ccfd

v1.0

Generated by Doxygen 1.9.1

Wed Jan 20 2021 08:09:08

Chapter 1

Main Page

ccfd

The ccfd code is a drop-in replacement for cfdfv, a CFD code written in Fortran by the Institute of Aerodynamics and Gas Dynamics at the University of Stuttgart for a CFD programming course. This program uses CGNS, for storing the calculation results.

ccfd is a two-dimensional finite volume solver for the Euler- or Navier-Stokes-equation systems on structured or unstructured grids. It can perform 1st or 2nd order spacial and temporal discretization, with explicit Euler or Runge-Kutta integration, or implicit Krylov subspace iteration. The code runs on multiple threads using OpenMP parallel for loops, if compiled with -fopenmp.

 

1.1 Dependencies

- git
- gcc, or any other C compiler
- make
- cmake
- gnuplot (optional, for displaying calculation residuals, available here)
- gmsh (optional, for mesh generation, available here)
- paraview (optional, for post-processing the results, available here)

1.2 Installation

The installation process is easiest on Linux, but possible on MacOS and Windows.

1.2.1 Linux

First make sure that all necessary dependencies are all installed. These can usually be obtained through your distributions package manager, on Arch based systems the following command should suffice
```
# pacman -S git base-devel cmake
```

For an Ubuntu based system the following command should be enough
```
# apt-get install git build-essential cmake libomp-dev
```

Next, navigate to the directory where you want to keep `ccfd`, clone the git repository and compile the code
```
$ cd path/to/directory
$ git clone https://github.com/hhh95/ccfd.git
$ cd ccfd
$ make
```

There should now be two new folders, `obj` and `bin`, the last one containing the `ccfd` executable.

You can test the compiled binary file, by executing the following command
```
$ make check
```

This will execute `ccfd` in the directory `check` on some small cases that test specific functions of the program.

Continue with `Usage`.

1.2.2 MacOS

I only had access to MacOS High Sierra, so some things might have changed, but the general procedure should still be the same on any MacOS version. First, install a package manager that can install all the necessary software for you. I suggest `Homebrew`. Head on over to their website and follow the installation instructions. Once you're done, install the necessary software to compile `ccfd`
```
$ brew update
$ brew upgrade
$ brew install git gcc make cmake libomp
```

Due to the fact, that on MacOS `gcc` is linked to `clang` per default, which does not work well with OpenMP, you will have to make a minor edit to the `Makefile`. Open the file with your favorite editor and find the line that defines the C compiler to be used
```
CC      = gcc
```

Replace `gcc` with the version that you have installed, most likely it is `gcc-9`. To find out which version you have, do the following
```
$ ls /usr/local/bin/gcc*
```

Next, navigate to the directory where you want to keep `ccfd`, clone the git repository and compile the code
```
$ cd path/to/directory
$ git clone https://github.com/hhh95/ccfd.git
$ cd ccfd
$ make
```

There should now be two new folders, `obj` and `bin`, the last one containing the `ccfd` executable.

You can test the compiled binary file, by executing the following command
```
$ make check
```

This will execute `ccfd` in the directory `check` on some small cases that test specific functions of the program.

Continue with `Usage`. When installing ParaView, do not choose the Linux version, but the MacOS version.

When using `ccfd` it is very handy to have the ability to open a terminal in a folder from Finder. This is possible, but has to be activated first. Go to *System-settings->Keyboard->Services* and then check the box in front of the option *New Terminal at Folder*. Now, you can right click a folder in Finder and open a terminal in that folder.

1.2.3 Windows

I am still working on installing it on Windows directly, but have not yet managed to do so. For now it only works with a `Linux Bash shell`. Get the latest Ubuntu shell and complete the installation process. Next, start the Ubuntu shell and install the necessary utilities (if you have never used Linux before, `$` in front of a command means the command can be executed as a normal user and `#` in front of a command means, you need administrative rights; these can be obtained by typing `sudo` in front of the command and entering the password)

```
# apt update
# apt upgrade
# apt install git build-essential cmake libomp-dev
```

Now change to your desired working directory

```
$ git clone https://github.com/hhh95/ccfd.git
$ cd ccfd
$ make
```

There should now be two new folders, `obj` and `bin`, the last one containing the `ccfd` executable.

You can test the compiled binary file, by executing the following command

```
$ make check
```

This will execute `ccfd` in the directory `check` on some small cases that test specific functions of the program.

After everything is set up, continue with `Usage`. However, when installing ParaView, do not install it in the Ubuntu shell, but rather install it normally for `Windows`. When you want to access the files created by `ccfd` from Windows, just type the following into the Ubuntu shell

```
$ explorer.exe .
```

This will open the directory in the Windows explorer and you can easily access the file with ParaView.

If, after trying to open ParaView, you get an error that a 'VCOMP140.DLL' library is missing, follow the explainatins of this `Forum Post`.

1.3 Usage

As a first step you should add the `ccfd` executable to your path. You can do so by running

```
$ source ccfdrc
```

Next, navigate to the `calc` folder. Here you will find example case files, contained in folder. First, try the Riemann problems. Navigate to the `riemann` folder with

```
$ cd riemann
```

Here you will find the SOD test case, as well as different versions of the case. Start a calculation with

```
$ ccfd sod.ini
```

and observe the output. There should be four new files. The initial condition of the case and the calculation results at t = 0.25 s. They should all be `.csv` files. You can examine them with any spreadsheet program you like. Alternatively you can use ParaView, a free post-processing program, that can visualize 1D, 2D, and 3D data. For Arch-based distributions you can install it from the package manager

```
# pacman -S paraview
```

On Ubuntu, the ParaView program in the repositories does not read CGNS files correctly for some reason. You will need to download ParaView 5.8 from the `ParaView website`. Next do the following

```
$ cd folder/where/you/downloaded/paraview
$ tar -xvf ParaView-5.8.0-MPI-Linux-Python3.7-64bit.tar.gz
# mv ParaView-5.8.0-MPI-Linux-Python3.7-64bit.tar.gz /opt
$ echo "export PATH:$PATH:/opt/ParaView-5.8.0-MPI-Linux-Python3.7-64bit/bin" » ~/.bashrc
```

Next open ParaView in the directory where you performed the calculations

```
$ cd path/to/ccfd/calc/riemann
$ paraview &
```

Now, click on *File*->*Open* and select both sets of `.csv` files. Because the results are 1D data, you need to change from *Render View* to *Line Chart View*. In the top right of the viewing area, click on the `X` button. Now select *Line Chart View* from the list. You should now see an empty grid with an x-, and a y-axis. In the *Pipeline Browser* to the left, click on the eye icons in front of the loaded files. A plot should appear on the axis grid. It will probably show the initial state. In the top bar, click on the play button. Now, the final state should be shown. You will see the analytical, or exact, solution, as well as the numerical solution.

For more information on the theory, maybe have a look at `Wikipedia`.

The procedure for running the other cases is the same. However, if the solution data is 2D, then you do not need to switch to *Line Chart View*. The 2D CGNS output files will usually have more than just the solution file. You can load everything at once by selecting the file that has `_Master` in its name. After loading the file, select all *Cell Arrays* in the *Pipeline Browser* and click on *Apply*. Then you can look at the different fields of the solution, by selecting them in the top bar (where it first says *Solid Color*).

Some files can only be run with the Navier-Stokes equations. In order the switch between Euler and Navier-Stokes equations, open the `Makefile` and change the `EQNSYS` parameter.

Chapter 2

Bug List

Global flux_ausmdv (double rhoL, double rhoR, double vxL, double vxR, double vyL, double vyR, double pL, double pR, double fluxLoc[4])

This function produces incorrect output, refrain from using it for the time being

Chapter 3

Data Structure Index

3.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 4

File Index

4.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 5

Data Structure Documentation

5.1 boundary_t Struct Reference

Structure that holds the information of a boundary condition.

```
#include <boundary.h>
```

Collaboration diagram for boundary_t:

Data Fields

- int BCtype
- int BCid
- int exactFunc
- double pVar [NVAR]
- bool isAdiabatic
- bool isTemperaturePrescribed
- double temperature
- double heatFlux
- double * connection
- boundary_t * next

5.1.1 Detailed Description

Structure that holds the information of a boundary condition.

5.1.2 Field Documentation

5.1.2.1 BCid

`int boundary_t::BCid`

boundary condition ID

Referenced by cgnsWriteMesh(), connectPeriodicBC(), createMesh(), initBoundary(), and initWing().

5.1.2.2 BCtype

`int boundary_t::BCtype`

boundary type:

- 1: slip wall
- 2: Navier-Stokes wall
- 3: supersonic inflow
- 4: supersonic outflow
- 5: characteristic
- 6: exact solution
- 7: periodic boundary condition
- 8: pressure outflow

Referenced by boundary(), cgnsWriteMesh(), connectPeriodicBC(), createMesh(), initBoundary(), and initWing().

5.1.2.3 connection

`double* boundary_t::connection`

connection coordinates for periodic BC

Referenced by connectPeriodicBC(), and initBoundary().

5.1.2.4 exactFunc

`int boundary_t::exactFunc`

exact boundary function identifier

Referenced by boundary(), and initBoundary().

5.1.2.5 heatFlux

```
double boundary_t::heatFlux
```

wall heat flux

5.1.2.6 isAdiabatic

```
bool boundary_t::isAdiabatic
```

adiabatic wall flag

Referenced by initBoundary().

5.1.2.7 isTemperaturePrescribed

```
bool boundary_t::isTemperaturePrescribed
```

is the temperature prescribed flag

Referenced by initBoundary().

5.1.2.8 next

```
boundary_t* boundary_t::next
```

pointer to next boundary condition

Referenced by cgnsWriteMesh(), createMesh(), freeBoundary(), initBoundary(), and initWing().

5.1.2.9 pVar

```
double boundary_t::pVar[NVAR]
```

inflow state

Referenced by boundary(), and initBoundary().

5.1.2.10 temperature

```
double boundary_t::temperature
```

wall temperature

Referenced by initBoundary().

The documentation for this struct was generated from the following file:

- ccfd/src/boundary.h

5.2 cartMesh_t Struct Reference

Structure holding the information for a cartesian mesh.

```
#include <mesh.h>
```

Data Fields

- int iMax
- int jMax
- int ∗ nBC
- int BCtype [2 ∗NDIM][NBC]
- int BCrange [2 ∗NDIM][NBC][2]

5.2.1 Detailed Description

Structure holding the information for a cartesian mesh.

5.2.2 Field Documentation

5.2.2.1 BCrange

```
int cartMesh_t::BCrange[2 *NDIM][NBC][2]
```

list of BC ranges per side

Referenced by createCartMesh(), and readMesh().

5.2.2.2 BCtype

```
int cartMesh_t::BCtype[2 *NDIM][NBC]
```

list of BC types per side

Referenced by createCartMesh(), and readMesh().

5.2.2.3 iMax

```
int cartMesh_t::iMax
```

number of cells in x-direction

Referenced by createCartMesh(), and readMesh().

5.2.2.4 jMax

```
int cartMesh_t::jMax
```

number of cells in y-direction

Referenced by createCartMesh(), and readMesh().

5.2.2.5 nBC

```
int* cartMesh_t::nBC
```

number of different BC per side

Referenced by createCartMesh(), and readMesh().

The documentation for this struct was generated from the following file:

- ccfd/src/mesh.h

5.3 cmd_t Struct Reference

A structure used to store the commands, read in from the parameter file.

Collaboration diagram for cmd_t:

Data Fields

- char key [STRLEN]
- char value [STRLEN]
- cmd_t ∗ next
- cmd_t ∗ prev

5.3.1 Detailed Description

A structure used to store the commands, read in from the parameter file.

5.3.2 Field Documentation

5.3.2.1 key

`char cmd_t::key[STRLEN]`

the key word of the command

Referenced by fillCmds(), and findCmd().

5.3.2.2 next

`cmd_t* cmd_t::next`

next command

Referenced by countKeys(), deleteCmd(), fillCmds(), findCmd(), freeCmds(), and ignoredCmds().

5.3.2.3 prev

`cmd_t* cmd_t::prev`

previous command

Referenced by deleteCmd(), fillCmds(), and freeCmds().

5.3.2.4 value

```
char cmd_t::value[STRLEN]
```

the vale of the command

Referenced by fillCmds(), and findCmd().

The documentation for this struct was generated from the following file:

- ccfd/src/readInTools.c

5.4 elem_t Struct Reference

Structure for a single element in the global element list.

```
#include <mesh.h>
```

Collaboration diagram for elem_t:

Data Fields

- int elemType
- long id
- int domain
- double bary [NDIM]
- double sx
- double sy
- double area
- double areaq
- double pVar [NVAR]
- double cVar [NVAR]
- double cVarStage [NVAR]
- double u_x [NVAR]
- double u_y [NVAR]
- double u_t [NVAR]
- double source [NVAR]
- double dt
- double dtLoc
- double venkEps_sq
- int innerSides
- int nGP
- double ** xGP
- double * wGP
- side_t * firstSide
- elem_t * next
- node_t ** node

5.4.1 Detailed Description

Structure for a single element in the global element list.

5.4.2 Field Documentation

5.4.2.1 area

`double elem_t::area`

area of the element

Referenced by calcTimeStep(), createElemInfo(), createReconstructionInfo(), and globalResidual().

5.4.2.2 areaq

`double elem_t::areaq`

inverse of element area

Referenced by createElemInfo(), and fvTimeDerivative().

5.4.2.3 bary

`double elem_t::bary[NDIM]`

coordinates ob element barycenter

Referenced by calcCoef(), calcErrors(), cgnsOutput(), connectPeriodicBC(), createElemInfo(), createMesh(), createReconstructionInfo(), createSideInfo(), csvOutput(), curveOutput(), initWing(), setBCatBarys(), setBCat↩ Sides(), and setInitialCondition().

5.4.2.4 cVar

`double elem_t::cVar[NVAR]`

conservative variables of element

Referenced by explicitTimeStepEuler(), explicitTimeStepRK(), implicitTimeStep(), matrixVector(), and setInitial↩ Condition().

5.4.2.5 cVarStage

```
double elem_t::cVarStage[NVAR]
```

conservative variables at initial Runge-Kutta stage

Referenced by explicitTimeStepRK().

5.4.2.6 domain

```
int elem_t::domain
```

flow domain number

Referenced by createMesh(), and setInitialCondition().

5.4.2.7 dt

```
double elem_t::dt
```

element time step

Referenced by analyze(), calcTimeStep(), evalRecordPoints(), and fvTimeDerivative().

5.4.2.8 dtLoc

```
double elem_t::dtLoc
```

local element time step

Referenced by fvTimeDerivative().

5.4.2.9 elemType

```
int elem_t::elemType
```

element type: triangle (3) or quadrangle (4)

Referenced by cgnsWriteMesh(), createElemInfo(), createMesh(), createReconstructionInfo(), and initRecord↩ Points().

5.4.2.10 firstSide

`side_t* elem_t::firstSide`

pointer to the first side of the element

Referenced by buildMatrix(), createMesh(), createReconstructionInfo(), freeMesh(), fvTimeDerivative(), limiter↩ BarthJespersen(), limiterVenkatakrishnan(), LUSGS(), and spatialReconstruction().

5.4.2.11 id

`long elem_t::id`

unique element Id

Referenced by buildMatrix(), cgnsOutput(), cgnsReadSolution(), createMesh(), LUSGS(), and setInitialCondition().

5.4.2.12 innerSides

`int elem_t::innerSides`

number of non-BC sides of element

5.4.2.13 next

`elem_t* elem_t::next`

pointer to the next element in global element list

Referenced by cgnsOutput(), cgnsReadSolution(), cgnsWriteMesh(), createMesh(), csvOutput(), curveOutput(), initRecordPoints(), and setInitialCondition().

5.4.2.14 nGP

`int elem_t::nGP`

number of Gaussian integration points

Referenced by calcErrors(), calcSource(), and createReconstructionInfo().

5.4.2.15 node

```
node_t** elem_t::node
```

pointer array of the element's nodes

Referenced by cgnsWriteMesh(), createElemInfo(), createMesh(), createReconstructionInfo(), freeMesh(), and initRecordPoints().

5.4.2.16 pVar

```
double elem_t::pVar[NVAR]
```

primitive variables of element

Referenced by buildMatrix(), calcErrors(), calcTimeStep(), cgnsOutput(), cgnsReadSolution(), csvOutput(), curve↩ Output(), evalRecordPoints(), explicitTimeStepEuler(), explicitTimeStepRK(), fluxCalculation(), implicitTime↩ Step(), limiterBarthJespersen(), limiterVenkatakrishnan(), matrixVector(), setBCatBarys(), setInitialCondition(), and spatialReconstruction().

5.4.2.17 source

```
double elem_t::source[NVAR]
```

source term

Referenced by calcSource(), fvTimeDerivative(), and initFV().

5.4.2.18 sx

```
double elem_t::sx
```

cell extension in x-direction

Referenced by calcTimeStep(), and createElemInfo().

5.4.2.19 sy

```
double elem_t::sy
```

cell extension in y-direction

Referenced by calcTimeStep(), and createElemInfo().

5.4.2.20 u_t

```
double elem_t::u_t[NVAR]
```

t-gradient of primitive variables

Referenced by buildMatrix(), explicitTimeStepEuler(), explicitTimeStepRK(), fvTimeDerivative(), globalResidual(), implicitTimeStep(), matrixVector(), and spatialReconstruction().

5.4.2.21 u_x

```
double elem_t::u_x[NVAR]
```

x-gradient of primitive variables

Referenced by calcErrors(), fluxCalculation(), limiterBarthJespersen(), limiterVenkatakrishnan(), and spatial↩
Reconstruction().

5.4.2.22 u_y

```
double elem_t::u_y[NVAR]
```

y-gradient of primitive variables

Referenced by calcErrors(), fluxCalculation(), limiterBarthJespersen(), limiterVenkatakrishnan(), and spatial↩
Reconstruction().

5.4.2.23 venkEps_sq

```
double elem_t::venkEps_sq
```

Venkatakrishnan limiter constant for element

Referenced by initFV(), and limiterVenkatakrishnan().

5.4.2.24 wGP

```
double* elem_t::wGP
```

Gaussian weights

Referenced by calcErrors(), calcSource(), createReconstructionInfo(), and freeMesh().

5.4.2.25 xGP

```
double** elem_t::xGP
```

Gaussian points for volume integral

Referenced by calcErrors(), calcSource(), createReconstructionInfo(), and freeMesh().

The documentation for this struct was generated from the following file:

- ccfd/src/mesh.h

5.5 node_t Struct Reference

Structure for a single node in a linked list of nodes.

```
#include <mesh.h>
```

Collaboration diagram for node_t:

Data Fields

- long id
- double x [NDIM]
- node_t * next

5.5.1 Detailed Description

Structure for a single node in a linked list of nodes.

5.5.2 Field Documentation

5.5.2.1 id

```
long node_t::id
```

unique node ID

Referenced by cgnsWriteMesh(), and createMesh().

5.5.2.2 next

```
node_t* node_t::next
```

next node in the list

Referenced by cgnsWriteMesh(), createMesh(), and freeMesh().

5.5.2.3 x

```
double node_t::x[NDIM]
```

coordinates of the node

Referenced by cgnsWriteMesh(), createElemInfo(), createMesh(), createReconstructionInfo(), createSideInfo(), and initRecordPoints().

The documentation for this struct was generated from the following file:

- ccfd/src/mesh.h

5.6 outputTime_t Struct Reference

Output times linked list.

```
#include <output.h>
```

Collaboration diagram for outputTime_t:

Data Fields

- long iter
- double time
- outputTime_t * next

5.6.1 Detailed Description

Output times linked list.

5.6.2 Field Documentation

5.6.2.1 iter

`long outputTime_t::iter`

iteration number at output

Referenced by cgnsFinalizeOutput(), and dataOutput().

5.6.2.2 next

`outputTime_t* outputTime_t::next`

pointer to next output time

Referenced by cgnsFinalizeOutput(), dataOutput(), and freeOutputTimes().

5.6.2.3 time

`double outputTime_t::time`

computational time at output

Referenced by cgnsFinalizeOutput(), and dataOutput().

The documentation for this struct was generated from the following file:

- ccfd/src/output.h

5.7 recordPoint_t Struct Reference

Recording point structure, used to output flow field at specific points.

`#include <analyze.h>`

Collaboration diagram for recordPoint_t:

Data Fields

- int nPoints
- double ** x
- elem_t ** elem
- FILE ** ioFile

5.7.1 Detailed Description

Recording point structure, used to output flow field at specific points.

5.7.2 Field Documentation

5.7.2.1 elem

`elem_t** recordPoint_t::elem`

array of elements that contain a RP

Referenced by evalRecordPoints(), and initRecordPoints().

5.7.2.2 ioFile

`FILE** recordPoint_t::ioFile`

array of output file pointers

Referenced by evalRecordPoints(), initRecordPoints(), and timeDisc().

5.7.2.3 nPoints

`int recordPoint_t::nPoints`

number of recording points

Referenced by analyze(), evalRecordPoints(), initAnalyze(), initRecordPoints(), and timeDisc().

5.7.2.4 x

`double** recordPoint_t::x`

`nPoints`x`NDIM` array of RP coordinates

Referenced by initRecordPoints().

The documentation for this struct was generated from the following file:

- ccfd/src/analyze.h

5.8 side_t Struct Reference

Structure for a single side in the global side list.

```
#include <mesh.h>
```

Collaboration diagram for side_t:

Data Fields

- long id
- int BCtype
- int BCid
- boundary_t * BC
- double pVar [NVAR]
- double n [NDIM]
- double len
- double baryBaryVec [NDIM]
- double baryBaryDist
- double GP [NDIM]
- double w [NDIM]
- double flux [NVAR]
- side_t * connection
- side_t * nextElemSide
- side_t * next
- node_t * node [2]
- elem_t * elem

5.8.1 Detailed Description

Structure for a single side in the global side list.

5.8.2 Field Documentation

5.8.2.1 baryBaryDist

```
double side_t::baryBaryDist
```

length of `baryBaryVec`

Referenced by createReconstructionInfo(), and fluxCalculation().

5.8.2.2 baryBaryVec

```
double side_t::baryBaryVec[NDIM]
```

vector from element barycenter to barycenter of neighbor element

Referenced by createReconstructionInfo(), and fluxCalculation().

