

SPHERLS Reference Manual

1.0

Generated by Doxygen 1.3.9.1

Tue Jan 10 13:58:37 2012

Contents

1	Using and Modifying SPHERLS	1
1.1	Overview	1
1.2	Program Flow	2
1.3	Compiling SPHERLS	2
1.4	Using SPHERLS	4
1.5	Modifing or Developing SPHERLS	4
1.6	Message Passing	6
2	SPHERLS Class Index	7
2.1	SPHERLS Class List	7
3	SPHERLS File Index	9
3.1	SPHERLS File List	9
4	SPHERLS Page Index	11
4.1	SPHERLS Related Pages	11
5	SPHERLS Class Documentation	13
5.1	eos Class Reference	13
5.2	Functions Class Reference	21
5.3	Global Class Reference	24
5.4	Grid Class Reference	26
5.5	Implicit Class Reference	36
5.6	MessPass Class Reference	40
5.7	Output Class Reference	42
5.8	Parameters Class Reference	45
5.9	Performance Class Reference	48
5.10	ProcTop Class Reference	49
5.11	Time Class Reference	52
5.12	WatchZone Class Reference	55

6	SPHERLS File Documentation	57
6.1	/home/cgeroux/SPHERLS/src/eos.cpp File Reference	57
6.2	/home/cgeroux/SPHERLS/src/eos.h File Reference	58
6.3	dataManipulation.cpp File Reference	59
6.4	dataManipulation.h File Reference	65
6.5	dataMonitoring.cpp File Reference	71
6.6	dataMonitoring.h File Reference	75
6.7	global.cpp File Reference	79
6.8	global.h File Reference	80
6.9	main.cpp File Reference	82
6.10	main.h File Reference	84
6.11	physEquations.cpp File Reference	86
6.12	physEquations.h File Reference	116
6.13	userguide.h File Reference	146
6.14	watchzone.cpp File Reference	147
6.15	watchzone.h File Reference	148
7	SPHERLS Page Documentation	149
7.1	Todo List	149
7.2	Boundary Conditions	151

Chapter 1

Using and Modifying SPHERLS

This manual is divided into two main parts the current chapter, and the rest of the chapters. All chapters other than the current, contain specific reference material for the SPHERLS code while the current chapter contains a more descriptive how-to approach explaining the usage and modification of SPHERLS. The chapters following chapter 1 will serve as a usefull reference when specific details need to be found, for example a discription of a particular variable in the code. The current chapter on the other hand is the best place to go to get a quick understanding of SPHERLS that will enable you to use it.

1.1 Overview

SPHERLS stands for Stellar Pulsation with a Horizontal Eulearian Radial Lagrangian Scheme. There are three components to SPHERLS: SPHERLS itself which does the hydodynamics calculations, SPHERLSgen which creates starting models, and SPHERLSanal which is able to manipulate the output files. Both SPHERLSgen and SPHERLSanal have there own manuals which can be consulted for their specific uses and installations.

1.1.1 The Basics

SPHERLS calculates the radial pulsation motions together with the horizontal convective flow. The radial pulsation can be described by a radial grid velocity `Grid::nU0`, moving the grid inward and outward with the pulsation. The movement of the grid is defined by the motion required to maintaining the mass in a spherical shell through out the calculation. This motion is determined so that it will change the volume of the shell so the newly calculated density when multiplied with the new volume will produce the same shell mass. The total motion of the stellar material is simply the combination of the three velocity components, radial `Grid::nU`, theta `Grid::nV`, and phi velocities `Grid::nW`. The convective motion is the radial velocity minus the grid velocity, combined with the theta and phi velocities. This is because the grid velocity describes the bulk motion of the pulsation so subtracting it out leaves only the convective motions.

SPHERLS solves the normal hydodynamic equations of, mass, momentum, and energy conservation. The form of the mass equation, momentum conservation, and energy conservation are:

$$\frac{dM}{dt} + \oint_S (\rho \vec{v}) \cdot \hat{n} d\sigma = 0$$
$$\frac{\partial \vec{v}}{\partial t} + (\vec{v} \cdot \nabla) \vec{v} = -\frac{1}{\rho} \nabla P + \nabla \cdot \boldsymbol{\tau} - \nabla \phi$$

$$\frac{\partial E}{\partial t} + (\vec{v} \cdot \nabla)E + P \frac{d\mathbb{V}}{dt} = \epsilon + \frac{1}{\rho} [-\nabla \cdot F + \nabla \cdot (\boldsymbol{\tau} \cdot \vec{v}) - (\nabla \cdot \boldsymbol{\tau}) \cdot \vec{v}]$$

where $\boldsymbol{\tau}$ is the stress tensor for zero bulk viscosity, E is the specific internal energy, \mathbb{V} is the specific volume, and F is the radiative flux. In addition to these conservation equations an equation of state is needed, in this case the OPAL equation of state and opacities, and the Alexander opacities at low temperatures are used. The equation of state tables are functions of density and temperature, and produce the energy, pressure, opacity, and adiabatic index of the gas for a given temperature and density. In adiabatic calculations, it is also possible to use a γ -law gas equation of state but in that case an energy profile must also be included.

The simulation grid is broken up into two main sections, the 1D region towards the center of the star, the multi-dimensional region towards the surface. The inner part of the multi-dimensional region solves all the conservation equations explicitly, in that the new values for the conserved quantities are directly calculated from the information in the previous time step. In the outer parts of the multi-dimensional region the energy conservation equation is calculated semi-implicitly, which means that the new values are dependent on the new values averaged with the old values to correctly time center the equation. This semi-implicit energy conservation equation can be perturbed and linearized producing a set of linear equations the size of the region being solved implicitly. The solution of these linear equations provide corrections for the temperature which can be applied and then resolved in an iterative approach until the value of the new temperature converges. The equation of state is a function of temperature and not energy which is why the temperature is perturbed and not the energy. This set of equations for the temperature corrections are solved using the PETSC library.

- Different ways in which SPHERLS can be used, 1D,2D,3D, Adiabatic,Non-adiabatic, implicit, debugging options/test

1.2 Program Flow

- Describe the grids
- The order of calculation
- When parts of the grid are updated

1.3 Compiling SPHERLS

Once the correct libraries are installed, and their paths added to your

```
LD_LIBRARY_PATH
```

environment variable, it should just require typing make in the correct directories. SPHERLS is broken up into 3 main codes. SPHERLS itself, which is the main hydrodynamics code which integrates the initial static model, SPHERLSgen which creates the static model, and SPHERLSanal which is used for processing the output of SPHERLS and SPHERLSgen.

To Add

- example `.bashrc` entries, showing `LD_LIBRARY_PATH` additions, and other SPHERLS related configuration options
- also the make files will need to know where the paths for the libraries are, either describe how the user can do this, or automate it some how.

1.3.1 Requirements

- openMPI
- gcc
- PETSc library
- python for analysis scripts
- fftw library for analysis

1.3.2 Installing PETSC Library

- Download PETSc library, from the PETSc [website](#). Version `petsc-lite-3.1-p8` has been tested to work with SPHERLS.
- The downloaded PETSc file (e.g. `petsc-lite-3.1-p8.tar.gz`) will need to be unzipped to do so type `gunzip petsc-lite-3.1-p8.tar.gz`
- Then untar it with `tar -xf petsc-lite-3.1-p8.tar`
- To install the library change into the directory made when you extracted the archive and type the following commands:
 1. `./configure --prefix=<path-to-final-location-of-library> --with-c++-support --with-c-support --with-shared --download-f-blas-lapack=1`
 2. `make all`
 3. `make install`
 4. `make PETSC_DIR=<path-to-final-location-of-library> test`

1.3.3 Installing FFTW Library

- Download the FFTW Library from the FFTS [website](#). Version `fftw-3.2.2` has been tested to work with SPHERLS.
- The downloaded FFTW file (e.g. `fftw-3.2.2.tar.gz`) will need to be unzipped to do so type `gunzip fftw-3.2.2.tar.gz`
- Then untar it with `tar -xf fftw-3.2.2.tar.gz`
- To install the library change into the directory made when you extracted the archive and type the following commands:

1. `./configure --prefix=<path-to-final-location-of-library>`
2. `make`
3. `make install`

1.4 Using SPHERLS

- Generating a starting model (see SPHERLSgen documentation for details)
- The XML configuration file
- Starting a calculation and the "makeFile"
- getting data
 - watchzones
 - peak KE tracking (might be removed at some point)
 - model dumps
- post calculation analysis (see SPHERLSanal documentation for details)
- Adiabatic Calculations
 - 1D, 2D, and 3D
 - γ -law gas
 - Sedov Blast wave test
- Non-Adiabatic Calculations
 - 1D, 2D, and 3D
 - Tabulate EOS
 - Different versions of the energy equation
 - LES models

1.5 Modifing or Developing SPHERLS

- Basic layout/design of the code
 - model output
 - data monitoring
 - * watch zones
 - * peak KE tracking
 - internal/versus external variables
 - message passing
 - grid layout
 - ranges of grids

- boundary regions
- grid updating
- How to document SPHERLS
- How to modify SPHERLS
 - Common changes
 - * How to add a new internal variable
 1. **Add to the internal variable count:** Decide in what cases the variable will be needed, 1D calculations, 2D calculations, when there is a gamma law gas or a tabulated equation of state, adiabatic or non-adiabatic etc. Then once decided it can be added to the total number of internal variables `Grid::nNumIntVars` by increasing the value by one in the function `modelRead` in the section below the comment "set number of internal variables ..." under the appropriate if block. If the specific if block for the situation you need isn't there, you can create your own, and add it there.
 2. **Create a new variable ID:** In the `grid::h` file under the `Grid` class are variable ID's. These ID's simply indicate the location of the variable in the array. One must add a new ID for the new variable as an integer. The value of the ID is set in the function `modelRead` in the same section as the number of internal variables. The value used should be the last integer after the last pre-existing variable ID. This should also be `Grid::nNumVars + Grid::nNumIntVars - 1`. The ID should also be initialized to -1, so that the code knows when it isn't being used. This is done in the grid class constructor, `Grid::Grid`. Simply add a line in the constructor setting your new ID = -1.
 3. **Set variable infos:** Decide what the dimensions of the new variable will be. It can be cell centered it can be or interface centered, it can also be only 1D, 2D, or 3D. Of course it will be only 1D if the entire calculation is 1D, or 2D if the calculation is 2D, but if the calculation is 3D it could also only be 2D, or 1D, and if 2D it could be only 1D. Also decide if the variable will change with time, dependent variables are only initialized and not updated during the calculations. This information is given to SPHERLS in the `setInternalVarInf` function in the `physEquations.cpp` file. The variable that is set is `Grid::nVariables`. It is a 2D array, the first index corresponds to the particular variable in question, the ID you made in the previous step can be used as the first index of this array. The second index refers to the three directions (0-2) and the time (3). If the variable is centered in the grid in direction 0 (r-direction) then this array element should have a value of 0. If the variable is interface centered in the grid in direction 0, then this array element should have a value of 1. If it isn't defined in direction 0, for example the theta independent variable isn't defined in the 0 direction then it should be -1. This is the same for the other 2 directions. The last element (3) should be either 0 not updated every time step, or 1 if updated every timestep.
 4. **Add functions:** Finally to do anything useful with your new internal variable functions must be added to initialize the values of the variables, and to update them with time if needed. Initialization functions are called within the `initInternalVars` function in the `physEquations.cpp` file. The details of these functions will depend on what the individual variables are intended for. Functions to be called every timestep must be called from the main program loop in the file `main.cpp` in the appropriate order.
 - * How to add a new external variable
 - * How to add a new physics functions
 - Function naming conventions
 - `Grid` variables
 - indices and their ranges

- SPHERLS debugging tips

1.6 Message Passing

- Explain message passing in SPHERLS

Chapter 2

SPHERLS Class Index

2.1 SPHERLS Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

eos	13
Functions	21
Global	24
Grid	26
Implicit	36
MessPass	40
Output	42
Parameters	45
Performance	48
ProcTop	49
Time	52
WatchZone	55

Chapter 3

SPHERLS File Index

3.1 SPHERLS File List

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

/home/cgeroux/SPHERLS/src/eos.cpp	57
/home/cgeroux/SPHERLS/src/eos.h	58
/home/cgeroux/SPHERLS/src/exception2.h	??
/home/cgeroux/SPHERLS/src/xmlFunctions.h	??
dataManipulation.cpp	59
dataManipulation.h	65
dataMonitoring.cpp	71
dataMonitoring.h	75
global.cpp	79
global.h	80
main.cpp	82
main.h	84
physEquations.cpp	86
physEquations.h	116
userguide.h	146
watchzone.cpp	147
watchzone.h	148

Chapter 4

SPHERLS Page Index

4.1 SPHERLS Related Pages

Here is a list of all related documentation pages:

Todo List	149
Boundary Conditions	151

Chapter 5

SPHERLS Class Documentation

5.1 eos Class Reference

```
#include <eos.h>
```

Public Member Functions

- `eos ()`
- `eos (int nNumT, int nNumRho)`
- `eos (const eos &ref)`
- `~eos ()`
- `eos & operator= (const eos &eosRightSide)`
- `void readAscii (std::string sFileName)`
- `void readBobsAscii (std::string sFileName)`
- `void writeAscii (std::string sFileName)`
- `void readBin (std::string sFileName) throw (exception2)`
- `void writeBin (std::string sFileName)`
- `double dGetPressure (double dT, double dRho)`
- `double dGetEnergy (double dT, double dRho)`
- `double dGetOpacity (double dT, double dRho)`
- `double dDRhoDP (double dT, double dRho)`
- `double dSoundSpeed (double dT, double dRho)`
- `void getEKappa (double dT, double dRho, double &dE, double &dKappa)`
- `void getPEKappa (double dT, double dRho, double &dP, double &dE, double &dKappa)`
- `void getPEKappaGamma (double dT, double dRho, double &dP, double &dE, double &dKappa, double &dGamma)`
- `void getPKappaGamma (double dT, double dRho, double &dP, double &dKappa, double &dGamma)`
- `void gamma1DelAdC_v (double dT, double dRho, double &dGamma1, double &dDelAd, double &dC_v)`
- `void getPAndDRhoDP (double dT, double dRho, double &dP, double &dDRhoDP)`
- `void getEAndDTDE (double dT, double dRho, double &dE, double &dTDE)`

Public Attributes

- int `nNumRho`
- int `nNumT`
- double `dXMassFrac`
- double `dYMassFrac`
- double `dLogRhoMin`
- double `dLogRhoDelta`
- double `dLogTMin`
- double `dLogTDelta`
- double ** `dLogP`
- double ** `dLogE`
- double ** `dLogKappa`

5.1.1 Detailed Description

This class holds an equation of state as well as many functions useful for manipulating it

5.1.2 Constructor & Destructor Documentation

5.1.2.1 `eos::eos ()`

Constructor, doesn't really do anything

5.1.2.2 `eos::eos (int nNumT, int nNumRho)`

Constructor, allocates memory for the 2D arrays

Parameters:

- ← *nNumT* number of temperatures in the equaiton of state table
- ← *nNumRho* number of densities in the equaiton of state table

5.1.2.3 `eos::eos (const eos & ref)`

Copy constructor, simply constructs a new eos object from another eos object

5.1.2.4 `eos::~~eos ()`

Destructor, deletes dynamic arrays

5.1.3 Member Function Documentation

5.1.3.1 double eos::dDRhoDP (double dT , double $dRho$)

This function calculates the partial derivative of density w.r.t. pressure

Parameters:

- ← dT temperature at which the derivative is to be computed
- ← $dRho$ density at which the derivative is to be computed

Returns:

the partial derivative of density w.r.t. pressure.

5.1.3.2 double eos::dGetEnergy (double dT , double $dRho$)

This function linearly interpolates the energy to a given temperature and density. Note that both dT and $dRho$ are not in log space.

Parameters:

- ← dT temperature to interpolate to.
- ← $dRho$ density to interpolate to.

Returns:

the interpolated energy.

5.1.3.3 double eos::dGetOpacity (double dT , double $dRho$)

This function linearly interpolates the opacity to a given temperature and density. Note that both dT and $dRho$ are not in log space.

Parameters:

- ← dT temperature to interpolate to.
- ← $dRho$ density to interpolate to.

Returns:

the interpolated opacity.

5.1.3.4 double eos::dGetPressure (double dT , double $dRho$)

This function linearly interpolates the pressure to a given temperature and density. Note that both dT and $dRho$ are not in log space.

Parameters:

- ← dT temperature to interpolate to.
- ← $dRho$ density to interpolate to.

Returns:

the interpolated pressure.

5.1.3.5 double eos::dSoundSpeed (double dT , double $dRho$)

This function calculates the adiabatic sound speed

Parameters:

- ← dT temperature at which the derivative is to be computed
- ← $dRho$ density at which the derivative is to be computed

Returns:

the sound speed.

5.1.3.6 void eos::gamma1DelAdC_v (double dT , double $dRho$, double & $dGamma1$, double & $dDelAd$, double & dC_v)

This function calculates gamma1 and the adiabatic gradient

Parameters:

- ← dT temperature at which the derivative is to be computed
- ← $dRho$ density at which the derivative is to be computed
- $dGamma1$ gamma1
- $dDelAd$ adiabatic gradient
- dC_v specific heat at constant volume

5.1.3.7 void eos::getEAndDTDE (double dT , double $dRho$, double & dE , double & $dDTDE$)

This function calculates the partial derivative of temperature w.r.t. energy and the energy

Parameters:

- ← dT temperature at which the derivative is to be computed
- ← $dRho$ density at which the derivative is to be computed
- dE energy at dT and $dRho$
- $dDTDE$ derivative of temperature w.r.t. energy at constant density

5.1.3.8 void eos::getEKappa (double dT , double $dRho$, double & dE , double & $dKappa$)

This function linearly interpolates the three dependent quantities (Pressure, Energy , Opacity) to a given temperature and density. Note that both dT and $dRho$ are not in log space.

Parameters:

- ← dT temperature to interpolate to.
- ← $dRho$ density to interpolate to.
- dE energy at dT and $dRho$.
- $dKappa$ opacity at dT and $dRho$.

5.1.3.9 void eos::getPAndDRhoDP (double *dT*, double *dRho*, double & *dP*, double & *ddRhoDP*)

This function calculates the partial derivative of density w.r.t. pressure and the pressure

Parameters:

- ← *dT* temperature at which the derivative is to be computed
- ← *dRho* density at which the derivative is to be computed
- *dP* pressure at *dT* and *dRho*
- *ddRhoDP* derivative of density w.r.t. pressure at constant temperature

5.1.3.10 void eos::getPEKappa (double *dT*, double *dRho*, double & *dP*, double & *dE*, double & *dKappa*)

This function linearly interpolates the three dependent quantities (Pressure, Energy , Opacity) to a given temperature and density. Note that both *dT* and *dRho* are not in log space.

Parameters:

- ← *dT* temperature to interpolate to.
- ← *dRho* density to interpolate to.
- *dP* pressure at *dT* and *dRho*.
- *dE* energy at *dT* and *dRho*.
- *dKappa* opacity at *dT* and *dRho*.

5.1.3.11 void eos::getPEKappaGamma (double *dT*, double *dRho*, double & *dP*, double & *dE*, double & *dKappa*, double & *dGamma*)

This function linearly interpolates the energy and opacity to a given temperature and density. Note that both *dT* and *dRho* are not in log space.

Parameters:

- ← *dT* temperature to interpolate to.
- ← *dRho* density to interpolate to.
- *dP* pressure at *dT* and *dRho*.
- *dE* energy at *dT* and *dRho*.
- *dKappa* opacity at *dT* and *dRho*.
- *dGamma* adiabatic index at *dT* and *dRho*.

5.1.3.12 void eos::getPKappaGamma (double *dT*, double *dRho*, double & *dP*, double & *dKappa*, double & *dGamma*)

This function linearly interpolates the energy and opacity to a given temperature and density. Note that both *dT* and *dRho* are not in log space.

Parameters:

- ← *dT* temperature to interpolate to.
- ← *dRho* density to interpolate to.
- *dP* pressure at *dT* and *dRho*.
- *dKappa* opacity at *dT* and *dRho*.
- *dGamma* adiabatic index at *dT* and *dRho*.

5.1.3.13 eos & eos::operator= (const eos & *eosRightSide*)

Assignment operator, assigns one eos object to another.

5.1.3.14 void eos::readAscii (std::string *sFileName*)

This fuction reads in an ascii file and stores it in the current object.

Parameters:

- ← *sFileName* name of the equation of state file to read from.

5.1.3.15 void eos::readBin (std::string *sFileName*) throw (exception2)

This fuction reads in a binary file and stores it in the current object.

Parameters:

- ← *sFileName* name of the equation of state file to read from.

5.1.3.16 void eos::readBobsAscii (std::string *sFileName*)

This fuction reads in an ascii file and stores it in the current object. The ascii file is in Bob's format.

Parameters:

- ← *sFileName* name of the equation of state file to read from.

5.1.3.17 void eos::writeAscii (std::string *sFileName*)

This function writes the equation of state stored in the current object to an ascii file.

Parameters:

← *sFileName* name of the file to write the equation of state to.

5.1.3.18 void eos::writeBin (std::string *sFileName*)

This function writes the equation of state stored in the current object to a binary file.

Parameters:

← *sFileName* name of the file to write the equation of state to.

5.1.4 Member Data Documentation**5.1.4.1 double** eos::dLogE**

2D array of log10 energies. `dLogE[i][j]` gives the log10 energy at log10 density of `eos::dLogRhoDelta*i+eos::dLogRhoMin`, and at log10 temperature of `eos::dLogTDelta*j+eos::dLogTMin`.

5.1.4.2 double eos::dLogKappa**

2D array of log10 opacities. `dLogKappa[i][j]` gives the log10 opacity at log10 density of `eos::dLogRhoDelta*i+eos::dLogRhoMin`, and at log10 temperature of `eos::dLogTDelta*j+eos::dLogTMin`.

5.1.4.3 double eos::dLogP**

2D array of log10 pressures. `dLogP[i][j]` gives the log10 pressure at log10 density of `eos::dLogRhoDelta*i+eos::dLogRhoMin`, and at log10 temperature of `eos::dLogTDelta*j+eos::dLogTMin`.

5.1.4.4 double eos::dLogRhoDelta

Increment of the density between table entries in log10.

5.1.4.5 double eos::dLogRhoMin

Minimum density of the table in log10.

5.1.4.6 double eos::dLogTDelta

Increment of the temperature between table entries in log10.

5.1.4.7 double [eos::dLogTMin](#)

Minimum temperature of the table in log10.

5.1.4.8 double [eos::dXMassFrac](#)

Hydrogen mass fraction of the composition used to generate the equation of state table.

5.1.4.9 double [eos::dYMassFrac](#)

Helium mass fraction of the composition used to generate the equation of state table.

5.1.4.10 int [eos::nNumRho](#)

Number of densities in the equation of state table

5.1.4.11 int [eos::nNumT](#)

Number of temperatures in the equation of state table

The documentation for this class was generated from the following files:

- [/home/cgeroux/SPHERLS/src/eos.h](#)
- [/home/cgeroux/SPHERLS/src/eos.cpp](#)

5.2 Functions Class Reference

```
#include <global.h>
```

Public Member Functions

- [Functions](#) ()

Public Attributes

- void(* [fpCalculateNewVelocities](#))(Grid &, Parameters &, Time &, ProcTop &)
- void(* [fpCalculateNewGridVelocities](#))(Grid &, Parameters &, Time &, ProcTop &, MessPass &)
- void(* [fpCalculateNewRadii](#))(Grid &, Time &)
- void(* [fpCalculateNewDensities](#))(Grid &, Parameters &, Time &, ProcTop &)
- void(* [fpCalculateNewEnergies](#))(Grid &, Parameters &, Time &, ProcTop &)
- void(* [fpCalculateDeltat](#))(Grid &, Parameters &, Time &, ProcTop &)
- void(* [fpCalculateAveDensities](#))(Grid &)
- void(* [fpCalculateNewEOSVars](#))(Grid &, Parameters &)
- void(* [fpCalculateNewAV](#))(Grid &, Parameters &)
- void(* [fpModelWrite](#))(std::string sFileName, ProcTop &, Grid &, Time &, Parameters &)
- void(* [fpWriteWatchZones](#))(Output &, Grid &, Parameters &, Time &, ProcTop &)
- void(* [fpUpdateLocalBoundaryVelocitiesNewGrid](#))(ProcTop &, MessPass &, Grid &)
- void(* [fpImplicitSolve](#))(Grid &, Implicit &, Parameters &, Time &, ProcTop &, MessPass &, [Functions](#) &)
- void(* [fpCalculateNewEddyVisc](#))(Grid &, Parameters &)

5.2.1 Detailed Description

This class holds function pointers used to indicate the functions which should be used to calculate the various needed quantities. These functions can be different from processor to processor. For example [ProcTop::nRank](#)=0 processor will have only 1D versions of the conservation equations, while the rest of the processors will have 3D versions. These functions will also change depending on what kind of model is being calculated and the number of dimensions the calculation includes.

5.2.2 Constructor & Destructor Documentation

5.2.2.1 Functions::Functions ()

Constructor for the class [Functions](#).

5.2.3 Member Data Documentation

5.2.3.1 void(* Functions::fpCalculateAveDensities)(Grid &)

Function pointer to the function used to calculate the new average density.

5.2.3.2 void(* Functions::fpCalculateDeltat)(Grid &, Parameters &, Time &, ProcTop &)

Function pointer to the function used to calculate the new time step.

5.2.3.3 void(* Functions::fpCalculateNewAV)(Grid &, Parameters &)

Function pointer to the function used to calculate new Artificial viscosity.

5.2.3.4 void(* Functions::fpCalculateNewDensities)(Grid &, Parameters &, Time &, ProcTop &)

Function pointer to the function used to calculate the new densities.

5.2.3.5 void(* Functions::fpCalculateNewEddyVisc)(Grid &, Parameters &)

Function pointer to the function that is used to calculate the new eddy viscosity.

5.2.3.6 void(* Functions::fpCalculateNewEnergies)(Grid &, Parameters &, Time &, ProcTop &)

Function pointer to the function used to calculate the new energies.

5.2.3.7 void(* Functions::fpCalculateNewEOSVars)(Grid &, Parameters &)

Function pointer to the function used to calculate the new variables depending on the equation of state.

5.2.3.8 void(* Functions::fpCalculateNewGridVelocities)(Grid &, Parameters &, Time &, ProcTop &, MessPass &)

Function pointer to the function used to calculate new grid velocities.

5.2.3.9 void(* Functions::fpCalculateNewRadii)(Grid &, Time &)

Function pointer to the function used to calculate new radii.

5.2.3.10 void(* Functions::fpCalculateNewVelocities)(Grid &, Parameters &, Time &, ProcTop &)

Function pointer to the function used to calculate new velocities.

5.2.3.11 void(* Functions::fpImplicitSolve)(Grid &, Implicit &, Parameters &, Time &, ProcTop &, MessPass &, Functions &)

Function pointer to the function that is used to implicitly solve for the temperature, then uses the equation of state to solve for energy, opacity, and pressure.

5.2.3.12 void(* [Functions::fpModelWrite](#))(std::string sFileName, [ProcTop](#) &, [Grid](#) &, [Time](#) &, [Parameters](#) &)

Function pointer to the function used to write out models.

