Talisman 4.0

A finite volume–finite element tool for numerical simulation of subsurface flow and contaminant transport with a posteriori error control and adaptive mesh refinement

Developer's manual

Libor Inovecký[1] & Martin Vohralík[2]

February 28, 2007

[1] AZPIRO s.r.o.
Tulipánová 2805–10, 10106 Prague, Czech Republic
`liburda@centrum.cz`

[2] Laboratoire Jacques-Louis Lions
Université Pierre et Marie Curie (Paris 6)
175 rue du Chevaleret, 75 013 Paris, France
`vohralik@ann.jussieu.fr`

Contents

1 Introduction **3**
- 1.1 Directory structure . . . 3
- 1.2 Project setup . . . 3
- 1.3 Using CodeGuard for memory checking . . . 4

2 Class THeader **4**

3 Finite volume mesh **4**
- 3.1 Mesh-related classes . . . 4
- 3.2 Cell parameters . . . 4
- 3.3 Cell subdivisions . . . 5

4 Imposed conditions **5**

5 Linear algebra classes **6**

6 Mesh and conditions visualization classes **7**

7 Other important classes **7**

8 Source documentation using DoxyGen **7**

9 Some rules of development **8**

1 Introduction

This is the developer's guide to the Talisman project and will give you basic overview of the project and especially code structure. Talisman is written in C++, namely Borland C++ version 5.01.

1.1 Directory structure

The whole project is contained in the directory `Talisman`. In this directory you will find the following sub-directories:

Doc Contains the documentation for Talisman. The source documentation is in `DoxyGen/Html`, where you must open the file `index.html`.

Include Contains the header files wih extension .h, file `wnflwapp.rh`, which contains the definition of some constants necessary for the resources, and file `params.def`, which contains definition of cell parameters.

Object Contains the object files, produced during compilation.

Resource Contains the resource files, namely the icons used in the menu and file `talisman.rc`, which is the resource file with the description of the menu, dialog boxes, etc.

Src Contains the source files with extension .cpp.

The directory `Talisman` also contains two files, file `talisman.ide`, which is the project file for Borland C++, and file `doxyfile.txt`, which serves for the source code documentation using DoxyGen (see Section 8 for details).

1.2 Project setup

You can open the project using *Project/Open project* in Borland C++ development environment. Before you can compile the project, you must set the project directories. This is done in *Options/Project/Directories*. The directories relevant to Talisman are fine, since they are set relative to the `Talisman` directory. However, you must set the correct (absolute) path to Borland C++ include and libraries directory.

When you debug your code, it is useful to set debugging options on in *Options/Project/Compiler/Debugging*, especially 'Debug information in OBJs' and in *Options/Project/Linker/General* 'Include debug information' and disable all optimisations in *Options/Project/Optimizations*.

On the other hand, when you build the release version of Talisman, it is good to disable all debugging information and enable some optimizations (optimize for speed will be the preferred option). After you switch from debug to release version or vice-versa, you must rebuild the whole project using *Project/Build all*.

Additionally, for the release version, you should add the definition NDEBUG in *Options/Project/Compiler/Defines*, which will disable the assert macros. In the development stage, asserts can be very helpful, so do not forget to delete the NDEBUG definition for debugging purposes. Try to use asserts wherever possible to avoid silly programmer's mistakes like NULL pointer dereference, invalid parameter values passed to functions etc.

1.3 Using CodeGuard for memory checking

CodeGuard is a software supplied with Borland C++ version 5.01 which serves for detection of invalid memory access, memory leaks, and some other common mistakes. Although it is a separate product, it integrates very well with the Borland development environment.

In order to enable CodeGuard checking, you must enable everything in *Options/Project/CodeGuard*, then open the project with list of all project files and in the popup menu for `talisman.exe` you choose *Target expert* and in section *Libraries* you must enable CodeGuard. With all this set you must completely rebuild the project. Finally, you must enable CodeGuard in the runtime using the menu option *Tools/CodeGuard config*. Now you are all set and you CodeGuard will inform you of the detected memory problems in the *Messages* window.

2 Class THeader

The `THeader` class is the central class of Talisman as it groups together all the mathematical properties of the simulation. It contains the mesh, the conditions at all time layers and some other data. It also contains the methods for data import/export in various formats, functions that take care of the matrix assembly, functions that handle the conditions and many more.

3 Finite volume mesh

3.1 Mesh-related classes

The mesh in Talisman consists of several horizontal layers with each layer consisting of several columns and rows of cells. The dimension of the columns and of the rows is the same in all layers.

The classes relevant to the meshing structure are `TMesh`, `TLayer` and `TCell`. `TMesh` contains all the layers and the dimensions of all columns and rows. `TLayer` contains a list of all cells that belong to one layer and some additional parameters. `TCell` represents one cell of the layer. The cells can be further divided into subcells, which is discussed below in more detail. Each cell contains a list of its neighbours (horizontal and vertical) and a list of its subdivisions. The cell subdivisions are also of type `TCell`. `TCell` also contains the physical properties of the media that is enclosed in the cell.

