the program in order to provide user with control during runs. The name of those routines follows the pattern USRxxxxx. The routine USROBJ routine allows users to specify their own object for grid generation. The USRLAND routine gives control on the sea-land top cells segregation. The USRINI is an initialization routine while the routines USRDT, USRBOSS, USRPROP, USRSLV and USROUT are called from the iteration loops when solving equations. Each equation set solution calls the user’s subroutines accordingly to the following algorithm:

- For each equation of the equation set
- build properties
- call URSPROP
- build operator
- set boundary conditions
- call URSBOSS
- set MIGAL parameters
- call USRSLV
- solve with MIGAL and update the variable
- call USROUT

USROBJ

This routine is called by the program GRIDGEN for each of the user defined objects specified by a CIF command <usrobj>. The object name is provided to users as well as the dimension specified in the command <usrobj>. When a user object is defined, users must program the USROBJ routine for setting the nodes coordinates and the elements connectivity of the object.

```
subroutine usrobj(name, nsz, nbe, nbn, nodes, elts)
character(len=*),               intent(in)    :: name
integer(4),                     intent(in)    :: nsz
integer(4),                     intent(in)    :: nbe
integer(4),                     intent(in)    :: nbn
real(4),    dimension(3,nbn),   intent(inout) :: nodes
integer(4), dimension(nsz,nbe), intent(inout) :: elts
end subroutine
```

USRLAND

Some boundary conditions and models (e.g. GWT) need to know which cells of the domain are on the land and which cells are under the sea. To manage this cell segregation, DarcyTools (and GRIDGEN for statistics only) automatically detects the cells laying on the top of the domain and having a minimum altitude (z-coordinate) strictly above zero (for land cells) or negative or null (for sea cell). Once this automatic cell segregation is accomplished, the program calls the routine URSLAND, for each cell on top of the domain, and provides users with the cell index ICELL and

the automatic detection result ISLAND. The ISLAND value is TRUE for land-cells and FALSE otherwise i.e. for sea-cells. Changing the ISLAND value inside the USRLAND routine gives users the control of the land and sea definition.

```
subroutine usrland(icell, island)
integer, intent(in)    :: icell
logical, intent(inout) :: island
end subroutine
```

USRINI

The USRINI routine is called at the beginning of the program after the default and the CIF specified initializations. It provides users with the possibility to modify these initializations by accessing directly to the variables (val, vm1, vm2) with the GET_VAR_xxx routines.

```
subroutine usrini()
end subroutine
```

USRDT

This routine is called at the beginning of each time step, after the computation of the CIF specified time step. It gives the opportunity to change the current time step DT during the integration process. The time or the step index can be retrieved using the GET_TIME and the GET_STEP routines. For accuracy reason, DarcyTools always use 8 bytes reals for time values.

```
subroutine usrdt(dt)
real*8, intent(inout) :: dt
end subroutine
```

USRPROP

This routine is dedicated to user's modifications of physical properties such as conductivity, porosity or diffusivity. It is called at the beginning of each equation solution just after the default setting and automatic property computation by models (e.g. TUN, GWT…). The equation name, provided as input argument, indicates users which equation calls the routine. The properties should be modified by retrieving the corresponding variables array with the GET_VAR_VAL routine.

```
subroutine usrprop(eqname)
character(len=*), intent(in) :: eqname
end subroutine
```

USRBSS

This routine gives access to the boundary conditions and source/sink terms for each equation.
It is called by the program just after the default setting, the CIF specified setting and the automatic computation by models (e.g. TUN, FRAME…). The equation name, provided as input argument, indicates users which equation calls the routine. The QSRC and QPHI arrays represent the operator source and diagonal terms such as:

$$S_\phi = Q_{src} - Q_{phi}\,\phi_P \quad \text{with} \quad Q_{phi} > 0 \qquad \text{(A-1)}$$

where QPHI should be positive in order to keep the algebraic set of equations diagonally dominant for a stable iterative convergence.

```
subroutine usrbss(eqname, qsrc, qphi, ndim)
character(len=*),      intent(in)    :: eqname
real, dimension(ndim), intent(inout) :: qsrc
real, dimension(ndim), intent(inout) :: qphi
end subroutine
```

USRSLV

This routine is dedicated to the adaptation of the solver parameters to the current equation or iteration. It is called after the default and CIF specified parameters settings and just before calling MIGAL. Users can overwrite these parameters by calling the specific MIGAL functions MGL_SET_xxx.

```
subroutine usrslv(eqsname)
character(len=*), intent(in) :: eqsname
end subroutine
```

USROUT

This routine is called at the end of each iteration for each equation, just after the variables update. It is a routine dedicated to users needs for printout or data saving.

```
subroutine usrout(eqsname)
character(len=*), intent(in) :: eqsname
end subroutine
```

USRSTOP

This routine is called when a fatal error ocurrs, just before stopping the program. It informs users of the reason of the crash by providing a message ID and a message string. The message IDs are printed in the solve.log files.

```
subroutine usrstop(id, message)
integer, intent(in) :: id
character(len=*), intent(in) :: message
end subroutine
```

MIGAL functions

To solve the successive algebraic set of equations resulting from discretisation, DarcyTools uses the MIGAL unstructured multi-grid solver. This solver offers the possibility of setting its internal parameters through a set of routines whose names follows the MGL_SET_xxx patterns, xxx being the parameter name. All those routines have a single argument made of the parameter value to specify.

subroutine MGL_SET_REFAC(resfac)

Sets the minimum residual reduction before automatic return.

subroutine MGL_SET_RELAX(relax)

Sets the smoother relaxation parameter.

subroutine MGL_SET_ALPHA(alpha)

Sets the agglomeration threshold parameter.

subroutine MGL_SET_RATIO(ratio)

Sets the end of coarsening ratio parameter.

subroutine MGL_SET_ALPHA(alpha)

Sest the agglomeration threshold parameter.

subroutine MGL_SET_DIAG(diag)

Sets the artificial diagonal dominance parameter.

subroutine MGL_SET_LITER(liter)

Sets the maximum number of cycles to perform.

subroutine MGL_SET_NBRELAX(nbrelax)

Sets the number of post-prolongation relaxations.

subroutine MGL_SET_IGMRES(igmres)

Sets the dimension of the GMRES Krylov subspace.

subroutine MGL_SET_IPRECO(ipreco)

Sets the number of preconditioning multi-grid cycles for GMRES.

subroutine MGL_SET_NBGRID(nbgrid)

Sets the maximum number of grid coarsening level.

subroutine MGL_SET_ICYCLE(icycle)

Sets the type of multi-grid cycle to use: 1 for W cycle, 0 for V cycle.

subroutine MGL_SET_NBPRER(nbprer)

Sets the number of pre-restriction relaxations.

subroutine MGL_SET_IGMS(igms)

Sets the dimension of the smoother GMRES Krylov subspace.

subroutine MGL_SET_ILINK(ilink)

Sets the kind of agglomeration criterion.

subroutine MGL_SET_NBMAX(nbmax)

Sets the maximum coarse block size (0=auto).

subroutine MGL_SET_NBMIN(nbmin)

Sets the minimum coarse block size (0=auto).

State laws functions

The evaluation of physical quantities such as fluid density or viscosity as functions of variables such as salinity or temperature may be necessary for programming user's functions. These quantities are not stored as variables by the solver but result from state laws functions. These functions are:

real function FN_RHO(salinity, temperature)

Returns the density of the fluid.

real function FN_MU(salinity, temperature)

Returns the dynamic viscosity of the fluid.

real function FN_CP(salinity)

Returns the specific mass heat of the fluid.

real function FN_COMP(pressure, density, storativity, pref)

Returns the compaction of the rock matrix.

To call these functions users must provide variable values for the pressure, the density, etc... Those values are retrieved using the GET_VAR_VAL routine and users are responsible for checking the size of the returned array. Actually, DarcyTools does not save uniform variables as arrays but as a single value. For example when a problem does not solve the temperature, the default value is stored in VAL(1) and the size of the array is only 1 while the salinity array size may be equal to the number of cells. A safe way to feed the FN_xxx arguments follows:

```
use M_UTIL
real, dimension(:), pointer :: salt, temp

salt => GET_VAR_VAL('salinity')
temp => GET_VAR_VAL('temperature')

ns = size(salt)
nt = size(temp)

...

rho = FN_RHO(salt(min(icell,ns)), temp(min(icell,nt))
```

Figure A-4. *Programming example for calling state laws.*

M_BCI module

DarcyTools implements the Intel MPI library to ensure block to block communications. All the MPI calls have been encapsulated into a single Block Communication Interface (BCI) module. Some of the public routines of this module are dedicated to user programming.