5.8.2.3 BC

```
boundary_t* side_t::BC
```

pointer to the boundary condition

Referenced by boundary(), cgnsWriteMesh(), connectPeriodicBC(), createMesh(), createSideInfo(), and initWing().

5.8.2.4 BCid

```
int side_t::BCid
```

boundary condition Sub-ID

5.8.2.5 BCtype

```
int side_t::BCtype
```

boundary condition type

5.8.2.6 connection

```
side_t* side_t::connection
```

neighbor side

Referenced by boundary(), buildMatrix(), connectPeriodicBC(), createMesh(), createReconstructionInfo(), create↩ SideInfo(), fluxCalculation(), initWing(), limiterBarthJespersen(), limiterVenkatakrishnan(), LUSGS(), setBCat↩ Barys(), setBCatSides(), and spatialReconstruction().

5.8.2.7 elem

```
elem_t* side_t::elem
```

pointer to the element of the side

Referenced by buildMatrix(), calcCoef(), connectPeriodicBC(), createMesh(), createSideInfo(), fluxCalculation(), freeMesh(), initWing(), limiterBarthJespersen(), limiterVenkatakrishnan(), LUSGS(), setBCatBarys(), setBCat↩ Sides(), and spatialReconstruction().

5.8.2.8 flux

```
double side_t::flux[NVAR]
```

numerical flux of the side

Referenced by fluxCalculation(), and fvTimeDerivative().

5.8.2.9 GP

```
double side_t::GP[NDIM]
```

vector from element barycenter to the Gaussian point of the side

Referenced by calcCoef(), connectPeriodicBC(), createMesh(), createReconstructionInfo(), createSideInfo(), init↩ Wing(), limiterBarthJespersen(), limiterVenkatakrishnan(), setBCatSides(), and spatialReconstruction().

5.8.2.10 id

```
long side_t::id
```

unique side ID

Referenced by createMesh().

5.8.2.11 len

```
double side_t::len
```

length of the side

Referenced by calcCoef(), createSideInfo(), and fluxCalculation().

5.8.2.12 n

```
double side_t::n[NDIM]
```

normal vector of side

Referenced by boundary(), calcCoef(), createMesh(), createSideInfo(), fluxCalculation(), and initWing().

5.8.2.13 next

```
side_t* side_t::next
```

point to the next side in the global side list

Referenced by connectPeriodicBC(), and createMesh().

5.8.2.14 nextElemSide

```
side_t* side_t::nextElemSide
```

pointer to the next side of the element

Referenced by buildMatrix(), createMesh(), createReconstructionInfo(), freeMesh(), fvTimeDerivative(), limiter↩ BarthJespersen(), limiterVenkatakrishnan(), LUSGS(), and spatialReconstruction().

5.8.2.15 node

```
node_t* side_t::node[2]
```

pointer array to the nodes of the side

Referenced by cgnsWriteMesh(), createMesh(), and createSideInfo().

5.8.2.16 pVar

```
double side_t::pVar[NVAR]
```

primitive variables state at side

Referenced by calcCoef(), fluxCalculation(), setBCatSides(), and spatialReconstruction().

5.8.2.17 w

```
double side_t::w[NDIM]
```

omegaX and omegaY entries for 2nd order gradient reconstruction

Referenced by createReconstructionInfo(), and spatialReconstruction().

The documentation for this struct was generated from the following file:

- ccfd/src/mesh.h

5.9 sideList_t Struct Reference

Helper structure for reading in the sides and deviding them into BC sides and non-BC sides.

Collaboration diagram for sideList_t:

Data Fields

- long node [2]
- bool BC
- side_t ∗ side
- bool isRotated

5.9.1 Detailed Description

Helper structure for reading in the sides and deviding them into BC sides and non-BC sides.

5.9.2 Field Documentation

5.9.2.1 BC

```
bool sideList_t::BC
```

flag for if the side is a BC side

Referenced by compare(), and createMesh().

5.9.2.2 isRotated

```
bool sideList_t::isRotated
```

flag for if the side is rotated

Referenced by createMesh().

5.9.2.3 node

```
long sideList_t::node[2]
```

node ID array of the side

Referenced by compare(), and createMesh().

5.9.2.4 side

```
side_t* sideList_t::side
```

pointer to the side

Referenced by createMesh().

The documentation for this struct was generated from the following file:

- ccfd/src/mesh.c

5.10 sidePtr_t Struct Reference

Secondary side lists used for various things.

```
#include <mesh.h>
```

Collaboration diagram for sidePtr_t:

Data Fields

- side_t * side
- sidePtr_t * next

5.10.1 Detailed Description

Secondary side lists used for various things.

5.10.2 Field Documentation

5.10.2.1 next

```
sidePtr_t* sidePtr_t::next
```

pointer to the next side in the secondary list

Referenced by calcCoef(), cgnsWriteMesh(), connectPeriodicBC(), createMesh(), freeAnalyze(), freeMesh(), and initWing().

5.10.2.2 side

```
side_t* sidePtr_t::side
```

pointer to a side

Referenced by calcCoef(), cgnsWriteMesh(), connectPeriodicBC(), createMesh(), freeMesh(), and initWing().

The documentation for this struct was generated from the following file:

- ccfd/src/mesh.h

5.11 wing_t Struct Reference

Collection of all necessary values for the calculation of CL and CD.

```
#include <analyze.h>
```

Collaboration diagram for wing_t:

Data Fields

- double refLength
- int wallId
- double cl
- double cd
- boundary_t ∗ wingBC
- sidePtr_t ∗ firstPressureSide
- sidePtr_t ∗ firstSuctionSide

5.11.1 Detailed Description

Collection of all necessary values for the calculation of CL and CD.

5.11.2 Field Documentation

5.11.2.1 cd

`double wing_t::cd`

drag coefficient

Referenced by analyze(), calcCoef(), and timeDisc().

5.11.2.2 cl

`double wing_t::cl`

lift coefficient

Referenced by analyze(), calcCoef(), and timeDisc().

5.11.2.3 firstPressureSide

`sidePtr_t* wing_t::firstPressureSide`

pointer to the first pressure side

Referenced by calcCoef(), freeAnalyze(), and initWing().

5.11.2.4 firstSuctionSide

`sidePtr_t* wing_t::firstSuctionSide`

pointer to the first suction side

Referenced by calcCoef(), freeAnalyze(), and initWing().

5.11.2.5 refLength

`double wing_t::refLength`

reference length

Referenced by calcCoef(), and readWing().

5.11.2.6 wallId

```
int wing_t::wallId
```

BCid of the wall that represents the wing

Referenced by initWing(), and readWing().

5.11.2.7 wingBC

```
boundary_t* wing_t::wingBC
```

pointer to the BC of the wing

The documentation for this struct was generated from the following file:

- ccfd/src/analyze.h

Chapter 6

File Documentation

6.1 ccfd/src/analyze.c File Reference

Contains functions for analyzing flow results.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdbool.h>
#include <math.h>
#include "analyze.h"
#include "readInTools.h"
#include "timeDiscretization.h"
#include "output.h"
#include "memTools.h"
#include "reconstruction.h"
#include "exactFunction.h"
#include "equation.h"
#include "initialCondition.h"
```

Include dependency graph for analyze.c:

Functions

- void initRecordPoints (void)

  Initialize recording points.
- void initWing (void)

  Initialize required data for calculation of CL and CD.
- void readWing (void)

  Get parameters for calculating aerodynamic coefficients.
- void initAnalyze (void)

  Initialize the analysis function.
- void calcCoef (void)

  Calculate CL and CD around the specified wall.
- void evalRecordPoints (double time)

  Evaluation of recording points.
- void analyze (double time, long iter, double resIter[NVAR+2])

  Compute aerodynamic coefficients and extract values at record points.

- void calcErrors (double time)

  Calculate L1, L2, and Linf error norms, between the flow solution and the exact solution.
- void globalResidual (double resIter[NVAR+2])

  Calculate the global residual of the conservative variables.
- void freeAnalyze (void)

  Free all memory that was allocated for the wing evaluation.

Variables

- bool doCalcWing
- wing_t wing
- recordPoint_t recordPoint
- bool hasExactSolution

6.1.1 Detailed Description

Contains functions for analyzing flow results.

Author

hhh

Date

Sun 29 Mar 2020 06:29:36 PM CEST

6.1.2 Function Documentation

6.1.2.1 analyze()

```
void analyze (
            double time,
            long iter,
            double resIter[NVAR+2] )
```

Compute aerodynamic coefficients and extract values at record points.

Parameters

in	*time*	Calculation time at output
in	*iter*	Iteration count at output
in,out	*resIter*	The residual vector containing the CL and CD residuals at 4th and 5th index position

References abortVariable, calcCoef(), wing_t::cd, wing_t::cl, doCalcWing, elem_t::dt, E, evalRecordPoints(), first↩ Elem, isStationary, recordPoint_t::nPoints, recordPoint, resFile, RHO, VX, VY, and wing.

Referenced by timeDisc().

6.1.2.2 calcErrors()

```
void calcErrors (
            double time )
```

Calculate L1, L2, and Linf error norms, between the flow solution and the exact solution.

Parameters

in	*time*	Calculation time at output

References elem_t::bary, elem, exactFunc(), intExactFunc, NDIM, nElems, elem_t::nGP, NVAR, P, elem_t::pVar, RHO, spatialReconstruction(), totalArea_q, elem_t::u_x, elem_t::u_y, VX, VY, elem_t::wGP, X, elem_t::xGP, and Y.

Referenced by timeDisc().

6.1.2.3 evalRecordPoints()

```
void evalRecordPoints (
            double time )
```

Evaluation of recording points.

Parameters

in	*time*	Calculation time at output

References elem_t::dt, recordPoint_t::elem, recordPoint_t::ioFile, recordPoint_t::nPoints, P, elem_t::pVar, record↩
Point, RHO, VX, and VY.

Referenced by analyze().

6.1.2.4 globalResidual()

```
void globalResidual (
            double resIter[NVAR+2] )
```

Calculate the global residual of the conservative variables.

Parameters

`in,out`	*resIter*	The residual vector containing the residuals of the conservative variables at the first four positions

References elem_t::area, E, elem, MX, MY, nElems, NVAR, RHO, totalArea_q, and elem_t::u_t.

Referenced by explicitTimeStepEuler(), explicitTimeStepRK(), and implicitTimeStep().

6.1.3 Variable Documentation

6.1.3.1 doCalcWing

```
bool doCalcWing
```

calculate CL CD flag

Referenced by analyze(), freeAnalyze(), initAnalyze(), and timeDisc().

6.1.3.2 hasExactSolution

```
bool hasExactSolution
```

exact solution existence flag

Referenced by dataOutput(), initAnalyze(), and timeDisc().

6.1.3.3 recordPoint

```
recordPoint_t recordPoint
```

record flow field at a specific point

Referenced by analyze(), evalRecordPoints(), initAnalyze(), initRecordPoints(), and timeDisc().

6.1.3.4 wing

```
wing_t wing
```

holds data for coefficient calculation

Referenced by analyze(), calcCoef(), freeAnalyze(), initWing(), readWing(), and timeDisc().

6.2 ccfd/src/analyze.h File Reference

Contains the structure definitions of wing_t and recordPoint_t

```
#include <stdlib.h>
#include <stdbool.h>
#include "main.h"
#include "boundary.h"
#include "mesh.h"
```

Include dependency graph for analyze.h: This graph shows which files directly or indirectly include this file:

Data Structures

- struct wing_t

  Collection of all necessary values for the calculation of CL and CD.
- struct recordPoint_t

  Recording point structure, used to output flow field at specific points.

Functions

- void initAnalyze (void)

  Initialize the analysis function.
- void analyze (double time, long iter, double resIter[NVAR+2])

  Compute aerodynamic coefficients and extract values at record points.
- void calcErrors (double time)

  Calculate L1, L2, and Linf error norms, between the flow solution and the exact solution.
- void globalResidual (double resIter[NVAR+2])

  Calculate the global residual of the conservative variables.
- void freeAnalyze (void)

  Free all memory that was allocated for the wing evaluation.

Variables

- bool doCalcWing
- wing_t wing
- recordPoint_t recordPoint
- bool hasExactSolution

6.2.1 Detailed Description

Contains the structure definitions of wing_t and recordPoint_t

Author

hhh

Date

Sun 29 Mar 2020 05:51:23 PM CEST

6.2.2 Function Documentation

6.2.2.1 analyze()

```
void analyze (
            double time,
            long iter,
            double resIter[NVAR+2] )
```

Compute aerodynamic coefficients and extract values at record points.

Parameters

in	*time*	Calculation time at output
in	*iter*	Iteration count at output
in,out	*resIter*	The residual vector containing the CL and CD residuals at 4th and 5th index position

References abortVariable, calcCoef(), wing_t::cd, wing_t::cl, doCalcWing, elem_t::dt, E, evalRecordPoints(), first↩ Elem, isStationary, recordPoint_t::nPoints, recordPoint, resFile, RHO, VX, VY, and wing.

Referenced by timeDisc().

6.2.2.2 calcErrors()

```
void calcErrors (
            double time )
```

Calculate L1, L2, and Linf error norms, between the flow solution and the exact solution.

Parameters

in	*time*	Calculation time at output

References elem_t::bary, elem, exactFunc(), intExactFunc, NDIM, nElems, elem_t::nGP, NVAR, P, elem_t::pVar, RHO, spatialReconstruction(), totalArea_q, elem_t::u_x, elem_t::u_y, VX, VY, elem_t::wGP, X, elem_t::xGP, and Y.

Referenced by timeDisc().

6.2.2.3 globalResidual()

```
void globalResidual (
            double resIter[NVAR+2] )
```

Calculate the global residual of the conservative variables.

Parameters

in,out	*resIter*	The residual vector containing the residuals of the conservative variables at the first four positions

References elem_t::area, E, elem, MX, MY, nElems, NVAR, RHO, totalArea_q, and elem_t::u_t.

Referenced by explicitTimeStepEuler(), explicitTimeStepRK(), and implicitTimeStep().

6.2.3 Variable Documentation

6.2.3.1 doCalcWing

`bool doCalcWing [extern]`

calculate CL CD flag

Referenced by analyze(), freeAnalyze(), initAnalyze(), and timeDisc().

6.2.3.2 hasExactSolution

`bool hasExactSolution [extern]`

exact solution existence flag

Referenced by dataOutput(), initAnalyze(), and timeDisc().

6.2.3.3 recordPoint

`recordPoint_t recordPoint [extern]`

record flow field at a specific point

Referenced by analyze(), evalRecordPoints(), initAnalyze(), initRecordPoints(), and timeDisc().

6.2.3.4 wing

`wing_t wing [extern]`

holds data for coefficient calculation

Referenced by analyze(), calcCoef(), freeAnalyze(), initWing(), readWing(), and timeDisc().

6.3 ccfd/src/boundary.c File Reference

Contains the functions for initializing and applying boundary conditions.

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "main.h"
#include "boundary.h"
#include "readInTools.h"
#include "equation.h"
#include "mesh.h"
#include "equationOfState.h"
#include "exactFunction.h"
```

Include dependency graph for boundary.c:

Functions

- void initBoundary (void)

  Initialize boundary conditions.
- void boundary (side_t ∗aSide, double time, double int_pVar[NVAR], double ghost_pVar[NVAR], double x[NDIM])

  Set boundary condition value at x.
- void setBCatSides (double time)

  Set the ghost values at sides.
- void setBCatBarys (double time)

  Set the ghost values at elements.
- void freeBoundary (void)

  Free all memory that was allocated for the boundary conditions.

Variables

- boundary_t ∗ firstBC
- int nBC
- bool isPeriodic

6.3.1 Detailed Description

Contains the functions for initializing and applying boundary conditions.

Author

hhh

Date

Tue 24 Mar 2020 10:10:51 AM CET

6.3.2 Function Documentation

6.3.2.1 boundary()

```
void boundary (
            side_t * aSide,
            double time,
            double int_pVar[NVAR],
            double ghost_pVar[NVAR],
            double x[NDIM] )
```

Set boundary condition value at x.

Parameters

in	*aSide*	Pinter to a boundary side
in	*time*	Computation time at calculation
in	*int_pVar*	Internal cell primitive variables state
out	*ghost_pVar*	Ghost cell primitive variables state
in	*x*	Barycenter coordinates of the ghost cell

References side_t::BC, boundary_t::BCtype, CHARACTERISTIC, charCons(), side_t::connection, consChar(), consPrim(), boundary_t::exactFunc, exactFunc(), EXACTSOL, gam, INFLOW, mu, MY, side_t::n, NDIM, NVAR, OUTFLOW, P, PRESSURE_OUT, primCons(), boundary_t::pVar, RHO, SLIPWALL, VX, VY, WALL, X, and Y.

Referenced by setBCatBarys(), and setBCatSides().

6.3.2.2 setBCatBarys()

```
void setBCatBarys (
            double time )
```

Set the ghost values at elements.

Parameters

in	*time*	Computation time at calculation

References elem_t::bary, BCside, boundary(), side_t::connection, side_t::elem, nBCsides, and elem_t::pVar.

Referenced by spatialReconstruction().

6.3.2.3 setBCatSides()

```
void setBCatSides (
            double time )
```

Set the ghost values at sides.

Parameters

in	*time*	Computation time at calculation

References elem_t::bary, BCside, boundary(), side_t::connection, side_t::elem, side_t::GP, nBCsides, NDIM, side↩_t::pVar, X, and Y.

Referenced by fvTimeDerivative().

6.3.3 Variable Documentation

6.3.3.1 firstBC

```
boundary_t* firstBC
```

pointer to the first boundary condition

Referenced by cgnsWriteMesh(), createMesh(), freeBoundary(), initBoundary(), and initWing().

6.3.3.2 isPeriodic

```
bool isPeriodic
```

periodic boundary condition flag

Referenced by cgnsWriteMesh(), and connectPeriodicBC().

6.3.3.3 nBC

```
int nBC
```

number of boundary conditions

Referenced by cgnsWriteMesh(), and initBoundary().

6.4 ccfd/src/boundary.h File Reference

Contains the structure definition of a boundary.

```
#include <stdbool.h>
#include "main.h"
#include "mesh.h"
```

Include dependency graph for boundary.h: This graph shows which files directly or indirectly include this file:

Data Structures

- struct boundary_t

  Structure that holds the information of a boundary condition.

Functions

- void initBoundary (void)

  Initialize boundary conditions.
- void setBCatSides (double time)

  Set the ghost values at sides.
- void setBCatBarys (double time)

  Set the ghost values at elements.
- void boundary (side_t *aSide, double time, double int_pVar[NVAR], double ghost_pVar[NVAR], double x[NDIM])

  Set boundary condition value at x.
- void freeBoundary (void)

  Free all memory that was allocated for the boundary conditions.

Variables

- boundary_t * firstBC
- int nBC
- bool isPeriodic

6.4.1 Detailed Description

Contains the structure definition of a boundary.

Author

hhh

Date

Tue 24 Mar 2020 10:02:10 AM CET

6.4.2 Function Documentation

6.4.2.1 boundary()

```
void boundary (
            side_t * aSide,
            double time,
            double int_pVar[NVAR],
            double ghost_pVar[NVAR],
            double x[NDIM] )
```

Set boundary condition value at x.

Parameters

in	*aSide*	Pinter to a boundary side
in	*time*	Computation time at calculation
in	*int_pVar*	Internal cell primitive variables state
out	*ghost_pVar*	Ghost cell primitive variables state
in	*x*	Barycenter coordinates of the ghost cell

References side_t::BC, boundary_t::BCtype, CHARACTERISTIC, charCons(), side_t::connection, consChar(), consPrim(), boundary_t::exactFunc, exactFunc(), EXACTSOL, gam, INFLOW, mu, MY, side_t::n, NDIM, NVAR, OUTFLOW, P, PRESSURE_OUT, primCons(), boundary_t::pVar, RHO, SLIPWALL, VX, VY, WALL, X, and Y.

Referenced by setBCatBarys(), and setBCatSides().

6.4.2.2 setBCatBarys()

```
void setBCatBarys (
            double time )
```

Set the ghost values at elements.

Parameters

in	*time*	Computation time at calculation

References elem_t::bary, BCside, boundary(), side_t::connection, side_t::elem, nBCsides, and elem_t::pVar.

Referenced by spatialReconstruction().

6.4.2.3 setBCatSides()

```
void setBCatSides (
            double time )
```

Set the ghost values at sides.

Parameters

in	*time*	Computation time at calculation

References elem_t::bary, BCside, boundary(), side_t::connection, side_t::elem, side_t::GP, nBCsides, NDIM, side←_t::pVar, X, and Y.

Referenced by fvTimeDerivative().

6.4.3 Variable Documentation

6.4.3.1 firstBC

```
boundary_t* firstBC  [extern]
```

pointer to the first boundary condition

Referenced by cgnsWriteMesh(), createMesh(), freeBoundary(), initBoundary(), and initWing().

6.4.3.2 isPeriodic

```
bool isPeriodic  [extern]
```

periodic boundary condition flag

Referenced by cgnsWriteMesh(), and connectPeriodicBC().

6.4.3.3 nBC

```
int nBC  [extern]
```

number of boundary conditions

Referenced by cgnsWriteMesh(), and initBoundary().

6.5 ccfd/src/equation.c File Reference

Contains the function for initializing the physical constants.