5.2.3.13 void(* [Functions::fpUpdateLocalBoundaryVelocitiesNewGrid](#))([ProcTop](#) &, [MessPass](#) &, [Grid](#) &)

Function pointer to the function that is used to update velocities across boundaries.

5.2.3.14 void(* [Functions::fpWriteWatchZones](#))([Output](#) &, [Grid](#) &, [Parameters](#) &, [Time](#) &, [ProcTop](#) &)

Function pointer to the function that is used to write out watch zone files

The documentation for this class was generated from the following files:

- [global.h](#)
- [global.cpp](#)

5.3 Global Class Reference

```
#include <global.h>
```

Public Member Functions

- [Global \(\)](#)

Public Attributes

- [ProcTop procTop](#)
- [MessPass messPass](#)
- [Grid grid](#)
- [Time time](#)
- [Parameters parameters](#)
- [Output output](#)
- [Performance performance](#)
- [Functions functions](#)
- [Implicit implicit](#)

5.3.1 Detailed Description

This class is simply a class that holds the other classes.

5.3.2 Constructor & Destructor Documentation

5.3.2.1 [Global::Global \(\)](#)

Constructor for the class [Global](#).

5.3.3 Member Data Documentation

5.3.3.1 [Functions Global::functions](#)

An instance of the [Functions](#) class.

5.3.3.2 [Grid Global::grid](#)

An instance of the [Grid](#) class.

5.3.3.3 [Implicit Global::implicit](#)

An instance of the [Implicit](#) class.

5.3.3.4 [MessPass Global::messPass](#)

An instance of the [MessPass](#) class.

5.3.3.5 [Output Global::output](#)

An instance of the [Output](#) class.

5.3.3.6 [Parameters Global::parameters](#)

An instance of the [Parameters](#) class.

5.3.3.7 [Performance Global::performance](#)

An instance of the [Performance](#) class.

5.3.3.8 [ProcTop Global::procTop](#)

An instance of the [ProcTop](#) class.

5.3.3.9 [Time Global::time](#)

An instance of the [Time](#) class.

The documentation for this class was generated from the following files:

- [global.h](#)
- [global.cpp](#)

5.4 Grid Class Reference

```
#include <global.h>
```

Public Member Functions

- [Grid\(\)](#)

Public Attributes

- int [nM](#)
- int [nTheta](#)
- int [nPhi](#)
- int [nDM](#)
- int [nR](#)
- int [nD](#)
- int [nU](#)
- int [nU0](#)
- int [nV](#)
- int [nW](#)
- int [nT](#)
- int [nE](#)
- int [nP](#)
- int [nKappa](#)
- int [nGamma](#)
- int [nDenAve](#)
- int [nQ0](#)
- int [nQ1](#)
- int [nQ2](#)
- int [nDTheta](#)
- int [nDPhi](#)
- int [nSinThetaIJK](#)
- int [nSinThetaIjp1halfK](#)
- int [nCotThetaIjp1halfK](#)
- int [nCotThetaIJK](#)
- int [nDCosThetaIJK](#)
- int [nEddyVisc](#)
- int [nNumDims](#)
- int [nNumVars](#)
- int [nNumIntVars](#)
- int [nNumIDZones](#)
- int [nNumGhostCells](#)
- int * [nGlobalGridDims](#)
- int ** [nVariables](#)
- int *** [nLocalGridDims](#)
- double **** [dLocalGridNew](#)

- double *** dLocalGridOld
- int ** nStartUpdateExplicit
- int ** nEndUpdateExplicit
- int ** nStartUpdateImplicit
- int ** nEndUpdateImplicit
- int *** nStartGhostUpdateExplicit
- int *** nEndGhostUpdateExplicit
- int *** nStartGhostUpdateImplicit
- int *** nEndGhostUpdateImplicit
- int * nCenIntOffset
- int nGlobalGridPositionLocalGrid [3]

5.4.1 Detailed Description

This class manages information which pertains to grid data.

External variables used with Gamma Law (GL) gas equation of state and their array indexes:

1D (nNumVars=7)		2D (nNumVars=9)		3D (nNumVars=11)	
Variable	Index	Variable	Index	Variable	Index
nM	0	nM	0	nM	0
nDM	1	nTheta	1	nTheta	1
nR	2	nDM	2	nPhi	2
nD	3	nR	3	nDM	3
nU	4	nD	4	nR	4
nU0	5	nU	5	nD	5
nE	6	nU0	6	nU	6
		nV	7	nU0	7
		nE	8	nV	8
				nW	9
				nE	10

External variables used with Tabulated Equation Of State (TEOS) and their array indexes:

1D (nNumVars=7)		2D (nNumVars=9)		3D (nNumVars=11)	
Variable	Index	Variable	Index	Variable	Index
nM	0	nM	0	nM	0
nDM	1	nTheta	1	nTheta	1
nR	2	nDM	2	nPhi	2
nD	3	nR	3	nDM	3
nU	4	nD	4	nR	4
nU0	5	nU	5	nD	5
nT	6	nU0	6	nU	6
		nV	7	nU0	7
		nT	8	nV	8
				nW	9
				nT	10

Internal variables with GL gas equation of state:

1D (nNumIntVars=2)		2D (nNumIntVars=8)	
Variable	Index	Variable	Index
nP	nNumVars+0	nP	nNumVars+0
nQ0	nNumVars+1	nQ0	nNumVars+1
		nDenAve	nNumVars+2
		nDCosThetaIJK	nNumVars+3
		nQ1	nNumVars+4
		nDTheta	nNumVars+5
		nSinThetaIJK	nNumVars+6
		nSinThetaIjp1halfK	nNumVars+7
3D (nNumIntVars=12)			
Variable	Index		
nP	nNumVars+0		
nQ0	nNumVars+1		
nDenAve	nNumVars+2		
nDPhi	nNumVars+3		
nDCosThetaIJK	nNumVars+4		
nQ1	nNumVars+5		
nDTheta	nNumVars+6		
nSinThetaIJK	nNumVars+7		
nSinThetaIjp1halfK	nNumVars+8		
nCotThetaIJK	nNumVars+9		
nCotThetaIjp1halfK	nNumVars+10		
nQ2	nNumVars+11		

Internal variables with TEOS:

1D (nNumIntVars=5)		2D (nNumIntVars=11)	
Variable	Index	Variable	Index
nP	nNumVars+0	nP	nNumVars+0
nQ0	nNumVars+1	nQ0	nNumVars+1
nE	nNumVars+2	nDenAve	nNumVars+2
nKappa	nNumVars+3	nDCosThetaIJK	nNumVars+3
nGamma	nNumVars+4	nE	nNumVars+4
		nKappa	nNumVars+5
		nGamma	nNumVars+6
		nQ1	nNumVars+7
		nDTheta	nNumVars+8
		nSinThetaIJK	nNumVars+9
		nSinThetaIjp1halfK	nNumVars+10
3D (nNumIntVars=15)			
Variable	Index		
nP	nNumVars+0		
nQ0	nNumVars+1		
nDenAve	nNumVars+2		
nDPhi	nNumVars+3		
nDCosThetaIJK	nNumVars+4		
nE	nNumVars+5		
nKappa	nNumVars+6		
nGamma	nNumVars+7		
nQ1	nNumVars+8		
nDTheta	nNumVars+9		
nSinThetaIJK	nNumVars+10		
nSinThetaIjp1halfK	nNumVars+11		
nCotThetaIJK	nNumVars+12		
nCotThetaIjp1halfK	nNumVars+13		
nQ2	nNumVars+14		

The variable indexes are set in [modelRead](#) based on the input model.

5.4.2 Constructor & Destructor Documentation

5.4.2.1 [Grid::Grid \(\)](#)

Constructor for the class [Grid](#).

5.4.3 Member Data Documentation

5.4.3.1 double*** [Grid::dLocalGridNew](#)

Updated local grid values. An array of size [Grid::nNumVars](#)+[Grid::nNumIntVars](#) by [Grid::nLocalGridDims](#)[0]+2*[Grid::nNumGhostCells](#) by [Grid::nLocalGridDims](#)[1]+2*[Grid::nNumGhostCells](#) by [Grid::nLocalGridDims](#)[2]+2*[Grid::nNumGhostCells](#) provided that the variable is defined in all 3 directions. Variables that are not defined in all 3 directions will have the additional two ghost cells left out in that direction and will also have a dimension of size 1 in that direction. This array contains the current grid state as it is being updated through calculations. This is a processor dependent variable and contains only the local grid for the current processor plus ghost cells.

5.4.3.2 double*** [Grid::dLocalGridOld](#)

Grid values from previous time step. An array the same size as [Grid::dLocalGridNew](#) but instead of containing the current grid state, it contains the last complete grid state. This is a processor dependent variable and contains only the local grid for the current processor plus ghost cells.

5.4.3.3 int* [Grid::nCenIntOffset](#)

Indicates the offset between interface and center quantities. If [nCenIntOffset](#)[1]=0 then the outer interface quantities have the same index as zone centered quantities in direction 1. If [nCenIntOffset](#)[1]=1 then the outer interface quantities are given by the index for the zone centered quantities +1, in direction 1. The values are dependent on [ProcTop::nRank](#) and [ProcTop::nPeriodic](#).

5.4.3.4 int [Grid::nCotThetaIJK](#)

Index of $\cot \theta$ at cell centers of grids. This is an internal grid variable and is included in the count of [Grid::nNumIntVars](#).

5.4.3.5 int [Grid::nCotThetaIjp1halfK](#)

Index of $\cot \theta$ at θ interfaces in grids. This is an internal grid variable and is included in the count of [Grid::nNumIntVars](#).

5.4.3.6 int [Grid::nD](#)

Index of ρ in grids, [Grid::dLocalGridOld](#) and [Grid::dLocalGridNew](#). This is an external grid variable included in the count [Grid::nNumVars](#)

5.4.3.7 int [Grid::nDCosThetaIJK](#)

Index of $\Delta \cos \theta$ defined at zone center in grids. This is an internal grid variable and is included in the count of [Grid::nNumIntVars](#).

5.4.3.8 int [Grid::nDenAve](#)

Index of $\langle \rho \rangle$ in grids, [Grid::dLocalGridOld](#) and [Grid::dLocalGridNew](#). This is an internal grid variable and is included in the count of [Grid::nNumIntVars](#). This variable is defined at cell centers only in the radial direction.

5.4.3.9 int `Grid::nDM`

Index of δM in grids, `Grid::dLocalGridOld` and `Grid::dLocalGridNew`. This is an external grid variable included in the count `Grid::nNumVars`

5.4.3.10 int `Grid::nDPhi`

Index of $\Delta\phi$ in grids, `Grid::dLocalGridOld` and `Grid::dLocalGridNew`. This is an internal grid variable and is included in the count of `Grid::nNumIntVars`. This variable is defined at cell centers.

5.4.3.11 int `Grid::nDTheta`

Index of $\Delta\theta$ in grids, `Grid::dLocalGridOld` and `Grid::dLocalGridNew`. This is an internal grid variable and is included in the count of `Grid::nNumIntVars`. This variable is defined at cell centers.

5.4.3.12 int `Grid::nE`

Index of E in grids, `Grid::dLocalGridOld` and `Grid::dLocalGridNew`. This is an internal grid variable included in the count `Grid::nNumIntVars`, unless the calculation is adiabatic in which case it is an external grid variable. This variable is defined at cell centers.

5.4.3.13 int `Grid::nEddyVisc`

Index of the eddy viscosity in the grid, it is defined at zone centers in the grids. This is an internal grid variable and is included in the count of `Grid::nNumIntVars`.

5.4.3.14 int* `Grid::nEndGhostUpdateExplicit`**

Positions to end updating ghost cells with explicit calculations. Is an array of size `Grid::nNumVars+Grid::nNumIntVars` by 2*3 by 3. The second dimension corresponds to which ghost region, since each dimension can have two ghost regions. The ghost region 0, is the outer ghost region in direction 0, 1 is the inner ghost region in direction 0, etc.

5.4.3.15 int* `Grid::nEndGhostUpdateImplicit`**

Positions to end updating ghost cells with implicit calculations. Is an array of size `Grid::nNumVars+Grid::nNumIntVars` by 2*3 by 3. The second dimension corresponds to which ghost region, since each dimension can have two ghost regions. The ghost region 0, is the outer ghost region in direction 0, 1 is the inner ghost region in direction 0, etc.

5.4.3.16 int `Grid::nEndUpdateExplicit`**

Positions to stop updating grid with explicit calculations. It is an array of size `nNumVars+nNumIntVars` by 3. The end positions are defined in `initUpdateLocalBoundaries()`. These start values are dependent on processor `ProcTop::nRank`.

5.4.3.17 `int** Grid::nEndUpdateImplicit`

Positions to stop updating grid with implicit calculations. It is an array of size `nNumVars+nNumIntVars` by 3. The end positions are defined in `initUpdateLocalBoundaries()`. These start values are dependent on processor `ProcTop::nRank`.

5.4.3.18 `int Grid::nGamma`

Index of adiabatic index in grids, `Grid::dLocalGridOld` and `Grid::dLocalGridNew`. This is an internal grid variable and is included in the count of `Grid::nNumIntVars`. This variable is defined at cell centers.

5.4.3.19 `int* Grid::nGlobalGridDims`

Size of the entire global grid. It is an array of size 3 to hold size of each dimension of global grid. This size does not include `Grid::nNumGhostCells` or the extra size required for interface centered quantities. The values of this variable are independent of processor `ProcTop::nRank`. In the case of 1D or 2D calculations the θ and ϕ dimensions are set to 1 or just the ϕ dimensions is set to 1 depending on the number of dimensions. The r , θ and ϕ dimensions are in the 0, 1 and 2 indices of the array respectively.

5.4.3.20 `int Grid::nGlobalGridPositionLocalGrid[3]`

The location at which the local grid starts in the global grid

5.4.3.21 `int Grid::nKappa`

Index of Opacity in grids, `Grid::dLocalGridOld` and `Grid::dLocalGridNew`. This is an internal grid variable and is included in the count of `Grid::nNumIntVars`. This variable is defined at cell centers.

5.4.3.22 `int*** Grid::nLocalGridDims`

Local grid dimensions. It is An array of size `ProcTop::nNumProcs` by `Grid::nNumVars+Grid::nNumIntVars` by 3. `nLocalGridDims[p][n][l]` gives the dimension of the local grid on processor p for variable n in direction l . This variable does not include `Grid::nNumGhostCells`. The values of this variable are independent of processor `ProcTop::nRank`.

5.4.3.23 `int Grid::nM`

Index of M_r independent variable in grid `Grid::dLocalGridOld` and `Grid::dLocalGridNew`. This is an external grid variable included in the count `Grid::nNumVars`. This is an independent grid variable.

5.4.3.24 `int Grid::nNum1DZones`

Number of zones in 1D region of grid. The number of zones in 3D region is (`Grid::nGlobalGridDims[0]-Grid::nNum1DZones`). This is set when reading in the model input file in the function `modelRead`. The value of this variable is independent of processor `ProcTop::nRank`.

5.4.3.25 int [Grid::nNumDims](#)

Number of dimensions of the grid. It is used to chose the appropriate conservation equations. The value of this variable is independent of processor [ProcTop::nRank](#).

5.4.3.26 int [Grid::nNumGhostCells](#)

Number of cells which are not included in local grid updating. This number is used in all dimensions to add to each local grid. When variables are not defined in a given direction ghost cells are not included in that direction. This is set when reading in the model input file in the function [modelRead](#). The value of this variable is independent of processor [ProcTop::nRank](#).

5.4.3.27 int [Grid::nNumIntVars](#)

Number of internal variables. Internal variables are variables which are not reported in model dumps, and are not required to fully specify a starting model. They are used to save important information required during computation, an example is $\sin \theta$. The value of this variable is independent of processor [ProcTop::nRank](#). This variable is set depending on the model read in (adiabatic/non-adiabatic/number of dimensions) in the function [modelRead](#) located in the file [dataManipulation.cpp](#).

5.4.3.28 int [Grid::nNumVars](#)

Number of grid variables. This is set when reading in the model input file in the function [modelRead](#). It is the number of variables that are printed and read from a file. The total number of variables also includes [Grid::nNumIntVars](#). The value of this variable is independent of processor [ProcTop::nRank](#).

5.4.3.29 int [Grid::nP](#)

Index of Pressure in grids, [Grid::dLocalGridOld](#) and [Grid::dLocalGridNew](#). This is an internal grid variable and is included in the count of [Grid::nNumIntVars](#). This variable is defined at cell centers.

5.4.3.30 int [Grid::nPhi](#)

Index of ϕ independent variable in grid [Grid::dLocalGridOld](#) and [Grid::dLocalGridNew](#). This is an external grid variable included in the count [Grid::nNumVars](#). This is an independent grid variable.

5.4.3.31 int [Grid::nQ0](#)

Index of the radial artificial viscosity in grids, [Grid::dLocalGridOld](#) and [Grid::dLocalGridNew](#). This is an internal grid variable and is included in the count of [Grid::nNumIntVars](#). This variable is defined at cell centers.

5.4.3.32 int [Grid::nQ1](#)

Index of the theta artificial viscosity in grids, [Grid::dLocalGridOld](#) and [Grid::dLocalGridNew](#). This is an internal grid variable and is included in the count of [Grid::nNumIntVars](#). This variable is defined at cell centers.

5.4.3.33 int [Grid::nQ2](#)

Index of the ϕ artificial viscosity in grids, [Grid::dLocalGridOld](#) and [Grid::dLocalGridNew](#). This is an internal grid variable and is included in the count of [Grid::nNumIntVars](#). This variable is defined at cell centers.

5.4.3.34 int [Grid::nR](#)

Index of r in grids, [Grid::dLocalGridOld](#) and [Grid::dLocalGridNew](#). This is an external grid variable included in the count [Grid::nNumVars](#)

5.4.3.35 int [Grid::nSinThetaIJK](#)

Index of $\sin \theta$ defined at zone center in grids, [Grid::dLocalGridOld](#) and [Grid::dLocalGridNew](#). This is an internal grid variable and is included in the count of [Grid::nNumIntVars](#).

5.4.3.36 int [Grid::nSinThetaIjp1halfK](#)

Index of $\sin \theta$ at θ interfaces in grids. This is an internal grid variable and is included in the count of [Grid::nNumIntVars](#).

5.4.3.37 int* [Grid::nStartGhostUpdateExplicit](#)**

Positions to begin updating ghost cells with explicit calculations. It is an array of size [Grid::nNumVars+Grid::nNumIntVars](#) by 2*3 by 3. The second dimension indicates a particular ghost region. There are 2*3 since each direction can have two ghost regions. The ghost region 0, is the outer ghost region in direction 0, 1 is the inner ghost region in direction 0, etc.

5.4.3.38 int* [Grid::nStartGhostUpdateImplicit](#)**

Positions to begin updating ghost cells with implicit calculations. It is an array of size [Grid::nNumVars+Grid::nNumIntVars](#) by 2*3 by 3. The second dimension indicates a particular ghost region. There are 2*3 since each direction can have two ghost regions. The ghost region 0, is the outer ghost region in direction 0, 1 is the inner ghost region in direction 0, etc.

5.4.3.39 int [Grid::nStartUpdateExplicit](#)**

Positions to begin updating grid with explicit calculations. It is an array of size [nNumVars+nNumIntVars](#) by 3. The start positions are defined in [initUpdateLocalBoundaries\(\)](#). These start values are dependent on processor [ProcTop::nRank](#).

5.4.3.40 int [Grid::nStartUpdateImplicit](#)**

Positions to begin updating grid with implicit calculations. It is an array of size [nNumVars+nNumIntVars](#) by 3. The start positions are defined in [initUpdateLocalBoundaries\(\)](#). These start values are dependent on processor [ProcTop::nRank](#).

5.4.3.41 int `Grid::nT`

Index of T in grids, `Grid::dLocalGridOld` and `Grid::dLocalGridNew`. This is an external grid variable included in the count `Grid::nNumVars`. This variable is defined at cell centers.

5.4.3.42 int `Grid::nTheta`

Index of θ independent variable in grid `Grid::dLocalGridOld` and `Grid::dLocalGridNew`. This is an external grid variable included in the count `Grid::nNumVars`. This is an independent grid variable.

5.4.3.43 int `Grid::nU`

Index of u in grids, `Grid::dLocalGridOld` and `Grid::dLocalGridNew`. This is an external grid variable included in the count `Grid::nNumVars`

5.4.3.44 int `Grid::nU0`

Index of u_0 in grids, `Grid::dLocalGridOld` and `Grid::dLocalGridNew`. This is an external grid variable included in the count `Grid::nNumVars`

5.4.3.45 int `Grid::nV`

Index of v in grids, `Grid::dLocalGridOld` and `Grid::dLocalGridNew`. This is an external grid variable included in the count `Grid::nNumVars`

5.4.3.46 int `Grid::nVariables`**

Provides information on grid variables. A 2D array of size `Grid::nNumVars+Grid::nNumIntVars` by 3+1. `nVariables[n][l]` has values:

- -1: indicating that variable n is not defined
- 0: indicating that variable n is zone centered quantity
- 1: indicating that variable n is an interface centered quantity

in directions $l=0, 1, 2$ which corresponding to \hat{r} , $\hat{\theta}$, and $\hat{\phi}$ respectively. `nVariables[n][l]` with $l=3$ is used to indicate if a variable is dependent on time (1) or not(0). The values of this variable are independent of processor `ProcTop::nRank`.

5.4.3.47 int `Grid::nW`

Index of w in grids, `Grid::dLocalGridOld` and `Grid::dLocalGridNew`. This is an external grid variable included in the count `Grid::nNumVars`

The documentation for this class was generated from the following files:

- `global.h`
- `global.cpp`

5.5 Implicit Class Reference

```
#include <global.h>
```

Public Member Functions

- [Implicit \(\)](#)

Public Attributes

- int [nNumImplicitZones](#)
- Mat [matCoeff](#)
- Vec [vecTCorrections](#)
- Vec [vecRHS](#)
- Vec [vecTCorrectionsLocal](#)
- KSP [kspContext](#)
- VecScatter [vecscatTCorrections](#)
- int [nMaxNumIterations](#)
- double [dTolerance](#)
- int [nNumRowsALocal](#)
- int [nNumRowsALocalSB](#)
- int * [nNumDerPerRow](#)
- int ** [nTypeDer](#)
- int *** [nLocDer](#)
- int ** [nLocFun](#)
- double [dDerivativeStepFraction](#)
- double [dCurrentRelTError](#)
- int [nCurrentNumIterations](#)
- int [nMaxNumSolverIterations](#)
- double [dMaxErrorInRHS](#)
- double [dAverageRHS](#)

5.5.1 Detailed Description

This class holds data required for the implicit calculation.

5.5.2 Constructor & Destructor Documentation

5.5.2.1 Implicit::Implicit ()

constructor the the class [Implicit](#).

5.5.3 Member Data Documentation

5.5.3.1 double `Implicit::dAverageRHS`

Holds the average value of the right hand side for the timestep where the error in the RHS is the largest `dMaxErrorInRHS`. Only set if `TRACKMAXSOLVERERROR` is set to 1.

5.5.3.2 double `Implicit::dCurrentRelTError`

keeps track of the largest relative error in the calculation of the temperature

5.5.3.3 double `Implicit::dDerivativeStepFraction`

Dicates the size of the step that should be used to evaluate the numerical derivitves of the energy equation, for solving for the temperature implicitly. This value multiplies the temperature to produce the step size. A good value is around 5e-7.

5.5.3.4 double `Implicit::dMaxErrorInRHS`

If `TRACKMAXSOLVERERROR` set to 1, then this will be the current maximum absolute error between the RHS as calculated from the solution and the coeffecient matrix, and the actual RHS. This value is the maximum from all values at each iteration of the solution, from each time step since the last model dump.

5.5.3.5 double `Implicit::dTolerance`

The amount of relative error that is allowed in the calculation of the temperature with the implicit calculation.

5.5.3.6 KSP `Implicit::kspContext`

PETSc solver context.

5.5.3.7 Mat `Implicit::matCoeff`

Parallel coeffecient matrix (spread across all processors)

5.5.3.8 int `Implicit::nCurrentNumIterations`

keeps track of the number of iterations needed to converge to a solution

5.5.3.9 int*** `Implicit::nLocDer`

An array of size `nNumRowsALocal` by 2 by `nNumDerPerRow` [q] , where q is a row index. This array holds the global position of the current row q for the current derivative e.g. the p th derivative in the q th row would be in row and column (`nLocDer[q][0][p]` , `nLocDer[q][1][p]`). The value of this variable is set in the function `initImplicitCalculation` .

5.5.3.10 int Implicit::nLocFun**

An array of size `nNumRowsALocal` by 3 [q] , where q is a row index. This array holds the local grid position of the current row q e.g. the (i,j,k) location of the the current row in the local grid. The value of this variable is set in the function `initImplicitCalculation` .

5.5.3.11 int Implicit::nMaxNumIterations

The maximum number of iterations to try to get the largest value of `vecTCorrections` relative to the temperature below `dTolerance`. After which the calculation continues.

5.5.3.12 int Implicit::nMaxNumSolverIterations

If `TRACKMAXSOLVERERROR` set to 1, then this will be the current maximum number of iterations required for the linear equation solver to solve for the temperature correction over all iterations and time steps since the last model dump.

5.5.3.13 int* Implicit::nNumDerPerRow

An array of size `nNumRowsALocal` which contains the number of non-zero derivatives for a given row of A.

5.5.3.14 int Implicit::nNumImplicitZones

The number of zones in the region near the surface which should use the implicit calculation of the energy equation. If zero no zones will use the implicit calculation of energy.

5.5.3.15 int Implicit::nNumRowsALocal

The number of rows of the coefficient matrix which is on the local processor.

5.5.3.16 int Implicit::nNumRowsALocalSB

The number of rows of the coefficient matrix which is on the local processor, and that are in the surface boundary region.

5.5.3.17 int Implicit::nTypeDer**

An array of size `nNumRowsALocal` by `nNumDerPerRow` [q] , where q is a row index. Thus each row of the array can have a different length. This gives the type of derivative of row q for each derivative in that row. The value of this variable is set in the function `initImplicitCalculation` .

5.5.3.18 Vec Implicit::vecRHS

RHS vector (spread across all processors)

5.5.3.19 VecScatter [Implicit::vecscatTCorrections](#)

Scatter context, used to hold information about retrieving the distributed temperature corrections from `vecTCorrections` and placing them into the local vector `vecTCorrectionsLocal`.

5.5.3.20 Vec [Implicit::vecTCorrections](#)

Temperature corrections solution vector (spread across all processors)

5.5.3.21 Vec [Implicit::vecTCorrectionsLocal](#)

Corrections to local temperatures only (on local processor only).

The documentation for this class was generated from the following files:

- [global.h](#)
- [global.cpp](#)

5.6 MessPass Class Reference

```
#include <global.h>
```

Public Member Functions

- [MessPass \(\)](#)

Public Attributes

- MPI::Datatype * [typeSendNewGrid](#)
- MPI::Datatype * [typeRecvOldGrid](#)
- MPI::Datatype ** [typeSendNewVar](#)
- MPI::Datatype ** [typeRecvNewVar](#)
- MPI::Request * [requestSend](#)
- MPI::Request * [requestRecv](#)
- MPI::Status * [statusSend](#)
- MPI::Status * [statusRecv](#)

5.6.1 Detailed Description

This class manages information which pertains to message passing between processors.