BCI_MYRANK()

This function returns the rank of the current process. By convention the rank of the process correspond to the block ID minus one.

BCI_MASTER()

This function returns true when the current process is the master process or when the run is sequential.

BCI_NBPROCS()

This function returns the number of processes involved by the run.

BCI_GATHER(val, nbvals)

This function gathers the local VAL arrays on the MASTER process and returns the pointer of the gathered array. User is in charge of deleting the returned array when it becomes useless. A null pointer is returned for slave processes. A copy of VAL is returned in sequential mode. When specified, NBVALS sets the number of values to take into account in VAL. VAL can be an array of real*4, real*8 or integers. This function must be called by each process of the run.

BCI_BCAST(val)

This subroutine broadcasts the local value VAL from the MASTER process to all processes. VAL can be either a single value or an array of values. Values can be real*4, real*8, integer, logical, character or string. VAL is an input on MASTER process and an output on slave processes. This subroutine must be called by each process of the run.

BCI_MIN(val, domval, blkval)

This subroutine computes the minimum value of the value VAL and returns it in BLKVAL. For parallel run, DOMVAL returns the minimum value of the entire domain by reducing BLKVAL values for all processes. VAL can be either a single value or an array of values. VAL values can be real*4, real*8 or integer. This subroutine must be called by each process of the run.

BCI_MAX(val, domval, blkval)

This subroutine computes the maximum value of the value VAL and returns it in BLKVAL. For parallel run, DOMVAL returns the maximum value of the entire domain by reducing BLKVAL values for all processes. VAL can be either a single value or an array of values. VAL values can be real*4, real*8 or integer. This subroutine must be called by each process of the run.

BCI_MEAN(val, domval, blkval)

This subroutine computes the mean value of the value VAL and returns it in BLKVAL. For parallel run, DOMVAL returns the mean value of the entire domain by reducing BLKVAL values for all processes. VAL can be either a single value or an array of values. VAL values can be real*4, real*8 or integer. This subroutine must be called by each process of the run.

BCI_SUM(val, domval, blkval)

This subroutine computes the sum of the value VAL and returns it in BLKVAL. For parallel run, DOMVAL returns the sum on the entire domain by reducing BLKVAL values for all processes. VAL can be either a single value or an array of values. VAL values can be real*4, real*8 or integer. This subroutine must be called by each process of the run.

BCI_SUFFIX(string)

This function copies STRING and returns it with the block suffix inserted. When the string contains a dot character, the suffix is inserted before it. Otherwise, the suffix is added at the end of the string. The suffix is made of the block ID according to the '#nnnn' pattern. No suffix is added for sequential runs.

BCI_HALO_ISIN(nbcells)

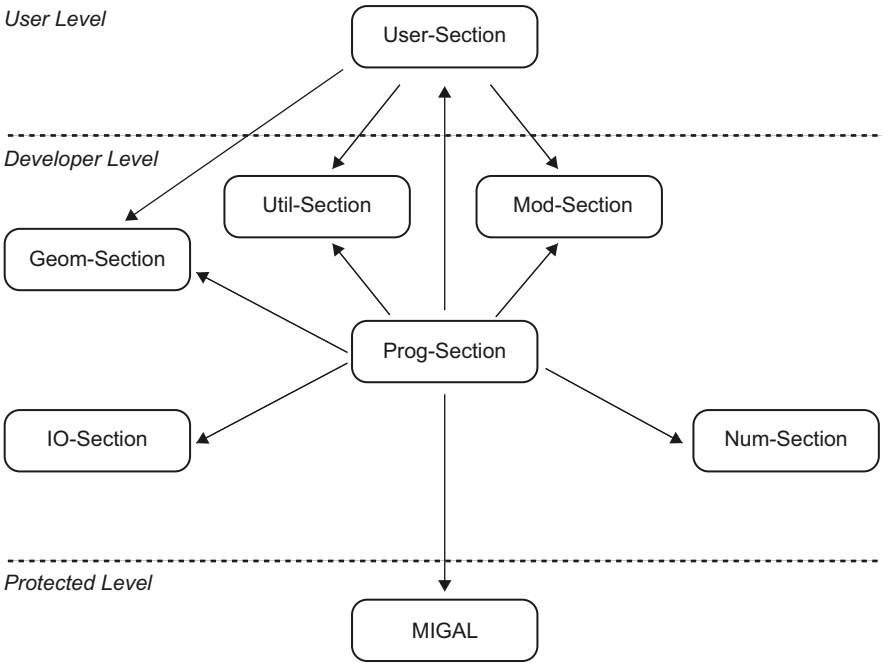
This function allocates an integer array according to NBCELLS the number of cells of the local grid and fills it with the rank (i.e. block ID minus 1) of the neighbouring block when the cell is in the halo and with -1 otherwise. A zero length array is allocated for sequential runs. Users are in charge of deleting the returned array when it becomes useless.

Appendix B

Structure of SOLVE

Code structure

DarcyTools is written in FORTRAN 95 while allowing users to program with FORTRAN 77 for greater ease. The code is object oriented and module encapsulated. Modules are gathered into sections and accessibility levels. A top user level is made of template user's routines in a FIF file, a mid developer level is made of the GEOM, IO, MOD, NUM, PROG, and UTIL sections and a low protected level is made of the MIGAL's subroutines.



The USER-section is a set of template routines automatically called by the program that users can complete to define and control the calculations. Those routines, whose names follow the pattern URSxxxxx, are gathered in a Fortran-Input-File named FIF and should be mainly programmed by using the modules provided in the UTIL-section when possible.

Sections	
num	Numerical discretization of the governing equations, boundary conditions, time management, solver implementation
geom	Grid generation, storage and searching facilities. Objects and patches location
io	Input and output facilities for command reading, or results writing (rof, tecplot, hist…)
mod	Physical models such as state laws encoding, constants definition and special models such as water table, tunnels, sub-grid process (frame) or particles tracking
util	Utility modules for safe programming, object nameing and message handling
prog	Main programs such as solver or grid generator
user	Templates routines for user control

geom-section

The GEOM-section gathers modules related to geometry definitions, grid generation and fast search algorithms.

Modules	
m_adt	Alternating digital tree
m_bgn	Block generation
m_blk	Block for domain decomposition
m_cfc	Cell to face connectivity
m_cgt	Construction grid tree
m_dist	Distance to boundary face
m_frog	Fast and robust geometrical predicates
m_gat	Grid alternating tree
m_geo	Geometry grid encapsulation
m_ggi	Grid generation integrator
m_ggn	Grid generator
m_ggr	Grid generator restart
m_gnd	Ground
m_grd	Grid
m_inter	Intersection predicates
m_loc	Cell and face locations
m_obj	Geometrical objects
m_pat	Patches
m_rgn	Roughness generator
m_rrc	Refinement-Removal Criteria
m_sfc	Space Filling Curves
m_zne	Zones

IO-section

The IO-section gathers modules related to the input-output facilities such as CIF command reading, Tecplot file writing or management of results and restart files.

Modules	
m_cmd	CIF commands
m_cof	Cartesian output files
m_file	Files
m_hist	History monitoring
m_rof	Result and restart output files
m_scr	Screen output control
m_seq	Time sequencers
m_sgf	Single grid files
m_stl	STL object files
m_svf	Single variable files
m_tec	Tecplot output files
m_tkf	Tracking output files
m_tof	Tunnel output files

mod-section

The MOD-section gathers the modules related to physical laws and special models encapsulations such as FRAME, GRT, GWT, TUNNEL, PARTRACK, Coriolis forces or turbulence.

Modules	
m_cmp	Matrix compaction model
m_corio	Coriolis forces
m_cst	Constants
m_dbc	Default boundary conditions for DarcyTools solver
m_dcy	Darcy velocity
m_frm	Frame sub-grid model
m_grt	Grouting model
m_gwt	Ground water table model
m_ice	Permafrost model
m_law	Base fluid properties
m_mat	Materials properties
m_mix	Mixture properties
m_pks	Partrack engine
m_por	Porous drag model for Navier-Stokes model
m_ptk	Particles tracking model
m_spt	Default salinity-pressure-temperature initialization for DarcyTools Solver
m_tun	Tunnel model
m_turb	Turbulence models

num-section

The NUM-section gathers modules related to the numerical discretization of the governing equations, the boundary conditions, the time management, and the solver implementation.