3.2 Cell parameters

With the exception of some basic variables, the cell parameters are grouped into structures, which are indirectly referenced with pointers, so that each group is allocated only when necessary (in order to save memory space). As of now, the cell parameters' groups are:

ParamGeom Contains parameters related to the geometry of the cell.

ParamFlow Contains parameters related to the fluid flow in general and to the the Dupuit and Hantush approximation.

ParamFlowRich Contains parameters related to the fluid flow in the Richards approximation.

ParamTrans Contains parameters related to the transport of contaminants.

ParamApost Contains parameters related to the a posteriori calculations and estimates.

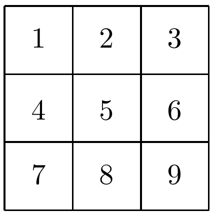

Figure 1: Numbering of the possible positions of subcells

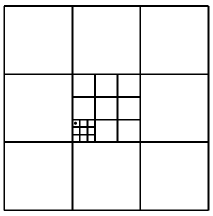

Figure 2: Cell with position 175 is identified with point

All these groups and parameters are defined in the file `param.def`, which is located in the `Include` directory. The definition is done using macros. For example

```
DEFINE_PARAM (double, perm, MP_PERM, 1e-3)
```

will define variable perm (the permeability of the media) of type double, with the initial value of 0.001 and macro MP_PERM, which is used to access the variable using the `TCell` methods SetValue and GetValue. These methods are mainly used to access the cell parameters from class `TTableur` (see below). Otherwise you can access the variable directly. The parameters which are not set or accessed by the user from the tableur and are used only for internal purposes of the solver should be defined using

```
DEFINE_PARAM_SIMPLE (double, resupw, 0.0)
```

where the macro identifier for SetValue/GetValue is omitted.

3.3 Cell subdivisions

Each basic cell can be divided into 3x3 equal rectangular cells. Each cell division can be again divided into 3x3 subcells and this process can continue up to 9 levels of subdivisions, which means that the basic cell can be divided to 9^9 cells. The reason for limiting cell subdivisions to 9 levels will become clear from the following.

The cell subdivisions on the first level are adressed by the column and row of the parent basic cell and by an integer number between 1 and 9, which uniquely determines which of the 9 possible cells we mean. Figure 1 shows clearly how the subdivisions are numbered.

Now, to determine the position of a subdivision on second level, we must know its position of the cell in the parent on the first level, and the parent's position in the basic cell. These numbers are appended into one integer number (often referred to as "position" in the code). This process is repeated for all levels between the subdivision and the basic cell. The last decimal digit determines the position of the first-level parent in the basic cell, the second digit the position of the second-level parent in the first-level parent and so on. Figure 2 gives an example of a cell and the integer that determines its position. The basic cells have their position integer equal to 0.

Since the position is contained in parameter of type `unsigned int`, which is a 32-bit unsigned integer with the maximum value of 4'294'967'295, we cannot address more than 9 levels of subdivisions. In more modern compilers, 64-bit integer numbers are available, so the passage to more (precisely 19) levels of subdivisions will easily be possible.

4 Imposed conditions

The conditions are in Talisman imposed for each period separately. The class `TStress` holds all the information about one time period: number, duration, number of time steps, and also the lists

Conditions						
TStress		TMCond	TCell			
ID	list	value	loc. ID	loc. var.	ID	variable
SP_WELL	fwelllist	SP_VAL1	MP_WELL	well	MP_SOUR	sour
SP_RECH	frechlist	SP_VAL1	MP_ZRECH	zrech	MP_RECH	rech
SP_PIEZIMP	fpiezimplist	SP_VAL1	MP_PIEZIMP	piezimp	MP_PIEZ	piez
SP_CONCIMP	fconcimplist	SP_VAL1	MP_CONCIMP	concimp	MP_CONC	conc
SP_CONCNEU	fconcneulist	SP_VAL1	MP_CONCNEU	concneu	MP_CONCFLUX	concflux
SP_WELL	fwelllist	SP_VAL2	MP_WELL	well	MP_CONCSOUR	concsour

Table 1: Imposing conditions (top to bottom): flow sources, recharge, Dirichlet BC's for flow, Dirichlet BC's for transport, flux BC's for transport, sources concentrations

of conditions imposed on this time period. Each list contains conditions of one certain type (list of flow sources, list of recharge zones, list of flow Dirichlet boundary conditions, list of transport flux boundary conditions, and sources concentrations actually).

The conditions themselves are of type `TMCond`. Certain conditions apply to individual cells; in such case, `TMCond` contains the layer, column, row, and position of the corresponding cell plus a pointer to this cell. Other conditions apply to a zone of cells. In this case, `TMCond` contains the number of zone to which it is applied (in the variable that normally corresponds to the layer number). One condition can hold up to three real values (of type double). This is useful, for example, in order to fully define the sources of contaminants, where one needs to know the source supply-rate and the source concentration.