5.6.2 Constructor & Destructor Documentation

5.6.2.1 MessPass::MessPass ()

Constructor for class [MessPass](#).

5.6.3 Member Data Documentation

5.6.3.1 MPI::Request* [MessPass::requestRecv](#)

Message handles.

5.6.3.2 MPI::Request* [MessPass::requestSend](#)

Message handles.

5.6.3.3 MPI::Status* [MessPass::statusRecv](#)

Message status.

5.6.3.4 MPI::Status* [MessPass::statusSend](#)

Message status.

5.6.3.5 MPI::Datatype [MessPass::typeRecvNewVar](#)**

Recieve data types for variables. It is of size [ProcTop::nNumNeighbors](#) by [Grid::nNumVars](#).

5.6.3.6 MPI::Datatype* [MessPass::typeRecvOldGrid](#)

Recv data types for entire grid. It is of sizee [ProcTop::nNumNeighbors](#).

5.6.3.7 MPI::Datatype* [MessPass::typeSendNewGrid](#)

Send data types for entire grid. It is of size [ProcTop::nNumNeighbors](#).

5.6.3.8 MPI::Datatype [MessPass::typeSendNewVar](#)**

Send data types for variables. It is of size [ProcTop::nNumNeighbors](#) by [Grid::nNumVars](#).

The documentation for this class was generated from the following files:

- [global.h](#)
- [global.cpp](#)

5.7 Output Class Reference

```
#include <global.h>
```

Public Member Functions

- [Output \(\)](#)

Public Attributes

- int [nDumpFrequencyStep](#)
- double [dDumpFrequencyTime](#)
- double [dTimeLastDump](#)
- int [nNumTimeStepsSinceLastPrint](#)
- bool [bDump](#)
- bool [bPrint](#)
- int [nPrintMode](#)
- std::string [sBaseOutputFileName](#)
- std::ofstream * [ofWatchZoneFiles](#)
- std::vector< [WatchZone](#) > [watchzoneList](#)
- int [nPrintFrequencyStep](#)
- double [dPrintFrequencyTime](#)
- double [dTimeLastPrint](#)

5.7.1 Detailed Description

This class manages information pertaining to the output of data to files.

5.7.2 Constructor & Destructor Documentation

5.7.2.1 [Output::Output \(\)](#)

Constructor for this class.

5.7.3 Member Data Documentation

5.7.3.1 bool [Output::bDump](#)

Should the grid state be written to a file at a frequency of [Output::nDumpFrequencyStep](#) timesteps, and/or every [Output::dDumpFrequencyTime](#) seconds of simulation time. This is set to true by putting a "<dump>" node into the "SPHERLS.xml" configuration file.

5.7.3.2 bool [Output::bPrint](#)

Should status updates be printed to the screen.

5.7.3.3 double [Output::dDumpFrequencyTime](#)

How often a the grid state should be written to a file according to simulation time in seconds. If it is 0 no dumps will be made according to simulation time.

5.7.3.4 double [Output::dPrintFrequencyTime](#)

How often the status is printed to the screen in simulation time.

5.7.3.5 double [Output::dTimeLastDump](#)

The simulation time at which the last dump was made using the [Output::dDumpFrequencyTime](#) criterion.

5.7.3.6 double [Output::dTimeLastPrint](#)

Simulation time when last status was printed.

5.7.3.7 int [Output::nDumpFrequencyStep](#)

How often a the grid state should be written to a file according to time step index. If it is 1 the will state will be written every time step, if it equals 2 it will be written every other time step etc. If it is 0 no dumps will be made according to the time step index.

5.7.3.8 int [Output::nNumTimeStepsSinceLastPrint](#)

The number of time steps since the last model dump.

5.7.3.9 int [Output::nPrintFrequencyStep](#)

How often the status is printed to the screen in time steps.

5.7.3.10 int [Output::nPrintMode](#)

Sets the way in which information should be printed to the standard output during the run. If it is 0, it will print the standard information reporting on the progress of the code. If it is 1 it will print out information to diagnose timestepping problems.

5.7.3.11 [std::ofstream*](#) [Output::ofWatchZoneFiles](#)

An array of output streams of size [Output::watchzoneList](#).size() which are used to write out the information of the watched zones.

5.7.3.12 `std::string` [Output::sBaseOutputFileName](#)

Base filename used for output, default is "out". All model dumps, and output information will contain this file name and extend it to indicate their specific information. The value of this variable is independent of processor [ProcTop::nRank](#).

5.7.3.13 `std::vector<WatchZone>` [Output::watchzoneList](#)

A vector used to keep information used to specify the zones to be watched.

The documentation for this class was generated from the following files:

- [global.h](#)
- [global.cpp](#)

5.8 Parameters Class Reference

```
#include <global.h>
```

Public Member Functions

- [Parameters](#) ()

Public Attributes

- bool [bEOSGammaLaw](#)
- bool [bAdiabatic](#)
- int [nTypeTurbulenceMod](#)
- double [dPi](#)
- double [dSigma](#)
- double [dG](#)
- double [dGamma](#)
- std::string [sEOSFileName](#)
- [eos](#) [eosTable](#)
- double [dA](#)
- double [dAVThreshold](#)
- double [dDonorFrac](#)
- double [dAlpha](#)
- double [dTolerance](#)
- int [nMaxIterations](#)
- double [dEddyViscosity](#)
- double [dMaxConvectiveVelocity](#)
- double [dMaxConvectiveVelocity_c](#)

5.8.1 Detailed Description

This class holds parameters and constants used for calculation.

5.8.2 Constructor & Destructor Documentation

5.8.2.1 [Parameters::Parameters](#) ()

Constructor for the class [Parameters](#)

5.8.3 Member Data Documentation

5.8.3.1 bool [Parameters::bAdiabatic](#)

If true SPHERLS will use adiabatic functions to calculate the energy. This can be used for both gamma law gas and tabulated equations of state (see [Parameters::bEOSGammaLaw](#)).

5.8.3.2 bool [Parameters::bEOSGammaLaw](#)

If true SPHERLS will use a gamma law gas instead of a tabulated equation of state. This is set in the starting model.

5.8.3.3 double [Parameters::dA](#)

Artificial viscosity parameter, reasonable values range from 0 to ~ 3 .

5.8.3.4 double [Parameters::dAlpha](#)

This parameter controls the amount of extra mass above the outter interface. it is read in from the starting model, so that it will be consistent with the value used in calculating the starting model.

5.8.3.5 double [Parameters::dAVThreshold](#)

The amount of compression before AV is turned on. It is in terms of a velocity difference between zone sides and is in fractions of the local sound speed.

5.8.3.6 double [Parameters::dDonorFrac](#)

Fraction of the upwind gradient to contribute to the advection term

5.8.3.7 double [Parameters::dEddyViscosity](#)

Used in calculating the eddy viscosity, larger values will produce a larger value of the eddy viscosity, causing the rethermalization to happen at larger scales. This value should be kept small, a good value is 0.17, which seems to correspond with experiments.

5.8.3.8 double [Parameters::dG](#)

The Gravitational constant G .

5.8.3.9 double [Parameters::dGamma](#)

The adiabatic γ , used in calculating the equation of state. If using a gamma law gas.

5.8.3.10 double [Parameters::dMaxConvectiveVelocity](#)

Holds the maximum convective velocity, it is set in the functions which calculate the timestep (see [calDelt_R_GL](#), [calDelt_R_TEOS](#), [calDelt_RT_GL](#), [calDelt_RT_TEOS](#), [calDelt_RTP_GL](#), [calDelt_RTP_TEOS](#), [calDelt_CONST](#)).

5.8.3.11 double [Parameters::dMaxConvectiveVelocity_c](#)

Holds the maximum of convective velocity divided by the sound speed. It is set in the functions which calculate the timestep (see [calDelt_R_GL](#), [calDelt_R_TEOS](#), [calDelt_RT_GL](#), [calDelt_RT_TEOS](#), [calDelt_RTP_GL](#), [calDelt_RTP_TEOS](#), [calDelt_CONST](#)).

5.8.3.12 double [Parameters::dPi](#)

The value of π .

5.8.3.13 double [Parameters::dSigma](#)

The value of σ , the Stefan-Boltzmann constant.

5.8.3.14 double [Parameters::dTolerance](#)

Amount of error to tolerate when calculating temperature from the equation of state.

5.8.3.15 eos [Parameters::eosTable](#)

Holds the equation of state table. If using a tabulated equation of state.

5.8.3.16 int [Parameters::nMaxIterations](#)

The maximum number of iterations to try to get the the relative error in the temperture below [parameters::dTolerance](#).

5.8.3.17 int [Parameters::nTypeTurbulenceMod](#)

This variable indicates the type of turbulence model to be used. If 0, no turbulence model will be used, if 1 it will use a constant times the zoning size, and if 2 it will use the Smagorinsky turbulence model which increases the value of the eddy viscosity parameter when there are large amounts of shear, and decrease it when there isn't.

5.8.3.18 std::string [Parameters::sEOSFileName](#)

File name of equation of state table. This value is set either by the configuration file, SPHERLS.xml or in the model file read in. If it is specified in SPHERLS.xml it will override the file name set in the model.

The documentation for this class was generated from the following files:

- [global.h](#)
- [global.cpp](#)

5.9 Performance Class Reference

```
#include <global.h>
```

Public Member Functions

- [Performance \(\)](#)

Public Attributes

- double [dStartTimer](#)
- double [dEndTimer](#)

5.9.1 Detailed Description

This class manages information pertaining to performance analysis of the code.

5.9.2 Constructor & Destructor Documentation

5.9.2.1 [Performance::Performance \(\)](#)

Constructor for the class [Performance](#).

5.9.3 Member Data Documentation

5.9.3.1 double [Performance::dEndTimer](#)

The time that the code timer was ended. The difference between [Performance::dStartTimer](#) and `dEndTimer` gives the total run time

5.9.3.2 double [Performance::dStartTimer](#)

The time that the code timer was started.

The documentation for this class was generated from the following files:

- [global.h](#)
- [global.cpp](#)

5.10 ProcTop Class Reference

```
#include <global.h>
```

Public Member Functions

- [ProcTop\(\)](#)

Public Attributes

- int [nNumProcs](#)
- int * [nProcDims](#)
- int * [nPeriodic](#)
- int ** [nCoords](#)
- int [nRank](#)
- int [nNumNeighbors](#)
- int * [nNeighborRanks](#)
- int [nNumRadialNeighbors](#)
- int * [nRadialNeighborRanks](#)
- int * [nRadialNeighborNeighborIDs](#)

5.10.1 Detailed Description

This class manages information which pertains to the processor topology.

5.10.2 Constructor & Destructor Documentation

5.10.2.1 ProcTop::ProcTop()

Constructor for class [ProcTop](#).

5.10.3 Member Data Documentation

5.10.3.1 int** [ProcTop::nCoords](#)

Coordinates of the processors. It is of size [ProcTop::nNumProcs](#) by 3. The values of this variable are independent of processor [ProcTop::nRank](#).

5.10.3.2 int* [ProcTop::nNeighborRanks](#)

[ProcTop::nRank](#)s of the neighboring processors. An array of size [nNumNeighbors](#) to hold ranks of neighbouring processors.

5.10.3.3 int [ProcTop::nNumNeighbors](#)

The number of neighbors surrounding the current processor. The maximum number of neighbors possible is 27, 3x3x3 don't forget the current processor itself can be its own neighbor because of periodic boundary conditions. The value of this variable is dependent on processor [ProcTop::nRank](#).

5.10.3.4 int [ProcTop::nNumProcs](#)

Number of processors in global communicator MPI::COMM_WORLD. The value of this variable is independent of processor [ProcTop::nRank](#).

5.10.3.5 int [ProcTop::nNumRadialNeighbors](#)

The number of neighbors in the radial direction. Can range from 1 to 2 depending on whether there is a processor beneath or above the current processor.

5.10.3.6 int* [ProcTop::nPeriodic](#)

Periodic boundary conditions. It is an array of size 3 to tell if a dimension is periodic (wraps) or not. It contains an integer value of 0 or 1. 0, the boundary condition is not periodic, 1 the boundary condition is periodic. The value of this variable is set in the configuration file "config.xml" which is parsed by the function [init](#). The values of this variable are independent of processor [ProcTop::nRank](#).

5.10.3.7 int* [ProcTop::nProcDims](#)

Dimensions of the processor topology. It is an array of size 3 to hold the size of the processor grid in each dimension. The value of this variable is set in the configuration file "config.xml" which is parsed by the function [init](#). The values of this variable are independent of processor [ProcTop::nRank](#).

5.10.3.8 int* [ProcTop::nRadialNeighborNeighborIDs](#)

Holds the ID of a radial neighbor, to be used to obtain their [ProcTop::nRank](#) from [ProcTop::nNeighborRanks](#)

5.10.3.9 int* [ProcTop::nRadialNeighborRanks](#)

[ProcTop::nRank](#)s of the neighboring radial processors. It is an array of size [ProcTop::nNumRadialNeighbors](#).

5.10.3.10 int [ProcTop::nRank](#)

Is a unique integer which identifies the processor. The values of [ProcTop::nRank](#) range from 0 to [ProcTop::nNumProcs](#)-1 depending on the processor.

The documentation for this class was generated from the following files:

- [global.h](#)
- [global.cpp](#)

5.11 Time Class Reference

```
#include <global.h>
```

Public Member Functions

- [Time \(\)](#)

Public Attributes

- double [dDeltat_np1half](#)
- double [dDeltat_nm1half](#)
- double [dDeltat_n](#)
- double [dt](#)
- double [dEndTime](#)
- double [dTimeStepFactor](#)
- int [nTimeStepIndex](#)
- bool [bVariableTimeStep](#)
- double [dConstTimeStep](#)
- double [dPerChange](#)
- double [dDelRho_t_Rho_max](#)
- double [dDelT_t_T_max](#)
- double [dDelE_t_E_max](#)

5.11.1 Detailed Description

This class manages information which pertains to time variables.

5.11.2 Constructor & Destructor Documentation

5.11.2.1 [Time::Time \(\)](#)

Constructor for the class [Time](#).

5.11.3 Member Data Documentation

5.11.3.1 bool [Time::bVariableTimeStep](#)

If true a variable time step is used as specified by the Courant condition, times the [dTimeStepFactor](#).

5.11.3.2 double [Time::dConstTimeStep](#)

If set to a value other than 0, will use that constant time step in place of the courant time step.

5.11.3.3 double [Time::dDelE_t_E_max](#)

Keeps track of the maximum relative change in energy from one time step to the next. This quantity is only tracked if the calculation is adiabatic, else the temperature is tracked instead, see [Time::dDelT_t_T_max](#)

5.11.3.4 double [Time::dDelRho_t_Rho_max](#)

Keeps track of the maximum relative change in density from one time step to the next.

5.11.3.5 double [Time::dDelT_t_T_max](#)

Keeps track of the maximum relative change in temperature from one time step to the next. This quantity is only tracked if the calculation is non-adiabatic, else the energy is tracked instead, see [Time::dDelE_t_E_max](#)

5.11.3.6 double [Time::dDeltat_n](#)

The time step centered at n in seconds. It is used for calculating new variables defined at time step $n + 1/2$, e.g. the radial velocity [Grid::nU](#). This value is determined by averaging the current [Time::dDeltat_np1half](#), and the last [Time::dDeltat_np1half](#).

5.11.3.7 double [Time::dDeltat_nm1half](#)

The previously used timestep centered at $n - 1/2$ in seconds. It is used for calculating [dDeltat_n](#) the n centered time step.

5.11.3.8 double [Time::dDeltat_np1half](#)

The time step centered at $n + 1/2$ in seconds. It is used for calculating new variables defined at time step n , e.g. the density [Grid::nD](#).

5.11.3.9 double [Time::dEndTime](#)

The end time of the current calculation in seconds.

5.11.3.10 double [Time::dPerChange](#)

A percentage amount to allow the maximum horizontal temperature variation and radial, theta and phi convective velocities to change by from one time step to the next. The time step is reduced accordingly to keep this percent change intact.

5.11.3.11 double [Time::dt](#)

The current time of the simulation in seconds.

5.11.3.12 double [Time::dTimeStepFactor](#)

Used for determining the time step. It is the factor which the courant time step is multiplied by in order to determine [Time::dDeltat_np1half](#).

5.11.3.13 int [Time::nTimeStepIndex](#)

An index indicating the current time step. An index of zero corresponds to a [Time::dt=0](#).

The documentation for this class was generated from the following files:

- [global.h](#)
- [global.cpp](#)

5.12 WatchZone Class Reference

```
#include <watchzone.h>
```

5.12.1 Detailed Description

This class contains information used to monitor a particular zone of the grid.

The documentation for this class was generated from the following files:

- [watchzone.h](#)
- [watchzone.cpp](#)

Chapter 6

SPHERLS File Documentation

6.1 /home/cgeroux/SPHERLS/src/eos.cpp File Reference

```
#include <string>
#include <fstream>
#include <sstream>
#include <iostream>
#include <cmath>
#include "eos.h"
#include "exception2.h"
```

6.1.1 Detailed Description

Implements the eos (equation of state) class defined in [eos.h](#)

6.2 /home/cgeroux/SPHERLS/src/eos.h File Reference

```
#include <string>
#include "exception2.h"
```

Classes

- class [eos](#)

6.2.1 Detailed Description

Header file for [eos.cpp](#)

6.3 dataManipulation.cpp File Reference

```
#include <cmath>
#include <sstream>
#include <fstream>
#include <iomanip>
#include <vector>
#include <fcntl.h>
#include "dataManipulation.h"
#include "global.h"
#include "xmlFunctions.h"
#include "exception2.h"
#include "dataMonitoring.h"
#include "physEquations.h"
#include <string>
```

Functions

- void [init](#) ([ProcTop](#) &procTop, [Grid](#) &grid, [Output](#) &output, [Time](#) &time, [Parameters](#) ¶meters, [MessPass](#) &messPass, [Performance](#) &performance, [Implicit](#) &implicit, int nNumArgs, char *cArgs[])
- void [setupLocalGrid](#) ([ProcTop](#) &procTop, [Grid](#) &grid)
- void [fin](#) (bool bWriteCurrentStateToFile, [Time](#) &time, [Output](#) &output, [ProcTop](#) &procTop, [Grid](#) &grid, [Parameters](#) ¶meters, [Functions](#) &functions, [Performance](#) &performance, [Implicit](#) &implicit)
- void [modelWrite_GL](#) (std::string sFileName, [ProcTop](#) &procTop, [Grid](#) &grid, [Time](#) &time, [Parameters](#) ¶meters)
- void [modelWrite_TEOS](#) (std::string sFileName, [ProcTop](#) &procTop, [Grid](#) &grid, [Time](#) &time, [Parameters](#) ¶meters)
- void [modelRead](#) (std::string sFileName, [ProcTop](#) &procTop, [Grid](#) &grid, [Time](#) &time, [Parameters](#) ¶meters)
- void [initUpdateLocalBoundaries](#) ([ProcTop](#) &procTop, [Grid](#) &grid, [MessPass](#) &messPass, [Implicit](#) &implicit)
- void [updateLocalBoundaries](#) ([ProcTop](#) &procTop, [MessPass](#) &messPass, [Grid](#) &grid)
- void [updateLocalBoundariesNewGrid](#) (int nVar, [ProcTop](#) &procTop, [MessPass](#) &messPass, [Grid](#) &grid)
- void [updateOldGrid](#) ([ProcTop](#) &procTop, [Grid](#) &grid)
- void [updateNewGridWithOld](#) ([Grid](#) &grid, [ProcTop](#) &procTop)
- void [average3DTo1DBoundariesOld](#) ([Grid](#) &grid)
- void [average3DTo1DBoundariesNew](#) ([Grid](#) &grid, int nVar)
- void [updateLocalBoundaryVelocitiesNewGrid_R](#) ([ProcTop](#) &procTop, [MessPass](#) &messPass, [Grid](#) &grid)

- void `updateLocalBoundaryVelocitiesNewGrid_RT` (`ProcTop` &procTop, `MessPass` &messPass, `Grid` &grid)
- void `updateLocalBoundaryVelocitiesNewGrid_RTP` (`ProcTop` &procTop, `MessPass` &messPass, `Grid` &grid)
- void `initImplicitCalculation` (`Implicit` &implicit, `Grid` &grid, `ProcTop` &procTop, int nNumArgs, char *cArgs[])

6.3.1 Detailed Description

This file holds functions for manipulating data. This includes initializing the program, parsing the configuration file "config.xml", allocating memory for the model to be read in, reading in the input model, etc.

6.3.2 Function Documentation

6.3.2.1 void `average3DTo1DBoundariesNew` (`Grid` & *grid*, int *nVar*)

This function averages the 3D boundary recieved by the 1D processor (`ProcTop::nRank` ==0) into 1D. This average is volume weighted. This function only needs to be called by the 1D processor, and if called by other processors may have unexpected results. This function calculates the average from the new grid, and places the average into new old grid. It does so for only the specified variable. This function is used every time the grid boundaries are updated with `updateLocalBoundariesNewGrid`.

Parameters:

- ↔ *grid* supplies the information for calculating the averages and recieves the averages.
- ← *nVar* index of the variable to be averaged with in the grid.

6.3.2.2 void `average3DTo1DBoundariesOld` (`Grid` & *grid*)

This function averages the 3D boundary recieved by the 1D processor (`ProcTop::nRank` ==0) into 1D. This average is volume weighted. This function only needs to be called by the 1D processor, and if called by other processors may have unexpected results. This function calculates the average from the old grid, and places the average into the old grid. It does so for all variables external and internal. This function is used every time the grid boundaries are updated with `updateLocalBoundaries`.

Parameters:

- ↔ *grid* supplies the information for calculating the averages and recieves the averages.

6.3.2.3 void `fin` (bool *bWriteCurrentStateToFile*, `Time` & *time*, `Output` & *output*, `ProcTop` & *procTop*, `Grid` & *grid*, `Parameters` & *parameters*, `Functions` & *functions*, `Performance` & *performance*, `Implicit` & *implicit*)

Finishes program execution by writing out last grid state, closing output files, and writting out run time.

Parameters:

- ← ***bWriteCurrentStateToFile*** is a bool value which indicates wheather or not to write out current model state.
- ← ***time***
- ← ***output***
- ← ***procTop***
- ← ***grid***
- ← ***parameters***
- ← ***functions***
- ← ***performance***
- ← ***implicit***

6.3.2.4 void init (**ProcTop** & *procTop*, **Grid** & *grid*, **Output** & *output*, **Time** & *time*, **Parameters** & *parameters*, **MessPass** & *messPass*, **Performance** & *performance*, **Implicit** & *implicit*, int *argc*, char * *argv* [])

Initializes the program. It does this by reading a number of configuration options from the config file "SPHERLS.xml". It also reads in the starting model, as specified in the "SPHERLS.xml" file, using the function [modelRead](#). During the reading of the initial model the [modelRead](#) function also calls [setupLocalGrid](#) to determine the sizes of the local grids and allocate memory for them.

Other things of note that are done in this function are:

- the calulation timer is started, [Performance::dStartTimer](#)
- It also reads in the equation of state table if using a tabulated equation of state ([Parameters::bEOSGammaLaw](#) = false) by calling [eos::readBin](#)
- Initilizes the watchZones, i.e. figure out which processors have which watch zones, opens the files and prints headers.

Parameters:

- ***procTop*** all parts of this stucture are set, and do not change throughout the rest of the calculation.
- ***grid*** through the function [modelRead](#) the function [setupLocalGrid](#) is called to allocate memory for the grid, and set sizes of it.
- ***output***
- ***time***
- ***parameters***
- ***messPass***
- ***performance***
- ***implicit***
- ← ***argc***
- ← ***argv***

6.3.2.5 void initImplicitCalculation (**Implicit** & *implicit*, **Grid** & *grid*, **ProcTop** & *procTop*, int *nNumArgs*, char * *cArgs*[])

Todo

isFrom, isTo, matCoeff, vecTCorrections, vecTCorrections, vecRHS, vecTCorrectionsLocal, ksp-Context, vecscatTCorrections all need to be destroyed before program finishes.

6.3.2.6 void initUpdateLocalBoundaries (**ProcTop** & *procTop*, **Grid** & *grid*, **MessPass** & *messPass*, **Implicit** & *implicit*)

Sets up MPI derived data types used for updating the local grid boundaries between processors. It sets where the local grids should start/stop updating the local grids (**Grid::nStartUpdateExplicit**, **Grid::nEndUpdateExplicit**, **Grid::nStartUpdateImplicit**, **Grid::nEndUpdateImplicit**, **Grid::nStartGhostUpdateExplicit**, **Grid::nEndGhostUpdateExplicit**, **Grid::nStartGhostUpdateImplicit**, **Grid::nEndGhostUpdateImplicit**). It sets the radial processor neighbors (**ProcTop::nNumRadialNeighbors**).

It also allocates memory for:

- **MessPass::requestSend**
- **MessPass::requestRecv**
- **MessPass::statusSend**
- **MessPass::statusRecv**

Parameters:

↔ *procTop*

↔ *grid*

↔ *messPass*

↔ *implicit*

6.3.2.7 void modelRead (std::string *sFileName*, **ProcTop** & *procTop*, **Grid** & *grid*, **Time** & *time*, **Parameters** & *parameters*)

Todo

At some point should get it working with only 1 processor

6.3.2.8 void modelWrite_GL (std::string *sFileName*, **ProcTop** & *procTop*, **Grid** & *grid*, **Time** & *time*, **Parameters** & *parameters*)

Writes out a model in distributed model format, meaning that each processor writes its own local grid to a file in binary format. They can be combined, and or converted to ascii format using SPHERLSanal. This is for a gamma-law gas model.

Parameters:

- ← *sFileName* base name of the output files
- ← *procTop*
- ← *grid*
- ← *time*
- ← *parameters*

6.3.2.9 void modelWrite_TEOS (std::string *sFileName*, **ProcTop** & *procTop*, **Grid** & *grid*, **Time** & *time*, **Parameters** & *parameters*)

Writes out a model in distributed model format, meaning that each processor writes it's own local grid to a file in binary format. They can be combined, and or converted to ascii format using SPHERLSanal. This is for a tabulated equation of state model.

Parameters:

- ← *sFileName* base name of the output files
- ← *procTop*
- ← *grid*
- ← *time*
- ← *parameters*

6.3.2.10 void setupLocalGrid (**ProcTop** & *procTop*, **Grid** & *grid*)

Determines size of local grids (**Grid::nLocalGridDims**) based on processor topology, and allocates memory for the local grids (**Grid::dLocalGridNew**, **Grid::dLocalGridOld**). It sets various other quantities aswell such as,

- the coordinates of all processors (**ProcTop::nCoords**)
- the offset for interface centered quantities (**Grid::nCenIntOffset**, which depends on zoning and boundary conditions
- the position the local grid is in relative to the global grid (**Grid::nGlobalGridPositionLocalGrid**).

Parameters:

- ↔ *procTop* contains information about the processor topology
- ↔ *grid* contains information about grid

6.3.2.11 void updateLocalBoundaries (**ProcTop** & *procTop*, **MessPass** & *messPass*, **Grid** & *grid*)

Todo

Shouldn't need MPI::COMM_WORLD.Barrier() may want to test out removing this at some point as it might produce a bit of a speed up.