Modules	
m_bci	Block Communication Interface
m_bfv	Boundary face values
m_bnc	Face boundary conditions
m_bss	Source-sink boundary conditions
m_eqn	Equations
m_eqs	Equation sets
m_flm	Mass fluxes
m_flu	Location mass fluxes
m_grad	Gradients
m_ifv	Internal face values
m_loop	Algorithm loops
m_mlf	Multi-linear function
m_mop	Migal first order operators
m_mrs	Modified row storage
m_setop4	Darcy operator discretization
m_setop5	Navier-Stokes operator discretization
m_tgn	Time generator
m_var	Variables

prog-section

The PROG-section gathers main programs. Among programs, three solvers are implemented to solve Darcy flows, pore scale flows or surface water flows. This section also contains utility programs to generate objects or Cartesian grids.

Programs	
distgen	Distance generation
dts	Darcytools solver
gridgen	Cartesian grid generation
blockgen	Block generation
objgen	Object generation
pss	Pore scale solver
sws	Surface water solver
attool	Automatic testing tool

user-section

The USER-section contains a single file named FIF (Fortran Input File). This file gathers subroutines called by solvers to provide a user control during the run execution. These subroutines, whose names follow the pattern USRxxxx, are dummy subroutines with no instruction in. Users are invited to use the M_UTIL module for safely programming them.

Files

fif.f	Fortran input file

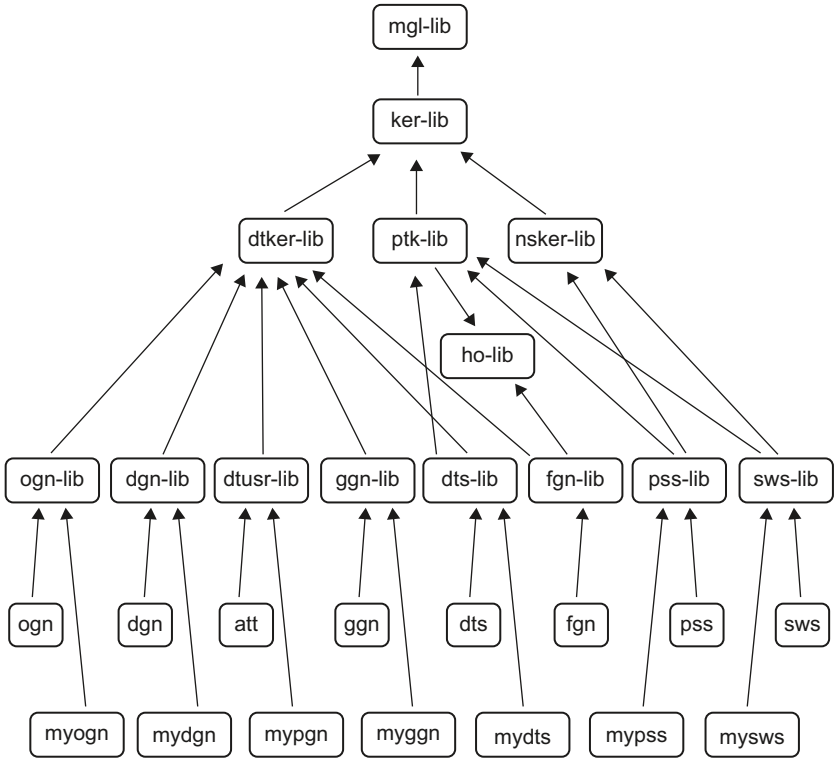
util-section

The UTIL-section gathers modules related to programming facilities such as run’s characteristics, names, messages management or user interface facilities.

Modules	
m_att	Automatic testing tool configuration
m_case	Case to be run by the automatic testing tool
m_chkp	Check points for program testing
m_def	Defined names and parameters
m_dgn	Distance generator
m_get	Retrieving facilities
m_hof	Fracgen and Partrack interface facilities
m_indg	In-grid bit cell tagging
m_msg	Messages output facilities
m_nom	Naming facilities
m_ogn	Object generator
m_prog	Private compilation parameters for test case's programs
m_run	Run's characteristics
m_set	Setting facilities
m_slv	Data exchange controller for solvers
m_tkr	Tecplot rotator
m_util	User facilities including get and set facilities

Libraries

Similarly to the code structure, the library structure is split into three accessibility levels. A user level concerns only the MYxxxx program built from available libraries and user's Fortran input file. The protected level concerns only the MIGAL library. In between, libraries and programs define the developer level and their dependencies follow:



Libraries	
mgl-lib	Migal
ker-lib	Darcy and Navier-Stokes common kernel
dtker-lib	Darcytools kernel
ptk-lib	Particle tracking
nsker-lib	Navier-Stokes kernel
ho-lib	Fracgen-Partrack kernel
ogn-lib	Object generation
dgn-lib	Distance generation
dtusr-lib	DarcyTools user library
ggn-lib	Grid generation generation
dts-lib	DarcyTools solver
fgn-lib	Fracgen library
pss-lib	Pore scale solver
sws-lib	Surface water solver