The specification of the conditions by the user is done with the aid of the class `TEditStress`, see Section 6 below; Talisman here shows a table given by all cells which have a localization variable for the given condition and all time periods. One incorporates the conditions during the Talisman execution as follows: on each time period, the function `StressPeriodConditions` is called. If one needs to know the value of a condition at the end of the given period, stored values are directly used. If, in the time adaptive mode, one needs to know a value of a condition for a time which is in the first (period length)/`TIMEINTERP` part of the period, interpolation with the conditions specified on the previous period (zero values for the first period) is used; in the rest of the time period, again stored values are directly used. Finally, in `StressPeriodConditions`, the resulting values of the conditions for the desired time are directly assigned to the corresponding cell variables. For a detailed list of conditions actually used in Talisman and the relations between the classes `TStress`, `TMCond`, and `TCell` concerning the conditions, see Table 1.

5 Linear algebra classes

`TVecteur` is a class that represents a real vector, `TMatrice` represents a sparse matrix and is stored in the compressed row format (there is and array of the nonzero matrix values stored row by row, and array of columns corresponding to these values and an array that stores the beginning of each row in the values array). The matrix rows are filled in the order from the first row to the last using the function `SetLine`. The function `SetLine` can take `TVecteur` as an argument (in such case the zero values are omitted, of course). Another possibility is to use `TMatriceRow` as an argument, which is a class similar to `TVecteur`, but holds only the non-zero elements. The elements of

TMatriceRow are created and accessed using the indexing operator [] just like for TVecteur.

One important convenience of these classes is that you can change the dimension without having to reallocate the memory every time, only when the required size is larger than the currently allocated capacity, the memory is reallocated. The dimension is changed using the function ChangeDim.

Another class related to the linear algebra is the class TAlgo, which contains the linear system solver (Bi-CGStab algorithm) and some preconditioners, as well as the solver parameters (tolerance, maximum number of iterations, etc.). If you plan to add new solver algorithm or preconditioning routine to Talisman, this is the place to do it.

6 Mesh and conditions visualization classes

Class TTableur serves for visualization of the cell parameters and class TEditStress for visualization of imposed conditions. Class TBasicTableur is their common parent and implements the common functionalities of the two. This is the reason why TCell and TMCond are derived from one common parent TBasicObj.

These classes are the most complicated of the graphical user interface. Normally, there should be no need to modify these classes. If the opposite is true, be warned that many of the properties and methods have influence on one another, so do everything very carefully and check that the tableur works correctly after your changes. Especially if you change something in the base class TBasicTableur, check both TTableur and TEditStress afterwards.

7 Other important classes

The main application class is WinFlowApp, which has the handlers for most commands of the menu. WinFlowMDIClient is a class that is responsible for opening the correct tableur (function CmMeshEditor) or stress editor (CmHistorique), when you select the corresponding command in the menu. If you will add some parameter to TCell and you will want to allow the user to modify or visualize it in a tableur, you will have to modify one of the mentioned routines.

Most other class that were not explicitly mentioned in this document represent the various dialogs present in Talisman. Their meaning is described in the DoxyGen documentation.

8 Source documentation using DoxyGen

The Talisman source code is documented using the free software DoxyGen. This software can be downloaded at www.doxygen.org. It extracts special comment blocks, which begin with "//!" and are usually placed in the header files. Therefore, the documentation is written in the header files and in addition, DoxyGen produces documentation in the HTML format (other formats, including LaTeX, are also available).

The documentation produced by DoxyGen is by default placed in the directory Talisman\Doc \Html and can be viewed by opening the file index.html in any web-browser. The recommended way to use DoxyGen under MS Windows is to use the Doxywizard (shiped with the DoxyGen installation files for Windows) to open the file Talisman\doxygen.txt, set the base directory of the project (e.g. d:\Talisman), save the file and then run the processor.

You should try to document all files and all class members—data members and methods. For complicated methods you should also document the parameters and return value. Here is an example:

```
  //! Calculate the column of the given position in
  //! the 3x3 subdivision.
  //! \param pos Position in the subdivision (1 <= pos <= 9).
  //! \return Column corresponding to the position (1, 2, or 3).
  static uint GetXFromPos (uint pos);
```

The file documentation should consist of a very brief description of what the file contains and looks like this:

```
//! \file
//! Definition of class TMesh (finite volume mesh of geometry).
```

File documentation should be placed in both header files (.h) and source files (.cpp).

9 Some rules of development

Here are some rules that you should obey when developing new code in Talisman, in order to keep it well organized and clear.

1. Document any new class, file or function that you add to the code as soon as you finish writing it.
2. Try as hard as you can to limit the line length to 80 characters.
3. Align the code correctly. The blocks are always indented 2 spaces from the parent block.
4. Never (!) insert tabelator characters into the source. For someone who has set a different width of tab character, the code will be terribly misaligned. If you use the Borland IDE, the tabs are disabled in menu *Options/Environment/Editor/Options*, where you disable 'Use tab character'.
5. Try to keep the code simple and understandable. If you do something non-trivial, add a comment with some explanation.
6. Use CodeGuard or whatever tool you wish to prevent memory leaks and invalid memory access.
7. Use asserts wherever they are appropriate.