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "equation.h"
#include "readInTools.h"
```

Include dependency graph for equation.c:

Functions

- void initEquation (void)

  Initialize equations.

Variables

- double pi
- bool doCalcSource
- double R
- double gam
- double gam1
- double gam2
- double gam1q
- double cp
- double Pr
- double mu
- int iFlux
- int intExactFunc
- int sourceFunc
- double sqrt2
- double sqrt3
- double sqrt3q

6.5.1 Detailed Description

Contains the function for initializing the physical constants.

Date

Tue 24 Mar 2020 08:30:28 AM CET

Author

hhh

6.5.2 Variable Documentation

6.5.2.1 cp

```
double cp
```

specific heat capacity

Referenced by calcCoef().

6.5.2.2 doCalcSource

```
bool doCalcSource
```

calculate source flag

Referenced by fvTimeDerivative(), and initEquation().

6.5.2.3 gam

```
double gam
```

specific heat ratio

Referenced by boundary(), calcTimeStep(), charCons(), consChar(), evalSource(), exactFunc(), exactRiemann(), flux_ausmd(), flux_ausmdv(), flux_god(), flux_hll(), flux_hllc(), flux_hlle(), flux_lxf(), flux_roe(), flux_stw(), flux_↩ vanleer(), GMRES_M(), initBoundary(), initEquation(), and initInitialCondition().

6.5.2.4 gam1

```
double gam1
```

`gam` - 1

Referenced by consPrim(), evalSource(), exactFunc(), flux_god(), flux_hll(), flux_hllc(), flux_roe(), flux_stw(), flux↩ _vanleer(), and initEquation().

6.5.2.5 gam1q

```
double gam1q
```

1.0 / (`gam` - 1)

Referenced by charCons(), consChar(), flux_ausmd(), flux_ausmdv(), flux_cen(), flux_hll(), flux_hllc(), flux_hlle(), flux_lxf(), flux_roe(), flux_stw(), flux_vanleer(), initEquation(), and primCons().

6.5.2.6 gam2

```
double gam2
```

`gam` - 2

Referenced by flux_roe(), and initEquation().

6.5.2.7 iFlux

```
int iFlux
```

flux function control

Referenced by convectiveFlux(), and initEquation().

6.5.2.8 intExactFunc

```
int intExactFunc
```

exact function control

Referenced by calcErrors(), cgnsOutput(), csvOutput(), curveOutput(), initInitialCondition(), and setInitialCondition().

6.5.2.9 mu

```
double mu
```

dynamic viscosity

Referenced by boundary(), calcTimeStep(), evalSource(), and initEquation().

6.5.2.10 pi

```
double pi
```

pi

Referenced by calcCoef(), evalSource(), exactFunc(), initBoundary(), initEquation(), and initInitialCondition().

6.5.2.11 Pr

```
double Pr
```

Prandtl number

Referenced by calcTimeStep(), evalSource(), and initEquation().

6.5.2.12 R

```
double R
```

specific gas constant

Referenced by exactFunc(), and initEquation().

6.5.2.13 sourceFunc

```
int sourceFunc
```

source function control

Referenced by calcSource(), and initEquation().

6.5.2.14 sqrt2

```
double sqrt2
```

sqrt(2.0)

Referenced by initEquation().

6.5.2.15 sqrt3

```
double sqrt3
```

sqrt(3.0)

Referenced by initEquation().

6.5.2.16 sqrt3q

```
double sqrt3q
```

1.0 / sqrt(3.0)

Referenced by exactFunc(), and initEquation().

6.6 ccfd/src/equation.h File Reference

```
#include <stdbool.h>
```

Include dependency graph for equation.h: This graph shows which files directly or indirectly include this file:

Functions

- void initEquation (void)

  Initialize equations.

Variables

- double pi
- bool doCalcSource
- double R
- double gam
- double gam1
- double gam2
- double gam1q
- double cp
- double Pr
- double mu
- int iFlux
- int intExactFunc
- int sourceFunc
- double sqrt2
- double sqrt3
- double sqrt3q

6.6.1 Detailed Description

Author

hhh

Date

Tue 24 Mar 2020 08:23:21 AM CET

6.6.2 Variable Documentation

6.6.2.1 cp

```
double cp  [extern]
```

specific heat capacity

Referenced by calcCoef().

6.6.2.2 doCalcSource

```
bool doCalcSource  [extern]
```

calculate source flag

Referenced by fvTimeDerivative(), and initEquation().

6.6.2.3 gam

```
double gam  [extern]
```

specific heat ratio

Referenced by boundary(), calcTimeStep(), charCons(), consChar(), evalSource(), exactFunc(), exactRiemann(), flux_ausmd(), flux_ausmdv(), flux_god(), flux_hll(), flux_hllc(), flux_hlle(), flux_lxf(), flux_roe(), flux_stw(), flux_↩ vanleer(), GMRES_M(), initBoundary(), initEquation(), and initInitialCondition().

6.6.2.4 gam1

```
double gam1  [extern]
```

`gam` - 1

Referenced by consPrim(), evalSource(), exactFunc(), flux_god(), flux_hll(), flux_hllc(), flux_roe(), flux_stw(), flux↩ _vanleer(), and initEquation().

6.6.2.5 gam1q

```
double gam1q [extern]
```

1.0 / (`gam` - 1)

Referenced by charCons(), consChar(), flux_ausmd(), flux_ausmdv(), flux_cen(), flux_hll(), flux_hllc(), flux_hlle(), flux_lxf(), flux_roe(), flux_stw(), flux_vanleer(), initEquation(), and primCons().

6.6.2.6 gam2

```
double gam2 [extern]
```

`gam` - 2

Referenced by flux_roe(), and initEquation().

6.6.2.7 iFlux

```
int iFlux [extern]
```

flux function control

Referenced by convectiveFlux(), and initEquation().

6.6.2.8 intExactFunc

```
int intExactFunc [extern]
```

exact function control

Referenced by calcErrors(), cgnsOutput(), csvOutput(), curveOutput(), initInitialCondition(), and setInitialCondition().

6.6.2.9 mu

```
double mu [extern]
```

dynamic viscosity

Referenced by boundary(), calcTimeStep(), evalSource(), and initEquation().

6.6.2.10 pi

```
double pi [extern]
```

pi

Referenced by calcCoef(), evalSource(), exactFunc(), initBoundary(), initEquation(), and initInitialCondition().

6.6.2.11 Pr

```
double Pr [extern]
```

Prandtl number

Referenced by calcTimeStep(), evalSource(), and initEquation().

6.6.2.12 R

```
double R [extern]
```

specific gas constant

Referenced by exactFunc(), and initEquation().

6.6.2.13 sourceFunc

```
int sourceFunc [extern]
```

source function control

Referenced by calcSource(), and initEquation().

6.6.2.14 sqrt2

```
double sqrt2 [extern]
```

sqrt(2.0)

Referenced by initEquation().

6.6.2.15 sqrt3

```
double sqrt3 [extern]
```

sqrt(3.0)

Referenced by initEquation().

6.6.2.16 sqrt3q

```
double sqrt3q [extern]
```

1.0 / sqrt(3.0)

Referenced by exactFunc(), and initEquation().

6.7 ccfd/src/equationOfState.c File Reference

Contains conversion functions between the different variable types.

```
#include <math.h>
#include "main.h"
#include "equation.h"
```

Include dependency graph for equationOfState.c:

Functions

- void primCons (const double pVar[NVAR], double cVar[NVAR])

  Convert primitive variables into conservative variables.
- void consPrim (const double cVar[NVAR], double pVar[NVAR])

  Convert conservative variables into primitive variables.
- void consChar (double cVar[NVAR], double charac[3], double pVarRef[NVAR])

  Convert conservative variables to characteristic variables.
- void charCons (double charac[3], double cVar[NVAR], double pVarRef[NVAR])

  Convert characteristic variables to conservative variables.

6.7.1 Detailed Description

Contains conversion functions between the different variable types.

Author

hhh

Date

Sat 28 Mar 2020 09:45:30 PM CET

6.7.2 Function Documentation

6.7.2.1 charCons()

```
void charCons (
            double charac[3],
            double cVar[NVAR],
            double pVarRef[NVAR] )
```

Convert characteristic variables to conservative variables.

Parameters

in	*charac*	Characteristic variable vector
out	*cVar*	Conservative variable vector
in	*pVarRef*	Reference primitive variable vector

References E, gam, gam1q, MX, P, RHO, and VX.

Referenced by boundary().

6.7.2.2 consChar()

```
void consChar (
            double cVar[NVAR],
            double charac[3],
            double pVarRef[NVAR] )
```

Convert conservative variables to characteristic variables.

Parameters

in	*cVar*	Conservative variable vector
out	*charac*	Characteristic variable vector
in	*pVarRef*	Reference primitive variable vector

References E, gam, gam1q, P, RHO, and VX.

Referenced by boundary().

6.7.2.3 consPrim()

```
void consPrim (
            const double cVar[NVAR],
            double pVar[NVAR] )
```

Convert conservative variables into primitive variables.

This function is used during reconstruction, therefore it has to be checked if the resulting primitive variables are negative. It that is the case, they are set to zero.

Parameters

in	*cVar*	Conservative variable vector
out	*pVar*	Primitive variable vector

References E, gam1, MX, MY, P, RHO, VX, and VY.

Referenced by boundary(), exactFunc(), explicitTimeStepEuler(), explicitTimeStepRK(), implicitTimeStep(), and matrixVector().

6.7.2.4 primCons()

```
void primCons (
            const double pVar[NVAR],
            double cVar[NVAR] )
```

Convert primitive variables into conservative variables.

Parameters

in	*pVar*	Primitive variable vector
out	*cVar*	Conservative variable vector

References E, gam1q, MX, MY, P, RHO, VX, and VY.

Referenced by boundary(), and setInitialCondition().

6.8 ccfd/src/equationOfState.h File Reference

```
#include "main.h"
```

Include dependency graph for equationOfState.h: This graph shows which files directly or indirectly include this file:

Functions

- void primCons (const double pVar[NVAR], double cVar[NVAR])
  Convert primitive variables into conservative variables.
- void consPrim (const double cVar[NVAR], double pVar[NVAR])
  Convert conservative variables into primitive variables.
- void consChar (double cVar[NVAR], double charac[3], double pVarRef[NVAR])
  Convert conservative variables to characteristic variables.
- void charCons (double charac[3], double cVar[NVAR], double pVarRef[NVAR])
  Convert characteristic variables to conservative variables.

6.8.1 Detailed Description

Author

hhh

Date

Sat 28 Mar 2020 09:45:50 PM CET

6.8.2 Function Documentation

6.8.2.1 charCons()

```
void charCons (
            double charac[3],
            double cVar[NVAR],
            double pVarRef[NVAR] )
```

Convert characteristic variables to conservative variables.

Parameters

in	*charac*	Characteristic variable vector
out	*cVar*	Conservative variable vector
in	*pVarRef*	Reference primitive variable vector

References E, gam, gam1q, MX, P, RHO, and VX.

Referenced by boundary().

6.8.2.2 consChar()

```
void consChar (
            double cVar[NVAR],
            double charac[3],
            double pVarRef[NVAR] )
```

Convert conservative variables to characteristic variables.

Parameters

in	*cVar*	Conservative variable vector
out	*charac*	Characteristic variable vector
in	*pVarRef*	Reference primitive variable vector

References E, gam, gam1q, P, RHO, and VX.

Referenced by boundary().

6.8.2.3 consPrim()

```
void consPrim (
            const double cVar[NVAR],
            double pVar[NVAR] )
```

Convert conservative variables into primitive variables.

This function is used during reconstruction, therefore it has to be checked if the resulting primitive variables are negative. It that is the case, they are set to zero.

Parameters

in	*cVar*	Conservative variable vector
out	*pVar*	Primitive variable vector

References E, gam1, MX, MY, P, RHO, VX, and VY.

Referenced by boundary(), exactFunc(), explicitTimeStepEuler(), explicitTimeStepRK(), implicitTimeStep(), and matrixVector().

6.8.2.4 primCons()

```
void primCons (
            const double pVar[NVAR],
            double cVar[NVAR] )
```

Convert primitive variables into conservative variables.

Parameters

in	*pVar*	Primitive variable vector
out	*cVar*	Conservative variable vector

References E, gam1q, MX, MY, P, RHO, VX, and VY.

Referenced by boundary(), and setInitialCondition().

6.9 ccfd/src/exactFunction.c File Reference

Contains the exact function evaluation function.

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "main.h"
#include "mesh.h"
#include "equation.h"
#include "initialCondition.h"
#include "equationOfState.h"
#include "exactRiemann.h"
```

Include dependency graph for exactFunction.c:

Functions

- void exactFunc (int iExactFunc, double x[NDIM], double time, double pVar[NVAR])
  Calculate an exact function.

6.9.1 Detailed Description

Contains the exact function evaluation function.

Author

hhh

Date

Sat 28 Mar 2020 05:31:15 PM CET

6.9.2 Function Documentation

6.9.2.1 exactFunc()

```
void exactFunc (
            int iExactFunc,
            double x[NDIM],
            double time,
            double pVar[NVAR] )
```

Calculate an exact function.

This function contains the following exact functions:

- 1: Richtmyer-Meshkov
- 2: Gaussian pressure pulse
- 3: Sinewave
- 4: Double mach reflection
- 5: 1D Riemann problem
- 6: 1D sine wave

Parameters

in	*iExactFunc*	The exact function control
in	*x*	The coordinates for which to calculate the exact function
in	*time*	The time for which to compute the exact value
out	*pVar*	The resulting vector of primitive variables

References consPrim(), dxRef, E, exactRiemann(), gam, gam1, NVAR, P, pi, R, refState, RHO, rp1Dinterface, sqrt3q, VX, VY, X, xMax, xMin, Y, yMax, and yMin.

Referenced by boundary(), calcErrors(), cgnsOutput(), csvOutput(), curveOutput(), and setInitialCondition().

6.10 ccfd/src/exactFunction.h File Reference

This graph shows which files directly or indirectly include this file:

Functions

- void exactFunc (int iExactFunc, double x[NDIM], double time, double pVar[NVAR])
  Calculate an exact function.

6.10.1 Detailed Description

Author

hhh

Date

Sat 28 Mar 2020 05:55:25 PM CET

6.10.2 Function Documentation

6.10.2.1 exactFunc()

```
void exactFunc (
            int iExactFunc,
            double x[NDIM],
            double time,
            double pVar[NVAR] )
```

Calculate an exact function.

This function contains the following exact functions:

- 1: Richtmyer-Meshkov
- 2: Gaussian pressure pulse
- 3: Sinewave
- 4: Double mach reflection
- 5: 1D Riemann problem
- 6: 1D sine wave

Parameters

in	*iExactFunc*	The exact function control
in	*x*	The coordinates for which to calculate the exact function
in	*time*	The time for which to compute the exact value
out	*pVar*	The resulting vector of primitive variables

References consPrim(), dxRef, E, exactRiemann(), gam, gam1, NVAR, P, pi, R, refState, RHO, rp1Dinterface, sqrt3q, VX, VY, X, xMax, xMin, Y, yMax, and yMin.

Referenced by boundary(), calcErrors(), cgnsOutput(), csvOutput(), curveOutput(), and setInitialCondition().

6.11 ccfd/src/exactRiemann.c File Reference

Contains the function to calculate the exact Riemann flux.

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "equation.h"
```

Include dependency graph for exactRiemann.c:

Functions

- void preFun (double ∗f, double ∗fd, double p, double rhok, double pk, double ck)
    Helper function for `exactRiemann`
- void exactRiemann (double rhol, double rhor, double ∗rho, double ul, double ur, double ∗u, double pl, double pr, double ∗p, double al, double ar, double s)
    Calculate the exact solution to the Riemann problem.

Variables

- double G [9]
- double tol = 1e-6
- int nIter = 1000

6.11.1 Detailed Description

Contains the function to calculate the exact Riemann flux.

Author

hhh

Date

Sun 29 Mar 2020 12:35:17 PM CEST

6.11.2 Function Documentation

6.11.2.1 exactRiemann()

```
void exactRiemann (
            double rhol,
            double rhor,
            double * rho,
            double ul,
            double ur,
            double * u,
            double pl,
            double pr,
            double * p,
            double al,
            double ar,
            double s )
```

Calculate the exact solution to the Riemann problem.

Parameters

in	*rhol*	Left side density
in	*rhor*	Right side density
out	*rho*	The resulting density
in	*ul*	Left side velocity
in	*ur*	Right side velocity
out	*u*	Resulting velocity
in	*pl*	Left side pressure
in	*pr*	Right side pressure
out	*p*	Resulting pressure
in	*al*	Left side speed of sound
in	*ar*	Right side speed of sound
in	*s*	Speed of the discontinuity

References G, gam, nIter, preFun(), and tol.

Referenced by exactFunc(), and flux_god().

6.11.2.2 preFun()

```
void preFun (
            double * f,
            double * fd,
            double p,
            double rhok,
```

```
double pk,
double ck )
```

Helper function for `exactRiemann`

Parameters

out	*f*	Flux
out	*fd*	Flux difference
in	*p*	Pressure
in	*rhok*	Critical density
in	*pk*	Critical pressure
in	*ck*	Critical speed of sound

References G.

Referenced by exactRiemann().

6.11.3 Variable Documentation

6.11.3.1 G

```
double G[9]
```

gamma vector

Referenced by exactRiemann(), and preFun().

6.11.3.2 nIter

```
int nIter = 1000
```

maximum number of iterations

Referenced by exactRiemann().

6.11.3.3 tol

```
double tol = 1e-6
```

tolerance for the iteration

Referenced by exactRiemann().

6.12 ccfd/src/exactRiemann.h File Reference

This graph shows which files directly or indirectly include this file:

Functions

- void exactRiemann (double rhol, double rhor, double ∗rho, double ul, double ur, double ∗u, double pl, double pr, double ∗p, double al, double ar, double s)

  Calculate the exact solution to the Riemann problem.

6.12.1 Detailed Description

Date

Sun 29 Mar 2020 12:35:26 PM CEST

Author

hhh

6.12.2 Function Documentation

6.12.2.1 exactRiemann()

```
void exactRiemann (
            double rhol,
            double rhor,
            double * rho,
            double ul,
            double ur,
            double * u,
            double pl,
            double pr,
            double * p,
            double al,
            double ar,
            double s )
```

Calculate the exact solution to the Riemann problem.

Parameters

in	*rhol*	Left side density
in	*rhor*	Right side density
out	*rho*	The resulting density
in	*ul*	Left side velocity
in	*ur*	Right side velocity
out	*u*	Resulting velocity
in	*pl*	Left side pressure
in	*pr*	Right side pressure
out	*p*	Resulting pressure
in	*al*	Left side speed of sound
in	*ar*	Right side speed of sound
in	*s*	Speed of the discontinuity

References G, gam, nIter, preFun(), and tol.

Referenced by exactFunc(), and flux_god().

6.13 ccfd/src/finiteVolume.c File Reference

Finite volume time derivative functions.

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "main.h"
#include "finiteVolume.h"
#include "mesh.h"
#include "readInTools.h"
#include "reconstruction.h"
#include "timeDiscretization.h"
#include "fluxCalculation.h"
#include "equation.h"
#include "source.h"
```

Include dependency graph for finiteVolume.c:

Functions

- void initFV (void)

  Initialize the finite volume method.
- void fvTimeDerivative (double time)

  Perform the spacial operator of the finite volume scheme.

Variables

- int spatialOrder
- int fluxFunction

6.13.1 Detailed Description

Finite volume time derivative functions.

Author

hhh

Date

Fri 27 Mar 2020 05:10:43 PM CET

6.13.2 Function Documentation

6.13.2.1 fvTimeDerivative()

```
void fvTimeDerivative (
            double time )
```

Perform the spacial operator of the finite volume scheme.

First, the local time step is calculated, then spacial gradients inside of the cells are reconstructed. Following that, the boundary conditions at the sides are applied and the numerical flux is calculated, using the specified flux function. Finally, the source term is evaluated and the time derivatives of all the elements are calculated.

Parameters

in	*time*	Calculation time at which to perform the finite volume differentiation

References elem_t::areaq, calcSource(), doCalcSource, elem_t::dt, elem_t::dtLoc, E, elem, elem_t::firstSide, side_t::flux, fluxCalculation(), nElems, side_t::nextElemSide, RHO, setBCatSides(), elem_t::source, spatial←Reconstruction(), timeOrder, elem_t::u_t, VX, and VY.

Referenced by buildMatrix(), explicitTimeStepEuler(), explicitTimeStepRK(), implicitTimeStep(), and matrixVector().

6.13.3 Variable Documentation

6.13.3.1 fluxFunction

```
int fluxFunction
```

the flux function to be used

6.13.3.2 spatialOrder

```
int spatialOrder
```

the spacial order to be used

Referenced by initFV(), and spatialReconstruction().

6.14 ccfd/src/finiteVolume.h File Reference

This graph shows which files directly or indirectly include this file:

Functions

- void initFV (void)

  Initialize the finite volume method.
- void fvTimeDerivative (double time)

  Perform the spacial operator of the finite volume scheme.

Variables

- int spatialOrder
- int fluxFunction

6.14.1 Detailed Description

Author

hhh

Date

Fri 27 Mar 2020 05:09:57 PM CET

6.14.2 Function Documentation

6.14.2.1 fvTimeDerivative()

```
void fvTimeDerivative (
            double time )
```

Perform the spacial operator of the finite volume scheme.

First, the local time step is calculated, then spacial gradients inside of the cells are reconstructed. Following that, the boundary conditions at the sides are applied and the numerical flux is calculated, using the specified flux function. Finally, the source term is evaluated and the time derivatives of all the elements are calculated.

Parameters

in	*time*	Calculation time at which to perform the finite volume differentiation

References elem_t::areaq, calcSource(), doCalcSource, elem_t::dt, elem_t::dtLoc, E, elem, elem_t::firstSide, side_t::flux, fluxCalculation(), nElems, side_t::nextElemSide, RHO, setBCatSides(), elem_t::source, spatial←Reconstruction(), timeOrder, elem_t::u_t, VX, and VY.

Referenced by buildMatrix(), explicitTimeStepEuler(), explicitTimeStepRK(), implicitTimeStep(), and matrixVector().

6.14.3 Variable Documentation

6.14.3.1 fluxFunction

```
int fluxFunction [extern]
```

the flux function to be used

6.14.3.2 spatialOrder

```
int spatialOrder [extern]
```

the spacial order to be used

Referenced by initFV(), and spatialReconstruction().