Appendix C

Four CIF-examples

Introduction

These four examples are part of the verification studies, summarized in Report 2. The reader is hence refered to this report for a description of the case and sample results from simulations.

```
<cif>

<!--============================================================
============== MAIN                              ================
==============                                  ================
============== Case:    Verification Case E1      ================
============== Date:    2007-09-04               ================
============== Author : Urban Svensson           ================
============================================================-->

<!-- Physical models -->
<!-- *************** -->

<law_mu><a0>   2.E-3   </a0></law_mu>
<law_rho><a1>  0.0078  </a1></law_rho>

<!-- Variables and initial cond. -->
<!-- *************************** -->

<var><name>permx    </name> <ini> domain 2.10E-12  </ini></var>
<var><name>permz    </name> <ini> domain 2.10E-12  </ini></var>
<var><name>diffx    </name> <ini> domain 2.67E-7   </ini></var>
<var><name>diffz    </name> <ini> domain 2.67E-7   </ini></var>
<var><name>poros    </name> <ini> domain 0.35      </ini></var>

<spt><s0>3.205</s0><s1>3.205</s1><dsdz>0.</dsdz></spt>

<!-- Boundary cond., Sources & Sinks -->
<!-- ****************************** -->

<bss>
 <name> lowx_p     </name>
 <loc>  west       </loc>
 <qsrc> 0.9549E-3  </qsrc>
 <qphi> 0.         </qphi>
</bss>

<bss>
 <name> lowx_s      </name>
 <loc>  west        </loc>
 <qsrc> 0.          </qsrc>
 <qphi> 1.E20       </qphi>
</bss>



<!-- Equations -->
<!-- ********* -->

<eqn>
 <name> mass        </name>
 <bss>  freez_east  </bss>
 <bss>  lowx_p      </bss>
</eqn>
```

```
<eqn>
 <name> salt        </name>
 <bss>  freez_east  </bss>
 <bss>  lowx_s      </bss>
</eqn>

<!-- Solver -->
<!-- ****** -->

<time>
 <order>  1     </order>
 <dt>     1.E8  </dt>
</time>

<loop>
 <name>    step     </name>
 <nbit>    100      </nbit>
 <eqs>     mass-salt</eqs>
 <hist>    hist1    </hist>
 <hist>    hist2    </hist>
 <hist>    hist3    </hist>
 <hist>    hist4    </hist>
</loop>

<loop>
 <name>    main     </name>
 <tecplot> CaseE1   </tecplot>
</loop>

<!--============================================================
=============  INPUT-OUTPUT  ===================================
============================================================-->

<!-- Run % store -->
<!-- *********** -->

<run>
 <title>  "Henry's Problem"  </title>
</run>

<!-- Screen -->
<!-- ****** -->

<hist><name>    hist1     </name>
      <var>     pressure  </var>
      <var>     salinity  </var></hist>

<hist><name>    hist2     </name>
      <var>     pressure  </var>
      <var>     salinity  </var>
      <spot>    spot1     </spot></hist>

<hist><name>    hist3     </name>
      <var>     pressure  </var>
      <var>     salinity  </var>
      <profile> prof1     </profile></hist>

<hist><name>    hist4     </name>
      <var>     salinity  </var>
      <palette> 0. 3.2 5  </palette>
      <cut>    'plane y'  </cut></hist>
```

```
<!-- Tecplot -->
<!-- ******* -->

<tecplot>
 <name>    CaseE1    </name>
 <loc>    'plane y'  </loc>
 <var>     pressure  </var>
 <var>     salinity  </var>
 <var>     darcy-u   </var>
 <var>     darcy-v   </var>
 <var>     darcy-w   </var>
</tecplot>

<!--============================================================
=============  GRID GENERATION  ==============================
============================================================-->

<!-- ggn -->
<!-- *** -->

<gridgen>
 <ggn>      gg1                  </ggn>
 <span>     200.  4.  100.       </span>
 <origin>   0.    0. -100.       </origin>
</gridgen>

<ggn>
 <name>     gg1                  </name>
 <isotropy> F                    </isotropy>
 <ycut>     F                    </ycut>
 <dxmax>    4.                   </dxmax>
 <dzmax>    4.                   </dzmax>
</ggn>

<!--============================================================
=============  OBJECTS & LOCATIONS  ==========================
============================================================-->

<!-- spots -->
<loc>
 <name> 'spot1'             </name>
 <spot>  100. 2. -50.       </spot>
</loc>

<!-- profiles -->
<loc>
 <name>   'prof1'           </name>
 <zline>   50. 2.           </zline>
 </loc>

<!-- planes -->
<loc>
 <name>     'plane y'       </name>
 <yplane>    2.             </yplane>
</loc>

<!--============================================================
============================================================-->

</cif>
```

```
<cif>

<!--============================================================
============== MAIN                                ================
==============                                    ================
============== Case:    Verification Case C3       ================
============== Date:    2007-10-18                 ================
============== Author : Urban Svensson            ================
============================================================-->

<!-- Physical models -->
<!-- *************** -->

<law_mu><a0>  2.E-3   </a0></law_mu>

<!-- Variables and initial cond. -->
<!-- *************************** -->

<var>
 <name>    permx  </name>
 <inifile> PERMX  </inifile>
</var>

<var>
 <name>    permy  </name>
 <inifile> PERMY  </inifile>
</var>

<var>
 <name>    permz  </name>
 <inifile> PERMZ  </inifile>
</var>

<var>
 <name>    poros   </name>
 <inifile> PORO    </inifile>
</var>

<flux>
 <name> flux1     </name>
 <loc>  'plane y' </loc>
 <faces> north    </faces>
</flux>

<!-- Boundary cond., Sources & Sinks -->
<!-- ****************************** -->

<bss>
 <name>  inflow1    </name>
 <loc>   south      </loc>
 <qsrc>  9.81E+23   </qsrc>
 <qphi>  1.00E+20   </qphi>
</bss>

<bss>
 <name>  outflow1   </name>
 <loc>   north      </loc>
 <qphi>  1.00E+20   </qphi>
</bss>
```

```
<!-- Equations -->
<!-- ********* -->

<eqn>
 <name>  mass      </name>
 <bss>  inflow1    </bss>
 <bss>  outflow1   </bss>
</eqn>

<eqs>
 <name>  mass      </name>
 <liter> 10        </liter>
 <resfac> 0.       </resfac>
</eqs>

<!-- Solver -->
<!-- ****** -->

<loop>
 <name>    step     </name>
 <nbit>    60       </nbit>
 <eqs>     mass     </eqs>
 <hist>    hist1    </hist>
 <hist>    hist2    </hist>
 <hist>    hist3    </hist>
</loop>

<loop>
 <name>    main     </name>
 <tecplot> CaseC3   </tecplot>
</loop>

<!--============================================================
=============  INPUT-OUTPUT  ================================
============================================================-->

<!-- Run % store -->
<!-- *********** -->

<run>
 <title>   'Single Fracture in a Box'      </title>
</run>

<!-- Screen -->
<!-- ****** -->

<hist><name>    hist1     </name>
      <var>     pressure  </var>
      <var>     flux1     </var></hist>

<hist><name>    hist2     </name>
      <var>     pressure  </var>
      <spot>    center     </spot></hist>

<hist><name>    hist3     </name>
      <var>     pressure  </var>
      <profile> vertical  </profile></hist>
```

```
<!-- Tecplot -->
<!-- ******* -->

<tecplot>
 <name>    CaseC3    </name>
 <loc>    'plane y'  </loc>
 <var>     pressure  </var>
 <var>     darcy-u   </var>
 <var>     darcy-v   </var>
 <var>     darcy-w   </var>
 <var>     permx     </var>
 <var>     permy     </var>
 <var>     permz     </var>
 <var>     poros     </var>
</tecplot>

<!--============================================================
=============  GRID GENERATION  ==============================
============================================================-->

<gridgen>
 <ggn>       gg1                </ggn>
 <span>      100.   100.  100.  </span>
 <origin>    0.     0.   -100.  </origin>
</gridgen>

<!-- ggn -->
<!-- *** -->

<ggn>
 <name>      gg1                </name>
 <isotropy>  F                  </isotropy>
 <dxmax>     2.                 </dxmax>
 <dymax>     2.                 </dymax>
 <dzmax>     2.                 </dzmax>
</ggn>

<!--============================================================
=============  FRACTURE NETWORK  ==============================
============================================================-->

<fracgen>
  <seed>       0.12345                   </seed>
</fracgen>

<knwf>
 <thknes> 2.     </thknes>
 <frevol> 0.005  </frevol>
 <fwsurf> 1.     </fwsurf>
 <storat> 1.E-6  </storat>
 <conduc> 1.E-4  </conduc>
 <diffus> 1.E-9  </diffus>
 <triang> 9.2  -1.0 -12.1608  90.8 101.0 -73.3608  9.2 -1.0 -6.6392 </triang>
<!--<triang> 50.  -1.0 -47.5  50. 101.0 -47.5  50. -1.0 -52.5 </triang>-->
</knwf>


<knwf>
 <thknes> 2.     </thknes>
 <frevol> 0.005  </frevol>
 <fwsurf> 1.     </fwsurf>
 <storat> 1.E-6  </storat>
 <conduc> 1.E-4  </conduc>
```

```
 <diffus> 1.E-9  </diffus>
  <triang> 9.2 -1.0 -6.6392 90.8 101.0 -73.3608 90.8 101.0 -67.8392 </triang>
<!--<triang> 50.  -1.0 -52.5  50. 101.0 -52.5  50. 101.0 -47.5 </triang>-->
</knwf>

<fraccmds>
 <genknw>   keptfracs      </genknw>
 <cellprop> keptfracs .dat </cellprop>
</fraccmds>

<!--============================================================
=============  OBJECTS & LOCATIONS  ============================
============================================================-->

<!-- spots -->
<loc>
 <name>  'center'           </name>
 <spot>   50.   50. -50.    </spot>
</loc>

<!-- profiles -->
<loc>
 <name>  'vertical'        </name>
 <xline>  50. -50.         </xline>
</loc>

<!-- planes -->
<loc>
 <name>     'plane y'       </name>
 <yplane>    50.            </yplane>
</loc>

<!--============================================================
============================================================-->

</cif>
```

```
<cif>

<!--============================================================
============== MAIN                                ================
==============                                    ================
============== Case:    Verification Case D6       ================
============== Date:    2008-01-05                 ================
============== Author : Urban Svensson            ================
============================================================-->

<!-- Physical models -->
<!-- *************** -->

<law_mu><a0>  2.E-3   </a0></law_mu>

<!-- Variables and initial cond. -->