6.3.2.12 void updateLocalBoundariesNewGrid (int *nVar*, **ProcTop & *procTop*, **MessPass** & *messPass*, **Grid** & *grid*)**

Todo

May want to do some waiting on this message at some point before the end of the timestep, but it doesn't need to be done in this function. It might also be that this is built into the code by waiting at some other point. This is something that should be checked out at somepoint, perhaps once the preformance starts to be analyzed. I would think that if the send buffer was being modified before the send was completed, that there would be some errors popping up that would likely kill the program.

6.3.2.13 void updateLocalBoundaryVelocitiesNewGrid_R (ProcTop** & *procTop*, **MessPass** & *messPass*, **Grid** & *grid*)**

Updates velocity boundaries of the new grid in a 1D calculations after the velocities have been newly calculated.

6.3.2.14 void updateLocalBoundaryVelocitiesNewGrid_RT (ProcTop** & *procTop*, **MessPass** & *messPass*, **Grid** & *grid*)**

Updates velocity boundaries of the new grid in a 2D calculations after the velocities have been newly calculated.

6.3.2.15 void updateLocalBoundaryVelocitiesNewGrid_RTP (ProcTop** & *procTop*, **MessPass** & *messPass*, **Grid** & *grid*)**

Updates velocity boundaries of the new grid in a 3D calculations after the velocities have been newly calculated.

6.3.2.16 void updateNewGridWithOld (Grid** & *grid*, **ProcTop** & *procTop*)**

Copies the contents of the old grid to the new grid including ghost cells.

Parameters:

↔ *grid*

← *procTop*

6.3.2.17 void updateOldGrid (ProcTop** & *procTop*, **Grid** & *grid*)**

Updates the old grid with the new grid, not including boundaries.

Parameters:

← *procTop*

↔ *grid*

6.4 dataManipulation.h File Reference

```
#include <mpi.h>
#include "global.h"
```

Functions

- void [init](#) ([ProcTop](#) &procTop, [Grid](#) &grid, [Output](#) &output, [Time](#) &time, [Parameters](#) ¶meters, [MessPass](#) &messPass, [Performance](#) &performance, [Implicit](#) &implicit, int argc, char *argv[])
- void [setupLocalGrid](#) ([ProcTop](#) &procTop, [Grid](#) &grid)
- void [fin](#) (bool bWriteCurrentStateToFile, [Time](#) &time, [Output](#) &output, [ProcTop](#) &procTop, [Grid](#) &grid, [Parameters](#) ¶meters, [Functions](#) &functions, [Performance](#) &performance, [Implicit](#) &implicit)
- void [modelWrite_GL](#) (std::string sFileName, [ProcTop](#) &procTop, [Grid](#) &grid, [Time](#) &time, [Parameters](#) ¶meters)
- void [modelWrite_TEOS](#) (std::string sFileName, [ProcTop](#) &procTop, [Grid](#) &grid, [Time](#) &time, [Parameters](#) ¶meters)
- void [modelRead](#) (std::string sFileName, [ProcTop](#) &procTop, [Grid](#) &grid, [Time](#) &time, [Parameters](#) ¶meters)
- void [initUpdateLocalBoundaries](#) ([ProcTop](#) &procTop, [Grid](#) &grid, [MessPass](#) &messPass, [Implicit](#) &implicit)
- void [updateLocalBoundaries](#) ([ProcTop](#) &procTop, [MessPass](#) &messPass, [Grid](#) &grid)
- void [updateLocalBoundariesNewGrid](#) (int nVar, [ProcTop](#) &procTop, [MessPass](#) &messPass, [Grid](#) &grid)
- void [updateOldGrid](#) ([ProcTop](#) &procTop, [Grid](#) &grid)
- void [updateNewGridWithOld](#) ([Grid](#) &grid, [ProcTop](#) &procTop)
- void [average3DTo1DBoundariesOld](#) ([Grid](#) &grid)
- void [average3DTo1DBoundariesNew](#) ([Grid](#) &grid, int nVar)
- void [updateLocalBoundaryVelocitiesNewGrid_R](#) ([ProcTop](#) &procTop, [MessPass](#) &messPass, [Grid](#) &grid)
- void [updateLocalBoundaryVelocitiesNewGrid_RT](#) ([ProcTop](#) &procTop, [MessPass](#) &messPass, [Grid](#) &grid)
- void [updateLocalBoundaryVelocitiesNewGrid_RTP](#) ([ProcTop](#) &procTop, [MessPass](#) &messPass, [Grid](#) &grid)
- void [initImplicitCalculation](#) ([Implicit](#) &implicit, [Grid](#) &grid, [ProcTop](#) &procTop, int nNumArgs, char *cArgs[])

6.4.1 Detailed Description

Header file for [dataManipulation.cpp](#)

6.4.2 Function Documentation

6.4.2.1 void average3DTo1DBoundariesNew (**Grid** & *grid*, int *nVar*)

This function averages the 3D boundary recieved by the 1D processor (**ProcTop::nRank** ==0) into 1D. This average is volume weighted. This function only needs to be called by the 1D processor, and if called by other processors may have unexpected results. This function calculates the average from the new grid, and places the average into new old grid. It does so for only the specified variable. This function is used every time the grid boundaries are updated with [updateLocalBoundariesNewGrid](#).

Parameters:

- ↔ *grid* supplies the information for calculating the averages and recieves the averages.
- ← *nVar* index of the variable to be averaged with in the grid.

6.4.2.2 void average3DTo1DBoundariesOld (**Grid** & *grid*)

This function averages the 3D boundary recieved by the 1D processor (**ProcTop::nRank** ==0) into 1D. This average is volume weighted. This function only needs to be called by the 1D processor, and if called by other processors may have unexpected results. This function calculates the average from the old grid, and places the average into the old grid. It does so for all variables external and internal. This function is used every time the grid boundaries are updated with [updateLocalBoundaries](#).

Parameters:

- ↔ *grid* supplies the information for calculating the averages and recieves the averages.

6.4.2.3 void fin (bool *bWriteCurrentStateToFile*, **Time** & *time*, **Output** & *output*, **ProcTop** & *procTop*, **Grid** & *grid*, **Parameters** & *parameters*, **Functions** & *functions*, **Performance** & *performance*, **Implicit** & *implicit*)

Finishes program execution by writing out last grid state, closing output files, and writting out run time.

Parameters:

- ← *bWriteCurrentStateToFile* is a bool value which indicates wheather or not to write out current model state.
- ← *time*
- ← *output*
- ← *procTop*
- ← *grid*
- ← *parameters*
- ← *functions*
- ← *performance*
- ← *implicit*

6.4.2.4 void init (**ProcTop** & *procTop*, **Grid** & *grid*, **Output** & *output*, **Time** & *time*, **Parameters** & *parameters*, **MessPass** & *messPass*, **Performance** & *performance*, **Implicit** & *implicit*, int *argc*, char * *argv*[])

Initializes the program. It does this by reading a number of configuration options from the config file "SPHERLS.xml". It also reads in the starting model, as specified in the "SPHERLS.xml" file, using the function [modelRead](#). During the reading of the initial model the [modelRead](#) function also calls [setupLocalGrid](#) to determine the sizes of the local grids and allocate memory for them.

Other things of note that are done in this function are:

- the calculation timer is started, [Performance::dStartTimer](#)
- It also reads in the equation of state table if using a tabulated equation of state ([Parameters::bEOSGammaLaw](#) = false) by calling [eos::readBin](#)
- Initializes the watchZones, i.e. figure out which processors have which watch zones, opens the files and prints headers.

Parameters:

- ***procTop*** all parts of this stucture are set, and do not change throughout the rest of the calculation.
- ***grid*** through the function [modelRead](#) the function [setupLocalGrid](#) is called to allocate memory for the grid, and set sizes of it.
- ***output***
- ***time***
- ***parameters***
- ***messPass***
- ***performance***
- ***implicit***
- ← ***argc***
- ← ***argv***

6.4.2.5 void initImplicitCalculation (**Implicit** & *implicit*, **Grid** & *grid*, **ProcTop** & *procTop*, int *nNumArgs*, char * *cArgs*[])

Todo

isFrom, isTo, matCoeff, vecTCorrections, vecTCorrections, vecRHS, vecTCorrectionsLocal, ksp-Context, vecscatTCorrections all need to be destroyed before program finishes.

6.4.2.6 void initUpdateLocalBoundaries (**ProcTop** & *procTop*, **Grid** & *grid*, **MessPass** & *messPass*, **Implicit** & *implicit*)

Sets up MPI derived data types used for updating the local grid boundaries between processors. It sets where the local grids should start/stop updating the local grids ([Grid::nStartUpdateExplicit](#), [Grid::nEndUpdateExplicit](#), [Grid::nStartUpdateImplicit](#), [Grid::nEndUpdateImplicit](#), [Grid::nStartGhostUpdateExplicit](#), [Grid::nEndGhostUpdateExplicit](#),

`Grid::nStartGhostUpdateImplicit`, `Grid::nEndGhostUpdateImplicit`). It sets the radial processor neighbors (`ProcTop::nNumRadialNeighbors`).

It also allocates memory for:

- `MessPass::requestSend`
- `MessPass::requestRecv`
- `MessPass::statusSend`
- `MessPass::statusRecv`

Parameters:

↔ *procTop*

↔ *grid*

↔ *messPass*

↔ *implicit*

6.4.2.7 `void modelRead (std::string sFileName, ProcTop & procTop, Grid & grid, Time & time, Parameters & parameters)`

Todo

At some point should get it working with only 1 processor

6.4.2.8 `void modelWrite_GL (std::string sFileName, ProcTop & procTop, Grid & grid, Time & time, Parameters & parameters)`

Writes out a model in distributed model format, meaning that each processor writes its own local grid to a file in binary format. They can be combined, and or converted to ascii format using SPHERLSanal. This is for a gamma-law gas model.

Parameters:

← *sFileName* base name of the output files

← *procTop*

← *grid*

← *time*

← *parameters*

6.4.2.9 `void modelWrite_TEOS (std::string sFileName, ProcTop & procTop, Grid & grid, Time & time, Parameters & parameters)`

Writes out a model in distributed model format, meaning that each processor writes its own local grid to a file in binary format. They can be combined, and or converted to ascii format using SPHERLSanal. This is for a tabulated equation of state model.

Parameters:

- ← *sFileName* base name of the output files
- ← *procTop*
- ← *grid*
- ← *time*
- ← *parameters*

6.4.2.10 void setupLocalGrid (ProcTop & procTop, Grid & grid)

Determines size of local grids ([Grid::nLocalGridDims](#)) based on processor topology, and allocates memory for the local grids ([Grid::dLocalGridNew](#), [Grid::dLocalGridOld](#)). It sets various other quantities aswell such as,

- the coordinates of all processors ([ProcTop::nCoords](#))
- the offset for interface centered quantities ([Grid::nCenIntOffset](#), which depends on zoning and boundary conditions
- the position the local grid is in relative to the global grid ([Grid::nGlobalGridPositionLocalGrid](#)).

Parameters:

- ↔ *procTop* contains information about the processor topology
- ↔ *grid* contains information about grid

6.4.2.11 void updateLocalBoundaries (ProcTop & procTop, MessPass & messPass, Grid & grid)**Todo**

Shouldn't need MPI::COMM_WORLD.Barrier() may want to test out removing this at some point as it might produce a bit of a speed up.

6.4.2.12 void updateLocalBoundariesNewGrid (int nVar, ProcTop & procTop, MessPass & messPass, Grid & grid)**Todo**

May want to do some waiting on this message at some point before the end of the timestep, but it doesn't need to be done in this function. It might also be that this is built into the code by waiting at some other point. This is something that should be checked out at somepoint, perhaps once the performance starts to be analyzed. I would think that if the send buffer was being modified before the send was completed, that there would be some errors popping up that would likely kill the program.

6.4.2.13 void updateLocalBoundaryVelocitiesNewGrid_R (ProcTop & procTop, MessPass & messPass, Grid & grid)

Updates velocity boundaries of the new grid in a 1D calculations after the velocities have been newly calculated.

6.4.2.14 void updateLocalBoundaryVelocitiesNewGrid_RT (ProcTop & procTop, MessPass & messPass, Grid & grid)

Updates velocity boundaries of the new grid in a 2D calculations after the velocities have been newly calculated.

6.4.2.15 void updateLocalBoundaryVelocitiesNewGrid_RTP (ProcTop & procTop, MessPass & messPass, Grid & grid)

Updates velocity boundaries of the new grid in a 3D calculations after the velocities have been newly calculated.

6.4.2.16 void updateNewGridWithOld (Grid & grid, ProcTop & procTop)

Copies the contents of the old grid to the new grid including ghost cells.

Parameters:

↔ *grid*

← *procTop*

6.4.2.17 void updateOldGrid (ProcTop & procTop, Grid & grid)

Updates the old grid with the new grid, not including boundaries.

Parameters:

← *procTop*

↔ *grid*

6.5 dataMonitoring.cpp File Reference

```
#include <mpi.h>
#include <sstream>
#include <fstream>
#include <iostream>
#include <cmath>
#include <iomanip>
#include <string>
#include "watchzone.h"
#include "exception2.h"
#include "xmlFunctions.h"
#include "dataMonitoring.h"
#include "global.h"
```

Namespaces

- namespace **std**

Functions

- void [initWatchZones](#) (XMLNode xParent, [ProcTop](#) &procTop, [Grid](#) &grid, [Output](#) &output, [Parameters](#) ¶meters, [Time](#) &time)
- void [writeWatchZones_RT_GL](#) ([Output](#) &output, [Grid](#) &grid, [Parameters](#) ¶meters, [Time](#) &time, [ProcTop](#) &procTop)
- void [writeWatchZones_RT_TEOS](#) ([Output](#) &output, [Grid](#) &grid, [Parameters](#) ¶meters, [Time](#) &time, [ProcTop](#) &procTop)
- void [writeWatchZones_RTP_GL](#) ([Output](#) &output, [Grid](#) &grid, [Parameters](#) ¶meters, [Time](#) &time, [ProcTop](#) &procTop)
- void [writeWatchZones_RTP_TEOS](#) ([Output](#) &output, [Grid](#) &grid, [Parameters](#) ¶meters, [Time](#) &time, [ProcTop](#) &procTop)
- void [writeWatchZones_RTP_GL](#) ([Output](#) &output, [Grid](#) &grid, [Parameters](#) ¶meters, [Time](#) &time, [ProcTop](#) &procTop)
- void [writeWatchZones_RTP_TEOS](#) ([Output](#) &output, [Grid](#) &grid, [Parameters](#) ¶meters, [Time](#) &time, [ProcTop](#) &procTop)
- void [finWatchZones](#) ([Output](#) &output)
- bool [bFileExists](#) (std::string sFilename)

6.5.1 Detailed Description

This file holds functions used for examining the grid data during execution. This includes initializing structures, handling watching zones during the execution of the program, opening files to write out the peak kinetic energy, etc.

6.5.2 Function Documentation

6.5.2.1 bool bFileExists (std::string *sFilename*)

Tests if the file exists by attempting to open the file for reading, if it fails it returns false, if it succeeds it returns true. This does not take into consideration permissions but that is ok for this project.

Parameters:

← *sFilename* file name of the file to check if it exists or not

Returns:

returns true or false depending on whether the file exists

6.5.2.2 void finWatchZones (**Output** & *output*)

Closes the files opened for writing out the watchzones

Parameters:

← *output*

6.5.2.3 void initWatchZones (XMLNode *xParent*, **ProcTop** & *procTop*, **Grid** & *grid*, **Output** & *output*, **Parameters** & *parameters*, **Time** & *time*)

Reads in watchzones set in configuration file "SPHERLS.xml". A list is created on each processor containing the watchzones on that processor's local grid. It also opens file streams for each watchzone and writes out a header.

Parameters:

← *xParent*

← *procTop*

← *grid*

↔ *output*

← *parameters*

← *time*

6.5.2.4 void writeWatchZones_R_GL (**Output** & *output*, **Grid** & *grid*, **Parameters** & *parameters*, **Time** & *time*, **ProcTop** & *procTop*)

Writes out the information for each watchzone specified in "SPHERLS.xml" in the case of a 1D gamma-law gas.

Parameters:

↔ *output*
← *grid*
← *parameters*
← *time*
← *procTop*

6.5.2.5 void writeWatchZones_R_TEOS (*Output* & *output*, *Grid* & *grid*, *Parameters* & *parameters*, *Time* & *time*, *ProcTop* & *procTop*)

Writes out the information for each watchzone specified in "SPHERLS.xml" in the case of a 1D tabulated equation of state.

Parameters:

↔ *output*
← *grid*
← *parameters*
← *time*
← *procTop*

6.5.2.6 void writeWatchZones_RT_GL (*Output* & *output*, *Grid* & *grid*, *Parameters* & *parameters*, *Time* & *time*, *ProcTop* & *procTop*)

Writes out the information for each watchzone specified in "SPHERLS.xml" in the case of a 2D gamma-law gas.

Parameters:

↔ *output*
← *grid*
← *parameters*
← *time*
← *procTop*

6.5.2.7 void writeWatchZones_RT_TEOS (*Output* & *output*, *Grid* & *grid*, *Parameters* & *parameters*, *Time* & *time*, *ProcTop* & *procTop*)

Writes out the information for each watchzone specified in "SPHERLS.xml" in the case of a 2D tabulated equation of state.

Parameters:

↔ *output*

← *grid*
← *parameters*
← *time*
← *procTop*

6.5.2.8 void writeWatchZones_RTP_GL (*Output* & *output*, *Grid* & *grid*, *Parameters* & *parameters*, *Time* & *time*, *ProcTop* & *procTop*)

Writes out the information for each watchzone specified in "SPHERLS.xml" in the case of a 3D gamma-law gas.

Parameters:

↔ *output*
← *grid*
← *parameters*
← *time*
← *procTop*

6.5.2.9 void writeWatchZones_RTP_TEOS (*Output* & *output*, *Grid* & *grid*, *Parameters* & *parameters*, *Time* & *time*, *ProcTop* & *procTop*)

Writes out the information for each watchzone specified in "SPHERLS.xml" in the case of a 3D tabulated equation of state.

Parameters:

↔ *output*
← *grid*
← *parameters*
← *time*
← *procTop*

6.6 dataMonitoring.h File Reference

```
#include <string>
#include "xmlParser.h"
#include "global.h"
```

Functions

- void [initWatchZones](#) (XMLNode xParent, [ProcTop](#) &procTop, [Grid](#) &grid, [Output](#) &output, [Parameters](#) ¶meters, [Time](#) &time)
- void [writeWatchZones_R_GL](#) ([Output](#) &output, [Grid](#) &grid, [Parameters](#) ¶meters, [Time](#) &time, [ProcTop](#) &procTop)
- void [writeWatchZones_R_TEOS](#) ([Output](#) &output, [Grid](#) &grid, [Parameters](#) ¶meters, [Time](#) &time, [ProcTop](#) &procTop)
- void [writeWatchZones_RT_GL](#) ([Output](#) &output, [Grid](#) &grid, [Parameters](#) ¶meters, [Time](#) &time, [ProcTop](#) &procTop)
- void [writeWatchZones_RT_TEOS](#) ([Output](#) &output, [Grid](#) &grid, [Parameters](#) ¶meters, [Time](#) &time, [ProcTop](#) &procTop)
- void [writeWatchZones_RTP_GL](#) ([Output](#) &output, [Grid](#) &grid, [Parameters](#) ¶meters, [Time](#) &time, [ProcTop](#) &procTop)
- void [writeWatchZones_RTP_TEOS](#) ([Output](#) &output, [Grid](#) &grid, [Parameters](#) ¶meters, [Time](#) &time, [ProcTop](#) &procTop)
- void [finWatchZones](#) ([Output](#) &output)
- bool [bFileExists](#) (std::string sFilename)

6.6.1 Detailed Description

Header file for [dataMonitoring.cpp](#)

6.6.2 Function Documentation

6.6.2.1 bool bFileExists (std::string sFilename)

Tests if the file exists by attempting to open the file for reading, if it fails it returns false, if it succeeds it returns true. This does not take into consideration permissions but that is ok for this project.

Parameters:

← *sFilename* file name of the file to check if it exists or not

Returns:

returns true or false depending on whether the file exists

6.6.2.2 void finWatchZones (**Output** & *output*)

Closes the files opened for writing out the watchzones

Parameters:

← *output*

6.6.2.3 void initWatchZones (XMLNode *xParent*, **ProcTop** & *procTop*, **Grid** & *grid*, **Output** & *output*, **Parameters** & *parameters*, **Time** & *time*)

Reads in watchzones set in configuration file "SPHERLS.xml". A list is created on each processor containing the watchzones on that processor's local grid. It also opens file streams for each watchzone and writes out a header.

Parameters:

← *xParent*

← *procTop*

← *grid*

↔ *output*

← *parameters*

← *time*

6.6.2.4 void writeWatchZones_R_GL (**Output** & *output*, **Grid** & *grid*, **Parameters** & *parameters*, **Time** & *time*, **ProcTop** & *procTop*)

Writes out the information for each watchzone specified in "SPHERLS.xml" in the case of a 1D gamma-law gas.

Parameters:

↔ *output*

← *grid*

← *parameters*

← *time*

← *procTop*

6.6.2.5 void writeWatchZones_R_TEOS (**Output** & *output*, **Grid** & *grid*, **Parameters** & *parameters*, **Time** & *time*, **ProcTop** & *procTop*)

Writes out the information for each watchzone specified in "SPHERLS.xml" in the case of a 1D tabulated equation of state.

Parameters:

↔ *output*

← *grid*
← *parameters*
← *time*
← *procTop*

6.6.2.6 void writeWatchZones_RT_GL (*Output* & *output*, *Grid* & *grid*, *Parameters* & *parameters*, *Time* & *time*, *ProcTop* & *procTop*)

Writes out the information for each watchzone specified in "SPHERLS.xml" in the case of a 2D gamma-law gas.

Parameters:

↔ *output*
← *grid*
← *parameters*
← *time*
← *procTop*

6.6.2.7 void writeWatchZones_RT_TEOS (*Output* & *output*, *Grid* & *grid*, *Parameters* & *parameters*, *Time* & *time*, *ProcTop* & *procTop*)

Writes out the information for each watchzone specified in "SPHERLS.xml" in the case of a 2D tabulated equation of state.

Parameters:

↔ *output*
← *grid*
← *parameters*
← *time*
← *procTop*

6.6.2.8 void writeWatchZones_RTP_GL (*Output* & *output*, *Grid* & *grid*, *Parameters* & *parameters*, *Time* & *time*, *ProcTop* & *procTop*)

Writes out the information for each watchzone specified in "SPHERLS.xml" in the case of a 3D gamma-law gas.

Parameters:

↔ *output*
← *grid*

← *parameters*

← *time*

← *procTop*

6.6.2.9 void writeWatchZones_RTP_TEOS (**Output** & *output*, **Grid** & *grid*, **Parameters** & *parameters*, **Time** & *time*, **ProcTop** & *procTop*)

Writes out the information for each watchzone specified in "SPHERLS.xml" in the case of a 3D tabulated equation of state.

Parameters:

↔ *output*

← *grid*

← *parameters*

← *time*

← *procTop*

6.7 global.cpp File Reference

```
#include "global.h"
```

6.7.1 Detailed Description

Declares global variables used across files and functions. This file contains the constructors used to initialize the classes defined in [global.h](#), and does little more than initialize the default values of various parameters.

6.8 global.h File Reference

```
#include <vector>
#include <mpi.h>
#include "watchzone.h"
#include "eos.h"
#include "petscksp.h"
#include <csignal>
```

Classes

- class [ProcTop](#)
- class [MessPass](#)
- class [Grid](#)
- class [Time](#)
- class [Parameters](#)
- class [Output](#)
- class [Performance](#)
- class [Implicit](#)
- class [Functions](#)
- class [Global](#)

Defines

- #define [SIGNEGDN](#) 0
- #define [SIGNEGENG](#) 0
- #define [SIGNEGTEMP](#) 0
- #define [TRACKMAXSOLVERERROR](#) 0
- #define [SEDOV](#) 0
- #define [VISCOUS_ENERGY_EQ](#) 1
- #define [DUMP_VERSION](#) 1

6.8.1 Detailed Description

Header file for [global.cpp](#).

This file contains definitions which are required throughout the program. The classes defined herein are used through out the program.

6.8.2 Define Documentation

6.8.2.1 **#define DUMP_VERSION 1**

Sets the version of the dump file. Should be incremented if changes are made to the information that is printed out in a dump.

6.8.2.2 **#define SEDOV 0**

If 1 we are performing the sedov test, which sets special boundary conditions, if 0 we use normal boundary conditions. It also handles artificial viscosity, and timestep slightly differently.

6.8.2.3 **#define SIGNEG DEN 0**

Raise signal on calculation of negative density if set to 1. Useful when debugging, it will stop the debugger at the location of the calculation of the negative density. If not 1, it will speed up calculation slightly and generate more useful output upon detection of negative densities. If 1 and not being run in the debugger, it likely won't generate any usefull output upon negative density, and wil simply abort the program.

6.8.2.4 **#define SIGNEG ENG 0**

Raise signal on calculation of negative energy if set to 1, else don't rais a signal. Otherwise it will be handled through the normal exception method. This is useful when debugging, it will stop the debugger at the location of the calculation of the negative energy. If not 1, it will speed up calculation slightly and generate more useful output upon detection of negative energy. If 1 and not being run in the debugger, it likely won't generate any usefull output upon negative energies, and wil simply abort the program.

6.8.2.5 **#define SIGNEG TEMP 0**

Raise signal on calculation of negative temperature if set to 1, else don't rais a signal. Otherwise it will be handled through the normal exception method. This is useful when debugging, it will stop the debugger at the location of the calculation of the negative energy. If not 1, it will speed up calculation slightly and generate more useful output upon detection of negative energy. If 1 and not being run in the debugger, it likely won't generate any usefull output upon negative energies, and wil simply abort the program.

6.8.2.6 **#define TRACKMAXSOLVERERROR 0**

Report the error of the linear equation solver if set to 1, else don't. Not tracking the error reduces the calculations per iteration and will speed up running, however if there is question of weather the solver is working accurately this is very handy to turn on.

6.8.2.7 **#define VISCOUS_ENERGY_EQ 1**

If 1 will include viscosity in the energy equation. If 0 it won't. This normally should be set to 1

6.9 main.cpp File Reference

```
#include <mpi.h>
#include <sstream>
#include <string>
#include <fstream>
#include <cmath>
#include <vector>
#include <algorithm>
#include <iomanip>
#include <csignal>
#include <fcntl.h>
#include "main.h"
#include "global.h"
#include "watchzone.h"
#include "exception2.h"
#include "xmlParser.h"
#include "xmlFunctions.h"
#include "dataManipulation.h"
#include "dataMonitoring.h"
#include "physEquations.h"
```

Functions

- [int main](#) (int argc, char *argv[])
- [void signalHandler](#) (int nSig)

6.9.1 Detailed Description

This file contains the main function which is the driver for SPHERLS.

6.9.2 Function Documentation

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

Main driving function of SPHERLS.

Parameters:

- ← *argc* number of arguments passed from the command line
- ← *argv* array of character strings of size *argc* containing the arguments from the command line.