6.15 ccfd/src/fluxCalculation.c File Reference

Contains the flux calculation functions.

```
#include <math.h>
#include "main.h"
#include "mesh.h"
#include "equation.h"
#include "exactRiemann.h"
#include "boundary.h"
#include "linearSolver.h"
```

Include dependency graph for fluxCalculation.c:

Functions

- void flux_god (double rhoL, double rhoR, double vxL, double vxR, double vyL, double vyR, double pL, double pR, double fluxLoc[4])

  Godunov flux, which is the exact flux.
- void flux_roe (double rhoL, double rhoR, double vxL, double vxR, double vyL, double vyR, double pL, double pR, double fluxLoc[4])

  Roe flux.
- void flux_hll (double rhoL, double rhoR, double vxL, double vxR, double vyL, double vyR, double pL, double pR, double fluxLoc[4])

  HLL flux.
- void flux_hlle (double rhoL, double rhoR, double vxL, double vxR, double vyL, double vyR, double pL, double pR, double fluxLoc[4])

  HLLE flux.
- void flux_hllc (double rhoL, double rhoR, double vxL, double vxR, double vyL, double vyR, double pL, double pR, double fluxLoc[4])

  HLLC flux.
- void flux_lxf (double rhoL, double rhoR, double vxL, double vxR, double vyL, double vyR, double pL, double pR, double fluxLoc[4])

  Lax-Friedrichs flux.
- void flux_stw (double rhoL, double rhoR, double vxL, double vxR, double vyL, double vyR, double pL, double pR, double fluxLoc[4])

  Steger-Warming flux.
- void flux_cen (double rhoL, double rhoR, double vxL, double vxR, double vyL, double vyR, double pL, double pR, double fluxLoc[4])

  Central flux.
- void flux_ausmd (double rhoL, double rhoR, double vxL, double vxR, double vyL, double vyR, double pL, double pR, double fluxLoc[4])

  AUSMD flux.

- void flux_ausmdv (double rhoL, double rhoR, double vxL, double vxR, double vyL, double vyR, double pL, double pR, double fluxLoc[4])

  AUSMDV flux.
- void flux_vanleer (double rhoL, double rhoR, double vxL, double vxR, double vyL, double vyR, double pL, double pR, double fluxLoc[4])

  Van Leer flux.
- void convectiveFlux (double rhoL, double rhoR, double vxL, double vxR, double vyL, double vyR, double pL, double pR, double fluxLoc[4])

  Select the convective flux.
- void fluxCalculation (void)

  Perform the flux calculation.

6.15.1 Detailed Description

Contains the flux calculation functions.

Author

hhh

Date

Tue 31 Mar 2020 05:18:40 PM CEST

6.15.2 Function Documentation

6.15.2.1 convectiveFlux()

```
void convectiveFlux (
            double rhoL,
            double rhoR,
            double vxL,
            double vxR,
            double vyL,
            double vyR,
            double pL,
            double pR,
            double fluxLoc[4] )
```

Select the convective flux.

Parameters

in	*rhoL*	Left side density
in	*rhoR*	Right side density
in	*vxL*	Left side x-velocity
in	*vxR*	Right side x-velocity
in	*vyL*	Left side y-velocity
in	*vyR*	Right side y-velocity
in	*pL*	Left side pressure
in	*pR*	Right side pressure
out	*fluxLoc*	The local numeric flux

References AUSMD, AUSMDV, CEN, flux_ausmd(), flux_ausmdv(), flux_cen(), flux_god(), flux_hll(), flux_hllc(), flux_hlle(), flux_lxf(), flux_roe(), flux_stw(), flux_vanleer(), GOD, HLL, HLLC, HLLE, iFlux, LXF, ROE, STW, and VANLEER.

Referenced by fluxCalculation().

6.15.2.2 flux_ausmd()

```
void flux_ausmd (
            double rhoL,
            double rhoR,
            double vxL,
            double vxR,
            double vyL,
            double vyR,
            double pL,
            double pR,
            double fluxLoc[4] )
```

AUSMD flux.

Parameters

in	*rhoL*	Left side density
in	*rhoR*	Right side density
in	*vxL*	Left side x-velocity
in	*vxR*	Right side x-velocity
in	*vyL*	Left side y-velocity
in	*vyR*	Right side y-velocity
in	*pL*	Left side pressure
in	*pR*	Right side pressure
out	*fluxLoc*	The local numeric flux

References gam, and gam1q.

Referenced by convectiveFlux().

6.15.2.3 flux_ausmdv()

```
void flux_ausmdv (
            double rhoL,
            double rhoR,
            double vxL,
            double vxR,
            double vyL,
            double vyR,
            double pL,
            double pR,
            double fluxLoc[4] )
```

AUSMDV flux.

Bug This function produces incorrect output, refrain from using it for the time being

Parameters

in	*rhoL*	Left side density
in	*rhoR*	Right side density
in	*vxL*	Left side x-velocity
in	*vxR*	Right side x-velocity
in	*vyL*	Left side y-velocity
in	*vyR*	Right side y-velocity
in	*pL*	Left side pressure
in	*pR*	Right side pressure
out	*fluxLoc*	The local numeric flux

References gam, and gam1q.

Referenced by convectiveFlux().

6.15.2.4 flux_cen()

```
void flux_cen (
            double rhoL,
            double rhoR,
            double vxL,
            double vxR,
            double vyL,
            double vyR,
            double pL,
            double pR,
            double fluxLoc[4] )
```

Central flux.

Note

This flux is unconditionally unstable, it can be stabilized by adding artificial viscosity (Jameson method). This is not implemented, however.

Parameters

in	*rhoL*	Left side density
in	*rhoR*	Right side density
in	*vxL*	Left side x-velocity
in	*vxR*	Right side x-velocity
in	*vyL*	Left side y-velocity
in	*vyR*	Right side y-velocity
in	*pL*	Left side pressure
in	*pR*	Right side pressure
out	*fluxLoc*	The local numeric flux

References gam1q.

Referenced by convectiveFlux().

6.15.2.5 flux_god()

```
void flux_god (
            double rhoL,
            double rhoR,
            double vxL,
            double vxR,
            double vyL,
            double vyR,
            double pL,
            double pR,
            double fluxLoc[4] )
```

Godunov flux, which is the exact flux.

Parameters

in	*rhoL*	Left side density
in	*rhoR*	Right side density
in	*vxL*	Left side x-velocity
in	*vxR*	Right side x-velocity
in	*vyL*	Left side y-velocity
in	*vyR*	Right side y-velocity
in	*pL*	Left side pressure
in	*pR*	Right side pressure
out	*fluxLoc*	The local numeric flux

References exactRiemann(), gam, and gam1.

Referenced by convectiveFlux().

6.15.2.6 flux_hll()

```
void flux_hll (
            double rhoL,
            double rhoR,
            double vxL,
            double vxR,
            double vyL,
            double vyR,
            double pL,
            double pR,
            double fluxLoc[4] )
```

HLL flux.

Parameters

in	*rhoL*	Left side density
in	*rhoR*	Right side density
in	*vxL*	Left side x-velocity
in	*vxR*	Right side x-velocity
in	*vyL*	Left side y-velocity
in	*vyR*	Right side y-velocity
in	*pL*	Left side pressure
in	*pR*	Right side pressure
out	*fluxLoc*	The local numeric flux

References gam, gam1, gam1q, MX, and NVAR.

Referenced by convectiveFlux().

6.15.2.7 flux_hllc()

```
void flux_hllc (
            double rhoL,
            double rhoR,
            double vxL,
            double vxR,
            double vyL,
            double vyR,
            double pL,
            double pR,
            double fluxLoc[4] )
```

HLLC flux.

Parameters

in	*rhoL*	Left side density
in	*rhoR*	Right side density
in	*vxL*	Left side x-velocity
in	*vxR*	Right side x-velocity
in	*vyL*	Left side y-velocity
in	*vyR*	Right side y-velocity
in	*pL*	Left side pressure
in	*pR*	Right side pressure
out	*fluxLoc*	The local numeric flux

References gam, gam1, gam1q, MX, and NVAR.

Referenced by convectiveFlux().

6.15.2.8 flux_hlle()

```
void flux_hlle (
            double rhoL,
            double rhoR,
            double vxL,
            double vxR,
            double vyL,
            double vyR,
            double pL,
            double pR,
            double fluxLoc[4] )
```

HLLE flux.

Parameters

in	*rhoL*	Left side density
in	*rhoR*	Right side density
in	*vxL*	Left side x-velocity
in	*vxR*	Right side x-velocity
in	*vyL*	Left side y-velocity
in	*vyR*	Right side y-velocity
in	*pL*	Left side pressure
in	*pR*	Right side pressure
out	*fluxLoc*	The local numeric flux

References gam, gam1q, MX, and NVAR.

Referenced by convectiveFlux().

6.15.2.9 flux_lxf()

```
void flux_lxf (
            double rhoL,
            double rhoR,
            double vxL,
            double vxR,
            double vyL,
            double vyR,
            double pL,
            double pR,
            double fluxLoc[4] )
```

Lax-Friedrichs flux.

Parameters

in	*rhoL*	Left side density
in	*rhoR*	Right side density
in	*vxL*	Left side x-velocity

Parameters

in	*vxR*	Right side x-velocity
in	*vyL*	Left side y-velocity
in	*vyR*	Right side y-velocity
in	*pL*	Left side pressure
in	*pR*	Right side pressure
out	*fluxLoc*	The local numeric flux

References gam, gam1q, and NVAR.

Referenced by convectiveFlux().

6.15.2.10 flux_roe()

```
void flux_roe (
            double rhoL,
            double rhoR,
            double vxL,
            double vxR,
            double vyL,
            double vyR,
            double pL,
            double pR,
            double fluxLoc[4] )
```

Roe flux.

Parameters

in	*rhoL*	Left side density
in	*rhoR*	Right side density
in	*vxL*	Left side x-velocity
in	*vxR*	Right side x-velocity
in	*vyL*	Left side y-velocity
in	*vyR*	Right side y-velocity
in	*pL*	Left side pressure
in	*pR*	Right side pressure
out	*fluxLoc*	The local numeric flux

References gam, gam1, gam1q, and gam2.

Referenced by convectiveFlux().

6.15.2.11 flux_stw()

```
void flux_stw (
            double rhoL,
```

```
            double rhoR,
            double vxL,
            double vxR,
            double vyL,
            double vyR,
            double pL,
            double pR,
            double fluxLoc[4] )
```

Steger-Warming flux.

Parameters

in	*rhoL*	Left side density
in	*rhoR*	Right side density
in	*vxL*	Left side x-velocity
in	*vxR*	Right side x-velocity
in	*vyL*	Left side y-velocity
in	*vyR*	Right side y-velocity
in	*pL*	Left side pressure
in	*pR*	Right side pressure
out	*fluxLoc*	The local numeric flux

References gam, gam1, and gam1q.

Referenced by convectiveFlux().

6.15.2.12 flux_vanleer()

```
void flux_vanleer (
            double rhoL,
            double rhoR,
            double vxL,
            double vxR,
            double vyL,
            double vyR,
            double pL,
            double pR,
            double fluxLoc[4] )
```

Van Leer flux.

Parameters

in	*rhoL*	Left side density
in	*rhoR*	Right side density
in	*vxL*	Left side x-velocity
in	*vxR*	Right side x-velocity
in	*vyL*	Left side y-velocity
in	*vyR*	Right side y-velocity
in	*pL*	Left side pressure
in	*pR*	Right side pressure
out	*fluxLoc*	The local numeric flux

References gam, gam1, and gam1q.

Referenced by convectiveFlux().

6.15.2.13 fluxCalculation()

```
void fluxCalculation (
            void )
```

Perform the flux calculation.

Calculation of left and right state, the velocity vector is transformed into the normal system of the cell interfaces. The function finishes with a back rotation of the velocity vector into global coordinate system.

References side_t::baryBaryDist, side_t::baryBaryVec, side_t::connection, convectiveFlux(), E, side_t::elem, side↩_t::flux, side_t::len, MX, MY, side_t::n, NDIM, nSides, NVAR, P, side_t::pVar, elem_t::pVar, RHO, side, elem_t::u_x, elem_t::u_y, VX, VY, X, and Y.

Referenced by fvTimeDerivative().

6.16 ccfd/src/fluxCalculation.h File Reference

This graph shows which files directly or indirectly include this file:

Functions

- void fluxCalculation (void)

  Perform the flux calculation.

6.16.1 Detailed Description

Author

hhh

Date

Tue 31 Mar 2020 05:18:46 PM CEST

6.16.2 Function Documentation

6.16.2.1 fluxCalculation()

```
void fluxCalculation (
            void )
```

Perform the flux calculation.

Calculation of left and right state, the velocity vector is transformed into the normal system of the cell interfaces. The function finishes with a back rotation of the velocity vector into global coordinate system.

References side_t::baryBaryDist, side_t::baryBaryVec, side_t::connection, convectiveFlux(), E, side_t::elem, side←_t::flux, side_t::len, MX, MY, side_t::n, NDIM, nSides, NVAR, P, side_t::pVar, elem_t::pVar, RHO, side, elem_t::u_x, elem_t::u_y, VX, VY, X, and Y.

Referenced by fvTimeDerivative().

6.17 ccfd/src/initialCondition.c File Reference

Functions involving the initialization and application of initial conditions.

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include "main.h"
#include "readInTools.h"
#include "initialCondition.h"
#include "equation.h"
#include "memTools.h"
#include "timeDiscretization.h"
#include "mesh.h"
#include "exactFunction.h"
#include "equationOfState.h"
#include "cgnslib.h"
```

Include dependency graph for initialCondition.c:

Functions

- void initInitialCondition (void)

  Get initial flow conditions from the parameter file.
- void cgnsReadSolution (void)

  Read a solution from a CGNS file, used at restart.
- void setInitialCondition (void)

  Set initial flow field in all cells.
- void freeInitialCondition (void)

  Free all memory that was allocated for the reference state.

Variables

- int icType
- int nDomains
- int ∗ domainID
- double rp1Dinterface
- double alpha
- double ∗∗ refState

6.17.1 Detailed Description

Functions involving the initialization and application of initial conditions.

Author

hhh

Date

Fri 27 Mar 2020 02:31:24 PM CET

6.17.2 Variable Documentation

6.17.2.1 alpha

```
double alpha
```

incident angle of the flow

Referenced by calcCoef(), GMRES_M(), implicitTimeStep(), initBoundary(), initInitialCondition(), and matrixVector().

6.17.2.2 domainID

```
int* domainID
```

domain ID vector, from 1 to `nDomains`

Referenced by initInitialCondition().

6.17.2.3 icType

```
int icType
```

type of initial condition

Referenced by initInitialCondition(), and setInitialCondition().

6.17.2.4 nDomains

```
int nDomains
```

number of domains where initial conditions are applied

Referenced by initInitialCondition(), and setInitialCondition().

6.17.2.5 refState

```
double** refState
```

primitive variable state in domain

Referenced by calcCoef(), exactFunc(), freeInitialCondition(), initInitialCondition(), and setInitialCondition().

6.17.2.6 rp1Dinterface

```
double rp1Dinterface
```

interface for a Riemann problem

Referenced by exactFunc(), and initInitialCondition().

6.18 ccfd/src/initialCondition.h File Reference

This graph shows which files directly or indirectly include this file:

Functions

- void initInitialCondition (void)

    Get initial flow conditions from the parameter file.
- void setInitialCondition (void)

    Set initial flow field in all cells.
- void freeInitialCondition (void)

    Free all memory that was allocated for the reference state.

Variables

- int icType
- int nDomains
- int ∗ domainID
- double rp1Dinterface
- double alpha
- double ∗∗ refState

6.18.1 Detailed Description

Author

hhh

Date

Fri 27 Mar 2020 02:28:25 PM CET

6.18.2 Variable Documentation

6.18.2.1 alpha

```
double alpha [extern]
```

incident angle of the flow

Referenced by calcCoef(), GMRES_M(), implicitTimeStep(), initBoundary(), initInitialCondition(), and matrixVector().

6.18.2.2 domainID

```
int* domainID [extern]
```

domain ID vector, from 1 to `nDomains`

Referenced by initInitialCondition().

6.18.2.3 icType

```
int icType [extern]
```

type of initial condition

Referenced by initInitialCondition(), and setInitialCondition().

6.18.2.4 nDomains

```
int nDomains [extern]
```

number of domains where initial conditions are applied

Referenced by initInitialCondition(), and setInitialCondition().

6.18.2.5 refState

`double** refState [extern]`

primitive variable state in domain

Referenced by calcCoef(), exactFunc(), freeInitialCondition(), initInitialCondition(), and setInitialCondition().

6.18.2.6 rp1Dinterface

`double rp1Dinterface [extern]`

interface for a Riemann problem

Referenced by exactFunc(), and initInitialCondition().

6.19 ccfd/src/linearSolver.c File Reference

Contains the functions for solving the linear system of equations during implicit calculations.

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <math.h>
#include <float.h>
#include <string.h>
#include "main.h"
#include "linearSolver.h"
#include "readInTools.h"
#include "mesh.h"
#include "timeDiscretization.h"
#include "memTools.h"
#include "fluxCalculation.h"
#include "equationOfState.h"
#include "finiteVolume.h"
```

Include dependency graph for linearSolver.c:

Functions

- void initLinearSolver (void)

    Initialize linear solver.
- double vectorDotProduct (double **A, double **B)

    Compute dot product for two state vectors A and B for every element.
- bool calcDinv (double **A, double **Ainv)

    Compute inverse of a 4x4 matrix.
- void buildMatrix (double time, double dt)

    Compute the global Jacobian matrix by use of finite differences.
- void LUSGS (double **B, double **delX)

    LUSGS preconditioner.
- void matrixVector (double time, double dt, double alpha, double **v, double **res)

    Computes matrix vector product using spatial operator and finite differences.
- void GMRES_M (double time, double dt, double alpha, double **B, double normB, double *abortCrit, double **delX)

    Uses matrix free to solve the linear system.
- void freeLinearSolver (void)

    Free all memory that was allocated for.

Variables

- int nKdim
- int nNewtonIter
- int nNewtonIterGlobal
- int nGMRESiterGlobal
- int nInnerNewton
- int nInnerGMRES
- bool usePrecond
- double rEps0
- double srEps0
- double eps2newton
- double eps2newton_sq
- double epsGMRES
- double gamEW
- double ** XK
- double ** R_XK
- double *** D
- double *** Dinv
- double ** dRdU
- double *** V
- double *** Z
- double ** R0
- double ** W
- double ** deltaXstar

6.19.1 Detailed Description

Contains the functions for solving the linear system of equations during implicit calculations.

Author

hhh

Date

Sat 28 Mar 2020 03:41:56 PM CET

6.19.2 Function Documentation

6.19.2.1 buildMatrix()

```
void buildMatrix (
            double time,
            double dt )
```

Compute the global Jacobian matrix by use of finite differences.

Parameters

in	*time*	Computation time at calculation
in	*dt*	Time step at calculation

References calcDinv(), side_t::connection, D, Dinv, dRdU, elem, side_t::elem, elem_t::firstSide, fvTimeDerivative(), elem_t::id, nElems, side_t::nextElemSide, NVAR, elem_t::pVar, R_XK, rEps0, srEps0, and elem_t::u_t.

Referenced by GMRES_M().

6.19.2.2 calcDinv()

```
bool calcDinv (
            double ** A,
            double ** Ainv )
```

Compute inverse of a 4x4 matrix.

Parameters

in	*A*	The 4x4 matrix to be inverted
out	*Ainv*	The 4x4 inverse matrix of A

Returns

0 = Inverse does not exist, 1 = Inverse computed

Referenced by buildMatrix().

6.19.2.3 GMRES_M()

```
void GMRES_M (
            double time,
            double dt,
            double alpha,
            double ** B,
            double normB,
            double * abortCrit,
            double ** delX )
```

Uses matrix free to solve the linear system.

Parameters

in	*time*	Computation time at calculation
in	*dt*	Time step at calculation
in	*alpha*	Relaxation parameter
in	*B*	Right hand side
in	*normB*	Norm of right hand side
in,out	*abortCrit*	GMRES abort criterium
out	*delX*	Resulting x vector of the linear system

References alpha, buildMatrix(), E, epsGMRES, gam, LUSGS(), matrixVector(), MX, MY, nElems, nGMRESiterGlobal, nInnerGMRES, nKdim, R0, RHO, t, usePrecond, V, vectorDotProduct(), W, and Z.

Referenced by implicitTimeStep().

6.19.2.4 LUSGS()

```
void LUSGS (
            double ** B,
            double ** delX )
```

LUSGS preconditioner.

Parameters

in	*B*	Old vector, to be preconditioned
out	*delX*	Preconditioned vector

Note

This function is slow to execute, it might be possible to speed it up via the use of omp locks for every element

References side_t::connection, deltaXstar, Dinv, dRdU, elem, side_t::elem, elem_t::firstSide, elem_t::id, nElems, side_t::nextElemSide, and NVAR.

Referenced by GMRES_M().

6.19.2.5 matrixVector()

```
void matrixVector (
            double time,
            double dt,
            double alpha,
            double ** v,
            double ** res )
```

Computes matrix vector product using spatial operator and finite differences.

Parameters

in	*time*	Computation time at calculation
in	*dt*	Time step at calculation
in	*alpha*	Relaxation parameter
in	*v*	Input vector for the matrix vector product
out	*res*	Resulting vector of the matrix vector product

References alpha, consPrim(), elem_t::cVar, E, elem, fvTimeDerivative(), MX, MY, nElems, elem_t::pVar, R_XK, rEps0, RHO, elem_t::u_t, vectorDotProduct(), and XK.

Referenced by GMRES_M().

6.19.2.6 vectorDotProduct()

```
double vectorDotProduct (
             double ** A,
             double ** B )
```

Compute dot product for two state vectors A and B for every element.

Parameters

in	*A*	First state vector for every element
in	*B*	Second state vector for every element

Returns

sum(sum(A_ij ∗ B_ij, i = RHO,MX,MY,E), i = 1..nElems)

References E, MX, MY, nElems, and RHO.

Referenced by GMRES_M(), implicitTimeStep(), and matrixVector().

6.19.3 Variable Documentation

6.19.3.1 D

```
double*** D
```

D-Matrix of LUSGS procedure

Referenced by buildMatrix(), freeLinearSolver(), and initLinearSolver().

6.19.3.2 deltaXstar

```
double** deltaXstar
```

temporary array, used in LUSGS

Referenced by freeLinearSolver(), initLinearSolver(), and LUSGS().

6.19.3.3 Dinv

```
double*** Dinv
```

inverse of D-Matrix

Referenced by buildMatrix(), freeLinearSolver(), initLinearSolver(), and LUSGS().

6.19.3.4 dRdU

```
double** dRdU
```

dR / dU

Referenced by buildMatrix(), freeLinearSolver(), initLinearSolver(), and LUSGS().