<!-- *************************** -->

<var>
 <name>    permx  </name>
 <inifile> PERMX  </inifile>
</var>

<var>
 <name>    permy  </name>
 <inifile> PERMY  </inifile>
</var>

<var>
 <name>    permz  </name>
 <inifile> PERMZ  </inifile>
</var>

<var>
 <name>    poros   </name>
 <inifile> PORO    </inifile>
</var>

<flux>
 <name>  flux1     </name>
 <loc>  'plane y'  </loc>
 <faces> north     </faces>
</flux>

<!-- Boundary cond., Sources & Sinks -->
<!-- ****************************** -->

<bss>
 <name>  inflow1    </name>
 <loc>   south      </loc>
 <qsrc>  9.81E+23   </qsrc>
 <qphi>  1.00E+20   </qphi>
</bss>

<bss>
 <name>  outflow1   </name>
 <loc>   north      </loc>
 <qphi>  1.00E+20   </qphi>
</bss>
```

```
<!-- Special models -->
<!-- ***************-->

<partrack>
 <method>  2                      </method>
 <sloc>    100000 startloc 0. 1. </sloc>
 <eloc>    endloc                 </eloc>
 <seed>    0.9                    </seed>
</partrack>

<!-- Equations -->
<!-- ********* -->

<eqn>
 <name>  mass      </name>
 <bss>  inflow1    </bss>
 <bss>  outflow1   </bss>
</eqn>

<eqs>
 <name>  mass      </name>
 <!--<liter> 10        </liter>
 <resfac> 0.       </resfac>-->
</eqs>

<!-- Solver -->
<!-- ****** -->

<time>
 <order>  1            </order>
<!-- <dt>     3600.        </dt>  for flow sim.  -->
 <dt>     3600.        </dt>
</time>

<loop>
 <name>    step     </name>
 <nbit>    600      </nbit>
 <eqs>     mass     </eqs>
 <hist>    hist1    </hist>
 <hist>    hist2    </hist>
 <hist>    hist3    </hist>
</loop>

<loop>
 <name>    main     </name>
 <tecplot> CaseD6   </tecplot>
</loop>

<!--============================================================
=============  INPUT-OUTPUT  ===================================
============================================================-->

<!-- Run % store -->
<!-- *********** -->

<run>
 <title>   'PARTRACK, Single fracture in a box'      </title>
</run>

<slv>
 <restart>  F  </restart>
 <reset>    T  </reset>
</slv>
```

```
<!-- Screen -->
<!-- ****** -->

<hist><name>    hist1     </name>
      <var>     npo       </var>
      <var>     npc       </var>
      <var>     pressure  </var>
      <var>     flux1     </var></hist>

<hist><name>    hist2     </name>
      <var>     pressure  </var>
      <var>     pdens     </var>
      <spot>    end       </spot></hist>

<hist><name>    hist3     </name>
      <var>     pdens     </var>
      <var>     pressure  </var>
      <profile> alongy  </profile></hist>

<!-- Tecplot -->
<!-- ******* -->

<tecplot>
 <name>    CaseD6    </name>
 <loc>    'plane y'  </loc>
 <var>     pressure  </var>
 <var>     darcy-u   </var>
 <var>     darcy-v   </var>
 <var>     darcy-w   </var>
 <var>     permx     </var>
 <var>     permy     </var>
 <var>     permz     </var>
 <var>     poros     </var>
</tecplot>

<!--============================================================
=============  GRID GENERATION  ==============================
============================================================-->

<gridgen>
 <ggn>       gg1               </ggn>
 <span>      10.   10.  10.     </span>
 <origin>    0.    0.  -10.     </origin>
</gridgen>

<!-- ggn -->
<!-- *** -->

<ggn>
 <name>      gg1               </name>
 <dxmax>     .2                </dxmax>
 <dymax>     .2                </dymax>
 <dzmax>     .2                </dzmax>
</ggn>

<!--============================================================
=============  FRACTURE NETWORK  =============================
============================================================-->

<fracgen>
  <seed>       0.12345                  </seed>
</fracgen>
```

```
<knwf>
 <thknes> .2     </thknes>
 <frevol> 0.05   </frevol>
 <fwsurf> 1.     </fwsurf>
 <storat> 1.E-6  </storat>
 <conduc> 5.E-5  </conduc>
 <diffus> 1.E-9  </diffus>
 <triang>1.0 0.0 -1.286  7.0 10.0 -5.786  1.0 0.0 -0.75  </triang>
<!--<triang>1.0 0.0 -1.302  9.0 10.0 -7.302  1.0 0.0 -0.75 </triang>-->
</knwf>


<knwf>
 <thknes> .2     </thknes>
 <frevol> 0.05   </frevol>
 <fwsurf> 1.     </fwsurf>
 <storat> 1.E-6  </storat>
 <conduc> 5.E-5  </conduc>
 <diffus> 1.E-9  </diffus>
 <triang> 1.0 0.0 -0.75  7.0 10.0 -5.786  7.0 10.0 -5.25 </triang>
<!--<triang>1.0 0.0 -0.75  9.0 10.0 -7.302  9.0 10.0 -6.75  </triang>-->
</knwf>

<fraccmds>
 <genknw>   keptfracs      </genknw>
 <cellprop> keptfracs .dat </cellprop>
</fraccmds>


<!--============================================================
=============  OBJECTS & LOCATIONS  ============================
============================================================-->

<loc>
 <name>  startloc                              </name>
 <patch>  1.0 1.0 0.01 0.01 -1.0  -1.0        </patch>
</loc>

<loc>
 <name>  endloc                                 </name>
 <patch>  0. 10.  9.7  10. -10.   0.0          </patch>
</loc>

<!-- spots -->
<loc>
 <name>  'end'              </name>
 <spot>   7.   9.9 -5.5     </spot>
</loc>

<!-- profiles -->
<loc>
 <name>  'alongy'        </name>
 <yline>  4. -3.5           </yline>
</loc>

<!-- planes -->
<loc>
 <name>     'plane y'       </name>
 <yplane>    5.             </yplane>
</loc>

<!--============================================================
============================================================-->

</cif>
```

```
<cif>

<!--============================================================
============== MAIN                                ================
==============                                    ================
============== Case:    Verification Case E5        ================
============== Date:    2007-09-04                  ================
============== Author : Urban Svensson             ================
============================================================-->

<!-- Physical models -->
<!-- *************** -->

<law_mu>   <a0>   2.E-3   </a0>  </law_mu>
<law_rho>  <a1>   0.0078  </a1>  </law_rho>

<!-- Variables and initial cond. -->
<!-- *************************** -->

<var><name>permx    </name> <ini> domain 2.04E-14  </ini></var>
<var><name>permz    </name> <ini> domain 4.00E-12  </ini></var>
<var><name>diffx    </name> <ini> domain 1.E-12    </ini></var>
<var><name>diffz    </name> <ini> domain 1.E-12    </ini></var>
<var><name>poros    </name> <ini> domain 0.001     </ini></var>
<var><name>salinity </name> <ini> domain 1.        </ini></var>

<!-- Boundary cond., Sources & Sinks -->
<!-- ****************************** -->

<defbc>
 <ssea> 1.0      </ssea>
 <pme>  3.16E-9  </pme>
</defbc>

<!-- Special models -->
<!-- ************** -->

<gwt>
 <relax> 0.8        </relax>
</gwt>

<!-- Equations -->
<!-- ********* -->

<eqn>
 <name> mass        </name>
</eqn>

<eqn>
 <name> salt        </name>
 <bss>  freez_low   </bss>
</eqn>

<!-- Solver -->
<!-- ****** -->

<time>
 <order>  1     </order>
 <dt>     1.E6  </dt>
</time>

<loop>
 <name>    step     </name>
 <nbit>    2000     </nbit>
 <eqs>     mass     </eqs>
```

```
 <eqs>     salt     </eqs>
 <hist>    hist1    </hist>
 <hist>    hist2    </hist>
 <hist>    hist3    </hist>
 <hist>    hist4    </hist>
</loop>

<loop>
 <name>    main     </name>
 <tecplot> CaseE5   </tecplot>
</loop>

<!--============================================================
=============  INPUT-OUTPUT  ===================================
============================================================-->

<!-- Run % store -->
<!-- *********** -->

<run>
 <title>  'The Floating Island'  </title>
</run>

<!-- Screen -->
<!-- ****** -->

<hist><name>    hist1          </name>
      <var>     pressure       </var>
      <var>     salinity       </var></hist>

<hist><name>    hist2          </name>
      <var>     pressure       </var>
      <var>     salinity       </var>
      <spot>    inside         </spot></hist>

<hist><name>    hist3          </name>
      <var>     pressure       </var>
      <var>     salinity       </var>
      <var>     gwt_fill       </var>
      <profile> vertical       </profile></hist>

<hist><name>    hist4          </name>
      <var>     pressure       </var>
       <var>    salinity       </var>
     <profile>  horizontal     </profile></hist>


<!-- Tecplot -->
<!-- ******* -->

<tecplot>
  <name>    CaseE5             </name>
  <title>  'E5'                </title>
  <var>     pressure           </var>
  <var>     salinity           </var>
  <var>     darcy-u            </var>
  <var>     darcy-v            </var>
  <var>     darcy-w            </var>
  <var>     permx              </var>
  <var>     permy              </var>
  <var>     gwt_fill           </var>
  <var>     gwt_gh             </var>
  <loc>     'plane y'          </loc>
</tecplot>
```

```
<!--============================================================
=============  GRID GENERATION  ==============================
============================================================-->

<!-- ggn -->
<!-- *** -->

<gridgen>
 <ggn>      gg1                    </ggn>
 <span>     1000.0  1.0   510.0    </span>
 <origin>   0.      0.   -500.     </origin>
</gridgen>

<ggn>
 <name>     gg1                    </name>
 <dxmax>    10.                    </dxmax>
 <dzmax>    10.                    </dzmax>
 <ycut>     F                      </ycut>
 <zone>     z5                     </zone>
</ggn>

<!-- Zones -->
<!-- ***** -->

<zone>
 <name>   z5          </name>
 <marker> -1          </marker>
 <dxmax>  1.          </dxmax>
 <dzmax>  1.          </dzmax>
 <mask>   high top    </mask>
</zone>

<!--============================================================
=============  OBJECTS & LOCATIONS  ==========================
============================================================-->

<!-- objects -->
<obj>
 <name>  top                   </name>
 <file>  E5top.dat             </file>
</obj>

<!-- planes -->
<loc>
 <name>      'plane y'         </name>
 <yplane>    0.5               </yplane>
</loc>

<!-- spots -->
<loc>
 <name>      'inside'          </name>
 <spot>      500. 0.5 -300.    </spot>
</loc>
```

```
<!-- profiles -->
<loc>
 <name>    'vertical'           </name>
 <zline>   500. 0.5             </zline>
</loc>


<loc>
 <name>    'horizontal'         </name>
 <xline>   0.5 0.               </xline>
</loc>

<!--============================================================
============================================================-->


</cif>
```