The flow of this function is as follows:

- Initialize program by calling `init()`
- Set function pointers by calling `setMainFunctions()`
- Update new grid with old grid by calling `updateNewGridWithOld()`
- Update boundaries of local grids
- Calculate the first time step by calling `Functions::fpCalculateDeltat()`
- Enter while loop until end time (`Time::dEndTime`) is reached, and for each iteration of the loop:
 - Test to see if a model dump is needed (by checking `Output::bDump` and `Output::nDump-Frequency`), if so dump one by calling `modelWrite()`
 - Write out information for any watchzones present by calling `writeWatchZones()`
 - Write out information for peak kinetic energy per period by calling `writePeakKE()`
 - calculate time step by calling function pointed to by `Functions::fpCalculateDeltat`
- Calculate new velocities by calling the function pointed to by `Functions::fpCalculateNewVelocities()`
- Update velocities on new grid boundaries between processors by calling `updateLocalBoundariesNewGrid()` three times indicating the r -velocity (U), θ -velocity (V) and the ϕ -velocity (W).
- Calculate new grid velocities with `Functions::fpCalculateNewGridVelocities()`.
- Calculate new radii with `Functions::fpCalculateNewRadii()`.
- Update radii on new grid boundaries between processors by calling `updateLocalBoundariesNewGrid()` indicating radius is to be updated (R).
- Calculate new densities with `Functions::fpCalculateNewDensities()`
- Calculate new energies with `Functions::fpCalculateNewEnergies()`
- Update the old grid boundaries and centers by calling `updateLocalBoundaries()`
- Calculating the next time step with `Functions::fpCalculateDeltat()`

Finish by dumping the last model computed

6.9.2.2 void signalHandler (int nSig)

Used for catching signals.

6.10 main.h File Reference

Functions

- void [signalHandler](#) (int nSig)
- int [main](#) (int argc, char *argv[])

6.10.1 Detailed Description

Header file for [main.cpp](#)

6.10.2 Function Documentation

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

Main driving function of SPHERLS.

Parameters:

- ← **argc** number of arguments passed from the command line
- ← **argv** array of character strings of size argc containing the arguments from the command line.

The flow of this function is as follows:

- Initialize program by calling [init\(\)](#)
- Set function pointers by calling [setMainFunctions\(\)](#)
- Update new grid with old grid by calling [updateNewGridWithOld\(\)](#)
- Update boundaries of local grids
- Calculate the first time step by calling [Functions::fpCalculateDeltat\(\)](#)
- Enter while loop until end time ([Time::dEndTime](#)) is reached, and for each iteration of the loop:
 - Test to see if a model dump is needed (by checking [Output::bDump](#) and [Output::nDumpFrequency](#)), if so dump one by calling [modelWrite\(\)](#)
 - Write out information for any watchzones present by calling [writeWatchZones\(\)](#)
 - Write out information for peak kinetic energy per period by calling [writePeakKE\(\)](#)
 - calculate time step by calling function pointed to by [Functions::fpCalculateDeltat](#)
- Calculate new velocities by calling the function pointed to by [Functions::fpCalculateNewVelocities\(\)](#)
- Update velocities on new grid boundaries between processors by calling [updateLocalBoundariesNewGrid\(\)](#) three times indicating the r -velocity (U), θ -velocity (V) and the ϕ -velocity (W).

- Calculate new grid velocities with [Functions::fpCalculateNewGridVelocities\(\)](#).
- Calculate new radii with [Functions::fpCalculateNewRadii\(\)](#).
- Update radii on new grid boundaries between processors by calling [updateLocalBoundariesNewGrid\(\)](#) indicating radius is to be updated (R).
- Calculate new densities with [Functions::fpCalculateNewDensities\(\)](#)
- Calculate new energies with [Functions::fpCalculateNewEnergies\(\)](#)
- Update the old grid boundaries and centers by calling [updateLocalBoundaries\(\)](#)
- Calculating the next time step with [Functions::fpCalculateDeltat\(\)](#)

Finish by dumping the last model computed

6.10.2.2 void signalHandler (int *nSig*)

Used for catching signals.

6.11 physEquations.cpp File Reference

```
#include <cmath>
#include <sstream>
#include <signal.h>
#include "exception2.h"
#include "physEquations.h"
#include "dataManipulation.h"
#include "dataMonitoring.h"
#include "global.h"
#include <limits>
```

Functions

- void [setMainFunctions](#) ([Functions](#) &functions, [ProcTop](#) &procTop, [Parameters](#) ¶meters, [Grid](#) &grid, [Time](#) &time, [Implicit](#) &implicit)
- void [setInternalVarInf](#) ([Grid](#) &grid, [Parameters](#) ¶meters)
- void [initInternalVars](#) ([Grid](#) &grid, [ProcTop](#) &procTop, [Parameters](#) ¶meters)
- void [calNewVelocities_R](#) ([Grid](#) &grid, [Parameters](#) ¶meters, [Time](#) &time, [ProcTop](#) &procTop)
- void [calNewVelocities_R_LES](#) ([Grid](#) &grid, [Parameters](#) ¶meters, [Time](#) &time, [ProcTop](#) &procTop)
- void [calNewVelocities_RT](#) ([Grid](#) &grid, [Parameters](#) ¶meters, [Time](#) &time, [ProcTop](#) &procTop)
- void [calNewVelocities_RT_LES](#) ([Grid](#) &grid, [Parameters](#) ¶meters, [Time](#) &time, [ProcTop](#) &procTop)
- void [calNewVelocities_RTP](#) ([Grid](#) &grid, [Parameters](#) ¶meters, [Time](#) &time, [ProcTop](#) &procTop)
- void [calNewVelocities_RTP_LES](#) ([Grid](#) &grid, [Parameters](#) ¶meters, [Time](#) &time, [ProcTop](#) &procTop)
- void [calNewU_R](#) ([Grid](#) &grid, [Parameters](#) ¶meters, [Time](#) &time, [ProcTop](#) &procTop)
- void [calNewU_R_LES](#) ([Grid](#) &grid, [Parameters](#) ¶meters, [Time](#) &time, [ProcTop](#) &procTop)
- void [calNewU_RT](#) ([Grid](#) &grid, [Parameters](#) ¶meters, [Time](#) &time, [ProcTop](#) &procTop)
- void [calNewU_RT_LES](#) ([Grid](#) &grid, [Parameters](#) ¶meters, [Time](#) &time, [ProcTop](#) &procTop)
- void [calNewU_RTP](#) ([Grid](#) &grid, [Parameters](#) ¶meters, [Time](#) &time, [ProcTop](#) &procTop)
- void [calNewU_RTP_LES](#) ([Grid](#) &grid, [Parameters](#) ¶meters, [Time](#) &time, [ProcTop](#) &procTop)
- void [calNewV_RT](#) ([Grid](#) &grid, [Parameters](#) ¶meters, [Time](#) &time, [ProcTop](#) &procTop)
- void [calNewV_RT_LES](#) ([Grid](#) &grid, [Parameters](#) ¶meters, [Time](#) &time, [ProcTop](#) &procTop)
- void [calNewV_RTP](#) ([Grid](#) &grid, [Parameters](#) ¶meters, [Time](#) &time, [ProcTop](#) &procTop)
- void [calNewV_RTP_LES](#) ([Grid](#) &grid, [Parameters](#) ¶meters, [Time](#) &time, [ProcTop](#) &procTop)
- void [calNewW_RTP](#) ([Grid](#) &grid, [Parameters](#) ¶meters, [Time](#) &time, [ProcTop](#) &procTop)
- void [calNewW_RTP_LES](#) ([Grid](#) &grid, [Parameters](#) ¶meters, [Time](#) &time, [ProcTop](#) &procTop)
- void [calNewU0_R](#) ([Grid](#) &grid, [Parameters](#) ¶meters, [Time](#) &time, [ProcTop](#) &procTop, [MessPass](#) &messPass)
- void [calNewU0_RT](#) ([Grid](#) &grid, [Parameters](#) ¶meters, [Time](#) &time, [ProcTop](#) &procTop, [MessPass](#) &messPass)

- void `calNewU0_RTP` (`Grid` &grid, `Parameters` ¶meters, `Time` &time, `ProcTop` &procTop, `MessPass` &messPass)
- void `calNewR` (`Grid` &grid, `Time` &time)
- void `calNewD_R` (`Grid` &grid, `Parameters` ¶meters, `Time` &time, `ProcTop` &procTop)
- void `calNewD_RT` (`Grid` &grid, `Parameters` ¶meters, `Time` &time, `ProcTop` &procTop)
- void `calNewD_RTP` (`Grid` &grid, `Parameters` ¶meters, `Time` &time, `ProcTop` &procTop)
- void `calNewE_R_AD` (`Grid` &grid, `Parameters` ¶meters, `Time` &time, `ProcTop` &procTop)
- void `calNewE_R_NA` (`Grid` &grid, `Parameters` ¶meters, `Time` &time, `ProcTop` &procTop)
- void `calNewE_R_NA_LES` (`Grid` &grid, `Parameters` ¶meters, `Time` &time, `ProcTop` &procTop)
- void `calNewE_RT_AD` (`Grid` &grid, `Parameters` ¶meters, `Time` &time, `ProcTop` &procTop)
- void `calNewE_RT_NA` (`Grid` &grid, `Parameters` ¶meters, `Time` &time, `ProcTop` &procTop)
- void `calNewE_RT_NA_LES` (`Grid` &grid, `Parameters` ¶meters, `Time` &time, `ProcTop` &procTop)
- void `calNewE_RTP_AD` (`Grid` &grid, `Parameters` ¶meters, `Time` &time, `ProcTop` &procTop)
- void `calNewE_RTP_NA` (`Grid` &grid, `Parameters` ¶meters, `Time` &time, `ProcTop` &procTop)
- void `calNewE_RTP_NA_LES` (`Grid` &grid, `Parameters` ¶meters, `Time` &time, `ProcTop` &procTop)
- void `calNewDenave_None` (`Grid` &grid)
- void `calNewDenave_R` (`Grid` &grid)
- void `calNewDenave_RT` (`Grid` &grid)
- void `calNewDenave_RTP` (`Grid` &grid)
- void `calNewP_GL` (`Grid` &grid, `Parameters` ¶meters)
- void `calNewTPKappaGamma_TEOS` (`Grid` &grid, `Parameters` ¶meters)
- void `calNewPEKappaGamma_TEOS` (`Grid` &grid, `Parameters` ¶meters)
- void `calNewQ0_R_TEOS` (`Grid` &grid, `Parameters` ¶meters)
- void `calNewQ0_R_GL` (`Grid` &grid, `Parameters` ¶meters)
- void `calNewQ0Q1_RT_TEOS` (`Grid` &grid, `Parameters` ¶meters)
- void `calNewQ0Q1_RT_GL` (`Grid` &grid, `Parameters` ¶meters)
- void `calNewQ0Q1Q2_RTP_TEOS` (`Grid` &grid, `Parameters` ¶meters)
- void `calNewQ0Q1Q2_RTP_GL` (`Grid` &grid, `Parameters` ¶meters)
- void `calNewEddyVisc_None` (`Grid` &grid, `Parameters` ¶meters)
- void `calNewEddyVisc_R_CN` (`Grid` &grid, `Parameters` ¶meters)
- void `calNewEddyVisc_RT_CN` (`Grid` &grid, `Parameters` ¶meters)
- void `calNewEddyVisc_RTP_CN` (`Grid` &grid, `Parameters` ¶meters)
- void `calNewEddyVisc_R_SM` (`Grid` &grid, `Parameters` ¶meters)
- void `calNewEddyVisc_RT_SM` (`Grid` &grid, `Parameters` ¶meters)
- void `calNewEddyVisc_RTP_SM` (`Grid` &grid, `Parameters` ¶meters)
- void `calOldDenave_None` (`Grid` &grid)
- void `calOldDenave_R` (`Grid` &grid)
- void `calOldDenave_RT` (`Grid` &grid)
- void `calOldDenave_RTP` (`Grid` &grid)
- void `calOldP_GL` (`Grid` &grid, `Parameters` ¶meters)
- void `calOldPEKappaGamma_TEOS` (`Grid` &grid, `Parameters` ¶meters)
- void `calOldQ0_R_GL` (`Grid` &grid, `Parameters` ¶meters)
- void `calOldQ0_R_TEOS` (`Grid` &grid, `Parameters` ¶meters)
- void `calOldQ0Q1_RT_GL` (`Grid` &grid, `Parameters` ¶meters)
- void `calOldQ0Q1_RT_TEOS` (`Grid` &grid, `Parameters` ¶meters)
- void `calOldQ0Q1Q2_RTP_GL` (`Grid` &grid, `Parameters` ¶meters)
- void `calOldQ0Q1Q2_RTP_TEOS` (`Grid` &grid, `Parameters` ¶meters)
- void `calOldEddyVisc_R_CN` (`Grid` &grid, `Parameters` ¶meters)

- void `calOldEddyVisc_RT_CN` (Grid &grid, Parameters ¶meters)
- void `calOldEddyVisc_RTP_CN` (Grid &grid, Parameters ¶meters)
- void `calOldEddyVisc_R_SM` (Grid &grid, Parameters ¶meters)
- void `calOldEddyVisc_RT_SM` (Grid &grid, Parameters ¶meters)
- void `calOldEddyVisc_RTP_SM` (Grid &grid, Parameters ¶meters)
- void `calDelt_R_GL` (Grid &grid, Parameters ¶meters, Time &time, ProcTop &procTop)
- void `calDelt_R_TEOS` (Grid &grid, Parameters ¶meters, Time &time, ProcTop &procTop)
- void `calDelt_RT_GL` (Grid &grid, Parameters ¶meters, Time &time, ProcTop &procTop)
- void `calDelt_RT_TEOS` (Grid &grid, Parameters ¶meters, Time &time, ProcTop &procTop)
- void `calDelt_RTP_GL` (Grid &grid, Parameters ¶meters, Time &time, ProcTop &procTop)
- void `calDelt_RTP_TEOS` (Grid &grid, Parameters ¶meters, Time &time, ProcTop &procTop)
- void `calDelt_CONST` (Grid &grid, Parameters ¶meters, Time &time, ProcTop &procTop)
- void `implicitSolve_None` (Grid &grid, Implicit &implicit, Parameters ¶meters, Time &time, ProcTop &procTop, MessPass &messPass, Functions &functions)
- void `implicitSolve_R` (Grid &grid, Implicit &implicit, Parameters ¶meters, Time &time, ProcTop &procTop, MessPass &messPass, Functions &functions)
- void `implicitSolve_RT` (Grid &grid, Implicit &implicit, Parameters ¶meters, Time &time, ProcTop &procTop, MessPass &messPass, Functions &functions)
- void `implicitSolve_RTP` (Grid &grid, Implicit &implicit, Parameters ¶meters, Time &time, ProcTop &procTop, MessPass &messPass, Functions &functions)
- double `dImplicitEnergyFunction_None` (Grid &grid, Parameters ¶meters, Time &time, double dTemps[], int i, int j, int k)
- double `dImplicitEnergyFunction_R` (Grid &grid, Parameters ¶meters, Time &time, double dTemps[], int i, int j, int k)
- double `dImplicitEnergyFunction_R_SB` (Grid &grid, Parameters ¶meters, Time &time, double dTemps[], int i, int j, int k)
- double `dImplicitEnergyFunction_RT` (Grid &grid, Parameters ¶meters, Time &time, double dTemps[], int i, int j, int k)
- double `dImplicitEnergyFunction_RT_SB` (Grid &grid, Parameters ¶meters, Time &time, double dTemps[], int i, int j, int k)
- double `dImplicitEnergyFunction_RTP` (Grid &grid, Parameters ¶meters, Time &time, double dTemps[], int i, int j, int k)
- double `dImplicitEnergyFunction_RTP_SB` (Grid &grid, Parameters ¶meters, Time &time, double dTemps[], int i, int j, int k)
- double `dImplicitEnergyFunction_R_LES` (Grid &grid, Parameters ¶meters, Time &time, double dTemps[], int i, int j, int k)
- double `dImplicitEnergyFunction_R_LES_SB` (Grid &grid, Parameters ¶meters, Time &time, double dTemps[], int i, int j, int k)
- double `dImplicitEnergyFunction_RT_LES` (Grid &grid, Parameters ¶meters, Time &time, double dTemps[], int i, int j, int k)
- double `dImplicitEnergyFunction_RT_LES_SB` (Grid &grid, Parameters ¶meters, Time &time, double dTemps[], int i, int j, int k)
- double `dImplicitEnergyFunction_RTP_LES` (Grid &grid, Parameters ¶meters, Time &time, double dTemps[], int i, int j, int k)
- double `dImplicitEnergyFunction_RTP_LES_SB` (Grid &grid, Parameters ¶meters, Time &time, double dTemps[], int i, int j, int k)
- double `dEOS_GL` (double dRho, double dE, Parameters parameters)
- void `initDonorFracAndMaxConVel_R_GL` (Grid &grid, Parameters ¶meters)
- void `initDonorFracAndMaxConVel_R_TEOS` (Grid &grid, Parameters ¶meters)
- void `initDonorFracAndMaxConVel_RT_GL` (Grid &grid, Parameters ¶meters)
- void `initDonorFracAndMaxConVel_RT_TEOS` (Grid &grid, Parameters ¶meters)
- void `initDonorFracAndMaxConVel_RTP_GL` (Grid &grid, Parameters ¶meters)
- void `initDonorFracAndMaxConVel_RTP_TEOS` (Grid &grid, Parameters ¶meters)

6.11.1 Detailed Description

This file is used to specify the functions which contain physics. This includes conservation equations, equation of state, etc.. It also sets function pointers for these functions, so that `main()` will know which functions to call. This implementation also allows the functions called to calculate, for example new densities, to be different depending on the processor. This allows one processor to handle the 1D region and other processors to handle a 3D region.

6.11.2 Function Documentation

6.11.2.1 `void calDelt_CONST (Grid & grid, Parameters & parameters, Time & time, ProcTop & procTop)`

This function is used when a constant tie step is desired.

6.11.2.2 `void calDelt_R_GL (Grid & grid, Parameters & parameters, Time & time, ProcTop & procTop)`

This function calculates the time step by considering the sound crossing time in the radial direction only and is compatible with a gamma law gas EOS.

Parameters:

- ← *grid* contains the local grid, and will hold the newly updated densities
- ← *parameters* various parameters needed for the calculation
- ↔ *time* contains time information, e.g. time step, current time etc.
- ← *procTop* contains information about the processor topology. This function uses `ProcTop::nRank` to pass messages.

6.11.2.3 `void calDelt_R_TEOS (Grid & grid, Parameters & parameters, Time & time, ProcTop & procTop)`

This function calculates the time step by considering the sound crossing time in the radial direction only and is compatible with a tabulated EOS.

Parameters:

- ← *grid* contains the local grid, and will hold the newly updated densities
- ← *parameters* various parameters needed for the calculation
- ↔ *time* contains time information, e.g. time step, current time etc.
- ← *procTop* contains information about the processor topology. This function uses `ProcTop::nRank` to pass messages.

6.11.2.4 void calDelt_RT_GL (**Grid** & *grid*, **Parameters** & *parameters*, **Time** & *time*, **ProcTop** & *procTop*)

This function calculates the time step by considering the sound crossing time in the radial and theta directions only and is compatible with a gamma law gas EOS.

Parameters:

- ← *grid* contains the local grid, and will hold the newly updated densities
- ← *parameters* various parameters needed for the calculation
- ↔ *time* contains time information, e.g. time step, current time etc.
- ← *procTop* contains information about the processor topology. This function uses [ProcTop::nRank](#) to pass messages.

6.11.2.5 void calDelt_RT_TEOS (**Grid** & *grid*, **Parameters** & *parameters*, **Time** & *time*, **ProcTop** & *procTop*)

This function calculates the time step by considering the sound crossing time in the radial and theta directions and is compatible with a tabulated EOS.

Parameters:

- ← *grid* contains the local grid, and will hold the newly updated densities
- ← *parameters* various parameters needed for the calculation
- ↔ *time* contains time information, e.g. time step, current time etc.
- ← *procTop* contains information about the processor topology. This function uses [ProcTop::nRank](#) to pass messages.

6.11.2.6 void calDelt_RTP_GL (**Grid** & *grid*, **Parameters** & *parameters*, **Time** & *time*, **ProcTop** & *procTop*)

This function calculates the time step by considering the sound crossing time in the radial, theta and phi directions only and is compatible with a gamma law gas EOS.

Parameters:

- ← *grid* contains the local grid, and will hold the newly updated densities
- ← *parameters* various parameters needed for the calculation
- ↔ *time* contains time information, e.g. time step, current time etc.
- ← *procTop* contains information about the processor topology. This function uses [ProcTop::nRank](#) to pass messages.

6.11.2.7 void calDelt_RTP_TEOS ([Grid](#) & *grid*, [Parameters](#) & *parameters*, [Time](#) & *time*, [ProcTop](#) & *procTop*)

This function calculates the time step by considering the sound crossing time in the radial, theta and phi directions and is compatible with a tabulated EOS.

Parameters:

- ← *grid* contains the local grid, and will hold the newly updated densities
- ← *parameters* various parameters needed for the calculation
- ↔ *time* contains time information, e.g. time step, current time etc.
- ← *procTop* contains information about the processor topology. This function uses [ProcTop::nRank](#) to pass messages.

6.11.2.8 void calNewD_R ([Grid](#) & *grid*, [Parameters](#) & *parameters*, [Time](#) & *time*, [ProcTop](#) & *procTop*)

This function calculates new densities using terms in the radial direction only

Parameters:

- ↔ *grid* contains the local grid, and will hold the newly updated densities
- ← *parameters* various parameters needed for the calculation
- ← *time* contains time information, e.g. time step, current time etc.
- ← *procTop* contains information about the processor topology, uses [ProcTop::nRank](#) when reporting negative densities

6.11.2.9 void calNewD_RT ([Grid](#) & *grid*, [Parameters](#) & *parameters*, [Time](#) & *time*, [ProcTop](#) & *procTop*)

Boundary Conditions

doesn't include flux through outer interface

6.11.2.10 void calNewD_RTP ([Grid](#) & *grid*, [Parameters](#) & *parameters*, [Time](#) & *time*, [ProcTop](#) & *procTop*)

Boundary Conditions

doesn't allow mass flux through outer interface

6.11.2.11 void calNewDenave_None (**Grid** & *grid*)

This function is a dummy function, and doesn't do anything. In the case of a 1D calculation the average density is undefined, and only the density is used. This is different from the case where the 1D region exists on the rank 0 processor, but the grid as a whole is really 2D or 3D. In which case [calNewDenave_R](#) should be used instead.

Parameters:

↔ *grid*

6.11.2.12 void calNewDenave_R (**Grid** & *grid*)

This function calculates the horizontal average density in a 3\1D region. This really just copies the density from the particular radial zone into the averaged density variable. This way it can be used exactly the same way in the 1D region as it is in the 3D region. This is done using the density in the new grid, and places the result into the new grid.

Parameters:

↔ *grid* supplies the information needed to calculate the horizontal density average, it also stores the calculated horizontally averaged density.

6.11.2.13 void calNewDenave_RT (**Grid** & *grid*)

This function calculates the horizontal average density in a 2D region from the new grid density and stores the result in the new grid.

Parameters:

↔ *grid* supplies the information needed to calculate the horizontal density average, it also stores the calculated horizontally averaged density.

6.11.2.14 void calNewDenave_RTP (**Grid** & *grid*)

This function calculates the horizontal average density in a 3D region from the new grid density and stores the result in the new grid.

Parameters:

↔ *grid* supplies the information needed to calculate the horizontal density average, it also stores the calculated horizontally averaged density.

6.11.2.15 void calNewE_R_AD (**Grid** & *grid*, **Parameters** & *parameters*, **Time** & *time*, **ProcTop** & *procTop*)

This function calculates new adiabatic energies using terms in the radial direction.

Parameters:

- ↔ *grid* contains the local grid, and will hold the newly updated densities
- ← *parameters* various parameters needed for the calculation
- ← *time* contains time information, e.g. time step, current time etc.
- ← *procTop*

6.11.2.16 void calNewE_R_NA (**Grid** & *grid*, **Parameters** & *parameters*, **Time** & *time*, **ProcTop** & *procTop*)

Boundary Conditions

Missing `grid.dLocalGridOld[grid.nE][i+1][j][k]` in calculation of $E_{i+1/2,j,k}$ setting it equal to value at *i*.
`grid.dLocalGridOld[grid.nDM][i+1][0][0]` and `grid.dLocalGridOld[grid.nE][i+1][j][k]` missing in the calculation of upwind gradient in `dA1`. Using the centered gradient instead.
 Missing `grid.dLocalGridOld[grid.nT][i+1][0][0]`

6.11.2.17 void calNewE_R_NA_LES (**Grid** & *grid*, **Parameters** & *parameters*, **Time** & *time*, **ProcTop** & *procTop*)

Boundary Conditions

Missing `grid.dLocalGridOld[grid.nE][i+1][j][k]` in calculation of $E_{i+1/2,j,k}$ setting it equal to value at *i*.
`grid.dLocalGridOld[grid.nDM][i+1][0][0]` and `grid.dLocalGridOld[grid.nE][i+1][j][k]` missing in the calculation of upwind gradient in `dA1`. Using the centered gradient instead.
 Missing `grid.dLocalGridOld[grid.nT][i+1][0][0]`

6.11.2.18 void calNewE_RT_AD (**Grid** & *grid*, **Parameters** & *parameters*, **Time** & *time*, **ProcTop** & *procTop*)

Boundary Conditions

`grid.dLocalGridOld[grid.nE][i+1][j][k]` is missing
`grid.dLocalGridOld[grid.nDM][i+1][0][0]` and `grid.dLocalGridOld[grid.nE][i+1][j][k]` missing using inner gradient for both

6.11.2.19 void calNewE_RT_NA (Grid & grid, Parameters & parameters, Time & time, ProcTop & procTop)

Boundary Conditions

Missing grid.dLocalGridOld[grid.nE][i+1][j][k] in calculation of $E_{i+1/2,j,k}$ setting it equal to value at i.

grid.dLocalGridOld[grid.nDM][i+1][0][0] and grid.dLocalGridOld[grid.nE][i+1][j][k] missing in the calculation of upwind gradient in dA1. Using the centered gradient instead.

Missing grid.dLocalGridOld[grid.nT][i+1][0][0] using flux equals $2\sigma T^4$ at surface.

6.11.2.20 void calNewE_RT_NA_LES (Grid & grid, Parameters & parameters, Time & time, ProcTop & procTop)

Boundary Conditions

Setting energy at surface equal to energy in last zone.

Missing grid.dLocalGridOld[grid.nT][i+1][0][0] using flux equals $2\sigma T^4$ at surface.

6.11.2.21 void calNewE_RTP_AD (Grid & grid, Parameters & parameters, Time & time, ProcTop & procTop)

Boundary Conditions

Missing grid.dLocalGridOld[grid.nE][i+1][j][k] in calculation of $E_{i+1/2,j,k}$ setting it equal to zero.

grid.dLocalGridOld[grid.nDM][i+1][0][0] and grid.dLocalGridOld[grid.nE][i+1][j][k] missing in the calculation of upwind gradient in dA1. Using the centered gradient.