6.19.3.5 eps2newton

```
double eps2newton
```

square of newton relative epsilon

Referenced by implicitTimeStep(), and initLinearSolver().

6.19.3.6 eps2newton_sq

```
double eps2newton_sq
```

newton relative epsilon

Referenced by implicitTimeStep(), and initLinearSolver().

6.19.3.7 epsGMRES

```
double epsGMRES
```

GMRES relative epsilon

Referenced by GMRES_M(), and initLinearSolver().

6.19.3.8 gamEW

```
double gamEW
```

gamma parameter for Eisenstat Walker

Referenced by implicitTimeStep(), and initLinearSolver().

6.19.3.9 nGMRESiterGlobal

```
int nGMRESiterGlobal
```

global number of GMRES iterations

Referenced by GMRES_M(), initLinearSolver(), and timeDisc().

6.19.3.10 nInnerGMRES

```
int nInnerGMRES
```

maximum number of GMRES iterations for one stage

Referenced by GMRES_M(), and initLinearSolver().

6.19.3.11 nInnerNewton

```
int nInnerNewton
```

maximum number of Newton iterations for one stage

Referenced by implicitTimeStep(), and initLinearSolver().

6.19.3.12 nKdim

```
int nKdim
```

number Krylov spaces

Referenced by GMRES_M(), and initLinearSolver().

6.19.3.13 nNewtonIter

```
int nNewtonIter
```

maximum number of Newton iterations

Referenced by implicitTimeStep(), and initLinearSolver().

6.19.3.14 nNewtonIterGlobal

```
int nNewtonIterGlobal
```

global number of Newton iterations

Referenced by implicitTimeStep(), initLinearSolver(), and timeDisc().

6.19.3.15 R0

```
double** R0
```

temporary array, used in GMRES

Referenced by freeLinearSolver(), GMRES_M(), and initLinearSolver().

6.19.3.16 R_XK

```
double** R_XK
```

residual of kth vector array

Referenced by buildMatrix(), freeLinearSolver(), implicitTimeStep(), initLinearSolver(), and matrixVector().

6.19.3.17 rEps0

```
double rEps0
```

DBL_EPSILON

Referenced by buildMatrix(), initLinearSolver(), and matrixVector().

6.19.3.18 srEps0

```
double srEps0
```

sqrt(DBL_EPSILON)

Referenced by buildMatrix(), and initLinearSolver().

6.19.3.19 usePrecond

```
bool usePrecond
```

use LUSGS preconditioner flag

Referenced by freeLinearSolver(), GMRES_M(), and initLinearSolver().

6.19.3.20 V

```
double*** V
```

temporary array, used in GMRES

Referenced by freeLinearSolver(), GMRES_M(), and initLinearSolver().

6.19.3.21 W

```
double** W
```

temporary array, used in GMRES

Referenced by freeLinearSolver(), GMRES_M(), and initLinearSolver().

6.19.3.22 XK

```
double** XK
```

kth X vector array

Referenced by freeLinearSolver(), implicitTimeStep(), initLinearSolver(), and matrixVector().

6.19.3.23 Z

```
double*** Z
```

temporary array, used in GMRES

Referenced by freeLinearSolver(), GMRES_M(), and initLinearSolver().

6.20 ccfd/src/linearSolver.h File Reference

```
#include <stdbool.h>
#include "main.h"
#include "mesh.h"
```

Include dependency graph for linearSolver.h: This graph shows which files directly or indirectly include this file:

Functions

- void initLinearSolver (void)

  Initialize linear solver.
- double vectorDotProduct (double **A, double **B)

  Compute dot product for two state vectors A and B for every element.
- void GMRES_M (double time, double dt, double alpha, double **B, double normB, double *abortCrit, double **deltaX)

  Uses matrix free to solve the linear system.
- void freeLinearSolver (void)

  Free all memory that was allocated for.

Variables

- int nKdim
- int nNewtonIter
- int nNewtonIterGlobal
- int nGMRESiterGlobal
- int nInnerNewton
- int nInnerGMRES
- bool usePrecond
- double rEps0
- double srEps0
- double eps2newton
- double eps2newton_sq
- double epsGMRES
- double gamEW
- double ** XK
- double ** R_XK

6.20.1 Detailed Description

Author

hhh

Date

Sat 28 Mar 2020 03:09:46 PM CET

6.20.2 Function Documentation

6.20.2.1 GMRES_M()

```
void GMRES_M (
            double time,
            double dt,
            double alpha,
            double ** B,
            double normB,
            double * abortCrit,
            double ** delX )
```

Uses matrix free to solve the linear system.

Parameters

in	*time*	Computation time at calculation
in	*dt*	Time step at calculation
in	*alpha*	Relaxation parameter
in	*B*	Right hand side
in	*normB*	Norm of right hand side
in,out	*abortCrit*	GMRES abort criterium
out	*delX*	Resulting x vector of the linear system

References alpha, buildMatrix(), E, epsGMRES, gam, LUSGS(), matrixVector(), MX, MY, nElems, nGMRESiter↩ Global, nInnerGMRES, nKdim, R0, RHO, t, usePrecond, V, vectorDotProduct(), W, and Z.

Referenced by implicitTimeStep().

6.20.2.2 vectorDotProduct()

```
double vectorDotProduct (
            double ** A,
            double ** B )
```

Compute dot product for two state vectors A and B for every element.

Parameters

in	*A*	First state vector for every element
in	*B*	Second state vector for every element

Returns

sum(sum(A_ij ∗ B_ij, i = RHO,MX,MY,E), i = 1..nElems)

References E, MX, MY, nElems, and RHO.

Referenced by GMRES_M(), implicitTimeStep(), and matrixVector().

6.20.3 Variable Documentation

6.20.3.1 eps2newton

```
double eps2newton  [extern]
```

square of newton relative epsilon

Referenced by implicitTimeStep(), and initLinearSolver().

6.20.3.2 eps2newton_sq

```
double eps2newton_sq  [extern]
```

newton relative epsilon

Referenced by implicitTimeStep(), and initLinearSolver().

6.20.3.3 epsGMRES

```
double epsGMRES  [extern]
```

GMRES relative epsilon

Referenced by GMRES_M(), and initLinearSolver().

6.20.3.4 gamEW

```
double gamEW [extern]
```

gamma parameter for Eisenstat Walker

Referenced by implicitTimeStep(), and initLinearSolver().

6.20.3.5 nGMRESiterGlobal

```
int nGMRESiterGlobal [extern]
```

global number of GMRES iterations

Referenced by GMRES_M(), initLinearSolver(), and timeDisc().

6.20.3.6 nInnerGMRES

```
int nInnerGMRES [extern]
```

maximum number of GMRES iterations for one stage

Referenced by GMRES_M(), and initLinearSolver().

6.20.3.7 nInnerNewton

```
int nInnerNewton [extern]
```

maximum number of Newton iterations for one stage

Referenced by implicitTimeStep(), and initLinearSolver().

6.20.3.8 nKdim

```
int nKdim [extern]
```

number Krylov spaces

Referenced by GMRES_M(), and initLinearSolver().

6.20.3.9 nNewtonIter

`int nNewtonIter [extern]`

maximum number of Newton iterations

Referenced by implicitTimeStep(), and initLinearSolver().

6.20.3.10 nNewtonIterGlobal

`int nNewtonIterGlobal [extern]`

global number of Newton iterations

Referenced by implicitTimeStep(), initLinearSolver(), and timeDisc().

6.20.3.11 R_XK

`double** R_XK [extern]`

residual of kth vector array

Referenced by buildMatrix(), freeLinearSolver(), implicitTimeStep(), initLinearSolver(), and matrixVector().

6.20.3.12 rEps0

`double rEps0 [extern]`

DBL_EPSILON

Referenced by buildMatrix(), initLinearSolver(), and matrixVector().

6.20.3.13 srEps0

`double srEps0 [extern]`

sqrt(DBL_EPSILON)

Referenced by buildMatrix(), and initLinearSolver().

6.20.3.14 usePrecond

```
bool usePrecond [extern]
```

use LUSGS preconditioner flag

Referenced by freeLinearSolver(), GMRES_M(), and initLinearSolver().

6.20.3.15 XK

```
double** XK [extern]
```

kth X vector array

Referenced by freeLinearSolver(), implicitTimeStep(), initLinearSolver(), and matrixVector().

6.21 ccfd/src/main.c File Reference

Contains the main function of `ccfd`

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdbool.h>
#include "readInTools.h"
#include "timeDiscretization.h"
#include "output.h"
#include "equation.h"
#include "boundary.h"
#include "mesh.h"
#include "initialCondition.h"
#include "finiteVolume.h"
#include "linearSolver.h"
#include "analyze.h"
```

Include dependency graph for main.c:

Functions

- int main (int argc, char *argv[ ])
  Main function.

6.21.1 Detailed Description

Contains the main function of `ccfd`

Author

hhh

Date

Sat 21 Mar 2020 10:41:51 AM CET

6.21.2 Function Documentation

6.21.2.1 main()

```
int main (
            int argc,
            char * argv[] )
```

Main function.

This function starts the program by initializing all the necessary global variables and initializing the time discretization loop. The function finishes by deallocating all the allocated memory.

Parameters

in	*argc*	The number of command line arguments passed to `main`
in	*argv*	The argument vector containing the command line arguments

Returns

0 = Success, 1 = Error during execution

References fillCmds(), freeAnalyze(), freeBoundary(), freeInitialCondition(), freeLinearSolver(), freeMesh(), freeOutputTimes(), getBool(), ignoredCmds(), iniIterationNumber, initAnalyze(), initBoundary(), initEquation(), initFV(), initInitialCondition(), initLinearSolver(), initMesh(), initOutput(), initTimeDisc(), isRestart, isStationary, outputTimes, restartTime, setInitialCondition(), startTime, strIniCondFile, and timeDisc().

6.22 ccfd/src/main.h File Reference

Contains the global constants and definitions.

This graph shows which files directly or indirectly include this file:

Macros

- #define STRLEN 256

Enumerations

- enum conservativeVariables {
  RHO ,
  MX ,
  MY ,
  E }

  Index aliases for the conservative variables vector.

- enum primitiveVariables {
  VX = 1 ,
  VY ,
  P }

  Index aliases for the primitive variables vector.
- enum directions {
  X ,
  Y }

  Index aliases for the x- and y-components of a vector.
- enum boundaryConditionType {
  SLIPWALL = 1 ,
  WALL ,
  INFLOW ,
  OUTFLOW ,
  CHARACTERISTIC ,
  EXACTSOL ,
  PERIODIC ,
  PRESSURE_OUT }

  Aliases for the different boundary condition types.
- enum cartesianMeshSides {
  BOTTOM ,
  RIGHT ,
  TOP ,
  LEFT }

  Aliases for the sides of a cartesian mesh.
- enum meshType {
  UNSTRUCTURED ,
  CARTESIAN }

  Flag for the mesh type.
- enum fluxFunction {
  GOD = 1 ,
  ROE ,
  HLL ,
  HLLE ,
  HLLC ,
  LXF ,
  STW ,
  CEN ,
  AUSMD ,
  AUSMDV ,
  VANLEER }

  Aliases for the different flux functions.
- enum limiterFunction {
  BARTHJESPERSEN = 1 ,
  VENKATAKRISHNAN }

  Flag for the limiter function.
- enum generalParameters {
  NDIM = 2 ,
  NVAR = 4 ,
  NBC = 20 }

  General parameters for the Program.
- enum ioFormat {
  CGNS = 1 ,
  CURVE ,
  CSV }

  Output format for the results.

- enum clcdResiduals {
  CL = 4 ,
  CD }

  Index aliases for the residual vector.

6.22.1 Detailed Description

Contains the global constants and definitions.

Author

hhh

Date

Sat 21 Mar 2020 10:44:50 AM CET

6.22.2 Macro Definition Documentation

6.22.2.1 STRLEN

```
#define STRLEN 256
```

string length

6.22.3 Enumeration Type Documentation

6.22.3.1 boundaryConditionType

```
enum boundaryConditionType
```

Aliases for the different boundary condition types.

Enumerator

SLIPWALL	slip wall, or Euler wall
WALL	no slip wall, or Navier-Stokes wall
INFLOW	supersonic inflow
OUTFLOW	supersonic outflow
CHARACTERISTIC	characteristic BC, or subsonic inflow
EXACTSOL	exact solution
PERIODIC	periodic BC
PRESSURE_OUT	subsonic pressure outflow

6.22.3.2 cartesianMeshSides

```
enum cartesianMeshSides
```

Aliases for the sides of a cartesian mesh.

Enumerator

BOTTOM	bottom side
RIGHT	right side
TOP	top side
LEFT	left side

6.22.3.3 clcdResiduals

```
enum clcdResiduals
```

Index aliases for the residual vector.

Enumerator

CL	lift coefficient
CD	drag coefficient

6.22.3.4 conservativeVariables

```
enum conservativeVariables
```

Index aliases for the conservative variables vector.

Enumerator

RHO	density
MX	momentum in x-direction
MY	momentum in y-direction
E	energy

6.22.3.5 directions

```
enum directions
```

Index aliases for the x- and y-components of a vector.

Enumerator

X	x-direction
Y	y-direction

6.22.3.6 fluxFunction

```
enum fluxFunction
```

Aliases for the different flux functions.

Enumerator

GOD	Godunov flux function
ROE	Roe flux function
HLL	HLL flux function
HLLE	HLLE flux function
HLLC	HLLC flux function
LXF	Lax-Friedrichs flux function
STW	Steger-Warming flux function
CEN	central flux function
AUSMD	AUSMD flux function
AUSMDV	AUSMDV flux function
VANLEER	Van Leer flux function

6.22.3.7 generalParameters

```
enum generalParameters
```

General parameters for the Program.

Enumerator

NDIM	number of dimensions, cannot be changed
NVAR	number of variables, primitive or conservative
NBC	maximum number of boundary conditions

6.22.3.8 ioFormat

```
enum ioFormat
```

Output format for the results.

Enumerator

CGNS	.CGNS file format
CURVE	.curve file format
CSV	.csv file format

6.22.3.9 limiterFunction

```
enum limiterFunction
```

Flag for the limiter function.

Enumerator

BARTHJESPERSEN	Barth & Jespersen limiter
VENKATAKRISHNAN	Venkatakrishnan limiter

6.22.3.10 meshType

```
enum meshType
```

Flag for the mesh type.

Enumerator

UNSTRUCTURED	unstructured
CARTESIAN	cartesian

6.22.3.11 primitiveVariables

```
enum primitiveVariables
```

Index aliases for the primitive variables vector.

Enumerator

VX	velocity in x-direction
VY	velocity in y-direction
P	pressure

6.23 ccfd/src/memTools.c File Reference

Memory management functions.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "cgnslib.h"
```

Include dependency graph for memTools.c:

Functions

- long ** dyn2DintArray (long I, long J)
  Allocate a dynamic 2D array of integers.
- cgsize_t ** dyn2DcgsizeArray (long I, long J)
  Allocate a dynamic 2D array of cgsize_t.
- double ** dyn2DdblArray (long I, long J)
  Allocate a dynamic 2D array of doubles.
- long *** dyn3DintArray (long I, long J, long K)
  Allocate a dynamic 3D array of integers.
- double *** dyn3DdblArray (long I, long J, long K)
  Allocate a dynamic 3D array of doubles.
- double **** dyn4DdblArray (long I, long J, long K, long L)
  Allocate a dynamic 4D array of doubles.
- char ** dynStringArray (long I, long J)
  Allocate a dynamic array of strings.

6.23.1 Detailed Description

Memory management functions.

Author

hhh

Date

Fri 27 Mar 2020 02:38:03 PM CET

Note

Manual Memory Management:
"The manual type involves malloc and free, and is where most of your segfaults happen. This memory model is why Jesus weeps when he has to code in C."
- Ben Klemens

6.23.2 Function Documentation

6.23.2.1 dyn2DcgsizeArray()

```
cgsize_t** dyn2DcgsizeArray (
            long I,
            long J )
```

Allocate a dynamic 2D array of cgsize_t.

Parameters

in	*I*	Number of elements in the first dimension
in	*J*	Number of elements in the second dimension

Returns

Pointer to a 2D cgsize_t array

Referenced by cgnsWriteMesh().

6.23.2.2 dyn2DdblArray()

```
double** dyn2DdblArray (
            long I,
            long J )
```

Allocate a dynamic 2D array of doubles.

Parameters

in	*I*	Number of elements in the first dimension
in	*J*	Number of elements in the second dimension

Returns

Pointer to a 2D double array

Referenced by cgnsWriteMesh(), createCartMesh(), createReconstructionInfo(), csvOutput(), curveOutput(), init↩ InitialCondition(), initLinearSolver(), initRecordPoints(), initTimeDisc(), readCGNS(), readEMC2(), and readGmsh().

6.23.2.3 dyn2DintArray()

```
long** dyn2DintArray (
            long I,
            long J )
```

Allocate a dynamic 2D array of integers.

Parameters

in	*I*	Number of elements in the first dimension
in	*J*	Number of elements in the second dimension

Returns

Pointer to a 2D integer array

Referenced by createCartMesh(), readCGNS(), readEMC2(), and readGmsh().

6.23.2.4 dyn3DdblArray()

```
double*** dyn3DdblArray (
            long I,
            long J,
            long K )
```

Allocate a dynamic 3D array of doubles.

Parameters

in	*I*	Number of elements in the first dimension
in	*J*	Number of elements in the second dimension
in	*K*	Number of elements in the third dimension

Returns

Pointer to a 3D double array

Referenced by initLinearSolver().

6.23.2.5 dyn3DintArray()

```
long*** dyn3DintArray (
            long I,
            long J,
            long K )
```

Allocate a dynamic 3D array of integers.

Parameters

in	*I*	Number of elements in the first dimension
in	*J*	Number of elements in the second dimension
in	*K*	Number of elements in the third dimension

Returns

Pointer to a 3D integer array

6.23.2.6 dyn4DdblArray()

```
double**** dyn4DdblArray (
            long I,
            long J,
            long K,
            long L )
```

Allocate a dynamic 4D array of doubles.

Parameters

in	*I*	Number of elements in the first dimension
in	*J*	Number of elements in the second dimension
in	*K*	Number of elements in the third dimension
in	*L*	Number of elements in the fourth dimension

Returns

Pointer to a 4D double array

6.23.2.7 dynStringArray()

```
char** dynStringArray (
            long I,
            long J )
```

Allocate a dynamic array of strings.

Parameters

in	*I*	Number of elements in the first dimension
in	*J*	String length of each element

Returns

Pointer to a string array

6.24 ccfd/src/memTools.h File Reference

`#include "cgnslib.h"`
Include dependency graph for memTools.h: This graph shows which files directly or indirectly include this file:

Functions

- long ** dyn2DintArray (long I, long J)

Allocate a dynamic 2D array of integers.
- cgsize_t ** dyn2DcgsizeArray (long I, long J)

  Allocate a dynamic 2D array of cgsize_t.
- double ** dyn2DdblArray (long I, long J)

  Allocate a dynamic 2D array of doubles.
- long *** dyn3DintArray (long I, long J, long K)

  Allocate a dynamic 3D array of integers.
- double *** dyn3DdblArray (long I, long J, long K)

  Allocate a dynamic 3D array of doubles.
- double **** dyn4DdblArray (long I, long J, long K, long L)

  Allocate a dynamic 4D array of doubles.
- char ** dynStringArray (long I, long J)

  Allocate a dynamic array of strings.

6.24.1 Detailed Description

Author

hhh

Date

Fri 27 Mar 2020 02:38:45 PM CET

6.24.2 Function Documentation

6.24.2.1 dyn2DcgsizeArray()

```
cgsize_t** dyn2DcgsizeArray (
            long I,
            long J )
```

Allocate a dynamic 2D array of cgsize_t.

Parameters

in	*I*	Number of elements in the first dimension
in	*J*	Number of elements in the second dimension

Returns

Pointer to a 2D cgsize_t array

Referenced by cgnsWriteMesh().

6.24.2.2 dyn2DdblArray()

```
double** dyn2DdblArray (
            long I,
            long J )
```

Allocate a dynamic 2D array of doubles.

Parameters

in	*I*	Number of elements in the first dimension
in	*J*	Number of elements in the second dimension

Returns

Pointer to a 2D double array

Referenced by cgnsWriteMesh(), createCartMesh(), createReconstructionInfo(), csvOutput(), curveOutput(), init↩ InitialCondition(), initLinearSolver(), initRecordPoints(), initTimeDisc(), readCGNS(), readEMC2(), and readGmsh().

6.24.2.3 dyn2DintArray()

```
long** dyn2DintArray (
            long I,
            long J )
```

Allocate a dynamic 2D array of integers.

Parameters

in	*I*	Number of elements in the first dimension
in	*J*	Number of elements in the second dimension

Returns

Pointer to a 2D integer array

Referenced by createCartMesh(), readCGNS(), readEMC2(), and readGmsh().

6.24.2.4 dyn3DdblArray()

```
double*** dyn3DdblArray (
            long I,
            long J,
            long K )
```

Allocate a dynamic 3D array of doubles.

Parameters

in	*I*	Number of elements in the first dimension
in	*J*	Number of elements in the second dimension
in	*K*	Number of elements in the third dimension

Returns

Pointer to a 3D double array

Referenced by initLinearSolver().

6.24.2.5 dyn3DintArray()

```
long*** dyn3DintArray (
            long I,
            long J,
            long K )
```

Allocate a dynamic 3D array of integers.

Parameters

in	*I*	Number of elements in the first dimension
in	*J*	Number of elements in the second dimension
in	*K*	Number of elements in the third dimension

Returns

Pointer to a 3D integer array

6.24.2.6 dyn4DdblArray()

```
double**** dyn4DdblArray (
            long I,
            long J,
            long K,
            long L )
```

Allocate a dynamic 4D array of doubles.

Parameters

in	*I*	Number of elements in the first dimension
in	*J*	Number of elements in the second dimension
in	*K*	Number of elements in the third dimension
in	*L*	Number of elements in the fourth dimension

Returns

Pointer to a 4D double array

6.24.2.7 dynStringArray()

```
char** dynStringArray (
            long I,
            long J )
```

Allocate a dynamic array of strings.