CIF commands and arguments

```
*=====================================================================*
*
* <blockgen>
*  <nblocks>    : number of blocks                  [1=]
*  <load>       : block id, load in percent         [0+]
*  <partfile>   : partition file name               [1-] (default='')
*  <tecplot>    : tecplot name                      [1-] (default='')
*  <binary>     : grid output file format           [1-] (default=T)
*  <gridpart>   : grid partitionning                [1-] (default=T)
*  <partition>  : file to partition                 [0+] (default='')
*  <merge>      : file to merge (no #xxx suffix)    [0+] (default='')
*
*=====================================================================*
*
* <bss>
*  <name>       : reference name               [1=]
*  <loc>        : location name                [1=]
*  <type>       : type source terms            [1-] (default='point')
*  <qsrc>       : constant source coefficient  [1-] (default=0.0)
*  <qphi>       : linear source coefficient    [1-] (default=0.0)
*  <tmin>       : starting time                [1-] (default=-oo)
*  <tmax>       : ending time                  [1-] (default=+oo)
*  <fixval>     : value to fix                 [1-]
*  <fixflu>     : flux value to fix            [1-]
*
*  The type of source terms can take the following values: 'point',
*  'volume', 'xarea', 'yarea', 'zarea', 'freezing'
*
*  When <fixval> or <fixflu> are set <qsrc> and <qphi> are
*  automatically overwritten. <type> applies for <fixflu>.
*
*=====================================================================*
*
* <cof>
*  <name>       : reference name            [1=]
*  <var>        : add a variable to output  [1=]
*  <nodes>      : nx, ny, nz                [1=]
*  <xspan>      : xmin, xmax                [1-] (default=domain xsize)
*  <yspan>      : ymin, ymax                [1-] (default=domain ysize)
*  <zspan>      : zmin, zmax                [1-] (default=domain zsize)
*  <file>       : file name                 [1-] (default='name.cof')
*  <binary>     : kind of output format     [1-] (default=T)
*  <seq>        : output time sequence      [1-]
*
*=====================================================================*
*
* <cst>
*  <g>          : default gravity acceleration   [1-] (default= 9.81)
*
*=====================================================================*
*
* <defbc>
*  <ssea>       : salinity of the sea                        [1-]
*  <tsea>       : temperature of the sea                     [1-]
*  <tland>      : temperature of the land                    [1-]
*  <pme>        : precipitation-evaporation velocity on land [1-]
*
*=====================================================================*
*
* <dgn>
*  <obj>      : obj name                          [1+]
*  <binary>   : output files format               [1-] (default=T)
*  <method>   : distance computational method     [1-] (default=1)
*  <nbsteps>  : number of iterations for method 2 [1-] (default=5)
*  <distfile> : output var file name              [1-]
*  <locfile>  : dmin, dmax, output loc file name  [1-]
*  <objfile>  : dmin, dmax, output obj file name  [1-]
*  <tecplot>  : tecplot name                      [1-]
*
```

```
*=====================================================================*
*
* <eqn>
*  <name>       : reference name                      [1=]
*  <var>        : variable name                       [1-]
*  <relin>      : linear relaxation factor            [1-] (default=1.0)
*  <nbgrad>     : number of evaluations for gradient [1-] (default=3)
*  <bss>        : bss name                            [0+]
*
*=====================================================================*
*
* <eqs>
*  <name>       : reference name                  [1=]
*  <relin>      : solution relaxation factor      [1-] (default=1)
*  <resfac>     : rms ending reduction ratio      [1-] (default=0.1)
*  <relax>      : smoother relaxation             [1-] (default=0.95)
*  <alpha>      : agglomeration threshold         [1-] (default=0)
*  <ratio>      : end coarsening ratio threshold [1-] (default=1.5)
*  <diag>       : artificial diagonal dominance   [1-] (default=0)
*  <liter>      : maximum number of cycles        [1-] (default=2)
*  <nbrelax>    : number of post-relaxations      [1-] (default=3)
*  <ipreco>     : number of MG-cycles for GMRES   [1-] (default=1)
*  <igmres>     : GMRES subspace dimension        [1-] (default=-1)
*  <nbgrid>     : maximum number of grids         [1-] (default=20)
*  <icycle>     : MG cycle type (0=V, 1=W)        [1-] (default=0)
*  <nbprer>     : number of pre-relaxations       [1-] (default=1)
*  <igms>       : smoother subspace dimension     [1-] (default=0)
*  <ilink>      : kind of agglomeration           [1-] (default=2)
*  <nbmax>      : maximum coarse cell size        [1-] (default=0)
*  <nbmin>      : minimum coarse cell size        [1-] (default=0)
*
*=====================================================================*
*
* <flux>
*  <name>       : reference name                [1=]
*  <loc>        : location name                 [1=]
*  <faces>      : type boundary faces           [1-] (default='all')
*
*  The type of boundary faces can be: 'all', 'east', 'west', 'north'
*  'south', 'high', 'low'.
*
*=====================================================================*
*
* <frame>
*  <var>     : variable name                        [1=]
*  <nalint>  : number of alpha interval             [1=]
*  <alphd>   : single rate alpha diffusion          [1-] (default=-1.0)
*  <alphl>   : low alpha limit                      [1-] (default=1.E-9)
*  <alphh>   : high alpha limit                     [1-] (default=0.05)
*  <bettot>  : global volume ratio immobile/mobile [1-] (default=1.0)
*  <frmk>    : FRAME minus late time slope          [1-] (default=1.5)
*  <iniccv>  : ccv initialization from variable     [1-] (default=T)
*
*  alphd>0 switch ON the single rate model then alphl, alphh and frmk
*  are ignored.
*
*=====================================================================*
*
* <ggn>
*  <name>       : ggn name                         [1=]
*  <nbcells>    : maximun number of cells          [1-] (default=10000000)
*  <maxlevel>   : maximun level                      [1-] (default=32)
*  <isotropy>   : isotropic cutting                  [1-] (default=T)
*  <c221>       : cross 221 rule application         [1-] (default=T)
*  <l221>       : longitudinal 221 rule application [1-] (default=T)
*  <blur>       : blurring factor                    [1-] (default=0)
*  <xcut>       : x direction cutting                [1-] (default=T)
*  <ycut>       : y direction cutting                [1-] (default=T)
*  <zcut>       : z direction cutting                [1-] (default=T)
*  <dxmax>      : maximum cell size in x direction   [1-] (default=-1)
*  <dymax>      : maximum cell size in y direction   [1-] (default=-1)
*  <dzmax>      : maximum cell size in z direction   [1-] (default=-1)
*  <zone>       : zone name                          [0+]
*
```

```
*=====================================================================*
*
* <ggr>
*  <name>     : ggr name                                [1=]
*  <ggn>      : ggn name                                [1+]
*  <tecplot>  : tecplot name                            [0+]
*  <cluster>  : zone name                               [0+]
*  <rfile>    : refinement-removal criteria file name [1-]
*
*=====================================================================*
*
* <gridgen>
*  <span>     : x, y and z domain span            [1-] (default=1,1,1)
*  <origin>   : x, y and z origin coordinates     [1-] (default=0,0,0)
*  <binary>   : grid output file format           [1-] (default=T)
*  <ggn>      : ggn name                          [0+]
*  <tecplot>  : tecplot name                      [0+]
*  <cluster>  : zone name                         [0+]
*  <ggr>      : grid generation restart name      [0+]
*
*=====================================================================*
*
* <grouting>
*  <type>   : type of grout ('silicasol', 'bingham')    [1=]
*
*=====================================================================*
*
* <gwt>
*  <relax>  : under-relaxation parameter           [1=]
*  <facmin> : minimum proterty reduction factor    [1-] (default=0.001)
*  <deltsy> : yield time cste for delayed response [1-] (default=-1)
*  <vlfrsy> : volume fraction of yield water       [1-] (default=0)
*  <betasy> : immobile to mobile volume fraction   [1-] (default=0)
*  <zmin>   : altitude above which gwt activates   [1-] (default=-oo)
*
*=====================================================================*
*
* <hist>
*  <name>       : reference name               [1=]