6.11.2.22 void calNewE_RTP_NA (Grid & grid, Parameters & parameters, Time & time, ProcTop & procTop)

Boundary Conditions

Missing grid.dLocalGridOld[grid.nE][i+1][j][k] in calculation of $E_{i+1/2,j,k}$ setting it equal to the value at i.

grid.dLocalGridOld[grid.nDM][i+1][0][0] and grid.dLocalGridOld[grid.nE][i+1][j][k] missing in the calculation of upwind gradient in dA1. Using the centered gradient instead.

Missing grid.dLocalGridOld[grid.nT][i+1][0][0]

6.11.2.23 void calNewE_RTP_NA_LES (Grid & grid, Parameters & parameters, Time & time, ProcTop & procTop)

Boundary Conditions

Missing W at i+1, assuming the same as at i

Missing grid.dLocalGridOld[grid.nE][i+1][j][k] in calculation of $E_{i+1/2,j,k}$ setting it equal to the value at i.

grid.dLocalGridOld[grid.nDM][i+1][0][0] and grid.dLocalGridOld[grid.nE][i+1][j][k] missing in the calculation of upwind gradient in dA1. Using the centered gradient instead.
Missing grid.dLocalGridOld[grid.nT][i+1][0][0]

6.11.2.24 void calNewEddyVisc_None (**Grid** & *grid*, **Parameters** & *parameters*)

This function is a empty function used as a place holder when no eddy viscosity model is being used.

Parameters:

↔ *grid*

← *parameters*

6.11.2.25 void calNewEddyVisc_R_CN (**Grid** & *grid*, **Parameters** & *parameters*)

This function calculates the eddy viscosity using a constant times the zoning with only the radial terms.

Parameters:

↔ *grid* supplies the input for calculating the eddy viscosity.

← *parameters* contains parameters used in calculating the eddy viscosity.

6.11.2.26 void calNewEddyVisc_R_SM (**Grid** & *grid*, **Parameters** & *parameters*)

This function calculates the eddy viscosity with only the radial terms.

Parameters:

↔ *grid* supplies the input for calculating the eddy viscosity.

← *parameters* contains parameters used in calculating the eddy viscosity.

6.11.2.27 void calNewEddyVisc_RT_CN (**Grid** & *grid*, **Parameters** & *parameters*)

This function calculates the eddy viscosity using a constant times the zoning with only the radial and theta terms.

Parameters:

↔ *grid* supplies the input for calculating the eddy viscosity.

← *parameters* contains parameters used in calculating the eddy viscosity.

6.11.2.28 void calNewEddyVisc_RT_SM (**Grid** & *grid*, **Parameters** & *parameters*)

This function calculates the eddy viscosity with only the radial and theta terms.

Parameters:

- ↔ *grid* supplies the input for calculating the eddy viscosity.
- ← *parameters* contains parameters used in calculating the eddy viscosity.

6.11.2.29 void calNewEddyVisc_RTP_CN (**Grid** & *grid*, **Parameters** & *parameters*)

This function calculates the eddy viscosity using a constant times the zoning with only the radial, theta, and phi terms.

Parameters:

- ↔ *grid* supplies the input for calculating the eddy viscosity.
- ← *parameters* contains parameters used in calculating the eddy viscosity.

6.11.2.30 void calNewEddyVisc_RTP_SM (**Grid** & *grid*, **Parameters** & *parameters*)

Boundary Conditions

- assuming that theta velocity is constant across surface
- assume phi velocity is constant across surface

6.11.2.31 void calNewP_GL (**Grid** & *grid*, **Parameters** & *parameters*)

This function calculates the pressure. It is calculated using the new values of quantities and places the result in the new grid. It uses a gamma law gas give in [dEOS_GL](#) to calculate the pressure.

Parameters:

- ↔ *grid* supplies the input for calculating the pressure and also accepts the result of the pressure calculations.
- ← *parameters* contains parameters used in calculating the pressure, namely the adiabatic gamma that is used.

6.11.2.32 void calNewPEKappaGamma_TEOS (**Grid** & *grid*, **Parameters** & *parameters*)

This function calculates the Energy, pressure and opacity of a cell. It calculates it using the new vaules of quantities and places the result in the new grid.

Parameters:

- ↔ *grid* supplies the input for calculating the pressure and also accepts the result of the pressure calculation

← *parameters* contains parameters used in calculating the pressure.

6.11.2.33 void calNewQ0_R_GL (*Grid* & *grid*, *Parameters* & *parameters*)

This function calculates the artificial viscosity of a cell. It calculates it using the new values of quantities and places the result in the new grid. It does this for the radial component of the viscosity only. It uses the sound speed derived from the adiabatic gamma given for the gamma law gas equation of state.

Parameters:

- ↔ *grid* supplies the input for calculating the artificial viscosity and also accepts the result of the artificial viscosity calculation.
- ← *parameters* contains parameters used when calculating the artificial viscosity, namely the adiabatic gamma.

6.11.2.34 void calNewQ0_R_TEOS (*Grid* & *grid*, *Parameters* & *parameters*)

This function calculates the artificial viscosity of a cell. It calculates it using the old values of quantities and places the result in the old grid. It does this for the radial component of the viscosity only. It uses a sound speed derived from a tabulated equation of state for the calculation.

Parameters:

- ↔ *grid* supplies the input for calculating the artificial viscosity and also accepts the result of the artificial viscosity calculation
- ← *parameters* contains parameters used in calculating the artificial viscosity.

6.11.2.35 void calNewQ0Q1_RT_GL (*Grid* & *grid*, *Parameters* & *parameters*)

This function calculates the artificial viscosity of a cell. It calculates it using the new values of quantities and places the result in the new grid. It does this for the radial and theta components of the viscosity. It uses the sound speed derived from the adiabatic gamma given for the gamma law gas equation of state.

Parameters:

- ↔ *grid* supplies the input for calculating the artificial viscosity and also accepts the result of the artificial viscosity calculations.
- ← *parameters* contains parameters used when calculating the artificial viscosity, namely the adiabatic gamma.

6.11.2.36 void calNewQ0Q1_RT_TEOS (*Grid* & *grid*, *Parameters* & *parameters*)

This function calculates the artificial viscosity of a cell. It calculates it using the old values of quantities and places the result in the old grid. It does this for two component of the viscosity.

Parameters:

- ↔ *grid* supplies the input for calculating the artificial viscosity and also accepts the result of the artificial viscosity calculation
- ← *parameters* contains parameters used in calculating the artificial viscosity.

6.11.2.37 void calNewQ0Q1Q2_RTP_GL (Grid & grid, Parameters & parameters)

This function calculates the artificial viscosity of a cell. It calculates it using the new values of quantities and places the result in the new grid. It does this for the radial, theta, and phi componenets of the viscosity. It uses the sound speed derived from the adiabatic gamma given for the gamma law gas equation of state.

Parameters:

- ↔ *grid* supplies the input for calculating the artificial viscosity and also accepts the result of the artificial viscosity calculations.
- ← *parameters* contains parameters used when calculating the artificial viscosity, namely the adiabatic gamma.

6.11.2.38 void calNewQ0Q1Q2_RTP_TEOS (Grid & grid, Parameters & parameters)

This function calculates the artificial viscosity of a cell. It calculates it using the old values of quantities and places the result in the old grid. It does this for the three component of the viscosity.

Parameters:

- ↔ *grid* supplies the input for calculating the artificial viscosity and also accepts the result of the artificial viscosity calculation
- ← *parameters* contains parameters used in calculating the artificial viscosity.

6.11.2.39 void calNewR (Grid & grid, Time & time)

This function calculates the radii, from the new radial grid velocities

Parameters:

- ↔ *grid* contains the local grid, and will hold the newly updated radial velocities
- ← *time* contains time information, e.g. time step, current time etc.

6.11.2.40 void calNewTPKappaGamma_TEOS (Grid & grid, Parameters & parameters)

This function calculates the Temperature, pressure and opacity of a cell. It calculates it using the new vaules of quantities and places the result in the new grid.

Parameters:

- ↔ *grid* supplies the input for calculating the pressure and also accepts the result of the pressure calculation
- ← *parameters* contains parameters used in calculating the pressure.

6.11.2.41 void calNewU0_R (**Grid** & *grid*, **Parameters** & *parameters*, **Time** & *time*, **ProcTop** & *procTop*, **MessPass** & *messPass*)

Todo

At some point I will likely want to make this function compatible with a 3D domain decomposition instead of a purely radial domain decomposition.

6.11.2.42 void calNewU0_RT (**Grid** & *grid*, **Parameters** & *parameters*, **Time** & *time*, **ProcTop** & *procTop*, **MessPass** & *messPass*)

Todo

At some point I will likely want to make this function compatible with a 3D domain decomposition instead of a purely radial domain decomposition.

Boundary Conditions

grid.dLocalGridOld[grid.nD][i+1][j][k] is missing

6.11.2.43 void calNewU0_RTP (**Grid** & *grid*, **Parameters** & *parameters*, **Time** & *time*, **ProcTop** & *procTop*, **MessPass** & *messPass*)

Todo

At some point I will likely want to make this function compatible with a 3D domain decomposition instead of a purely radial domain decomposition.

Boundary Conditions

grid.dLocalGridOld[grid.nD][i+1][j][k] is missing

6.11.2.44 void calNewU_R (**Grid** & *grid*, **Parameters** & *parameters*, **Time** & *time*, **ProcTop** & *procTop*)

Boundary Conditions

Missing grid.dLocalGridOld[grid.nD][nICen+1][j][k] in calculation of $\rho_{i+1/2}$, setting it to 0.0
 Missing grid.dLocalGridOld[grid.nP][nICen+1][j][k] in calculation of S_1 , setting it to -1.0*grid.dLocalGridOld[grid.nP][nICen][j][k].
 Missing grid.dLocalGridOld[grid.nDM][nICen+1][0][0] in calculation of centered A_1 gradient, setting it to zero.
 Missing grid.dLocalGridOld[grid.nU][i+1][j][k] and grid.dLocalGridOld[grid.nDM][nICen+1][0][0] in calculation of upwind gradient, when moving inward. Using centered gradient instead.

6.11.2.45 void calNewU_R_LES (**Grid** & *grid*, **Parameters** & *parameters*, **Time** & *time*, **ProcTop** & *procTop*)

Boundary Conditions

Missing `grid.dLocalGridOld[grid.nD][nICen+1][j][k]` in calculation of $\rho_{i+1/2}$, setting it to 0.0
 Missing `grid.dLocalGridOld[grid.nU][i+1][j][k]` using velocity at *i*
 Assuming eddy viscosity outside model is zero.
 Missing `grid.dLocalGridOld[grid.nP][nICen+1][j][k]` in calculation of S_1 , setting it to $-1.0 * \text{grid.dLocalGridOld[grid.nP][nICen][j][k]}$.
 Missing `grid.dLocalGridOld[grid.nDM][nICen+1][0][0]` in calculation of centered A_1 gradient, setting it to zero.
 Missing `grid.dLocalGridOld[grid.nU][i+1][j][k]` and `grid.dLocalGridOld[grid.nDM][nICen+1][0][0]` in calculation of upwind gradient, when moving inward. Using centered gradient instead.

6.11.2.46 void calNewU_RT (**Grid** & *grid*, **Parameters** & *parameters*, **Time** & *time*, **ProcTop** & *procTop*)

Boundary Conditions

Missing `grid.dLocalGridOld[grid.nD][nICen+1][j][k]` in calculation of $\rho_{i+1/2,j,k}$, setting it to zero.
 assuming theta velocity is constant across surface
 Missing `grid.dLocalGridOld[grid.nDenAve][nICen+1][0][0]` in calculation of $\langle \rho \rangle_{i+1/2}$, setting it to zero.
 Missing `grid.dLocalGridOld[grid.nP][nICen+1][j][k]` in calculation of S_1 , setting it to $-1.0 * \text{dP_ijk_n}$.
 Missing `grid.dLocalGridOld[grid.nDM][nICen+1][0][0]` in calculation of centered A_1 gradient, setting it to zero.
 Missing `grid.dLocalGridOld[grid.nU][i+1][j][k]` and `grid.dLocalGridOld[grid.nDM][nICen+1][0][0]` in calculation of upwind gradient, when moving inward. Using centered gradient instead.
 Missing `grid.dLocalGridOld[grid.nDM][i+1][0][0]` in calculation of S_1 using `Parameters::dAlpha * grid.dLocalGridOld[grid.nDM][nICen][0][0]` instead.

6.11.2.47 void calNewU_RT_LES (**Grid** & *grid*, **Parameters** & *parameters*, **Time** & *time*, **ProcTop** & *procTop*)

Boundary Conditions

Missing `grid.dLocalGridOld[grid.nDM][i+1][0][0]` in calculation of S_1 using `Parameters::dAlpha * grid.dLocalGridOld[grid.nDM][nICen][0][0]` instead.
 Missing density outside of surface, setting it to zero.
 Assuming theta velocities are constant across surface.
 Missing density outside model, setting it to zero.
 Missing pressure outside surface setting it equal to negative pressure in the center of the first cell so that it will be zero at surface.
 eddy viscosity outside the star is zero.
 Missing mass outside model, setting it to zero.
 Missing `grid.dLocalGridOld[grid.nU][i+1][j][k]` and `grid.dLocalGridOld[grid.nDM][nICen+1][0][0]` in calculation of upwind gradient, when moving inward. Using centered gradient instead.

6.11.2.48 void calNewU_RTP (**Grid** & *grid*, **Parameters** & *parameters*, **Time** & *time*, **ProcTop** & *procTop*)

Boundary Conditions

missing grid.dLocalGridOld[grid.nU][i+1][j][k] in calculation of $u_{i+1,j,k}$ setting $u_{i+1,j,k}=u_{i+1/2,j,k}$.

Missing grid.dLocalGridOld[grid.nD][i+1][j][k] in calculation of $\rho_{i+1/2,j,k}$, setting it to zero.

assuming theta velocity is constant across the surface.

assuming phi velocity is constant across the surface.

Missing grid.dLocalGridOld[grid.nDenAve][nICen+1][0][0] in calculation of $\langle \rho \rangle_{i+1/2}$ setting it to zero.

Missing grid.dLocalGridOld[grid.nDM][nICen+1][0][0] in calculation of centered A_1 gradient, setting it equal to **Parameters::dAlpha** grid.dLocalGridOld[grid.nDM][nICen][0][0].

Missing grid.dLocalGridOld[grid.nU][i+1][j][k] and grid.dLocalGridOld[grid.nDM][nICen+1][0][0] in calculation of upwind gradient, when moving inward. Using centered gradient instead.

Missing grid.dLocalGridOld[grid.nDM][i+1][0][0] in calculation of S_1 using **Parameters::dAlpha** *grid.dLocalGridOld[grid.nDM][nICen][0][0] instead.

6.11.2.49 void calNewU_RTP_LES (**Grid** & *grid*, **Parameters** & *parameters*, **Time** & *time*, **ProcTop** & *procTop*)

Boundary Conditions

assuming theta and phi velocity same outside star as inside.

assuming that $\$V\$$ at $\$i+1\$$ is equal to $\$v\$$ at $\$i\$$.

Missing pressure outside surface setting it equal to negative pressure in the center of the first cell so that it will be zero at surface.

eddy viscosity outside the star is zero.

Missing mass outside model, setting it to zero.

Missing grid.dLocalGridOld[grid.nU][i+1][j][k] and grid.dLocalGridOld[grid.nDM][nICen+1][0][0] in calculation of upwind gradient, when moving inward. Using centered gradient instead.

6.11.2.50 void calNewV_RT (**Grid** & *grid*, **Parameters** & *parameters*, **Time** & *time*, **ProcTop** & *procTop*)

Boundary Conditions

grid.dLocalGridOld[grid.nV][i+1][j+1][k] is missing

missing upwind gradient, using centred gradient instead

6.11.2.51 void calNewV_RT_LES (**Grid** & *grid*, **Parameters** & *parameters*, **Time** & *time*, **ProcTop** & *procTop*)

Boundary Conditions

Assuming theta is constant across surface.

Assuming eddy viscosity is zero at surface.

6.11.2.52 void calNewV_RTP (**Grid** & *grid*, **Parameters** & *parameters*, **Time** & *time*, **ProcTop** & *procTop*)

Boundary Conditions

Assuming theta and phi velocities are the same at the surface of the star as just inside the star.
 using centered gradient for upwind gradient outside star at surface.

6.11.2.53 void calNewV_RTP_LES (**Grid** & *grid*, **Parameters** & *parameters*, **Time** & *time*, **ProcTop** & *procTop*)

Boundary Conditions

Assuming density outside star is zero

6.11.2.54 void calNewVelocities_R (**Grid** & *grid*, **Parameters** & *parameters*, **Time** & *time*, **ProcTop** & *procTop*)

This function simply calls a function that calculate the radial velocity. Calls the function [calNewU_R](#) to calculate radial velocity, including only radial terms.

Parameters:

- ↔ ***grid*** contains the local grid data and supplies the needed data to calculate the new velocities as well as holding the new velocities.
- ← ***parameters*** contains parameters used in the calculation of the new velocities.
- ← ***time*** contains time step information, current time step, and current time
- ← ***procTop*** contains processor topology information

6.11.2.55 void calNewVelocities_R_LES (**Grid** & *grid*, **Parameters** & *parameters*, **Time** & *time*, **ProcTop** & *procTop*)

This function simply calls a function that calculate the radial velocity. Calls the function [calNewU_R](#) to calculate radial velocity, including only radial terms.

Parameters:

- ↔ ***grid*** contains the local grid data and supplies the needed data to calculate the new velocities as well as holding the new velocities.
- ← ***parameters*** contains parameters used in the calculation of the new velocities.
- ← ***time*** contains time step information, current time step, and current time
- ← ***procTop*** contains processor topology information

6.11.2.56 void calNewVelocities_RT (**Grid** & *grid*, **Parameters** & *parameters*, **Time** & *time*, **ProcTop** & *procTop*)

This function simply calls two other functions that calculate the radial and theta velocities. Calls the two functions [calNewU_RT](#) and [calNewV_RT](#) to calculate radial and theta velocities, including both radial and theta terms.

Parameters:

- ↔ ***grid*** contains the local grid data and supplies the needed data to calculate the new velocities as well as holding the new velocities.
- ← ***parameters*** contains parameters used in the calculation of the new velocities.
- ← ***time*** contains time step information, current time step, and current time
- ← ***procTop*** contains processor topology information

6.11.2.57 void calNewVelocities_RT_LES (**Grid** & *grid*, **Parameters** & *parameters*, **Time** & *time*, **ProcTop** & *procTop*)

This function simply calls two other functions that calculate the radial and theta velocities. Calls the two functions [calNewU_RT](#) and [calNewV_RT](#) to calculate radial and theta velocities, including both radial and theta terms.

Parameters:

- ↔ ***grid*** contains the local grid data and supplies the needed data to calculate the new velocities as well as holding the new velocities.
- ← ***parameters*** contains parameters used in the calculation of the new velocities.
- ← ***time*** contains time step information, current time step, and current time
- ← ***procTop*** contains processor topology information

6.11.2.58 void calNewVelocities_RTP (**Grid** & *grid*, **Parameters** & *parameters*, **Time** & *time*, **ProcTop** & *procTop*)

This function simply calls three other functions that calculate the radial, theta and phi velocities. Calls the two functions [calNewU_RTP](#), [calNewV_RTP](#) and [calNewW_RTP](#) to calculate radial, theta, and phi velocities, including radial, theta, and phi terms.

Parameters:

- ↔ ***grid*** contains the local grid data and supplies the needed data to calculate the new velocities as well as holding the new velocities.
- ← ***parameters*** contains parameters used in the calculation of the new velocities.
- ← ***time*** contains time step information, current time step, and current time
- ← ***procTop*** contains processor topology information

6.11.2.59 void calNewVelocities_RTP_LES (**Grid** & *grid*, **Parameters** & *parameters*, **Time** & *time*, **ProcTop** & *procTop*)

This function simply calls three other functions that calculate the radial, theta and phi velocities. Calls the two functions `calNewU_RTP`, `calNewV_RTP` and `calNewW_RTP` to calculate radial, theta, and phi velocities, including radial, theta, and phi terms.

Parameters:

- ↔ *grid* contains the local grid data and supplies the needed data to calculate the new velocities as well as holding the new velocities.
- ← *parameters* contains parameters used in the calculation of the new velocities.
- ← *time* contains time step information, current time step, and current time
- ← *procTop* contains processor topology information

6.11.2.60 void calNewW_RTP (**Grid** & *grid*, **Parameters** & *parameters*, **Time** & *time*, **ProcTop** & *procTop*)

Boundary Conditions

missing `grid.dLocalGridOld[grid.nW][i+1][j][k]` assuming that the phi velocity at the outer most interface is the same as the phi velocity in the center of the zone.
 missing `grid.dLocalGridOld[grid.nW][i+1][j][k]` in outer most zone. This is needed to calculate the upwind gradient for donor cell. The centered gradient is used instead when moving in the negative direction.

6.11.2.61 void calNewW_RTP_LES (**Grid** & *grid*, **Parameters** & *parameters*, **Time** & *time*, **ProcTop** & *procTop*)

Boundary Conditions

assume theta and phi velocities are constant across surface
 assume eddy viscosity is zero at surface
 assume upwind gradient is the same as centered gradient across surface

6.11.2.62 void calOldDenave_None (**Grid** & *grid*)

This function is a dummy function, and doesn't do anything. In the case of a 1D calculation the average density is undefined, and only the density is used. This is different from the case where the 1D region exists on the rank 0 processor, but the grid as a whole is really 2D or 3D. In which case `calOldDenave_R` should be used instead.

6.11.2.63 void calOldDenave_R (**Grid** & *grid*)

This function does nothing as the averaged density is not needed in 1D calculations.

Parameters:

↔ *grid* supplies the information needed to calculate the horizontal density average, it also stores the calculated horizontally averaged density.

6.11.2.64 void calOldDenave_RT (Grid & grid)

This function calculates the horizontal average density in a 2D region. This function differs from [calNewDenave_RT](#) in that it calculates the average density from the old grid density and stores the result in the old grid. While [calNewDenave_RT](#) calculates the average density from the new grid density and places the result in the new grid.

Parameters:

↔ *grid* supplies the information needed to calculate the horizontal density average, it also stores the calculated horizontally averaged density.

6.11.2.65 void calOldDenave_RTP (Grid & grid)

This function calculates the horizontal average density in a 3D region. This function differs from [calNewDenave_RTP](#) in that it calculates the average density from the old grid density and stores the result in the old grid. While [calNewDenave_RTP](#) calculates the average density from the new grid density and places the result in the new grid.

Parameters:

↔ *grid* supplies the information needed to calculate the horizontal density average, it also stores the calculated horizontally averaged density.

6.11.2.66 void calOldEddyVisc_R_CN (Grid & grid, Parameters & parameters)

Calculates the eddy viscosity using a constant times the zoning including only the radial terms. It puts the result into the old grid. This function is used to initialize the eddy viscosity when the code begins execution.

6.11.2.67 void calOldEddyVisc_R_SM (Grid & grid, Parameters & parameters)

Calculates the eddy viscosity including only the radial terms. It puts the result into the old grid. This function is used to initialize the eddy viscosity when the code begins execution. It uses the Smagorinsky model for calculating the eddy viscosity.

Parameters:

↔ *grid* supplies the input for calculating the eddy viscosity.

← *parameters* contains parameters used in calculating the eddy viscosity.

6.11.2.68 void calOldEddyVisc_RT_CN (**Grid** & *grid*, **Parameters** & *parameters*)

Calculates the eddy viscosity using a constant times the zoning including only the radial and theta terms. It puts the result into the old grid. This function is used to initialize the eddy viscosity when the code begins execution.

Parameters:

- ↔ *grid* supplies the input for calculating the eddy viscosity.
- ← *parameters* contains parameters used in calculating the eddy viscosity.

6.11.2.69 void calOldEddyVisc_RT_SM (**Grid** & *grid*, **Parameters** & *parameters*)

Calculates the eddy viscosity including only the radial and theta terms. It puts the result into the old grid. This function is used to initialize the eddy viscosity when the code begins execution. It uses the Smagorinsky model for calculating the eddy viscosity.

Parameters:

- ↔ *grid* supplies the input for calculating the eddy viscosity.
- ← *parameters* contains parameters used in calculating the eddy viscosity.

6.11.2.70 void calOldEddyVisc_RTP_CN (**Grid** & *grid*, **Parameters** & *parameters*)

Calculates the eddy viscosity using a constant times the zoning including the radial, theta, and phi terms. It puts the result into the old grid. This function is used to initialize the eddy viscosity when the code begins execution.

Parameters:

- ↔ *grid* supplies the input for calculating the eddy viscosity.
- ← *parameters* contains parameters used in calculating the eddy viscosity.

6.11.2.71 void calOldEddyVisc_RTP_SM (**Grid** & *grid*, **Parameters** & *parameters*)

Boundary Conditions

- assuming that theta velocity is constant across surface
- assume phi velocity is constant across surface

6.11.2.72 void calOldP_GL (**Grid** & *grid*, **Parameters** & *parameters*)

This function calculates the pressure using a gamma law gas, calculate by [dEOS_GL](#).

Parameters:

- ↔ *grid* supplies the input for calculating the pressure and also accepts the results of the pressure calculations
- ← *parameters* contains parameters used in calculating the pressure, namely the value of the adiabatic gamma

6.11.2.73 void calOldPEKappaGamma_TEOS (Grid & grid, Parameters & parameters)

This function calculates the pressure, energy, opacity, and adiabatic index of a cell. It calculates it using the old vaules of quantities and places the result in the old grid. This function is used to initialize the internal variables pressure, energy and kappa, and is suitable for both 1D and 3D calculations.

Parameters:

- ↔ *grid* supplies the input for calculating the pressure and also accepts the result of the pressure calculation
- ← *parameters* contains parameters used in calculating the pressure.

6.11.2.74 void calOldQ0_R_GL (Grid & grid, Parameters & parameters)

This function calculates the artificial viscosity of a cell. It calculates it using the old vaules of quantities and places the result in the old grid. It does this for the radial component of the viscosity only. This function is used when using a gamma law gas equation of state.

Parameters:

- ↔ *grid* supplies the input for calculating the artificial viscosity and also accepts the result of the artificial viscosity calculation
- ← *parameters* contains parameters used in calculating the artificial viscosity.

6.11.2.75 void calOldQ0_R_TEOS (Grid & grid, Parameters & parameters)

This function calculates the artificial viscosity of a cell. It calculates it using the old vaules of quantities and places the result in the old grid. It does this for 1D viscosity only.

Parameters:

- ↔ *grid* supplies the input for calculating the pressure and also accepts the result of the pressure calculation
- ← *parameters* contains parameters used in calculating the artificial viscosity.

6.11.2.76 void calOldQ0Q1_RT_GL (**Grid** & *grid*, **Parameters** & *parameters*)

This function calculates the artificial viscosity of a cell. It calculates it using the old vaules of quantities and places the result in the old grid. It does this for the two components of the viscosity. This function is used when using a gamma law gas equation of state.

Parameters:

- ↔ *grid* supplies the input for calculating the artificial viscosity and also accepts the result of the artificial viscosity calculation
- ← *parameters* contains parameters used in calculating the artificial viscosity.

6.11.2.77 void calOldQ0Q1_RT_TEOS (**Grid** & *grid*, **Parameters** & *parameters*)

This function calculates the artificial viscosity of a cell. It calculates it using the old vaules of quantities and places the result in the old grid. It does this for two components of the viscosity. This function is used when using a tabulated equation of state.