Parameters

in	*I*	Number of elements in the first dimension
in	*J*	String length of each element

Returns

Pointer to a string array

6.25 ccfd/src/mesh.c File Reference

Contains all the functions for reading and creating meshes.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <stdbool.h>
#include <float.h>
#include "main.h"
#include "mesh.h"
#include "readInTools.h"
#include "output.h"
#include "timeDiscretization.h"
#include "memTools.h"
#include "initialCondition.h"
#include "cgnslib.h"
```

Include dependency graph for mesh.c:

Data Structures

- struct sideList_t

  Helper structure for reading in the sides and deviding them into BC sides and non-BC sides.

Functions

- void createReconstructionInfo (elem_t ∗aElem)

  Compute required vectors for reconstruction.
- void connectPeriodicBC (void)

  Create connection for periodic BCs.
- void createSideInfo (side_t ∗aSide)

  Create side info: normal vector, side length, and ghost cells.
- void createElemInfo (elem_t ∗aElem)

  Compute the cell specific values: barycenter, area, projection length of an element.
- int compare (const void ∗a, const void ∗b)

  Compare two elements of the `sideList` list, used for sorting the list.
- void createCartMesh (double ∗∗∗vertex, long ∗nVertices, long ∗∗∗BCedge, long ∗nBCedges, long ∗∗∗quad)

  Create a cartesian mesh.
- void readGmsh (char fileName[STRLEN], double ∗∗∗vertex, long ∗nVertices, long ∗∗∗BCedge, long ∗n←BCedges, long ∗∗∗tria, long ∗∗∗quad)

  Read in a gmsh mesh file.
- void readEMC2 (char fileName[STRLEN], double ∗∗∗vertex, long ∗nVertices, long ∗∗∗BCedge, long ∗n←BCedges, long ∗∗∗tria, long ∗∗∗quad)

  Read in EMC2 mesh file.
- void readCGNS (char fileName[STRLEN], double ∗∗∗vertex, long ∗nVertices, long ∗∗∗BCedge, long ∗n←BCedges, long ∗∗∗tria, long ∗∗∗quad)

  Read in a CGNS mesh.
- void createMesh (void)

  Create a cartesian or structured mesh.
- void readMesh (void)

  Read in and store the mesh parameters from the parameter file.
- void initMesh (void)

  Initialize the mesh and call `readMesh`
- void freeMesh (void)

  Free all allocated memory of the mesh.

Variables

- char strMeshFormat [STRLEN]
- char strMeshFile [STRLEN]
- char strIniCondFile [STRLEN]
- cartMesh_t cartMesh
- char gridFile [STRLEN]
- int meshType
- int meshFormat
- long nNodes
- long nElems
- long nTrias
- long nQuads
- long nSides
- long nBCsides
- long nInnerSides
- double totalArea_q
- double xMin
- double xMax
- double yMin

- double yMax
- double dxRef
- elem_t ** elem
- side_t ** side
- side_t ** BCside
- elem_t * firstElem
- node_t * firstNode
- side_t * firstSide
- sidePtr_t * firstBCside

6.25.1 Detailed Description

Contains all the functions for reading and creating meshes.

Author

hhh

Date

Tue 24 Mar 2020 12:45:57 PM CET

6.25.2 Function Documentation

6.25.2.1 compare()

```
int compare (
            const void * a,
            const void * b )
```

Compare two elements of the `sideList` list, used for sorting the list.

Sort sideList array in order to retrieve the connectivity info more efficiently. Basically we want to sort the sides on three different levels: first, by their first node ID, secondly, by their second node ID, and thirdly by if they are a boundary condition side. The result might look something like this: 3 3 1 1 2 0 2 1 1 1 3 0 1 2 0 --> 2 1 1 3 1 0 2 2 1 2 2 1 3 3 0 3 3 0 3 3 1

Parameters

in	*a*	Pointer an element of the side list
in	*b*	Pointer an element of the side list

Returns

The difference of a and b, if the values are the same, the move to next sorting metric

References sideList_t::BC, and sideList_t::node.

Referenced by createMesh().

6.25.2.2 createCartMesh()

```
void createCartMesh (
            double *** vertex,
            long * nVertices,
            long *** BCedge,
            long * nBCedges,
            long *** quad )
```

Create a cartesian mesh.

Parameters

in,out	*vertex*	Pointer to 2D array, used for the vertices
in,out	*nVertices*	Pointer to the number of total vertices in `vertex`
in,out	*BCedge*	Pointer to 2D array, used for the BC edges
in,out	*nBCedges*	Pointer to the number of total BC edges
in,out	*quad*	Pointer to a 2D array, used for the quadrangles

References cartMesh_t::BCrange, cartMesh_t::BCtype, BOTTOM, cartMesh, dyn2DdblArray(), dyn2DintArray(), cartMesh_t::iMax, cartMesh_t::jMax, LEFT, cartMesh_t::nBC, nQuads, RIGHT, TOP, X, xMax, xMin, Y, yMax, and yMin.

Referenced by createMesh().

6.25.2.3 createElemInfo()

```
void createElemInfo (
            elem_t * aElem )
```

Compute the cell specific values: barycenter, area, projection length of an element.

Parameters

in	*aElem*	A pointer to an element

References elem_t::area, elem_t::areaq, elem_t::bary, elem_t::elemType, elem_t::node, elem_t::sx, elem_t::sy, totalArea_q, X, node_t::x, and Y.

Referenced by createMesh().

6.25.2.4 createMesh()

```
void createMesh (
            void )
```

Create a cartesian or structured mesh.

Read in of all supported mesh types:

- ∗.msh
- ∗.msh2
- ∗.msh4
- ∗.emc2
- ∗.cgns

References elem_t::bary, sideList_t::BC, side_t::BC, boundary_t::BCid, BCside, boundary_t::BCtype, CARTESIAN, compare(), side_t::connection, connectPeriodicBC(), createCartMesh(), createElemInfo(), createReconstructionInfo(), createSideInfo(), elem_t::domain, elem, side_t::elem, elem_t::elemType, firstBC, firstBCside, firstElem, firstNode, firstSide, elem_t::firstSide, side_t::GP, node_t::id, side_t::id, elem_t::id, sideList_t::isRotated, side_t::n, nBCsides, nElems, boundary_t::next, node_t::next, side_t::next, sidePtr_t::next, elem_t::next, side_t::nextElemSide, nInnerSides, nNodes, sideList_t::node, side_t::node, elem_t::node, nQuads, nSides, nTrias, PERIODIC, readCGNS(), readEMC2(), readGmsh(), side, sideList_t::side, sidePtr_t::side, strMeshFile, strMeshFormat, totalArea_q, UNSTRUCTURED, X, node_t::x, xMax, xMin, Y, yMax, and yMin.

Referenced by initMesh().

6.25.2.5 createReconstructionInfo()

```
void createReconstructionInfo (
            elem_t * aElem )
```

Compute required vectors for reconstruction.

Parameters

in	*aElem*	A pointer to an element

References elem_t::area, elem_t::bary, side_t::baryBaryDist, side_t::baryBaryVec, side_t::connection, dyn2DdblArray(), elem_t::elemType, elem_t::firstSide, side_t::GP, NDIM, side_t::nextElemSide, elem_t::nGP, elem_t::node, side_t::w, elem_t::wGP, X, node_t::x, elem_t::xGP, and Y.

Referenced by createMesh().

6.25.2.6 createSideInfo()

```
void createSideInfo (
            side_t * aSide )
```

Create side info: normal vector, side length, and ghost cells.

Parameters

in	*aSide*	A pointer to a side

References elem_t::bary, side_t::BC, side_t::connection, side_t::elem, side_t::GP, side_t::len, side_t::n, NDIM, side_t::node, X, node_t::x, and Y.

Referenced by createMesh().

6.25.2.7 readCGNS()

```
void readCGNS (
            char fileName[STRLEN],
            double *** vertex,
            long * nVertices,
            long *** BCedge,
            long * nBCedges,
            long *** tria,
            long *** quad )
```

Read in a CGNS mesh.

Parameters

in	*fileName*	Name of the mesh file
in,out	*vertex*	Pointer to 2D array, used for the vertices
in,out	*nVertices*	Pointer to the number of total vertices in `vertex`
in,out	*BCedge*	Pointer to 2D array, used for the BC edges
in,out	*nBCedges*	Pointer to the number of total BC edges
in,out	*tria*	Pointer to a 2D array, used for the triangles
in,out	*quad*	Pointer to a 2D array, used for the quadrangles

References dyn2DdblArray(), dyn2DintArray(), nElems, nQuads, nTrias, X, and Y.

Referenced by createMesh().

6.25.2.8 readEMC2()

```
void readEMC2 (
            char fileName[STRLEN],
            double *** vertex,
            long * nVertices,
            long *** BCedge,
            long * nBCedges,
            long *** tria,
            long *** quad )
```

Read in EMC2 mesh file.

Parameters

in	*fileName*	Name of the mesh file
in,out	*vertex*	Pointer to 2D array, used for the vertices
in,out	*nVertices*	Pointer to the number of total vertices in `vertex`
in,out	*BCedge*	Pointer to 2D array, used for the BC edges
in,out	*nBCedges*	Pointer to the number of total BC edges
in,out	*tria*	Pointer to a 2D array, used for the triangles
in,out	*quad*	Pointer to a 2D array, used for the quadrangles

References dyn2DdblArray(), dyn2DintArray(), nQuads, nTrias, STRLEN, X, and Y.

Referenced by createMesh().

6.25.2.9 readGmsh()

```
void readGmsh (
            char fileName[STRLEN],
            double *** vertex,
            long * nVertices,
            long *** BCedge,
            long * nBCedges,
            long *** tria,
            long *** quad )
```

Read in a gmsh mesh file.

Parameters

in	*fileName*	Name of the mesh file
in,out	*vertex*	Pointer to 2D array, used for the vertices
in,out	*nVertices*	Pointer to the number of total vertices in `vertex`
in,out	*BCedge*	Pointer to 2D array, used for the BC edges
in,out	*nBCedges*	Pointer to the number of total BC edges
in,out	*tria*	Pointer to a 2D array, used for the triangles
in,out	*quad*	Pointer to a 2D array, used for the quadrangles

References dyn2DdblArray(), dyn2DintArray(), nQuads, nTrias, STRLEN, X, and Y.

Referenced by createMesh().

6.25.3 Variable Documentation

6.25.3.1 BCside

```
side_t** BCside
```

global BC side pointer array

Referenced by createMesh(), freeBoundary(), setBCatBarys(), and setBCatSides().

6.25.3.2 cartMesh

```
cartMesh_t cartMesh
```

cartesian mesh structure

Referenced by createCartMesh(), and readMesh().

6.25.3.3 dxRef

```
double dxRef
```

reference x length

Referenced by exactFunc(), and initMesh().

6.25.3.4 elem

```
elem_t** elem
```

global element pointer array

Referenced by buildMatrix(), calcErrors(), calcSource(), calcTimeStep(), createMesh(), explicitTimeStepEuler(), explicitTimeStepRK(), freeMesh(), fvTimeDerivative(), globalResidual(), implicitTimeStep(), initFV(), LUSGS(), matrixVector(), and spatialReconstruction().

6.25.3.5 firstBCside

```
sidePtr_t* firstBCside
```

pointer to first BC side

Referenced by cgnsWriteMesh(), connectPeriodicBC(), createMesh(), freeMesh(), and initWing().

6.25.3.6 firstElem

```
elem_t* firstElem
```

pointer to first element

Referenced by analyze(), cgnsOutput(), cgnsReadSolution(), cgnsWriteMesh(), createMesh(), csvOutput(), curve↩Output(), initRecordPoints(), and setInitialCondition().

6.25.3.7 firstNode

```
node_t* firstNode
```

pointer to first node

Referenced by cgnsWriteMesh(), createMesh(), and freeMesh().

6.25.3.8 firstSide

```
side_t* firstSide
```

pointer to first side

Referenced by connectPeriodicBC(), and createMesh().

6.25.3.9 gridFile

```
char gridFile[STRLEN]
```

complete name of the output mesh file

Referenced by cgnsFinalizeOutput(), cgnsOutput(), cgnsWriteMesh(), and initMesh().

6.25.3.10 meshFormat

```
int meshFormat
```

code for the mesh format

6.25.3.11 meshType

```
int meshType
```

code for the mesh type

6.25.3.12 nBCsides

```
long nBCsides
```

global number of BC sides

Referenced by cgnsWriteMesh(), createMesh(), setBCatBarys(), and setBCatSides().

6.25.3.13 nElems

```
long nElems
```

global number of elements

Referenced by buildMatrix(), calcErrors(), calcSource(), calcTimeStep(), cgnsFinalizeOutput(), cgnsOutput(), cgnsReadSolution(), cgnsWriteMesh(), createMesh(), csvOutput(), curveOutput(), explicitTimeStepEuler(), explicitTimeStepRK(), freeMesh(), fvTimeDerivative(), globalResidual(), GMRES_M(), implicitTimeStep(), init↩ FV(), initLinearSolver(), initMesh(), initTimeDisc(), LUSGS(), matrixVector(), readCGNS(), spatialReconstruction(), and vectorDotProduct().

6.25.3.14 nInnerSides

```
long nInnerSides
```

global number of non BC sides

Referenced by createMesh().

6.25.3.15 nNodes

```
long nNodes
```

global number of nodes

Referenced by cgnsFinalizeOutput(), cgnsOutput(), cgnsWriteMesh(), and createMesh().

6.25.3.16 nQuads

```
long nQuads
```

global number of quadrangles

Referenced by cgnsFinalizeOutput(), cgnsOutput(), cgnsWriteMesh(), createCartMesh(), createMesh(), read←CGNS(), readEMC2(), and readGmsh().

6.25.3.17 nSides

```
long nSides
```

global number of sides

Referenced by connectPeriodicBC(), createMesh(), and fluxCalculation().

6.25.3.18 nTrias

```
long nTrias
```

global number of triangles

Referenced by cgnsFinalizeOutput(), cgnsOutput(), cgnsWriteMesh(), createMesh(), readCGNS(), readEMC2(), and readGmsh().

6.25.3.19 side

```
side_t** side
```

global side pointer array

Referenced by createMesh(), fluxCalculation(), and freeMesh().

6.25.3.20 strIniCondFile

```
char strIniCondFile[STRLEN]
```

file name of the initial conditions file

Referenced by cgnsReadSolution(), and main().

6.25.3.21 strMeshFile

```
char strMeshFile[STRLEN]
```

mesh file base name

Referenced by createMesh(), and readMesh().

6.25.3.22 strMeshFormat

```
char strMeshFormat[STRLEN]
```

mesh format string

Referenced by createMesh(), and readMesh().

6.25.3.23 totalArea_q

```
double totalArea_q
```

inverse of the global area of the mesh

Referenced by calcErrors(), createElemInfo(), createMesh(), globalResidual(), and initMesh().

6.25.3.24 xMax

```
double xMax
```

minimum x-direction extension

Referenced by createCartMesh(), createMesh(), exactFunc(), and readMesh().

6.25.3.25 xMin

```
double xMin
```

maximum x-direction extension

Referenced by createCartMesh(), createMesh(), exactFunc(), and readMesh().

6.25.3.26 yMax

```
double yMax
```

minimum y-direction extension

Referenced by createCartMesh(), createMesh(), exactFunc(), and readMesh().

6.25.3.27 yMin

```
double yMin
```

maximum y-direction extension

Referenced by createCartMesh(), createMesh(), exactFunc(), and readMesh().

6.26 ccfd/src/mesh.h File Reference

Contains the definitions of all structs for the mesh handling.

```
#include "main.h"
#include "boundary.h"
```

Include dependency graph for mesh.h: This graph shows which files directly or indirectly include this file:

Data Structures

- struct node_t

    Structure for a single node in a linked list of nodes.
- struct side_t

    Structure for a single side in the global side list.
- struct sidePtr_t

    Secondary side lists used for various things.
- struct elem_t

    Structure for a single element in the global element list.
- struct cartMesh_t

    Structure holding the information for a cartesian mesh.

Functions

- void initMesh (void)

    Initialize the mesh and call `readMesh`
- void freeMesh (void)

    Free all allocated memory of the mesh.

Variables

- char strMeshFormat [STRLEN]
- char strMeshFile [STRLEN]
- char strIniCondFile [STRLEN]
- cartMesh_t cartMesh
- char gridFile [STRLEN]
- int meshType
- int meshFormat
- long nNodes
- long nElems
- long nTrias
- long nQuads
- long nSides
- long nBCsides
- long nInnerSides
- double totalArea_q
- double xMin
- double xMax
- double yMin
- double yMax
- double dxRef
- elem_t ** elem
- side_t ** side
- side_t ** BCside
- elem_t * firstElem
- node_t * firstNode
- side_t * firstSide
- sidePtr_t * firstBCside

6.26.1 Detailed Description

Contains the definitions of all structs for the mesh handling.

Author

hhh

Date

Tue 24 Mar 2020 12:45:49 PM CET

6.26.2 Variable Documentation

6.26.2.1 BCside

```
side_t** BCside [extern]
```

global BC side pointer array

Referenced by createMesh(), freeBoundary(), setBCatBarys(), and setBCatSides().

6.26.2.2 cartMesh

```
cartMesh_t cartMesh [extern]
```

cartesian mesh structure

Referenced by createCartMesh(), and readMesh().

6.26.2.3 dxRef

```
double dxRef [extern]
```

reference x length

Referenced by exactFunc(), and initMesh().

6.26.2.4 elem

```
elem_t** elem [extern]
```

global element pointer array

Referenced by buildMatrix(), calcErrors(), calcSource(), calcTimeStep(), createMesh(), explicitTimeStepEuler(), explicitTimeStepRK(), freeMesh(), fvTimeDerivative(), globalResidual(), implicitTimeStep(), initFV(), LUSGS(), matrixVector(), and spatialReconstruction().

6.26.2.5 firstBCside

```
sidePtr_t* firstBCside [extern]
```

pointer to first BC side

Referenced by cgnsWriteMesh(), connectPeriodicBC(), createMesh(), freeMesh(), and initWing().

6.26.2.6 firstElem

```
elem_t* firstElem [extern]
```

pointer to first element

Referenced by analyze(), cgnsOutput(), cgnsReadSolution(), cgnsWriteMesh(), createMesh(), csvOutput(), curve↩ Output(), initRecordPoints(), and setInitialCondition().

6.26.2.7 firstNode

```
node_t* firstNode [extern]
```

pointer to first node

Referenced by cgnsWriteMesh(), createMesh(), and freeMesh().

6.26.2.8 firstSide

```
side_t* firstSide [extern]
```

pointer to first side

Referenced by connectPeriodicBC(), and createMesh().

6.26.2.9 gridFile

```
char gridFile[STRLEN] [extern]
```

complete name of the output mesh file

Referenced by cgnsFinalizeOutput(), cgnsOutput(), cgnsWriteMesh(), and initMesh().

6.26.2.10 meshFormat

```
int meshFormat [extern]
```

code for the mesh format

6.26.2.11 meshType

```
int meshType [extern]
```

code for the mesh type

6.26.2.12 nBCsides

```
long nBCsides [extern]
```

global number of BC sides

Referenced by cgnsWriteMesh(), createMesh(), setBCatBarys(), and setBCatSides().

6.26.2.13 nElems

```
long nElems  [extern]
```

global number of elements

Referenced by buildMatrix(), calcErrors(), calcSource(), calcTimeStep(), cgnsFinalizeOutput(), cgnsOutput(), cgnsReadSolution(), cgnsWriteMesh(), createMesh(), csvOutput(), curveOutput(), explicitTimeStepEuler(), explicitTimeStepRK(), freeMesh(), fvTimeDerivative(), globalResidual(), GMRES_M(), implicitTimeStep(), initFV(), initLinearSolver(), initMesh(), initTimeDisc(), LUSGS(), matrixVector(), readCGNS(), spatialReconstruction(), and vectorDotProduct().

6.26.2.14 nInnerSides

```
long nInnerSides  [extern]
```

global number of non BC sides

Referenced by createMesh().

6.26.2.15 nNodes

```
long nNodes  [extern]
```

global number of nodes

Referenced by cgnsFinalizeOutput(), cgnsOutput(), cgnsWriteMesh(), and createMesh().

6.26.2.16 nQuads

```
long nQuads  [extern]
```

global number of quadrangles

Referenced by cgnsFinalizeOutput(), cgnsOutput(), cgnsWriteMesh(), createCartMesh(), createMesh(), readCGNS(), readEMC2(), and readGmsh().

6.26.2.17 nSides

```
long nSides  [extern]
```

global number of sides

Referenced by connectPeriodicBC(), createMesh(), and fluxCalculation().

6.26.2.18 nTrias

```
long nTrias [extern]
```

global number of triangles

Referenced by cgnsFinalizeOutput(), cgnsOutput(), cgnsWriteMesh(), createMesh(), readCGNS(), readEMC2(), and readGmsh().

6.26.2.19 side

```
side_t** side [extern]
```

global side pointer array

Referenced by createMesh(), fluxCalculation(), and freeMesh().

6.26.2.20 strIniCondFile

```
char strIniCondFile[STRLEN] [extern]
```

file name of the initial conditions file

Referenced by cgnsReadSolution(), and main().

6.26.2.21 strMeshFile

```
char strMeshFile[STRLEN] [extern]
```

mesh file base name

Referenced by createMesh(), and readMesh().

6.26.2.22 strMeshFormat

```
char strMeshFormat[STRLEN] [extern]
```

mesh format string

Referenced by createMesh(), and readMesh().

6.26.2.23 totalArea_q

```
double totalArea_q [extern]
```

inverse of the global area of the mesh

Referenced by calcErrors(), createElemInfo(), createMesh(), globalResidual(), and initMesh().

6.26.2.24 xMax

```
double xMax [extern]
```

minimum x-direction extension

Referenced by createCartMesh(), createMesh(), exactFunc(), and readMesh().

6.26.2.25 xMin

```
double xMin [extern]
```

maximum x-direction extension

Referenced by createCartMesh(), createMesh(), exactFunc(), and readMesh().

6.26.2.26 yMax

```
double yMax [extern]
```

minimum y-direction extension

Referenced by createCartMesh(), createMesh(), exactFunc(), and readMesh().

6.26.2.27 yMin

```
double yMin [extern]
```

maximum y-direction extension

Referenced by createCartMesh(), createMesh(), exactFunc(), and readMesh().

6.27 ccfd/src/output.c File Reference

Contains all functions used for writing flow solutions.