*  <var>        : add a variable to output     [1+]
*  <title>      : history graph title          [1-] (default='')
*  <profile>    : line or segment loc name     [1-]
*  <spot>       : spot loc name                [1-]
*  <cut>        : plane or plate loc name      [1-]
*  <palette>    : valmin, valmax, nbcolors     [1-] (default=0,0,256)
*  <showgrid>   : true for showing grid on cut [1-] (default=F)
*  <expand>     : h/w factor in cut views      [1-] (default=1.0)
*
*  When expand is negative or null DT automatically compute the value
*  for a nice canvas filling.
*
*=====================================================================*
*
* <ice>
*  <phi_fn>   : phi function ID                  [1=]
*  <phi_tl>   : liquid temperature               [1-] (default=0.0)
*  <phi_w>    : decay factor                     [1-] (default=1.0)
*  <phi_max>  : scaling value                    [1-] (default=1.0)
*  <alpha_fn> : alpha function ID                [1-] (default=1)
*  <alpha_a>  : phi exponent                     [1-] (default=5)
*  <alpha_min>: minimum reduction factor         [1-] (default=1.E-5)
*  <latent>   : latent heat of fusion            [1-] (default=3.33E+5)
*  <rho_a0>   : default ice density              [1-] (default=917.)
*  <rho_t0>   : reference temperature            [1-] (default=0.)
*  <rho_b1>   : linear temperature coefficient   [1-] (default=0.)
*  <rho_b2>   : quadratic temperature coefficient [1-] (default=0.)
*  <cp_a0>    : default ice specific mass heat   [1-] (default=2110.)
*  <cp_t0>    : reference temperature            [1-] (default=0.)
*  <cp_b1>    : linear temperature coefficient   [1-] (default=0.)
*  <cp_b2>    : quadratic temperature coefficient [1-] (default=0.)
*  <tcond_a0> : default ice thermal conductivity [1-] (default=2.14)
*  <tcond_t0> : reference temperature            [1-] (default=0.)
*  <tcond_b1> : linear temperature coefficient   [1-] (default=0.)
```

```
*  <tcond_b2> : quadratic temperature coefficient [1-] (default=0.)
*  <store>    : ice_xxx variable name to store    [0+]
*
*=====================================================================*
*
* <indg>
*  <infile>  : input file name  [1-]
*
*=====================================================================*
*
* <law_cp>
*  <a0>       : default fluid specific mass heat [1-] (default= 4182.)
*  <a1>       : linear salinity coefficient      [1-] (default= 0.13)
*  <a2>       : quadratic salinity coefficient   [1-] (default=-0.0001)
*  <store>    : storage trigger                  [1-] (default= F)
*
*=====================================================================*
*
* <law_mu>
*  <a0>         : default dynamic viscosity     [1-] (default=1.78E-3)
*  <t0>         : reference temperature              [1-] (default= 0)
*  <a1>         : linear salinity coefficient        [1-] (default= 0)
*  <a2>         : quadratic salinity coefficient     [1-] (default= 0)
*  <b1>         : linear temperature coefficient     [1-] (default= 0)
*  <b2>         : quadratic temperature coefficient  [1-] (default= 0)
*  <n>          : exponent value                     [1-] (default=-1)
*  <store>      : storage trigger                    [1-] (default= F)
*
*=====================================================================*
*
* <law_rho>
*  <a0>       : default fluid density              [1-] (default=1000)
*  <t0>       : reference temperature              [1-] (default=0)
*  <a1>       : linear salinity coefficient        [1-] (default=0.008)
*  <a2>       : quadratic salinity coefficient     [1-] (default=0)
*  <b1>       : linear temperature coefficient     [1-] (default=0)
*  <b2>       : quadratic temperature coefficient  [1-] (default=0)
*  <store>    : storage trigger                    [1-] (default=F)
*
*=====================================================================*
*
* <law_tcond>
*  <a0>       : default fluid therma conductivity [1-] (default= 2.14)
*  <t0>       : reference temperature             [1-] (default= 0.)
*  <b1>       : linear temperature coefficient    [1-] (default= 0.)
*  <b2>       : quadratic temperature coefficient [1-] (default= 0.)
*  <store>    : storage trigger                   [1-] (default= F)
*
*=====================================================================*
*
* <loc>
*  <name>       : reference name            [1=]
*  <type>       : location type             [1-] (default='cell')
*  <marker>     : marker type and value     [1-] (default='cell',-1)
*  <not>        : locname                   [0+]
*  <patch>      : xmin, xmax, ..., zmax     [1-]
*  <xplane>     : x                         [1-]
*  <yplane>     : y                         [1-]
*  <zplane>     : z                         [1-]
*  <xplate>     : x, ymin, ymax, zmin, zmax [1-]
*  <yplate>     : y, xmin, xmax, zmin, zmax [1-]
*  <zplate>     : z, xmin, xmax, ymin, ymax [1-]
*  <xline>      : y, z                      [1-]
*  <yline>      : x, z                      [1-]
*  <zline>      : x, y                      [1-]
*  <xseg>       : y, z, xmin, xmax          [1-]
*  <yseg>       : x, z, ymin, ymax          [1-]
*  <zseg>       : x, y, zmin, zmax          [1-]
*  <spot>       : x, y, z                   [1-]
*  <dplane>     : depth                     [1-]
*  <dplate>     : d, xmin, xmax, ymin, ymax [1-]
*  <file>       : filename                  [1-]
*
```

```
*  marker   type : 'cell', 'east', 'west', ... , 'low', 'face'
*  location type : 'cell', 'bndface'
*
*=====================================================================*
*
* <loop>
*  <name>       : reference name                  [1=]
*  <nbit>       : number of iteration             [1-] (default=0)
*  <eqs>        : equation set to solve           [0+]
*  <usrloop>    : user loop to run                [0+]
*  <hist>       : history file to write           [0+]
*  <rof>        : result output file to write     [0+]
*  <cof>        : cartesian output file to write  [0+]
*  <tof>        : tunnel output file to write     [0+]
*  <tecplot>    : tecplot file to write           [0+]
*
*=====================================================================*
*
* <mat>
*  <name>            : reference name           [1=]
*  <density>         : density                  [1-] (default=law_rho%a0)
*  <viscosity>       : dynamic viscosity        [1-] (default=law_mu%a0)
*  <bingham_stress>  : Bingham yield stress     [1-] (default=0.0)
*  <bingham_espilon> : Bingham regularisation   [1-] (default=1.E-4)
*  <diffusivity>     : diffusivity coefficient  [1-] (default=0.0)
*  <schmidt>         : Schmidt number           [1-] (default=0.7)
*  <schmidt_turb>    : Turbulent Schmidt number [1-] (default=0.7)
*  <gel>             : silicasol A, Tg, Vmax    [1-]
*
*=====================================================================*
*
* <mrpow>
*  <nalint>  : number of alpha interval (=naintt)     [1-] (default=0)
*  <alphl>   : low alpha limit                        [1-] (default=0)
*  <alphh>   : high alpha limit                       [1-] (default=0)
*  <ak>      : late time slope of breakthrough curve  [1-] (default=0)
*  <bettot>  : volume ratio value (immobile/mobile)   [1-] (default=0)
*  <fxdift>  : uniform cross diffusion coefficient    [1-] (default=0)
*  <retmob>  : retardation mobile                     [1-] (default=0)
*  <nmobst>  : number of a/d states in mobile zone    [1-] (default=0)
*  <naintn>  : number of a/d states in immobile zone  [1-] (default=0)
*
*=====================================================================*
*
* <obj>
*  <name>     : reference name                        [1=]
*  <file>     : object input file name                [1-]
*  <refine>   : refinement criteria (>0.5)            [1-]
*  <brick>    : xmin, xmax, ymin, ymax, zmin, zmax    [1-]
*  <cylinder> : x1, y1, z1, x2, y2, z2, r, nbs        [1-]
*  <sphere>   : x1, y1, z1, r, nbs                    [1-]
*  <plate>    : x1, y1, z1, x2, y2, z2, x3, y3, z3    [1-]
*  <line>     : x1, y1, z1, x2, y2, z2                [1-]
*  <spot>     : x1, y1, z1                            [1-]
*  <user>     : eltsize, nbelts, nbnodes              [1-]
*  <move>     : dx, dy, dz                            [1-]
*  <dilat>    : dilx, dily, dilz                      [1-]
*  <rot>      : degx, degy, degz                      [1-]
*
*  One of the argument <file>, <brick>, <cylinder>, <sphere>,
*  <plate>, <line>, <spot> or <user> must be present.
*
*=====================================================================*
*
* <ogn>
*  <stl>     : stl name                    [0+]
*  <outfile> : object output file name     [1-]
*  <binary>  : output file format          [1-] (default=T)
*  <fusion>  : tolerance for fusing vertex [1-] (default=0.0)