Parameters:

- ↔ *grid* supplies the input for calculating the artificial viscosity and also accepts the result of the artificial viscosity calculation
- ← *parameters* contains parameters used in calculating the artificial viscosity.

6.11.2.78 void calOldQ0Q1Q2_RTP_GL (**Grid** & *grid*, **Parameters** & *parameters*)

This function calculates the artificial viscosity of a cell. It calculates it using the old vaules of quantities and places the result in the old grid. It does this for the three components of the viscosity. This function is used when using a gamma law gas equation of state.

Parameters:

- ↔ *grid* supplies the input for calculating the artificial viscosity and also accepts the result of the artificial viscosity calculation
- ← *parameters* contains parameters used in calculating the artificial viscosity.

6.11.2.79 void calOldQ0Q1Q2_RTP_TEOS (**Grid** & *grid*, **Parameters** & *parameters*)

This function calculates the artificial viscosity of a cell. It calculates it using the old vaules of quantities and places the result in the old grid. It does this for the three components of the viscosity. This function is used when using a tabulated equation of state.

Parameters:

- ↔ *grid* supplies the input for calculating the artificial viscosity and also accepts the result of the artificial viscosity calculation
- ← *parameters* contains parameters used in calculating the artificial viscosity.

6.11.2.80 double dEOS_GL (double *dRho*, double *dE*, [Parameters](#) *parameters*)

Calculates the pressure from the energy and density using a γ -law gas.

Parameters:

- ← *dRho* the density of a cell
- ← *dE* the energy of a cell
- ← *parameters* contains various parameters, including γ needed to calculate the pressure.

Returns:

the pressure

This version of [dEOS_GL](#) uses the same value of γ through out the model. The equation of state is given by $\rho(\gamma - 1)E$.

6.11.2.81 double dImplicitEnergyFunction_None ([Grid](#) & *grid*, [Parameters](#) & *parameters*, [Time](#) & *time*, double *dTemps*[], int *i*, int *j*, int *k*)

This is an empty function, that isn't even called when no implicit solution is needed. This safe guards against future addition which may need to call an empty function when no implicit solve is being done.

6.11.2.82 double dImplicitEnergyFunction_R ([Grid](#) & *grid*, [Parameters](#) & *parameters*, [Time](#) & *time*, double *dTemps*[], int *i*, int *j*, int *k*)

This function is used to determine the agreement of the updated values at $n + 1$, with each other in the non-adiabatic energy equation. The *_R* version of the function contains only the radial terms, and should be used for purely radial calculations. This function can also be used for calculating numerical derivatives by varying the input temperatures.

Parameters:

- ← *grid*
- ← *parameters*
- ← *time*
- ← *dTemps* *dTemps*[0]=*dT_ijk_np1* is the temperature at radial position (i, j, k) and time $n + 1$, *dTemps*[1]=*dT_ip1jk_np1* is the temperature at radial position $(i + 1, j, k)$ and time $n + 1$, *dTemps*[2]=*dT_im1jk_np1* is the temperature at radial position $(i - 1, j, k)$ and time $n + 1$.
- ← *i* is the radial index to evaluate the function at.
- ← *j* is the theta index to evaluate the function at.
- ← *k* is the phi index to evaluate the function at.

6.11.2.83 double dImplicitEnergyFunction_R_LES ([Grid](#) & *grid*, [Parameters](#) & *parameters*, [Time](#) & *time*, double *dTemps*[], int *i*, int *j*, int *k*)

This function is used to determine the agreement of the updated values at $n + 1$, with each other in the non-adiabatic energy equation. The *_R* version of the function contains only the radial terms, and should be used for purely radial calculations. This function can also be used for calculating numerical derivatives by varying the input temperatures.

Parameters:

- ← *grid*
- ← *parameters*
- ← *time*
- ← *dTemps* dTemps[0]=dT_ijk_np1 is the temperature at radial position (i, j, k) and time $n + 1$, dTemps[1]=dT_ip1jk_np1 is the temperature at radial position $(i + 1, j, k)$ and time $n + 1$, dTemps[2]=dT_im1jk_np1 is the temperature at radial position $(i - 1, j, k)$ and time $n + 1$.
- ← *i* is the radial index to evaluate the function at.
- ← *j* is the theta index to evaluate the function at.
- ← *k* is the phi index to evaluate the function at.

6.11.2.84 double dImplicitEnergyFunction_R_LES_SB (Grid & grid, Parameters & parameters, Time & time, double dTemps[], int i, int j, int k)

Boundary Conditions

Missing grid.dLocalGridOld[grid.nE][i+1][j][k] in calculation of $E_{i+1/2,j,k}$ setting it equal to value at i.
 grid.dLocalGridOld[grid.nDM][i+1][0][0] and grid.dLocalGridOld[grid.nE][i+1][j][k] missing in the calculation of upwind gradient in dA1. Using the centered gradient instead.
 Missing grid.dLocalGridOld[grid.nT][i+1][0][0]

6.11.2.85 double dImplicitEnergyFunction_R_SB (Grid & grid, Parameters & parameters, Time & time, double dTemps[], int i, int j, int k)

Boundary Conditions

Missing grid.dLocalGridOld[grid.nE][i+1][j][k] in calculation of $E_{i+1/2,j,k}$ setting it equal to value at i.
 grid.dLocalGridOld[grid.nDM][i+1][0][0] and grid.dLocalGridOld[grid.nE][i+1][j][k] missing in the calculation of upwind gradient in dA1. Using the centered gradient instead.
 Missing grid.dLocalGridOld[grid.nT][i+1][0][0]

6.11.2.86 double dImplicitEnergyFunction_RT (Grid & grid, Parameters & parameters, Time & time, double dTemps[], int i, int j, int k)

This function is used to determine the agreement of the updated values at $n + 1$, with each other in the non-adiabatic energy equation. The _RT version of the function contains only the radial and theta terms, and should be used for radial-theta calculations. This function can also be used for calculating numerical derivatives by varying the input temperatures.

Parameters:

- ← *grid*
- ← *parameters*
- ← *time*

- ← **dTemps** dTemps[0]=dT_ijk_np1 is the temperature at radial position (i, j, k) and time $n + 1$, dTemps[1]=dT_ip1jk_np1 is the temperature at radial position $(i + 1, j, k)$ and time $n + 1$, dTemps[2]=dT_im1jk_np1 is the temperature at radial position $(i - 1, j, k)$ and time $n + 1$, dTemps[3]=dT_ijp1k_np1 is the temperature at radial position $(i, j + 1, k)$ and time $n + 1$, dTemps[4]=dT_ijm1k_np1 is the temperature at radial position $(i, j - 1, k)$ and time $n + 1$.
- ← **i** is the radial index to evaluate the function at.
- ← **j** is the theta index to evaluate the function at.
- ← **k** is the phi index to evaluate the function at.

6.11.2.87 double dImplicitEnergyFunction_RT_LES (**Grid** & *grid*, **Parameters** & *parameters*, **Time** & *time*, double *dTemps*[], int *i*, int *j*, int *k*)

This function is used to determine the agreement of the updated values at $n + 1$, with each other in the non-adiabatic energy equation. The _RT version of the function contains only the radial and theta terms, and should be used for radial-theta calculations. This function can also be used for calculating numerical derivatives by varying the input temperatures.

Parameters:

- ← *grid*
- ← *parameters*
- ← *time*
- ← **dTemps** dTemps[0]=dT_ijk_np1 is the temperature at radial position (i, j, k) and time $n + 1$, dTemps[1]=dT_ip1jk_np1 is the temperature at radial position $(i + 1, j, k)$ and time $n + 1$, dTemps[2]=dT_im1jk_np1 is the temperature at radial position $(i - 1, j, k)$ and time $n + 1$, dTemps[3]=dT_ijp1k_np1 is the temperature at radial position $(i, j + 1, k)$ and time $n + 1$, dTemps[4]=dT_ijm1k_np1 is the temperature at radial position $(i, j - 1, k)$ and time $n + 1$.
- ← **i** is the radial index to evaluate the function at.
- ← **j** is the theta index to evaluate the function at.
- ← **k** is the phi index to evaluate the function at.

6.11.2.88 double dImplicitEnergyFunction_RT_LES_SB (**Grid** & *grid*, **Parameters** & *parameters*, **Time** & *time*, double *dTemps*[], int *i*, int *j*, int *k*)

Boundary Conditions

Using centered gradient for upwind gradient when motion is into the star at the surface
Missing grid.dLocalGridOld[grid.nT][i+1][0][0] using flux equals $2\sigma T^4$ at surface.

6.11.2.89 double dImplicitEnergyFunction_RT_SB (**Grid** & *grid*, **Parameters** & *parameters*, **Time** & *time*, double *dTemps*[], int *i*, int *j*, int *k*)

Boundary Conditions

Using centered gradient for upwind gradient when motion is into the star at the surface
Missing grid.dLocalGridOld[grid.nT][i+1][0][0] using flux equals $2\sigma T^4$ at surface.

6.11.2.90 **double dImplicitEnergyFunction_RTP** (**Grid & grid, Parameters & parameters, Time & time, double dTemps[], int i, int j, int k**)

This function is used to determine the agreement of the updated values at $n + 1$, with each other in the non-adiabatic energy equation. The `_RTP` version of the function contains terms for all three directions, and should be used for calculations involving all three directions. This function can also be used for calculating numerical derivatives by varying the input temperatures. This function differs from the version without the "_SB" suffix (`dImplicitEnergyFunction_RT`) in that it is tailored to the surface boundary region.

Parameters:

- ← *grid*
- ← *parameters*
- ← *time*
- ← *dTemps, dTemps[0]=dT_ijk_np1* is the temperature at radial position (i, j, k) and time $n + 1$, *dTemps[1]=dT_ip1jk_np1* is the temperature at radial position $(i + 1, j, k)$ and time $n + 1$, *dTemps[2]=dT_im1jk_np1* is the temperature at radial position $(i - 1, j, k)$ and time $n + 1$, *dTemps[3]=dT_ijp1k_np1* is the temperature at radial position $(i, j + 1, k)$ and time $n + 1$, *dTemps[4]=dT_ijm1k_np1* is the temperature at radial position $(i, j - 1, k)$ and time $n + 1$, *dTemps[5]=dT_ijkp1_np1* is the temperature at radial position $(i, j, k + 1)$ and time $n + 1$, *dTemps[6]=dT_ijkm1_np1* is the temperature at radial position $(i, j, k - 1)$ and time $n + 1$.
- ← *i* is the radial index to evaluate the function at.
- ← *j* is the theta index to evaluate the function at.
- ← *k* is the phi index to evaluate the function at.

6.11.2.91 **double dImplicitEnergyFunction_RTP_LES** (**Grid & grid, Parameters & parameters, Time & time, double dTemps[], int i, int j, int k**)

This function is used to determine the agreement of the updated values at $n + 1$, with each other in the non-adiabatic energy equation. The `_RTP` version of the function contains terms for all three directions, and should be used for calculations involving all three directions. This function can also be used for calculating numerical derivatives by varying the input temperatures. This function differs from the version without the "_SB" suffix (`dImplicitEnergyFunction_RT`) in that it is tailored to the surface boundary region.

Parameters:

- ← *grid*
- ← *parameters*
- ← *time*
- ← *dTemps, dTemps[0]=dT_ijk_np1* is the temperature at radial position (i, j, k) and time $n + 1$, *dTemps[1]=dT_ip1jk_np1* is the temperature at radial position $(i + 1, j, k)$ and time $n + 1$, *dTemps[2]=dT_im1jk_np1* is the temperature at radial position $(i - 1, j, k)$ and time $n + 1$, *dTemps[3]=dT_ijp1k_np1* is the temperature at radial position $(i, j + 1, k)$ and time $n + 1$, *dTemps[4]=dT_ijm1k_np1* is the temperature at radial position $(i, j - 1, k)$ and time $n + 1$, *dTemps[5]=dT_ijkp1_np1* is the temperature at radial position $(i, j, k + 1)$ and time $n + 1$, *dTemps[6]=dT_ijkm1_np1* is the temperature at radial position $(i, j, k - 1)$ and time $n + 1$.
- ← *i* is the radial index to evaluate the function at.
- ← *j* is the theta index to evaluate the function at.
- ← *k* is the phi index to evaluate the function at.

6.11.2.92 `double dImplicitEnergyFunction_RTP_LES_SB (Grid & grid, Parameters & parameters, Time & time, double dTemps[], int i, int j, int k)`

Boundary Conditions

assuming V at ip1half is the same as V at i

assuming W at ip1half is the same as W at i

Using $E_{i,j,k}^{n+1/2}$ for $E_{i+1/2,j,k}^{n+1/2}$

Using centered gradient for upwind gradient when motion is into the star at the surface

Missing grid.dLocalGridOld[grid.nT][i+1][0][0] using flux equals $2\sigma T^4$ at surface.

6.11.2.93 `double dImplicitEnergyFunction_RTP_SB (Grid & grid, Parameters & parameters, Time & time, double dTemps[], int i, int j, int k)`

Boundary Conditions

Using $E_{i,j,k}^{n+1/2}$ for $E_{i+1/2,j,k}^{n+1/2}$

Using centered gradient for upwind gradient when motion is into the star at the surface

Missing grid.dLocalGridOld[grid.nT][i+1][0][0] using flux equals $2\sigma T^4$ at surface.

6.11.2.94 `void implicitSolve_None (Grid & grid, Implicit & implicit, Parameters & parameters, Time & time, ProcTop & procTop, MessPass & messPass, Functions & functions)`

This is an empty function, to be called when no implicit solution is needed. This allows the same code in the main program to be executed wheather or not an implicit solution is being preformed by setting the funciton pointer to this funciton if there is no implicit solution required.

6.11.2.95 `void implicitSolve_R (Grid & grid, Implicit & implicit, Parameters & parameters, Time & time, ProcTop & procTop, MessPass & messPass, Functions & functions)`

This function solves for temperature corrections based on derivatives of the radial non-adiabatic energy equation with respect to the new temperature. It then uses these derivatives as entries in the coefferient matrix. The discrepancy in the balance of the energy equation with the new temperature, energy, pressure, and opacity are included as the right hand side of the system of equaitons. Solving this system of equaitons provides the corrections needed for the new temperature. This processes is then repeated until the correc-tions are small. At this point the new temperature is used to update the energy, pressure, and opacity in the new grid via the equaiton of state.

6.11.2.96 `void implicitSolve_RT (Grid & grid, Implicit & implicit, Parameters & parameters, Time & time, ProcTop & procTop, MessPass & messPass, Functions & functions)`

This function solves for temperature corrections based on derivatives of the radial-theta non-adiabatic en-ergy equation with respect to the new temperature. It then uses these derivatives as entries in the coefferient matrix. The discrepancy in the balance of the energy equation with the new temperature, energy, pressure, and opacity are included as the right hand side of the system of equaitons. Solving this system of equaitons provides the corrections needed for the new temperature. This processes is then repeated until the correc-tions are small. At this point the new temperature is used to update the energy, pressure, and opacity in the new grid via the equaiton of state.

6.11.2.97 void implicitSolve_RTP (Grid & grid, Implicit & implicit, Parameters & parameters, Time & time, ProcTop & procTop, MessPass & messPass, Functions & functions)

This function solves for temperature corrections based on derivatives of the radial-theta-phi non-adiabatic energy equation with respect to the new temperature. It then uses these derivatives as entries in the coefficient matrix. The discrepancy in the balance of the energy equation with the new temperature, energy, pressure, and opacity are included as the right hand side of the system of equations. Solving this system of equations provides the corrections needed for the new temperature. This process is then repeated until the corrections are small. At this point the new temperature is used to update the energy, pressure, and opacity in the new grid via the equation of state.

6.11.2.98 void initDonorFracAndMaxConVel_R_GL (Grid & grid, Parameters & parameters)

Initializes the donor fraction, and the maximum convective velocity when starting a calculation. The donor fraction is used to determine the amount of upwinded donor cell to use in advection terms. The maximum convective velocity is used for calculation of constant eddy viscosity parameter. This version of the function is for 1D, gamma law calculations.

6.11.2.99 void initDonorFracAndMaxConVel_R_TEOS (Grid & grid, Parameters & parameters)

Initializes the donor fraction, and the maximum convective velocity when starting a calculation. The donor fraction is used to determine the amount of upwinded donor cell to use in advection terms. The maximum convective velocity is used for calculation of constant eddy viscosity parameter. This version of the function is for 1D, tabulated equation of state calculations.

6.11.2.100 void initDonorFracAndMaxConVel_RT_GL (Grid & grid, Parameters & parameters)

Initializes the donor fraction, and the maximum convective velocity when starting a calculation. The donor fraction is used to determine the amount of upwinded donor cell to use in advection terms. The maximum convective velocity is used for calculation of constant eddy viscosity parameter. This version of the function is for 2D, gamma law calculations.

6.11.2.101 void initDonorFracAndMaxConVel_RT_TEOS (Grid & grid, Parameters & parameters)

Initializes the donor fraction, and the maximum convective velocity when starting a calculation. The donor fraction is used to determine the amount of upwinded donor cell to use in advection terms. The maximum convective velocity is used for calculation of constant eddy viscosity parameter. This version of the function is for 2D, tabulated equation of state calculations.

6.11.2.102 void initDonorFracAndMaxConVel_RTP_GL (Grid & grid, Parameters & parameters)

Initializes the donor fraction, and the maximum convective velocity when starting a calculation. The donor fraction is used to determine the amount of upwinded donor cell to use in advection terms. The maximum convective velocity is used for calculation of constant eddy viscosity parameter. This version of the function is for 3D, gamma law calculations.

6.11.2.103 void initDonorFracAndMaxConVel_RTP_TEOS ([Grid](#) & *grid*, [Parameters](#) & *parameters*)

Initializes the donor fraction, and the maximum convective velocity when starting a calculation. The donor fraction is used to determine the amount of upwinded donor cell to use in advection terms. The maximum convective velocity is used for calculation of constant eddy viscosity parameter. This version of the function is for 3D, tabulated equation of state calculations.

6.11.2.104 void initInternalVars ([Grid](#) & *grid*, [ProcTop](#) & *procTop*, [Parameters](#) & *parameters*)

Warning:

$\Delta\theta$, $\Delta\phi$, $\sin\theta_{i,j,k}$, $\Delta\cos\theta_{i,j,k}$, all don't have the first zone calculated. At the moment this is a ghost cell that doesn't matter, but it may become a problem if calculations require this quantity. This is an issue for quantities that aren't updated in time, as those that are will have boundary cells updated with periodic boundary conditions.

6.11.2.105 void setInternalVarInf ([Grid](#) & *grid*, [Parameters](#) & *parameters*)

This function sets the information for internal variables. While external verabile information is derived from the starting model, internal variables infos are set in this function. In other words this function sets the values of [Grid::nVariables](#).

Parameters:

- ↔ *grid* supplies the information needed to calculate the horizontal density average, it also stores the calculated horizontally averaged density.
- ← *parameters* is used when setting variable infos, since one needs to know if the code is calculating using a gamma law gas, or a tabulated equation of state.

6.11.2.106 void setMainFunctions ([Functions](#) & *functions*, [ProcTop](#) & *procTop*, [Parameters](#) & *parameters*, [Grid](#) & *grid*, [Time](#) & *time*, [Implicit](#) & *implicit*)

Used to set the functions that [main\(\)](#) uses to evolve the input model.

Parameters:

- *functions* is of class [Functions](#) and is used to specify the functions called to calculate the evolution of the input model.
- ← *procTop* is of type [ProcTop](#). [ProcTop::nRank](#) is used to set different functions based on processor rank. For instance processor rank 1 requires 1D versions of the equations.
- ← *parameters* is of class [Parameters](#). It holds various constants and runtime parameters.
- ← *grid* of type [Grid](#). This function requires the number of dimensions, specified by [Grid::nNumDims](#).
- ← *time* of type [Time](#). This function requires knowledge of the type of time setp being used, specified by [Time::bVariableTimeStep](#).
- ← *implicit* of type [Implicit](#). This function needs to know if there is an implicit region, specified when [Implicit::nNumImplicitZones](#)>0.

The functions are picked based on model geometry, and the physics requested or required by the input model, and the configuration file. The specific functions pointers that are set are described in the [Functions](#) class.

6.12 physEquations.h File Reference

```
#include "global.h"
```

Functions

- void [setMainFunctions](#) ([Functions](#) &functions, [ProcTop](#) &procTop, [Parameters](#) ¶meters, [Grid](#) &grid, [Time](#) &time, [Implicit](#) &implicit)
- void [setInternalVarInf](#) ([Grid](#) &grid, [Parameters](#) ¶meters)
- void [initInternalVars](#) ([Grid](#) &grid, [ProcTop](#) &procTop, [Parameters](#) ¶meters)
- void [calNewVelocities_R](#) ([Grid](#) &grid, [Parameters](#) ¶meters, [Time](#) &time, [ProcTop](#) &procTop)
- void [calNewVelocities_R_LES](#) ([Grid](#) &grid, [Parameters](#) ¶meters, [Time](#) &time, [ProcTop](#) &procTop)
- void [calNewVelocities_RT](#) ([Grid](#) &grid, [Parameters](#) ¶meters, [Time](#) &time, [ProcTop](#) &procTop)
- void [calNewVelocities_RT_LES](#) ([Grid](#) &grid, [Parameters](#) ¶meters, [Time](#) &time, [ProcTop](#) &procTop)
- void [calNewVelocities_RTP](#) ([Grid](#) &grid, [Parameters](#) ¶meters, [Time](#) &time, [ProcTop](#) &procTop)
- void [calNewVelocities_RTP_LES](#) ([Grid](#) &grid, [Parameters](#) ¶meters, [Time](#) &time, [ProcTop](#) &procTop)
- void [calNewU_R](#) ([Grid](#) &grid, [Parameters](#) ¶meters, [Time](#) &time, [ProcTop](#) &procTop)
- void [calNewU_R_LES](#) ([Grid](#) &grid, [Parameters](#) ¶meters, [Time](#) &time, [ProcTop](#) &procTop)
- void [calNewU_RT](#) ([Grid](#) &grid, [Parameters](#) ¶meters, [Time](#) &time, [ProcTop](#) &procTop)
- void [calNewU_RT_LES](#) ([Grid](#) &grid, [Parameters](#) ¶meters, [Time](#) &time, [ProcTop](#) &procTop)
- void [calNewU_RTP](#) ([Grid](#) &grid, [Parameters](#) ¶meters, [Time](#) &time, [ProcTop](#) &procTop)
- void [calNewU_RTP_LES](#) ([Grid](#) &grid, [Parameters](#) ¶meters, [Time](#) &time, [ProcTop](#) &procTop)
- void [calNewV_RT](#) ([Grid](#) &grid, [Parameters](#) ¶meters, [Time](#) &time, [ProcTop](#) &procTop)
- void [calNewV_RT_LES](#) ([Grid](#) &grid, [Parameters](#) ¶meters, [Time](#) &time, [ProcTop](#) &procTop)
- void [calNewV_RTP](#) ([Grid](#) &grid, [Parameters](#) ¶meters, [Time](#) &time, [ProcTop](#) &procTop)
- void [calNewV_RTP_LES](#) ([Grid](#) &grid, [Parameters](#) ¶meters, [Time](#) &time, [ProcTop](#) &procTop)
- void [calNewW_RTP](#) ([Grid](#) &grid, [Parameters](#) ¶meters, [Time](#) &time, [ProcTop](#) &procTop)
- void [calNewW_RTP_LES](#) ([Grid](#) &grid, [Parameters](#) ¶meters, [Time](#) &time, [ProcTop](#) &procTop)
- void [calNewU0_R](#) ([Grid](#) &grid, [Parameters](#) ¶meters, [Time](#) &time, [ProcTop](#) &procTop, [MessPass](#) &messPass)
- void [calNewU0_RT](#) ([Grid](#) &grid, [Parameters](#) ¶meters, [Time](#) &time, [ProcTop](#) &procTop, [MessPass](#) &messPass)
- void [calNewU0_RTP](#) ([Grid](#) &grid, [Parameters](#) ¶meters, [Time](#) &time, [ProcTop](#) &procTop, [MessPass](#) &messPass)
- void [calNewR](#) ([Grid](#) &grid, [Time](#) &time)
- void [calNewD_R](#) ([Grid](#) &grid, [Parameters](#) ¶meters, [Time](#) &time, [ProcTop](#) &procTop)
- void [calNewD_RT](#) ([Grid](#) &grid, [Parameters](#) ¶meters, [Time](#) &time, [ProcTop](#) &procTop)
- void [calNewD_RTP](#) ([Grid](#) &grid, [Parameters](#) ¶meters, [Time](#) &time, [ProcTop](#) &procTop)
- void [calNewE_R_AD](#) ([Grid](#) &grid, [Parameters](#) ¶meters, [Time](#) &time, [ProcTop](#) &procTop)
- void [calNewE_R_NA](#) ([Grid](#) &grid, [Parameters](#) ¶meters, [Time](#) &time, [ProcTop](#) &procTop)
- void [calNewE_R_NA_LES](#) ([Grid](#) &grid, [Parameters](#) ¶meters, [Time](#) &time, [ProcTop](#) &procTop)
- void [calNewE_RT_AD](#) ([Grid](#) &grid, [Parameters](#) ¶meters, [Time](#) &time, [ProcTop](#) &procTop)
- void [calNewE_RT_NA](#) ([Grid](#) &grid, [Parameters](#) ¶meters, [Time](#) &time, [ProcTop](#) &procTop)