```
#include <stdio.h>
#include <string.h>
#include <omp.h>
#include <math.h>
#include "main.h"
#include "output.h"
#include "readInTools.h"
#include "timeDiscretization.h"
#include "analyze.h"
#include "equation.h"
#include "exactFunction.h"
#include "cgnslib.h"
#include "memTools.h"
```

Include dependency graph for output.c:

Functions

- void initOutput (void)
  Initialize output.
- void csvOutput (char fileName[STRLEN], double time, bool doExact)
  Tabular CSV output, only for 1D data.
- void cgnsOutput (char fileName[STRLEN], double time, bool doExact)
  Write solution to CGNS file.
- void curveOutput (char fileName[STRLEN], double time, bool doExact)
  Curve data output, only for 1D data.
- void dataOutput (double time, long iter)
  Perform a data output, dependent on the output format.
- void cgnsFinalizeOutput (void)
  Finalize CGNS output.
- void finalizeDataOutput (void)
  Finalize the data output, if necessary.
- void cgnsWriteMesh (void)
  Write the generated mesh to a CGNS mesh file.
- void freeOutputTimes (void)
  Free all memory that was allocated for the output times.

Variables

- char strOutFile [STRLEN]
- double IOtimeInterval
- int IOiterInterval
- int iVisuProg
- FILE ∗ resFile
- bool doErrorOutput
- outputTime_t ∗ outputTimes

6.27.1 Detailed Description

Contains all functions used for writing flow solutions.

Author

hhh

Date

Mon 23 Mar 2020 10:42:06 PM CET

6.27.2 Function Documentation

6.27.2.1 cgnsFinalizeOutput()

```
void cgnsFinalizeOutput (
            void  )
```

Finalize CGNS output.

This function creates a `<case>_Master.cgns` file that has links to all the output times of previous flow solutions. This makes it easier to load an entire case into ParaView.

References gridFile, isStationary, outputTime_t::iter, nElems, outputTime_t::next, nNodes, nQuads, nTrias, outputTimes, STRLEN, strOutFile, and outputTime_t::time.

Referenced by finalizeDataOutput().

6.27.2.2 cgnsOutput()

```
void cgnsOutput (
            char fileName[STRLEN],
            double time,
            bool doExact )
```

Write solution to CGNS file.

Parameters

in	*fileName*	The name of the output file
in	*time*	The computational time of the output result
in	*doExact*	If the exact exact solution should be written, instead of the computed flow results

References elem_t::bary, exactFunc(), firstElem, gridFile, elem_t::id, intExactFunc, nElems, elem_t::next, nNodes, nQuads, nTrias, NVAR, P, elem_t::pVar, RHO, STRLEN, timeOverall, VX, and VY.

Referenced by dataOutput().

6.27.2.3 csvOutput()

```
void csvOutput (
            char fileName[STRLEN],
            double time,
            bool doExact )
```

Tabular CSV output, only for 1D data.

Parameters

in	*fileName*	The name of the output file
in	*time*	The computational time of the output result
in	*doExact*	If the exact exact solution should be written, instead of the computed flow results

References elem_t::bary, dyn2DdblArray(), exactFunc(), firstElem, intExactFunc, nElems, elem_t::next, NVAR, P, elem_t::pVar, RHO, VX, and X.

Referenced by dataOutput().

6.27.2.4 curveOutput()

```
void curveOutput (
            char fileName[STRLEN],
            double time,
            bool doExact )
```

Curve data output, only for 1D data.

Parameters

in	*fileName*	The name of the output file
in	*time*	The computational time of the output result
in	*doExact*	If the exact exact solution should be written, instead of the computed flow results

References elem_t::bary, dyn2DdblArray(), exactFunc(), firstElem, intExactFunc, nElems, elem_t::next, NVAR, P, elem_t::pVar, RHO, VX, and X.

Referenced by dataOutput().

6.27.2.5 dataOutput()

```
void dataOutput (
            double time,
            long iter )
```

Perform a data output, dependent on the output format.

Parameters

in	*time*	The computational time of the output result
in	*iter*	The iteration number of the output result

References CGNS, cgnsOutput(), CSV, csvOutput(), CURVE, curveOutput(), hasExactSolution, isStationary, outputTime_t::iter, iVisuProg, outputTime_t::next, outputTimes, STRLEN, strOutFile, and outputTime_t::time.

Referenced by timeDisc().

6.27.3 Variable Documentation

6.27.3.1 doErrorOutput

```
bool doErrorOutput
```

error output flag

6.27.3.2 IOiterInterval

```
int IOiterInterval
```

iteration interval for data output

Referenced by initOutput(), initTimeDisc(), and timeDisc().

6.27.3.3 IOtimeInterval

```
double IOtimeInterval
```

time interval for data output

Referenced by initOutput(), initTimeDisc(), and timeDisc().

6.27.3.4 iVisuProg

```
int iVisuProg
```

output format code

Referenced by dataOutput(), finalizeDataOutput(), initMesh(), and initOutput().

6.27.3.5 outputTimes

```
outputTime_t* outputTimes
```

the first output time object

Referenced by cgnsFinalizeOutput(), dataOutput(), freeOutputTimes(), and main().

6.27.3.6 resFile

```
FILE* resFile
```

residual file pointer

Referenced by analyze(), initAnalyze(), and timeDisc().

6.27.3.7 strOutFile

```
char strOutFile[STRLEN]
```

name of the output file

Referenced by calcCoef(), cgnsFinalizeOutput(), dataOutput(), initAnalyze(), initMesh(), initOutput(), and init↩ RecordPoints().

6.28 ccfd/src/output.h File Reference

Contains `outputTime_t` struct definition.

```
#include <stdbool.h>
#include <stdlib.h>
```

Include dependency graph for output.h: This graph shows which files directly or indirectly include this file:

Data Structures

- struct outputTime_t

    Output times linked list.

Functions

- void initOutput (void)

    Initialize output.
- void dataOutput (double time, long iter)

    Perform a data output, dependent on the output format.
- void finalizeDataOutput (void)

    Finalize the data output, if necessary.
- void cgnsWriteMesh (void)

    Write the generated mesh to a CGNS mesh file.
- void freeOutputTimes (void)

    Free all memory that was allocated for the output times.

Variables

- char strOutFile [STRLEN]
- double IOtimeInterval
- int IOiterInterval
- int iVisuProg
- FILE * resFile
- bool doErrorOutput
- outputTime_t * outputTimes

6.28.1 Detailed Description

Contains `outputTime_t` struct definition.

Author

hhh

Date

Mon 23 Mar 2020 10:37:31 PM CET

6.28.2 Function Documentation

6.28.2.1 dataOutput()

```
void dataOutput (
            double time,
            long iter )
```

Perform a data output, dependent on the output format.

Parameters

in	*time*	The computational time of the output result
in	*iter*	The iteration number of the output result

References CGNS, cgnsOutput(), CSV, csvOutput(), CURVE, curveOutput(), hasExactSolution, isStationary, outputTime_t::iter, iVisuProg, outputTime_t::next, outputTimes, STRLEN, strOutFile, and outputTime_t::time.

Referenced by timeDisc().

6.28.3 Variable Documentation

6.28.3.1 doErrorOutput

```
bool doErrorOutput  [extern]
```

error output flag

6.28.3.2 IOiterInterval

```
int IOiterInterval  [extern]
```

iteration interval for data output

Referenced by initOutput(), initTimeDisc(), and timeDisc().

6.28.3.3 IOtimeInterval

```
double IOtimeInterval  [extern]
```

time interval for data output

Referenced by initOutput(), initTimeDisc(), and timeDisc().

6.28.3.4 iVisuProg

```
int iVisuProg  [extern]
```

output format code

Referenced by dataOutput(), finalizeDataOutput(), initMesh(), and initOutput().

6.28.3.5 outputTimes

```
outputTime_t* outputTimes [extern]
```

the first output time object

Referenced by cgnsFinalizeOutput(), dataOutput(), freeOutputTimes(), and main().

6.28.3.6 resFile

```
FILE* resFile [extern]
```

residual file pointer

Referenced by analyze(), initAnalyze(), and timeDisc().

6.28.3.7 strOutFile

```
char strOutFile[STRLEN] [extern]
```

name of the output file

Referenced by calcCoef(), cgnsFinalizeOutput(), dataOutput(), initAnalyze(), initMesh(), initOutput(), and init↩ RecordPoints().

6.29 ccfd/src/readInTools.c File Reference

Provides functions for reading data from the `.ini` parameter file.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <stdbool.h>
#include "main.h"
```

Include dependency graph for readInTools.c:

Data Structures

- struct cmd_t

    A structure used to store the commands, read in from the parameter file.

Functions

- void fillCmds (char iniFileName[STRLEN])
  Read parameter file and create commands list.
- void deleteCmd (cmd_t *aCmd)
  Delete a single node of the command list.
- char * findCmd (const char *key, char defMsg[8], const char *proposal)
  Find a command in the commands list.
- char * getStr (const char *key, const char *proposal)
  Get a string from the commands list.
- int countKeys (const char *key, const int proposal)
  Count how often a key appears.
- long getInt (const char *key, const char *proposal)
  Get an integer from the commands list.
- double getDbl (const char *key, const char *proposal)
  Get a double from the commands list.
- bool getBool (const char *key, const char *proposal)
  Get a boolean from the commands list.
- int * getIntArray (const char *key, const int N, const char *proposal)
  Get an integer array from the commands list.
- double * getDblArray (const char *key, const int N, const char *proposal)
  Get a double array from the commands list.
- void freeCmds (void)
  Delete all commands in the commands list.
- void ignoredCmds (void)
  Print out all remaining commands in the list.

Variables

- cmd_t * firstCmd

6.29.1 Detailed Description

Provides functions for reading data from the `.ini` parameter file.

Author

hhh

Date

Sat 21 Mar 2020 10:46:32 AM CET

6.29.2 Function Documentation

6.29.2.1 countKeys()

```
int countKeys (
            const char * key,
            const int proposal )
```

Count how often a key appears.

Count all occurrences of key in parameter file and return them. If the key is not specified, the proposal will be returned. If the proposal is -1, but the key is not in the list, an error will be thrown.

Parameters

in	*key*	Key string of the command to be found
in	*proposal*	The default value that is used if the key was not

Returns

How often the `key` appeared in the command list

References firstCmd, and cmd_t::next.

Referenced by initBoundary().

6.29.2.2 deleteCmd()

```
void deleteCmd (
            cmd_t * aCmd )
```

Delete a single node of the command list.

Before deleting the command, the previous command is connected to the next command and vice versa.

Parameters

in	*aCmd*	A pointer to the command that is to be deleted

References firstCmd, cmd_t::next, and cmd_t::prev.

Referenced by findCmd().

6.29.2.3 fillCmds()

```
void fillCmds (
            char iniFileName[STRLEN] )
```

Read parameter file and create commands list.

Read `.ini` file and parse each line into a `cmd_t` object. All `cmd_t` objects are connected in a list of commands starting with `firstCmd`.

Parameters

in	*iniFileName*	The name of the parameter file

References firstCmd, cmd_t::key, cmd_t::next, cmd_t::prev, STRLEN, and cmd_t::value.

Referenced by main().

6.29.2.4 findCmd()

```
char* findCmd (
            const char * key,
            char defMsg[8],
            const char * proposal )
```

Find a command in the commands list.

Find the provided key in the list of commands, and return the address of the corresponding value string. Return NULL if key was not found. Once a key was read from the commands list, it is deleted from the list.

Parameters

in	*key*	Key string of the command to be found
out	*defMsg*	String, that indicates if an actual value or the proposal was returned
in	*proposal*	The default value that is used if the key was not specified

Returns

Pointer to the `value` string, or `NULL`

References deleteCmd(), firstCmd, cmd_t::key, cmd_t::next, and cmd_t::value.

Referenced by getBool(), getDbl(), getDblArray(), getInt(), getIntArray(), and getStr().

6.29.2.5 getBool()

```
bool getBool (
            const char * key,
            const char * proposal )
```

Get a boolean from the commands list.

Find the key in the command list and return the corresponding value. If the key is not specified, the proposal will be returned. If the proposal is NULL, but the key is not in the list, an error will be thrown. The value in the parameter file is accepted as true, if it is a 'T', 't', 'True', or 'true', otherwise it is false.

Parameters

in	*key*	Key string of the command to be found
in	*proposal*	The default value that is used if the key was not

Returns

The value of the parameter, or the default value

References findCmd().

Referenced by initAnalyze(), initBoundary(), initEquation(), initLinearSolver(), initTimeDisc(), and main().

6.29.2.6 getDbl()

```
double getDbl (
            const char * key,
            const char * proposal )
```

Get a double from the commands list.

Find the key in the command list and return the corresponding value. If the key is not specified, the proposal will be returned. If the proposal is NULL, but the key is not in the list, an error will be thrown.

Parameters

in	*key*	Key string of the command to be found
in	*proposal*	The default value that is used if the key was not

Returns

The value of the parameter, or the default value

References findCmd().

Referenced by initBoundary(), initEquation(), initFV(), initInitialCondition(), initLinearSolver(), initOutput(), initTime↩ Disc(), and readWing().

6.29.2.7 getDblArray()

```
double* getDblArray (
            const char * key,
            const int N,
            const char * proposal )
```

Get a double array from the commands list.

Find the key in the command list and return the corresponding integer array. If the key is not specified, the proposal will be returned. If the proposal is NULL, but the key is not in the list, an error will be thrown.

Parameters

in	*key*	Key string of the command to be found
in	*N*	Length of the array that is to be read in
in	*proposal*	The default value that is used if the key was not

Returns

A pointer to the value array of the parameter, or the default value array

References findCmd(), and STRLEN.

Referenced by initBoundary(), initInitialCondition(), initRecordPoints(), and readMesh().

6.29.2.8 getInt()

```
long getInt (
            const char * key,
            const char * proposal )
```

Get an integer from the commands list.

Find the key in the command list and return the corresponding value. If the key is not specified, the proposal will be returned. If the proposal is NULL, but the key is not in the list, an error will be thrown.

Parameters

in	*key*	Key string of the command to be found
in	*proposal*	The default value that is used if the key was not

Returns

The value of the parameter, or the default value

References findCmd().

Referenced by initAnalyze(), initBoundary(), initEquation(), initFV(), initInitialCondition(), initLinearSolver(), init←
Output(), initTimeDisc(), readMesh(), and readWing().

6.29.2.9 getIntArray()

```
int* getIntArray (
            const char * key,
            const int N,
            const char * proposal )
```

Get an integer array from the commands list.

Find the key in the command list and return the corresponding integer array. If the key is not specified, the proposal will be returned. If the proposal is NULL, but the key is not in the list, an error will be thrown.

Parameters

in	*key*	Key string of the command to be found
in	*N*	Length of the array that is to be read in
in	*proposal*	The default value that is used if the key was not

Returns

A pointer to the value array of the parameter, or the default value array

References findCmd(), and STRLEN.

Referenced by readMesh().

6.29.2.10 getStr()

```
char* getStr (
            const char * key,
            const char * proposal )
```

Get a string from the commands list.

Find the key in the command list and return the corresponding value. If the key is not specified, the proposal will be returned. If the proposal is NULL, but the key is not in the list, an error will be thrown.

Parameters

in	*key*	Key string of the command to be found
in	*proposal*	The default value that is used if the key was not

Returns

Pointer to the `value` string, containing the parameter, or the default value, if the parameter was not specified

References findCmd().

Referenced by initOutput(), and readMesh().

6.29.3 Variable Documentation

6.29.3.1 firstCmd

```
cmd_t* firstCmd
```

first command of the list

Referenced by countKeys(), deleteCmd(), fillCmds(), findCmd(), freeCmds(), and ignoredCmds().

6.30 ccfd/src/readInTools.h File Reference

```
#include <stdbool.h>
#include "main.h"
```

Include dependency graph for readInTools.h: This graph shows which files directly or indirectly include this file:

Functions

- void fillCmds (char iniFileName[STRLEN])

  Read parameter file and create commands list.
- char ∗ getStr (const char ∗key, const char ∗proposal)

  Get a string from the commands list.
- int countKeys (const char ∗key, const int proposal)

  Count how often a key appears.
- long getInt (const char ∗key, const char ∗proposal)

  Get an integer from the commands list.
- double getDbl (const char ∗key, const char ∗proposal)

  Get a double from the commands list.
- bool getBool (const char ∗key, const char ∗proposal)

  Get a boolean from the commands list.
- int ∗ getIntArray (const char ∗key, const int N, const char ∗proposal)

  Get an integer array from the commands list.
- double ∗ getDblArray (const char ∗key, const int N, const char ∗proposal)

  Get a double array from the commands list.
- void ignoredCmds (void)

  Print out all remaining commands in the list.

6.30.1 Detailed Description

Author

hhh

Date

Sat 21 Mar 2020 10:51:13 AM CET

6.30.2 Function Documentation

6.30.2.1 countKeys()

```
int countKeys (
            const char * key,
            const int proposal )
```

Count how often a key appears.

Count all occurrences of key in parameter file and return them. If the key is not specified, the proposal will be returned. If the proposal is -1, but the key is not in the list, an error will be thrown.

Parameters

in	*key*	Key string of the command to be found
in	*proposal*	The default value that is used if the key was not

Returns

How often the `key` appeared in the command list

References firstCmd, and cmd_t::next.

Referenced by initBoundary().

6.30.2.2 fillCmds()

```
void fillCmds (
            char iniFileName[STRLEN] )
```

Read parameter file and create commands list.

Read `.ini` file and parse each line into a `cmd_t` object. All `cmd_t` objects are connected in a list of commands starting with `firstCmd`.

Parameters

in	*iniFileName*	The name of the parameter file

References firstCmd, cmd_t::key, cmd_t::next, cmd_t::prev, STRLEN, and cmd_t::value.

Referenced by main().

6.30.2.3 getBool()

```
bool getBool (
            const char * key,
            const char * proposal )
```

Get a boolean from the commands list.

Find the key in the command list and return the corresponding value. If the key is not specified, the proposal will be returned. If the proposal is NULL, but the key is not in the list, an error will be thrown. The value in the parameter file is accepted as true, if it is a 'T', 't', 'True', or 'true', otherwise it is false.

Parameters

in	*key*	Key string of the command to be found
in	*proposal*	The default value that is used if the key was not

Returns

The value of the parameter, or the default value

References findCmd().

Referenced by initAnalyze(), initBoundary(), initEquation(), initLinearSolver(), initTimeDisc(), and main().

6.30.2.4 getDbl()

```
double getDbl (
            const char * key,
            const char * proposal )
```

Get a double from the commands list.

Find the key in the command list and return the corresponding value. If the key is not specified, the proposal will be returned. If the proposal is NULL, but the key is not in the list, an error will be thrown.

Parameters

in	*key*	Key string of the command to be found
in	*proposal*	The default value that is used if the key was not

Returns

The value of the parameter, or the default value

References findCmd().

Referenced by initBoundary(), initEquation(), initFV(), initInitialCondition(), initLinearSolver(), initOutput(), initTime←Disc(), and readWing().

6.30.2.5 getDblArray()

```
double* getDblArray (
            const char * key,
            const int N,
            const char * proposal )
```

Get a double array from the commands list.

Find the key in the command list and return the corresponding integer array. If the key is not specified, the proposal will be returned. If the proposal is NULL, but the key is not in the list, an error will be thrown.

Parameters

in	*key*	Key string of the command to be found
in	*N*	Length of the array that is to be read in
in	*proposal*	The default value that is used if the key was not

Returns

A pointer to the value array of the parameter, or the default value array

References findCmd(), and STRLEN.

Referenced by initBoundary(), initInitialCondition(), initRecordPoints(), and readMesh().

6.30.2.6 getInt()

```
long getInt (
            const char * key,
            const char * proposal )
```

Get an integer from the commands list.

Find the key in the command list and return the corresponding value. If the key is not specified, the proposal will be returned. If the proposal is NULL, but the key is not in the list, an error will be thrown.

Parameters

in	*key*	Key string of the command to be found
in	*proposal*	The default value that is used if the key was not

Returns

The value of the parameter, or the default value

References findCmd().

Referenced by initAnalyze(), initBoundary(), initEquation(), initFV(), initInitialCondition(), initLinearSolver(), init↩ Output(), initTimeDisc(), readMesh(), and readWing().

6.30.2.7 getIntArray()

```
int* getIntArray (
            const char * key,
            const int N,
            const char * proposal )
```

Get an integer array from the commands list.

Find the key in the command list and return the corresponding integer array. If the key is not specified, the proposal will be returned. If the proposal is NULL, but the key is not in the list, an error will be thrown.

Parameters

in	*key*	Key string of the command to be found
in	*N*	Length of the array that is to be read in
in	*proposal*	The default value that is used if the key was not

Returns

A pointer to the value array of the parameter, or the default value array

References findCmd(), and STRLEN.

Referenced by readMesh().

6.30.2.8 getStr()

```
char* getStr (
            const char * key,
            const char * proposal )
```

Get a string from the commands list.

Find the key in the command list and return the corresponding value. If the key is not specified, the proposal will be returned. If the proposal is NULL, but the key is not in the list, an error will be thrown.

Parameters

in	*key*	Key string of the command to be found
in	*proposal*	The default value that is used if the key was not

Returns

Pointer to the `value` string, containing the parameter, or the default value, if the parameter was not specified

References findCmd().

Referenced by initOutput(), and readMesh().

6.31 ccfd/src/reconstruction.c File Reference

Contains the reconstruction and limiter functions.

```
#include <math.h>
#include "main.h"
#include "reconstruction.h"
#include "finiteVolume.h"
#include "mesh.h"
```

Include dependency graph for reconstruction.c:

Functions

- void limiterBarthJespersen (elem_t *aElem)

  Limiter after Barth & Jespersen.
- void limiterVenkatakrishnan (elem_t *aElem)

  Limiter after Venkatakrishnan, with additional limiting parameter k.
- void spatialReconstruction (double time)

  Compute the gradients of dU/dx.