*  <tecplot> : tecplot output file name    [1-]
*  <obj>     : input obj name              [1-]
*  <scale>   : roughness scale             [1-] (default=0.0)
*  <hurst>   : fractal Hurst exponent      [1-] (default=0.9)
```

```
*  <level>   : refinement level             [1-] (default=0)
*  <seed>    : random generator seed        [1-] (default=0.0)
*  <inidev>  : initial random deviation     [1-] (default=.false.)
*
*=====================================================================*
*
* <partrack>
*  <method>  : method (1 or 2)            [1=]
*  <seed>    : random generator seed      [1-] (default=clock time)
*  <traj>    : nbtrj,ntrjfr,lentrj,dltrj  [1-] (default=0,1,0,0.0)
*  <steph>   : tsteph, wnstph(1-3)        [1-] (default=1.E10,.1,.1,.1)
*  <nolond>  : no longitudinal dispersion [1-] (default=T)
*  <adsorp>  : adsorption                 [1-] (default=F)
*  <reverse> : reverse flow moving        [1-] (default=F)
*  <sloc>    : (nb particles, locname, tmin, tmax) [0+]
*  <eloc>    : loc name                          [0+]
*  <locfile> : file for reading sloc             [1-]
*
*=====================================================================*
*
* <rof>
*  <name>       : reference name             [1=]
*  <file>       : file name                  [1-] (default='name.rof')
*  <binary>     : kind of output format      [1-] (default=T)
*  <double>     : 8 bytes reals outputs      [1-] (default=F)
*  <seq>        : output time sequence       [1-]
*  <var>        : add a variable to output   [0+]
*
*=====================================================================*
*
* <rst>
*  <binary>     : kind of output format      [1-] (default= T)
*  <var>        : add a variable to output   [0+]
*
*=====================================================================*
*
* <run>
*  <title>      : title of the run           [1-] (default='')
*  <licence>    : licence string             [1-] (default='')
*  <gridfile>   : grid file name             [1-] (default='xyz')
*  <nbthreads>  : number of threads to use   [1-] (default=0)
*
*=====================================================================*
*
* <screen>
*  <error>   : error                     [1-] (default=T)
*  <head>    : header                    [1-] (default=T)
*  <cif>     : cif commands              [1-] (default=F)
*  <algo>    : algorithm checkup         [1-] (default=F)
*  <checkup> : checkup                   [1-] (default=F)
*  <init>    : initialization progress   [1-] (default=F)
*  <ggn>     : ggn progress              [1-] (default=T)
*  <stats>   : grid statistics           [1-] (default=T)
*  <slv>     : slv progress              [1-] (default=T)
*  <end>     : ending progress           [1-] (default=T)
*  <chkp>    : check point status        [1-] (default=F)
*  <all>     : apply to all triggers     [1-]
*
*=====================================================================*
*
* <seq>
*  <name>   : sequence name                [1=]
*  <step>   : step value                   [0+]
*  <time>   : time value                   [0+]
*
*=====================================================================*
*
* <slv>
*  <restart>    : restart from rst file      [1-] (default=F)
*  <reset>      : reset time and step to 0   [1-] (default=F)
*
```

```
*=====================================================================*
*
* <spt>
*  <s0>         : salinity for z=0              [1-] (default=0)
*  <t0>         : temperature for z=0           [1-] (default=0)
*  <s1>         : salinity for z>0              [1-] (default=0)
*  <t1>         : temperature for z>0           [1-] (default=0)
*  <dwt>        : depth of water table (>0)     [1-] (default=0)
*  <dsdz>       : salinity gradient (<0)        [1-] (default=0)
*  <dtdz>       : temperature gradient (<0)     [1-] (default=0)
*
*=====================================================================*
*
* <stl>
*  <name>    : reference name                  [1=]
*  <file>    : input file name                 [1-] (default='name'.stl)
*  <pos>     : dx, dy, dz displacements        [1-] (default=0,0,0)
*  <rot>     : degx,degy,degz rotation angles  [1-] (default=0,0,0)
*
*  Rotations are applied in the x->y->z order around the minimum corner
*  of the bounding box. They are anticlockwize in degree.
*
*=====================================================================*
*
* <tecplot>
*  <name>       : reference name             [1=]
*  <file>       : file name                  [1-] (default='name.plt')
*  <title>      : tecplot title              [1-] (default='')
*  <seq>        : output sequence name       [1-]
*  <loc>        : add a location zone                                 [0+]
*  <obj>        : add a tecplot zone made of the given object         [0+]
*  <traj>       : add a tecplot zone made of the given trajectory     [0+]
*  <swarm>      : add swarm of all trajectory particles               [0+]
*  <var>        : add a variable to output                            [0+]
*  <action>     : output variable action (varname, action, value)     [0+]
*
* The possible actions are strings among: '+', '*', 'log10' and 'ln'.
* When 'log10' and 'ln' are specified the action value may be omitted.
*
* When the trajectory index specified by <traj> is 0 all trajectories
* are added.
*
* Allowed locations are 'domain' and planes
*
*=====================================================================*
*
* <time>
*  <order>      : order of time discretisation [1-] (default=0)
*  <dt>         : time step                    [1-] (default=1.D+20)
*  <t0>         : initial time                 [1-] (default=0)
*  <zone1>      : n1, t1                       [1-] (default=0, t0)
*  <zone2>      : n2, t2                       [1-] (default=0, t1)
*
*=====================================================================*
*
* <tof>
*  <name>       : reference name             [1=]
*  <file>       : file name                  [1-] (default='name.tof')
*  <binary>     : kind of output format      [1-] (default=T)
*  <seq>        : output time sequence       [1-]
*  <var>        : add a variable to output   [0+]
*  <tunsec>     : add a section marker       [0+]
*
*=====================================================================*
*
* <tracks>
*  <file>    : file name for output    [1=]
*  <binary>  : kind of output format   [1-] (default=T)
*  <ievent>  : event to write in file  [0+] (from 1 to 4)
*  <var>     : variable name           [0+]
*
```

```
*=======================================================================*
*
* <tunsec>
*  <marker> : marker value                        [1=]
*  <sink>   : mass withdrawal (kg/s)              [1-] (default=0.0)
*  <skmax>  : maximun skin factor                 [1-] (default=1.E+10)
*  <skmin>  : minimun skin factor                 [1-] (default=1.E-10)
*  <relax>  : under relaxation parameter          [1-] (default=0.0)
*  <skin>   : initial skin factor                 [1-] (default=1.0)
*  <resat>  : resaturated status                  [1-] (default=F)
*  <permax> : fixed wall permeability             [1-]
*  <grout>  : laymin, laymax, pmin, pmax, pval    [0+]
*
*=======================================================================*
*
* <var>
*  <name>       : reference name          [1=]
*  <pos>        : cell position           [1-] (default='none')
*  <nbsteps>    : number of steps (1-3)   [1-] (default=1)
*  <nbvals>     : number of values (>0)   [1-] (default=1)
*  <blank>      : blanking output value   [1-] (default=-1.0)
*  <inifile>    : file name               [1-]
*  <ini>        : locname, val, vm1, vm2  [0+] (default='domain',0,0,0)
*  <ini>        : locname, val, vm1       [0+]
*  <ini>        : locname, val            [0+]
*
*  pos = 'cell', 'xface', 'yface' or 'zface'
*  locname is the location name. locname=domain fills the entire arrays
*  val is the value to fill the var%val array
*  vm1 is the value to fill the var%vm1 array i.e. step n-1
*  vm2 is the value to fill the var%vm2 array i.e. step n-2
*
*=======================================================================*
*
* <zone>
*  <name>    : zone name    [1=]
*  <mask>    : (maskid, object name)            [1+]
*  <marker>  : cell marker ((<=0 for removal)   [1-] (default= 1)
*  <dxmax>   : maximum cell size in x direction [1-] (default=-1)
*  <dymax>   : maximum cell size in y direction [1-] (default=-1)
*  <dzmax>   : maximum cell size in z direction [1-] (default=-1)
*  <blur>    : blurring factor                  [1-] (default= 0)
*  <operand> : mask operand (or, and)           [1-] (default='or')
*
*  maskid = BORDER, EAST, WEST, NORTH, SOUTH, HIGH, LOW, INSIDE, OUTSIDE
*
*=======================================================================*
```