Variables

- int limiter
- double venk_k

6.31.1 Detailed Description

Contains the reconstruction and limiter functions.

Author

hhh

Date

Sat 28 Mar 2020 10:17:02 AM CET

6.31.2 Function Documentation

6.31.2.1 limiterBarthJespersen()

```
void limiterBarthJespersen (
            elem_t * aElem )
```

Limiter after Barth & Jespersen.

Note

2D, unstructured limiter

Parameters

in	*aElem*	Pointer to an element

References side_t::connection, side_t::elem, elem_t::firstSide, side_t::GP, side_t::nextElemSide, NVAR, P, elem_t::pVar, RHO, elem_t::u_x, elem_t::u_y, VX, VY, X, and Y.

Referenced by spatialReconstruction().

6.31.2.2 limiterVenkatakrishnan()

```
void limiterVenkatakrishnan (
            elem_t * aElem )
```

Limiter after Venkatakrishnan, with additional limiting parameter k.

Note

2D, unstructured limiter

Parameters

in	*aElem*	Pointer to an element

References side_t::connection, side_t::elem, elem_t::firstSide, side_t::GP, side_t::nextElemSide, NVAR, P, elem_↩ t::pVar, RHO, elem_t::u_x, elem_t::u_y, elem_t::venkEps_sq, VX, VY, X, and Y.

Referenced by spatialReconstruction().

6.31.2.3 spatialReconstruction()

```
void spatialReconstruction (
            double time )
```

Compute the gradients of dU/dx.

Parameters

in	*time*	Calculation time at which to perform the spatial reconstruction

References BARTHJESPERSEN, side_t::connection, elem, side_t::elem, elem_t::firstSide, side_t::GP, limiter, limiterBarthJespersen(), limiterVenkatakrishnan(), nElems, side_t::nextElemSide, NVAR, P, side_t::pVar, elem_↩ t::pVar, RHO, setBCatBarys(), spatialOrder, elem_t::u_t, elem_t::u_x, elem_t::u_y, VENKATAKRISHNAN, VX, VY, side_t::w, X, and Y.

Referenced by calcErrors(), and fvTimeDerivative().

6.31.3 Variable Documentation

6.31.3.1 limiter

```
int limiter
```

limiter selection

Referenced by initFV(), and spatialReconstruction().

6.31.3.2 venk_k

```
double venk_k
```

constant for Venkatakrishnan limiter

Referenced by initFV().

6.32 ccfd/src/reconstruction.h File Reference

This graph shows which files directly or indirectly include this file:

Functions

- void spatialReconstruction (double time)
    Compute the gradients of dU/dx.

Variables

- int limiter
- double venk_k

6.32.1 Detailed Description

Author

hhh

Date

Sat 28 Mar 2020 10:16:16 AM CET

6.32.2 Function Documentation

6.32.2.1 spatialReconstruction()

```
void spatialReconstruction (
            double time )
```

Compute the gradients of dU/dx.

Parameters

in	*time*	Calculation time at which to perform the spatial reconstruction

References BARTHJESPERSEN, side_t::connection, elem, side_t::elem, elem_t::firstSide, side_t::GP, limiter, limiterBarthJespersen(), limiterVenkatakrishnan(), nElems, side_t::nextElemSide, NVAR, P, side_t::pVar, elem_↩ t::pVar, RHO, setBCatBarys(), spatialOrder, elem_t::u_t, elem_t::u_x, elem_t::u_y, VENKATAKRISHNAN, VX, VY, side_t::w, X, and Y.

Referenced by calcErrors(), and fvTimeDerivative().

6.32.3 Variable Documentation

6.32.3.1 limiter

```
int limiter [extern]
```

limiter selection

Referenced by initFV(), and spatialReconstruction().

6.32.3.2 venk_k

```
double venk_k [extern]
```

constant for Venkatakrishnan limiter

Referenced by initFV().

6.33 ccfd/src/source.c File Reference

Contains the functions for initializing and evaluating the source term.

```
#include <math.h>
#include "main.h"
#include "equation.h"
#include "mesh.h"
```

Include dependency graph for source.c:

Functions

- void evalSource (int iSource, double x[NDIM], double time, double source[NVAR])

    Evaluate the source term.
- void calcSource (double time)

    Calculate the contribution of the source terms.

6.33.1 Detailed Description

Contains the functions for initializing and evaluating the source term.

Author

hhh

Date

Wed 01 Apr 2020 12:25:11 PM CEST

6.33.2 Function Documentation

6.33.2.1 calcSource()

```
void calcSource (
            double time )
```

Calculate the contribution of the source terms.

Parameters

in	*time*	The computation time at which the evaluate the source term

References E, elem, evalSource(), nElems, elem_t::nGP, NVAR, RHO, elem_t::source, sourceFunc, VX, VY, elem↩_t::wGP, and elem_t::xGP.

Referenced by fvTimeDerivative().

6.33.2.2 evalSource()

```
void evalSource (
            int iSource,
            double x[NDIM],
            double time,
            double source[NVAR] )
```

Evaluate the source term.

Parameters

in	*iSource*	The source control integer
in	*x*	The coordinates at which to evaluate the source term
in	*time*	The computation time at which the evaluate the source term
out	*source*	The source term contribution

References E, gam, gam1, mu, pi, Pr, RHO, VX, VY, X, and Y.

Referenced by calcSource().

6.34 ccfd/src/source.h File Reference

This graph shows which files directly or indirectly include this file:

Functions

- void calcSource (double time)
  Calculate the contribution of the source terms.

6.34.1 Detailed Description

Author

hhh

Date

Wed 01 Apr 2020 12:25:19 PM CEST

6.34.2 Function Documentation

6.34.2.1 calcSource()

```
void calcSource (
            double time )
```

Calculate the contribution of the source terms.

Parameters

in	*time*	The computation time at which the evaluate the source term

References E, elem, evalSource(), nElems, elem_t::nGP, NVAR, RHO, elem_t::source, sourceFunc, VX, VY, elem←_t::wGP, and elem_t::xGP.

Referenced by fvTimeDerivative().

6.35 ccfd/src/timeDiscretization.c File Reference

Contains the functions for performing the time stepping process.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <omp.h>
#include <float.h>
#include "main.h"
#include "timeDiscretization.h"
#include "readInTools.h"
#include "output.h"
#include "mesh.h"
#include "equation.h"
#include "analyze.h"
#include "linearSolver.h"
#include "equationOfState.h"
#include "finiteVolume.h"
#include "memTools.h"
```

Include dependency graph for timeDiscretization.c:

Functions

- void initTimeDisc (void)

  Initialize the time discretization.
- void calcTimeStep (double pTime, double ∗dt, bool ∗viscousTimeStepDominates)

  Compute the time step.
- void explicitTimeStepEuler (double time, double dt, double resIter[NVAR+2])

  Performs explicit time step using Euler scheme.
- void explicitTimeStepRK (double time, double dt, double resIter[NVAR+2])

  Performs explicit time step using Runge-Kutta scheme `nRKstages` stages.
- void implicitTimeStep (double time, double dt, double resIter[NVAR+2])

  Euler implicit time integration.
- void timeDisc (void)

  Main time discretization loop.

Variables

- double cfl
- double dfl
- double t
- double timeOverall
- int timeOrder
- bool isTimeStep1D
- bool isStationary
- long maxIter
- double stopTime
- long iniIterationNumber
- double startTime
- double abortResidual
- int abortVariable
- char abortVariableName [4]
- double clAbortResidual
- double cdAbortResidual
- bool doAbortOnClResidual

- bool doAbortOnCdResidual
- bool isRestart
- double restartTime
- int printIter
- double printTime
- int nRKstages
- double RKcoeff [6] = {0.0}
- bool isImplicit
- double ** deltaX
- double ** Q
- double ** F_X0
- double ** F_XK

6.35.1 Detailed Description

Contains the functions for performing the time stepping process.

Author

hhh

Date

Sat 21 Mar 2020 07:52:42 PM CET

6.35.2 Function Documentation

6.35.2.1 calcTimeStep()

```
void calcTimeStep (
            double pTime,
            double * dt,
            bool * viscousTimeStepDominates )
```

Compute the time step.

Parameters

in	*pTime*	The print time interval
out	*dt*	The resulting time step
out	*viscousTimeStepDominates*	Flag for if the viscous time step is dominating

References elem_t::area, cfl, dfl, elem_t::dt, elem, gam, isTimeStep1D, mu, nElems, P, Pr, elem_t::pVar, RHO, stopTime, elem_t::sx, elem_t::sy, t, VX, and VY.

Referenced by timeDisc().

6.35.2.2 explicitTimeStepEuler()

```
void explicitTimeStepEuler (
            double time,
            double dt,
            double resIter[NVAR+2] )
```

Performs explicit time step using Euler scheme.

Parameters

in	*time*	Computation time at calculation
in	*dt*	Time step at calculation
out	*resIter*	Residual vector for time step

References consPrim(), elem_t::cVar, E, elem, fvTimeDerivative(), globalResidual(), MX, MY, nElems, elem_t::pVar, RHO, and elem_t::u_t.

Referenced by timeDisc().

6.35.2.3 explicitTimeStepRK()

```
void explicitTimeStepRK (
            double time,
            double dt,
            double resIter[NVAR+2] )
```

Performs explicit time step using Runge-Kutta scheme `nRKstages` stages.

Parameters

in	*time*	Computation time at calculation
in	*dt*	Time step at calculation
out	*resIter*	Residual vector for time step

References consPrim(), elem_t::cVar, elem_t::cVarStage, E, elem, fvTimeDerivative(), globalResidual(), MX, MY, nElems, nRKstages, elem_t::pVar, RHO, RKcoeff, and elem_t::u_t.

Referenced by timeDisc().

6.35.2.4 implicitTimeStep()

```
void implicitTimeStep (
            double time,
            double dt,
            double resIter[NVAR+2] )
```

Euler implicit time integration.

The non-linear equations require the use of a Newton method with internal sub-iteration, using a GMRES method.

Parameters

in	*time*	Computation time at calculation
in	*dt*	Time step at calculation
out	*resIter*	Residual vector for time step

References alpha, consPrim(), elem_t::cVar, deltaX, E, elem, eps2newton, eps2newton_sq, F_X0, F_XK, fvTime↩ Derivative(), gamEW, globalResidual(), GMRES_M(), MX, MY, nElems, nInnerNewton, nNewtonIter, nNewtonIter↩ Global, elem_t::pVar, Q, R_XK, RHO, t, elem_t::u_t, vectorDotProduct(), and XK.

Referenced by timeDisc().

6.35.2.5 timeDisc()

```
void timeDisc (
            void  )
```

Main time discretization loop.

Selection of temporal integration method, as well as management of data output and analysis tools.

References abortResidual, abortVariable, abortVariableName, analyze(), calcErrors(), calcTimeStep(), wing_t::cd, cdAbortResidual, wing_t::cl, clAbortResidual, CPU_TIME, dataOutput(), deltaX, doAbortOnCdResidual, doAbort↩ OnClResidual, doCalcWing, E, explicitTimeStepEuler(), explicitTimeStepRK(), F_X0, F_XK, finalizeDataOutput(), hasExactSolution, implicitTimeStep(), iniIterationNumber, recordPoint_t::ioFile, IOiterInterval, IOtimeInterval, is↩ Implicit, isStationary, maxIter, MX, MY, nGMRESiterGlobal, nNewtonIterGlobal, recordPoint_t::nPoints, nRKstages, NVAR, printIter, printTime, Q, recordPoint, resFile, RHO, stopTime, t, timeOrder, and wing.

Referenced by main().

6.35.3 Variable Documentation

6.35.3.1 abortResidual

```
double abortResidual
```

residual at which to abort the calculation

Referenced by initTimeDisc(), and timeDisc().

6.35.3.2 abortVariable

```
int abortVariable
```

abort variable

Referenced by analyze(), initTimeDisc(), and timeDisc().

6.35.3.3 abortVariableName

```
char abortVariableName[4]
```

string of the abort variable

Referenced by initAnalyze(), initTimeDisc(), and timeDisc().

6.35.3.4 cdAbortResidual

```
double cdAbortResidual
```

CD abort residual

Referenced by initTimeDisc(), and timeDisc().

6.35.3.5 cfl

```
double cfl
```

Courant-Friedrichs-Lewy number

Referenced by calcTimeStep(), and initTimeDisc().

6.35.3.6 clAbortResidual

```
double clAbortResidual
```

CL abort residual

Referenced by initTimeDisc(), and timeDisc().

6.35.3.7 deltaX

```
double** deltaX
```

variable used in implicit calculation

Referenced by implicitTimeStep(), initTimeDisc(), and timeDisc().

6.35.3.8 dfl

```
double dfl
```

diffusive Courant-Friedrichs-Lewy number

Referenced by calcTimeStep(), and initTimeDisc().

6.35.3.9 doAbortOnCdResidual

```
bool doAbortOnCdResidual
```

CD abort flag

Referenced by initTimeDisc(), and timeDisc().

6.35.3.10 doAbortOnClResidual

```
bool doAbortOnClResidual
```

CL abort flag

Referenced by initTimeDisc(), and timeDisc().

6.35.3.11 F_X0

```
double** F_X0
```

variable used in implicit calculation

Referenced by implicitTimeStep(), initTimeDisc(), and timeDisc().

6.35.3.12 F_XK

```
double** F_XK
```

variable used in implicit calculation

Referenced by implicitTimeStep(), initTimeDisc(), and timeDisc().

6.35.3.13 iniIterationNumber

```
long iniIterationNumber
```

initial iteration number

Referenced by initTimeDisc(), main(), and timeDisc().

6.35.3.14 isImplicit

```
bool isImplicit
```

implicit calculation flag

Referenced by freeLinearSolver(), initLinearSolver(), initTimeDisc(), and timeDisc().

6.35.3.15 isRestart

```
bool isRestart
```

restart flag

Referenced by initAnalyze(), initMesh(), initTimeDisc(), main(), and setInitialCondition().

6.35.3.16 isStationary

```
bool isStationary
```

flag for stationary problem

Referenced by analyze(), cgnsFinalizeOutput(), dataOutput(), initAnalyze(), initTimeDisc(), main(), and timeDisc().

6.35.3.17 isTimeStep1D

```
bool isTimeStep1D
```

flag for 1D problem

Referenced by calcTimeStep(), and initTimeDisc().

6.35.3.18 maxIter

```
long maxIter
```

maximum number of iterations

Referenced by initTimeDisc(), and timeDisc().

6.35.3.19 nRKstages

```
int nRKstages
```

number of Runge-Kutta stages

Referenced by explicitTimeStepRK(), initTimeDisc(), and timeDisc().

6.35.3.20 printIter

```
int printIter
```

iterations after which to output

Referenced by initTimeDisc(), and timeDisc().

6.35.3.21 printTime

```
double printTime
```

calculation time after which to output

Referenced by initTimeDisc(), and timeDisc().

6.35.3.22 Q

```
double** Q
```

variable used in implicit calculation

Referenced by implicitTimeStep(), initTimeDisc(), and timeDisc().

6.35.3.23 restartTime

```
double restartTime
```

calculation time for restart

Referenced by initTimeDisc(), and main().

6.35.3.24 RKcoeff

```
double RKcoeff[6] = {0.0}
```

array of Runge-Kutta coefficients

Referenced by explicitTimeStepRK(), and initTimeDisc().

6.35.3.25 startTime

```
double startTime
```

starting time

Referenced by main().

6.35.3.26 stopTime

```
double stopTime
```

simulation end time

Referenced by calcTimeStep(), initTimeDisc(), and timeDisc().

6.35.3.27 t

```
double t
```

global calculation time

Referenced by calcTimeStep(), cgnsReadSolution(), GMRES_M(), implicitTimeStep(), initTimeDisc(), and time←
Disc().

6.35.3.28 timeOrder

```
int timeOrder
```

order of the time integration

Referenced by fvTimeDerivative(), initTimeDisc(), and timeDisc().

6.35.3.29 timeOverall

```
double timeOverall
```

overall time

Referenced by cgnsOutput(), and cgnsReadSolution().

6.36 ccfd/src/timeDiscretization.h File Reference

```
#include <stdbool.h>
#include <time.h>
```

Include dependency graph for timeDiscretization.h: This graph shows which files directly or indirectly include this file:

Macros

- #define CPU_TIME() ((double)clock() / (double)CLOCKS_PER_SEC)

  Get the CPU time for a serial program.

Functions

- void initTimeDisc (void)

  Initialize the time discretization.
- void timeDisc (void)

  Main time discretization loop.

Variables

- double cfl
- double dfl
- double t
- double timeOverall
- int timeOrder
- bool isTimeStep1D
- bool isStationary
- long maxIter
- double stopTime
- long iniIterationNumber
- double startTime
- double abortResidual
- int abortVariable
- char abortVariableName [4]
- double clAbortResidual
- double cdAbortResidual
- bool doAbortOnClResidual
- bool doAbortOnCdResidual
- bool isRestart
- double restartTime
- int printIter
- double printTime
- int nRKstages
- double RKcoeff [6]
- bool isImplicit

6.36.1 Detailed Description

Author

hhh

Date

Sat 21 Mar 2020 07:48:34 PM CET

6.36.2 Function Documentation

6.36.2.1 timeDisc()

```
void timeDisc (
            void )
```

Main time discretization loop.

Selection of temporal integration method, as well as management of data output and analysis tools.

References abortResidual, abortVariable, abortVariableName, analyze(), calcErrors(), calcTimeStep(), wing_t::cd, cdAbortResidual, wing_t::cl, clAbortResidual, CPU_TIME, dataOutput(), deltaX, doAbortOnCdResidual, doAbort←OnClResidual, doCalcWing, E, explicitTimeStepEuler(), explicitTimeStepRK(), F_X0, F_XK, finalizeDataOutput(), hasExactSolution, implicitTimeStep(), iniIterationNumber, recordPoint_t::ioFile, IOiterInterval, IOtimeInterval, is←Implicit, isStationary, maxIter, MX, MY, nGMRESiterGlobal, nNewtonIterGlobal, recordPoint_t::nPoints, nRKstages, NVAR, printIter, printTime, Q, recordPoint, resFile, RHO, stopTime, t, timeOrder, and wing.

Referenced by main().

6.36.3 Variable Documentation

6.36.3.1 abortResidual

`double abortResidual [extern]`

residual at which to abort the calculation

Referenced by initTimeDisc(), and timeDisc().

6.36.3.2 abortVariable

`int abortVariable [extern]`

abort variable

Referenced by analyze(), initTimeDisc(), and timeDisc().

6.36.3.3 abortVariableName

`char abortVariableName[4] [extern]`

string of the abort variable

Referenced by initAnalyze(), initTimeDisc(), and timeDisc().

6.36.3.4 cdAbortResidual

`double cdAbortResidual [extern]`

CD abort residual

Referenced by initTimeDisc(), and timeDisc().

6.36.3.5 cfl

`double cfl [extern]`

Courant-Friedrichs-Lewy number

Referenced by calcTimeStep(), and initTimeDisc().

6.36.3.6 clAbortResidual

```
double clAbortResidual [extern]
```

CL abort residual

Referenced by initTimeDisc(), and timeDisc().

6.36.3.7 dfl

```
double dfl [extern]
```

diffusive Courant-Friedrichs-Lewy number

Referenced by calcTimeStep(), and initTimeDisc().

6.36.3.8 doAbortOnCdResidual

```
bool doAbortOnCdResidual [extern]
```

CD abort flag

Referenced by initTimeDisc(), and timeDisc().

6.36.3.9 doAbortOnClResidual

```
bool doAbortOnClResidual [extern]
```

CL abort flag

Referenced by initTimeDisc(), and timeDisc().

6.36.3.10 iniIterationNumber

```
long iniIterationNumber [extern]
```

initial iteration number

Referenced by initTimeDisc(), main(), and timeDisc().

6.36.3.11 isImplicit

```
bool isImplicit  [extern]
```

implicit calculation flag

Referenced by freeLinearSolver(), initLinearSolver(), initTimeDisc(), and timeDisc().

6.36.3.12 isRestart

```
bool isRestart  [extern]
```

restart flag

Referenced by initAnalyze(), initMesh(), initTimeDisc(), main(), and setInitialCondition().

6.36.3.13 isStationary

```
bool isStationary  [extern]
```

flag for stationary problem

Referenced by analyze(), cgnsFinalizeOutput(), dataOutput(), initAnalyze(), initTimeDisc(), main(), and timeDisc().

6.36.3.14 isTimeStep1D

```
bool isTimeStep1D  [extern]
```

flag for 1D problem

Referenced by calcTimeStep(), and initTimeDisc().

6.36.3.15 maxIter

```
long maxIter  [extern]
```

maximum number of iterations

Referenced by initTimeDisc(), and timeDisc().

6.36.3.16 nRKstages

```
int nRKstages [extern]
```

number of Runge-Kutta stages

Referenced by explicitTimeStepRK(), initTimeDisc(), and timeDisc().

6.36.3.17 printIter

```
int printIter [extern]
```

iterations after which to output

Referenced by initTimeDisc(), and timeDisc().

6.36.3.18 printTime

```
double printTime [extern]
```

calculation time after which to output

Referenced by initTimeDisc(), and timeDisc().

6.36.3.19 restartTime

```
double restartTime [extern]
```

calculation time for restart

Referenced by initTimeDisc(), and main().

6.36.3.20 RKcoeff

```
double RKcoeff[6] [extern]
```

array of Runge-Kutta coefficients

Referenced by explicitTimeStepRK(), and initTimeDisc().

6.36.3.21 startTime

```
double startTime  [extern]
```

starting time

Referenced by main().

6.36.3.22 stopTime

```
double stopTime  [extern]
```

simulation end time

Referenced by calcTimeStep(), initTimeDisc(), and timeDisc().

6.36.3.23 t

```
double t  [extern]
```

global calculation time

Referenced by calcTimeStep(), cgnsReadSolution(), GMRES_M(), implicitTimeStep(), initTimeDisc(), and time↩ Disc().

6.36.3.24 timeOrder

```
int timeOrder  [extern]
```

order of the time integration

Referenced by fvTimeDerivative(), initTimeDisc(), and timeDisc().

6.36.3.25 timeOverall

```
double timeOverall  [extern]
```

overall time

Referenced by cgnsOutput(), and cgnsReadSolution().

Index