- void [calNewE_RT_NA_LES](#) ([Grid](#) &grid, [Parameters](#) ¶meters, [Time](#) &time, [ProcTop](#) &procTop)
- void [calNewE_RTP_AD](#) ([Grid](#) &grid, [Parameters](#) ¶meters, [Time](#) &time, [ProcTop](#) &procTop)
- void [calNewE_RTP_NA](#) ([Grid](#) &grid, [Parameters](#) ¶meters, [Time](#) &time, [ProcTop](#) &procTop)
- void [calNewE_RTP_NA_LES](#) ([Grid](#) &grid, [Parameters](#) ¶meters, [Time](#) &time, [ProcTop](#) &procTop)
- void [calNewDenave_None](#) ([Grid](#) &grid)
- void [calNewDenave_R](#) ([Grid](#) &grid)
- void [calNewDenave_RT](#) ([Grid](#) &grid)
- void [calNewDenave_RTP](#) ([Grid](#) &grid)
- void [calNewP_GL](#) ([Grid](#) &grid, [Parameters](#) ¶meters)
- void [calNewTPKappaGamma_TEOS](#) ([Grid](#) &grid, [Parameters](#) ¶meters)
- void [calNewPEKappaGamma_TEOS](#) ([Grid](#) &grid, [Parameters](#) ¶meters)
- void [calNewQ0_R_GL](#) ([Grid](#) &grid, [Parameters](#) ¶meters)
- void [calNewQ0_R_TEOS](#) ([Grid](#) &grid, [Parameters](#) ¶meters)
- void [calNewQ0Q1_RT_GL](#) ([Grid](#) &grid, [Parameters](#) ¶meters)
- void [calNewQ0Q1_RT_TEOS](#) ([Grid](#) &grid, [Parameters](#) ¶meters)
- void [calNewQ0Q1Q2_RTP_GL](#) ([Grid](#) &grid, [Parameters](#) ¶meters)
- void [calNewQ0Q1Q2_RTP_TEOS](#) ([Grid](#) &grid, [Parameters](#) ¶meters)
- void [calNewEddyVisc_None](#) ([Grid](#) &grid, [Parameters](#) ¶meters)
- void [calNewEddyVisc_R_CN](#) ([Grid](#) &grid, [Parameters](#) ¶meters)
- void [calNewEddyVisc_RT_CN](#) ([Grid](#) &grid, [Parameters](#) ¶meters)
- void [calNewEddyVisc_RTP_CN](#) ([Grid](#) &grid, [Parameters](#) ¶meters)
- void [calNewEddyVisc_R_SM](#) ([Grid](#) &grid, [Parameters](#) ¶meters)
- void [calNewEddyVisc_RT_SM](#) ([Grid](#) &grid, [Parameters](#) ¶meters)
- void [calNewEddyVisc_RTP_SM](#) ([Grid](#) &grid, [Parameters](#) ¶meters)
- void [calOldDenave_None](#) ([Grid](#) &grid)
- void [calOldDenave_R](#) ([Grid](#) &grid)
- void [calOldDenave_RT](#) ([Grid](#) &grid)
- void [calOldDenave_RTP](#) ([Grid](#) &grid)
- void [calOldP_GL](#) ([Grid](#) &grid, [Parameters](#) ¶meters)
- void [calOldPEKappaGamma_TEOS](#) ([Grid](#) &grid, [Parameters](#) ¶meters)
- void [calOldQ0_R_GL](#) ([Grid](#) &grid, [Parameters](#) ¶meters)
- void [calOldQ0_R_TEOS](#) ([Grid](#) &grid, [Parameters](#) ¶meters)
- void [calOldQ0Q1_RT_GL](#) ([Grid](#) &grid, [Parameters](#) ¶meters)
- void [calOldQ0Q1_RT_TEOS](#) ([Grid](#) &grid, [Parameters](#) ¶meters)
- void [calOldQ0Q1Q2_RTP_GL](#) ([Grid](#) &grid, [Parameters](#) ¶meters)
- void [calOldQ0Q1Q2_RTP_TEOS](#) ([Grid](#) &grid, [Parameters](#) ¶meters)
- void [calOldEddyVisc_R_CN](#) ([Grid](#) &grid, [Parameters](#) ¶meters)
- void [calOldEddyVisc_RT_CN](#) ([Grid](#) &grid, [Parameters](#) ¶meters)
- void [calOldEddyVisc_RTP_CN](#) ([Grid](#) &grid, [Parameters](#) ¶meters)
- void [calOldEddyVisc_R_SM](#) ([Grid](#) &grid, [Parameters](#) ¶meters)
- void [calOldEddyVisc_RT_SM](#) ([Grid](#) &grid, [Parameters](#) ¶meters)
- void [calOldEddyVisc_RTP_SM](#) ([Grid](#) &grid, [Parameters](#) ¶meters)
- void [calDelt_R_GL](#) ([Grid](#) &grid, [Parameters](#) ¶meters, [Time](#) &time, [ProcTop](#) &procTop)
- void [calDelt_R_TEOS](#) ([Grid](#) &grid, [Parameters](#) ¶meters, [Time](#) &time, [ProcTop](#) &procTop)
- void [calDelt_RT_GL](#) ([Grid](#) &grid, [Parameters](#) ¶meters, [Time](#) &time, [ProcTop](#) &procTop)
- void [calDelt_RT_TEOS](#) ([Grid](#) &grid, [Parameters](#) ¶meters, [Time](#) &time, [ProcTop](#) &procTop)
- void [calDelt_RTP_GL](#) ([Grid](#) &grid, [Parameters](#) ¶meters, [Time](#) &time, [ProcTop](#) &procTop)
- void [calDelt_RTP_TEOS](#) ([Grid](#) &grid, [Parameters](#) ¶meters, [Time](#) &time, [ProcTop](#) &procTop)

- void [calDelt_CONST](#) (Grid &grid, [Parameters](#) ¶meters, [Time](#) &time, [ProcTop](#) &procTop)
- void [implicitSolve_None](#) (Grid &grid, [Implicit](#) &implicit, [Parameters](#) ¶meters, [Time](#) &time, [ProcTop](#) &procTop, [MessPass](#) &messPass, [Functions](#) &functions)
- void [implicitSolve_R](#) (Grid &grid, [Implicit](#) &implicit, [Parameters](#) ¶meters, [Time](#) &time, [ProcTop](#) &procTop, [MessPass](#) &messPass, [Functions](#) &functions)
- void [implicitSolve_RT](#) (Grid &grid, [Implicit](#) &implicit, [Parameters](#) ¶meters, [Time](#) &time, [ProcTop](#) &procTop, [MessPass](#) &messPass, [Functions](#) &functions)
- void [implicitSolve_RTP](#) (Grid &grid, [Implicit](#) &implicit, [Parameters](#) ¶meters, [Time](#) &time, [ProcTop](#) &procTop, [MessPass](#) &messPass, [Functions](#) &functions)
- double [dImplicitEnergyFunction_None](#) (Grid &grid, [Parameters](#) ¶meters, [Time](#) &time, double dTemps[], int i, int j, int k)
- double [dImplicitEnergyFunction_R](#) (Grid &grid, [Parameters](#) ¶meters, [Time](#) &time, double dTemps[], int i, int j, int k)
- double [dImplicitEnergyFunction_R_SB](#) (Grid &grid, [Parameters](#) ¶meters, [Time](#) &time, double dTemps[], int i, int j, int k)
- double [dImplicitEnergyFunction_RT](#) (Grid &grid, [Parameters](#) ¶meters, [Time](#) &time, double dTemps[], int i, int j, int k)
- double [dImplicitEnergyFunction_RT_SB](#) (Grid &grid, [Parameters](#) ¶meters, [Time](#) &time, double dTemps[], int i, int j, int k)
- double [dImplicitEnergyFunction_RTP](#) (Grid &grid, [Parameters](#) ¶meters, [Time](#) &time, double dTemps[], int i, int j, int k)
- double [dImplicitEnergyFunction_RTP_SB](#) (Grid &grid, [Parameters](#) ¶meters, [Time](#) &time, double dTemps[], int i, int j, int k)
- double [dImplicitEnergyFunction_R_LES](#) (Grid &grid, [Parameters](#) ¶meters, [Time](#) &time, double dTemps[], int i, int j, int k)
- double [dImplicitEnergyFunction_R_LES_SB](#) (Grid &grid, [Parameters](#) ¶meters, [Time](#) &time, double dTemps[], int i, int j, int k)
- double [dImplicitEnergyFunction_RT_LES](#) (Grid &grid, [Parameters](#) ¶meters, [Time](#) &time, double dTemps[], int i, int j, int k)
- double [dImplicitEnergyFunction_RT_LES_SB](#) (Grid &grid, [Parameters](#) ¶meters, [Time](#) &time, double dTemps[], int i, int j, int k)
- double [dImplicitEnergyFunction_RTP_LES](#) (Grid &grid, [Parameters](#) ¶meters, [Time](#) &time, double dTemps[], int i, int j, int k)
- double [dImplicitEnergyFunction_RTP_LES_SB](#) (Grid &grid, [Parameters](#) ¶meters, [Time](#) &time, double dTemps[], int i, int j, int k)
- double [dEOS_GL](#) (double dRho, double dE, [Parameters](#) parameters)
- void [initDonorFracAndMaxConVel_R_GL](#) (Grid &grid, [Parameters](#) ¶meters)
- void [initDonorFracAndMaxConVel_R_TEOS](#) (Grid &grid, [Parameters](#) ¶meters)
- void [initDonorFracAndMaxConVel_RT_GL](#) (Grid &grid, [Parameters](#) ¶meters)
- void [initDonorFracAndMaxConVel_RT_TEOS](#) (Grid &grid, [Parameters](#) ¶meters)
- void [initDonorFracAndMaxConVel_RTP_GL](#) (Grid &grid, [Parameters](#) ¶meters)
- void [initDonorFracAndMaxConVel_RTP_TEOS](#) (Grid &grid, [Parameters](#) ¶meters)

6.12.1 Detailed Description

Header file for [physEquations.cpp](#)

6.12.2 Function Documentation

6.12.2.1 void calDelt_CONST (**Grid** & *grid*, **Parameters** & *parameters*, **Time** & *time*, **ProcTop** & *procTop*)

This function is used when a constant tie step is desired.

6.12.2.2 void calDelt_R_GL (**Grid** & *grid*, **Parameters** & *parameters*, **Time** & *time*, **ProcTop** & *procTop*)

This function calculates the time step by considering the sound crossing time in the radial direction only and is compatible with a gamma law gas EOS.

Parameters:

- ← *grid* contains the local grid, and will hold the newly updated densities
- ← *parameters* various parameters needed for the calculation
- ↔ *time* contains time information, e.g. time step, current time etc.
- ← *procTop* contains information about the processor topology. This function uses [ProcTop::nRank](#) to pass messages.

6.12.2.3 void calDelt_R_TEOS (**Grid** & *grid*, **Parameters** & *parameters*, **Time** & *time*, **ProcTop** & *procTop*)

This function calculates the time step by considering the sound crossing time in the radial direction only and is compatible with a tabulated EOS.

Parameters:

- ← *grid* contains the local grid, and will hold the newly updated densities
- ← *parameters* various parameters needed for the calculation
- ↔ *time* contains time information, e.g. time step, current time etc.
- ← *procTop* contains information about the processor topology. This function uses [ProcTop::nRank](#) to pass messages.

6.12.2.4 void calDelt_RT_GL (**Grid** & *grid*, **Parameters** & *parameters*, **Time** & *time*, **ProcTop** & *procTop*)

This function calculates the time step by considering the sound crossing time in the radial and theta directions only and is compatible with a gamma law gas EOS.

Parameters:

- ← *grid* contains the local grid, and will hold the newly updated densities
- ← *parameters* various parameters needed for the calculation
- ↔ *time* contains time information, e.g. time step, current time etc.
- ← *procTop* contains information about the processor topology. This function uses [ProcTop::nRank](#) to pass messages.

6.12.2.5 void calDelt_RT_TEOS (**Grid** & *grid*, **Parameters** & *parameters*, **Time** & *time*, **ProcTop** & *procTop*)

This function calculates the time step by considering the sound crossing time in the radial and theta directions and is compatible with a tabulated EOS.

Parameters:

- ← *grid* contains the local grid, and will hold the newly updated densities
- ← *parameters* various parameters needed for the calculation
- ↔ *time* contains time information, e.g. time step, current time etc.
- ← *procTop* contains information about the processor topology. This function uses [ProcTop::nRank](#) to pass messages.

6.12.2.6 void calDelt_RTP_GL (**Grid** & *grid*, **Parameters** & *parameters*, **Time** & *time*, **ProcTop** & *procTop*)

This function calculates the time step by considering the sound crossing time in the radial, theta and phi directions only and is compatible with a gamma law gas EOS.

Parameters:

- ← *grid* contains the local grid, and will hold the newly updated densities
- ← *parameters* various parameters needed for the calculation
- ↔ *time* contains time information, e.g. time step, current time etc.
- ← *procTop* contains information about the processor topology. This function uses [ProcTop::nRank](#) to pass messages.

6.12.2.7 void calDelt_RTP_TEOS (**Grid** & *grid*, **Parameters** & *parameters*, **Time** & *time*, **ProcTop** & *procTop*)

This function calculates the time step by considering the sound crossing time in the radial, theta and phi directions and is compatible with a tabulated EOS.

Parameters:

- ← *grid* contains the local grid, and will hold the newly updated densities
- ← *parameters* various parameters needed for the calculation
- ↔ *time* contains time information, e.g. time step, current time etc.
- ← *procTop* contains information about the processor topology. This function uses [ProcTop::nRank](#) to pass messages.

6.12.2.8 void calNewD_R ([Grid](#) & *grid*, [Parameters](#) & *parameters*, [Time](#) & *time*, [ProcTop](#) & *procTop*)

This function calculates new densities using terms in the radial direction only

Parameters:

- ↔ *grid* contains the local grid, and will hold the newly updated densities
- ← *parameters* various parameters needed for the calculation
- ← *time* contains time information, e.g. time step, current time etc.
- ← *procTop* contains information about the processor topology, uses [ProcTop::nRank](#) when reporting negative densities

6.12.2.9 void calNewD_RT ([Grid](#) & *grid*, [Parameters](#) & *parameters*, [Time](#) & *time*, [ProcTop](#) & *procTop*)

Boundary Conditions

doesn't include flux through outter interface

6.12.2.10 void calNewD_RTP ([Grid](#) & *grid*, [Parameters](#) & *parameters*, [Time](#) & *time*, [ProcTop](#) & *procTop*)

Boundary Conditions

doesn't allow mass flux through outter interface

6.12.2.11 void calNewDenave_None ([Grid](#) & *grid*)

This function is a dumby functon, and doesn't do anything. In the case of a 1D calculation the average density is undefined, and only the density is used. This is different from the case where the 1D region exists on the rank 0 processor, but the grid as a whole is really 2D or 3D. In which case [calNewDenave_R](#) should be used instead.

Parameters:

- ↔ *grid*

6.12.2.12 void calNewDenave_R ([Grid](#) & *grid*)

This function calculates the horizontal average density in a 3\1D region. This really just copies the density from the particular radial zone into the averaged density variable. This way it can be used exactly the same way in the 1D region as it is in the 3D region. This is done using the density in the new grid, and places the result into the new grid.

Parameters:

↔ *grid* supplies the information needed to calculate the horizontal density average, it also stores the calculated horizontally averaged density.

6.12.2.13 void calNewDenave_RT (Grid & grid)

This function calculates the horizontal average density in a 2D region from the new grid density and stores the result in the new grid.

Parameters:

↔ *grid* supplies the information needed to calculate the horizontal density average, it also stores the calculated horizontally averaged density.

6.12.2.14 void calNewDenave_RTP (Grid & grid)

This function calculates the horizontal average density in a 3D region from the new grid density and stores the result in the new grid.

Parameters:

↔ *grid* supplies the information needed to calculate the horizontal density average, it also stores the calculated horizontally averaged density.

6.12.2.15 void calNewE_R_AD (Grid & grid, Parameters & parameters, Time & time, ProcTop & procTop)

This function calculates new adiabatic energies using terms in the radial direction.

Parameters:

↔ *grid* contains the local grid, and will hold the newly updated densities
 ← *parameters* various parameters needed for the calculation
 ← *time* contains time information, e.g. time step, current time etc.
 ← *procTop*

6.12.2.16 void calNewE_R_NA (Grid & grid, Parameters & parameters, Time & time, ProcTop & procTop)**Boundary Conditions**

Missing grid.dLocalGridOld[grid.nE][i+1][j][k] in calculation of $E_{i+1/2,j,k}$ setting it equal to value at i.

grid.dLocalGridOld[grid.nDM][i+1][0][0] and grid.dLocalGridOld[grid.nE][i+1][j][k] missing in the calculation of upwind gradient in dA1. Using the centered gradient instead.

Missing grid.dLocalGridOld[grid.nT][i+1][0][0]

6.12.2.17 void calNewE_R_NA_LES (**Grid** & *grid*, **Parameters** & *parameters*, **Time** & *time*, **ProcTop** & *procTop*)

Boundary Conditions

Missing grid.dLocalGridOld[grid.nE][i+1][j][k] in calculation of $E_{i+1/2,j,k}$ setting it equal to value at i.

grid.dLocalGridOld[grid.nDM][i+1][0][0] and grid.dLocalGridOld[grid.nE][i+1][j][k] missing in the calculation of upwind gradient in dA1. Using the centered gradient instead.

Missing grid.dLocalGridOld[grid.nT][i+1][0][0]

6.12.2.18 void calNewE_RT_AD (**Grid** & *grid*, **Parameters** & *parameters*, **Time** & *time*, **ProcTop** & *procTop*)

Boundary Conditions

grid.dLocalGridOld[grid.nE][i+1][j][k] is missing

grid.dLocalGridOld[grid.nDM][i+1][0][0] and grid.dLocalGridOld[grid.nE][i+1][j][k] missing using inner gradient for both

6.12.2.19 void calNewE_RT_NA (**Grid** & *grid*, **Parameters** & *parameters*, **Time** & *time*, **ProcTop** & *procTop*)

Boundary Conditions

Missing grid.dLocalGridOld[grid.nE][i+1][j][k] in calculation of $E_{i+1/2,j,k}$ setting it equal to value at i.

grid.dLocalGridOld[grid.nDM][i+1][0][0] and grid.dLocalGridOld[grid.nE][i+1][j][k] missing in the calculation of upwind gradient in dA1. Using the centered gradient instead.

Missing grid.dLocalGridOld[grid.nT][i+1][0][0] using flux equals $2\sigma T^4$ at surface.

6.12.2.20 void calNewE_RT_NA_LES (**Grid** & *grid*, **Parameters** & *parameters*, **Time** & *time*, **ProcTop** & *procTop*)

Boundary Conditions

Setting energy at surface equal to energy in last zone.

Missing grid.dLocalGridOld[grid.nT][i+1][0][0] using flux equals $2\sigma T^4$ at surface.

6.12.2.21 void calNewE_RTP_AD (**Grid** & *grid*, **Parameters** & *parameters*, **Time** & *time*, **ProcTop** & *procTop*)

Boundary Conditions

Missing grid.dLocalGridOld[grid.nE][i+1][j][k] in calculation of $E_{i+1/2,j,k}$ setting it equal to zero.

grid.dLocalGridOld[grid.nDM][i+1][0][0] and grid.dLocalGridOld[grid.nE][i+1][j][k] missing in the calculation of upwind gradient in dA1. Using the centered gradient.

6.12.2.22 void calNewE_RTP_NA (**Grid** & *grid*, **Parameters** & *parameters*, **Time** & *time*, **ProcTop** & *procTop*)

Boundary Conditions

Missing `grid.dLocalGridOld[grid.nE][i+1][j][k]` in calculation of $E_{i+1/2,j,k}$ setting it equal to the value at *i*.

`grid.dLocalGridOld[grid.nDM][i+1][0][0]` and `grid.dLocalGridOld[grid.nE][i+1][j][k]` missing in the calculation of upwind gradient in `dA1`. Using the centered gradient instead.

Missing `grid.dLocalGridOld[grid.nT][i+1][0][0]`

6.12.2.23 void calNewE_RTP_NA_LES (**Grid** & *grid*, **Parameters** & *parameters*, **Time** & *time*, **ProcTop** & *procTop*)

Boundary Conditions

Missing *W* at *i+1*, assuming the same as at *i*

Missing `grid.dLocalGridOld[grid.nE][i+1][j][k]` in calculation of $E_{i+1/2,j,k}$ setting it equal to the value at *i*.

`grid.dLocalGridOld[grid.nDM][i+1][0][0]` and `grid.dLocalGridOld[grid.nE][i+1][j][k]` missing in the calculation of upwind gradient in `dA1`. Using the centered gradient instead.

Missing `grid.dLocalGridOld[grid.nT][i+1][0][0]`

6.12.2.24 void calNewEddyVisc_None (**Grid** & *grid*, **Parameters** & *parameters*)

This function is a empty function used as a place holder when no eddy viscosity model is being used.

Parameters:

↔ *grid*

← *parameters*

6.12.2.25 void calNewEddyVisc_R_CN (**Grid** & *grid*, **Parameters** & *parameters*)

This function calculates the eddy viscosity using a constant times the zoning with only the radial terms.

Parameters:

↔ *grid* supplies the input for calculating the eddy viscosity.

← *parameters* contains parameters used in calculating the eddy viscosity.

6.12.2.26 void calNewEddyVisc_R_SM (**Grid** & *grid*, **Parameters** & *parameters*)

This function calculates the eddy viscosity with only the radial terms.

Parameters:

- ↔ *grid* supplies the input for calculating the eddy viscosity.
- ← *parameters* contains parameters used in calculating the eddy viscosity.

6.12.2.27 void calNewEddyVisc_RT_CN (Grid & grid, Parameters & parameters)

This function calculates the eddy viscosity using a constant times the zoning with only the radial and theta terms.

Parameters:

- ↔ *grid* supplies the input for calculating the eddy viscosity.
- ← *parameters* contains parameters used in calculating the eddy viscosity.

6.12.2.28 void calNewEddyVisc_RT_SM (Grid & grid, Parameters & parameters)

This function calculates the eddy viscosity with only the radial and theta terms.

Parameters:

- ↔ *grid* supplies the input for calculating the eddy viscosity.
- ← *parameters* contains parameters used in calculating the eddy viscosity.

6.12.2.29 void calNewEddyVisc_RTP_CN (Grid & grid, Parameters & parameters)

This function calculates the eddy viscosity using a constant times the zoning with only the radial, theta, and phi terms.

Parameters:

- ↔ *grid* supplies the input for calculating the eddy viscosity.
- ← *parameters* contains parameters used in calculating the eddy viscosity.

6.12.2.30 void calNewEddyVisc_RTP_SM (Grid & grid, Parameters & parameters)**Boundary Conditions**

- assuming that theta velocity is constant across surface
- assume phi velocity is constant across surface

6.12.2.31 void calNewP_GL (*Grid* & *grid*, *Parameters* & *parameters*)

This function calculates the pressure. It is calculated using the new values of quantities and places the result in the new grid. It uses a gamma law gas give in [dEOS_GL](#) to calculate the pressure.

Parameters:

- ↔ *grid* supplies the input for calculating the pressure and also accepts the result of the pressure calculations.
- ← *parameters* contains parameters used in calculating the pressure, namely the adiabatic gamma that is used.

6.12.2.32 void calNewPEKappaGamma_TEOS (*Grid* & *grid*, *Parameters* & *parameters*)

This function calculates the Energy, pressure and opacity of a cell. It calculates it using the new vaules of quantities and places the result in the new grid.

Parameters:

- ↔ *grid* supplies the input for calculating the pressure and also accepts the result of the pressure calculation
- ← *parameters* contains parameters used in calculating the pressure.

6.12.2.33 void calNewQ0_R_GL (*Grid* & *grid*, *Parameters* & *parameters*)

This functon calculates the artificial viscosity of a cell. It calculates it using the new values of quantities and places the result in the new grid. It does this for the radial component of the viscosity only. It uses the sound speed derived from the adiabatic gamma given for the gamma law gas equation of state.

Parameters:

- ↔ *grid* supplies the input for calculating the artificial viscosity and also accepts the result of the artificial viscosity calculation.
- ← *parameters* contains parameters used when calculating the artificial viscosity, namely the adiabatic gamma.

6.12.2.34 void calNewQ0_R_TEOS (*Grid* & *grid*, *Parameters* & *parameters*)

This function calculates the artificial viscosity of a cell. It calculates it using the old values of quantities and places the result in the old grid. It does this for the radial component of the viscosity only. It uses a sound speed derived from a tabulated equaiton of state for the calculation.

Parameters:

- ↔ *grid* supplies the input for calculating the artificial viscosity and also accepts the result of the artificial viscosity calculation
- ← *parameters* contains parameters used in calculating the artificial viscosity.

6.12.2.35 void calNewQ0Q1_RT_GL (Grid & grid, Parameters & parameters)

This function calculates the artificial viscosity of a cell. It calculates it using the new values of quantities and places the result in the new grid. It does this for the radial and theta componenets of the viscosity. It uses the sound speed derived from the adiabatic gamma given for the gamma law gas equation of state.

Parameters:

- ↔ *grid* supplies the input for calculating the artificial viscosity and also accepts the result of the artificial viscosity calculations.
- ← *parameters* contains parameters used when calculating the artificial viscosity, namely the adiabatic gamma.

6.12.2.36 void calNewQ0Q1_RT_TEOS (Grid & grid, Parameters & parameters)

This function calculates the artificial viscosity of a cell. It calculates it using the old values of quantities and places the result in the old grid. It does this for two component of the viscosity.

Parameters:

- ↔ *grid* supplies the input for calculating the artificial viscosity and also accepts the result of the artificial viscosity calculation
- ← *parameters* contains parameters used in calculating the artificial viscosity.

6.12.2.37 void calNewQ0Q1Q2_RTP_GL (Grid & grid, Parameters & parameters)

This function calculates the artificial viscosity of a cell. It calculates it using the new values of quantities and places the result in the new grid. It does this for the radial, theta, and phi componenets of the viscosity. It uses the sound speed derived from the adiabatic gamma given for the gamma law gas equation of state.

Parameters:

- ↔ *grid* supplies the input for calculating the artificial viscosity and also accepts the result of the artificial viscosity calculations.
- ← *parameters* contains parameters used when calculating the artificial viscosity, namely the adiabatic gamma.

6.12.2.38 void calNewQ0Q1Q2_RTP_TEOS (Grid & grid, Parameters & parameters)

This function calculates the artificial viscosity of a cell. It calculates it using the old values of quantities and places the result in the old grid. It does this for the three component of the viscosity.

Parameters:

- ↔ *grid* supplies the input for calculating the artificial viscosity and also accepts the result of the artificial viscosity calculation
- ← *parameters* contains parameters used in calculating the artificial viscosity.

6.12.2.39 void calNewR (Grid & grid, Time & time)

This function calculates the radii, from the new radial grid velocities

Parameters:

- ↔ *grid* contains the local grid, and will hold the newly updated radial velocities
- ← *time* contains time information, e.g. time step, current time etc.

6.12.2.40 void calNewTPKappaGamma_TEOS (Grid & grid, Parameters & parameters)

This function calculates the Temperature, pressure and opacity of a cell. It calculates it using the new values of quantities and places the result in the new grid.

Parameters:

- ↔ *grid* supplies the input for calculating the pressure and also accepts the result of the pressure calculation
- ← *parameters* contains parameters used in calculating the pressure.

6.12.2.41 void calNewU0_R (Grid & grid, Parameters & parameters, Time & time, ProcTop & procTop, MessPass & messPass)**Todo**

At some point I will likely want to make this function compatible with a 3D domain decomposition instead of a purely radial domain decomposition.

6.12.2.42 void calNewU0_RT (Grid & grid, Parameters & parameters, Time & time, ProcTop & procTop, MessPass & messPass)**Todo**

At some point I will likely want to make this function compatible with a 3D domain decomposition instead of a purely radial domain decomposition.

Boundary Conditions

grid.dLocalGridOld[grid.nD][i+1][j][k] is missing

6.12.2.43 void calNewU0_RTP (Grid & grid, Parameters & parameters, Time & time, ProcTop & procTop, MessPass & messPass)**Todo**

At some point I will likely want to make this function compatible with a 3D domain decomposition instead of a purely radial domain decomposition.

Boundary Conditions

grid.dLocalGridOld[grid.nD][i+1][j][k] is missing

6.12.2.44 void calNewU_R (Grid & grid, Parameters & parameters, Time & time, ProcTop & procTop)

Boundary Conditions

Missing grid.dLocalGridOld[grid.nD][nICen+1][j][k] in calculation of $\rho_{i+1/2}$, setting it to 0.0
 Missing grid.dLocalGridOld[grid.nP][nICen+1][j][k] in calculation of S_1 , setting it to $-1.0 \times \text{grid.dLocalGridOld[grid.nP][nICen][j][k]}$.
 Missing grid.dLocalGridOld[grid.nDM][nICen+1][0][0] in calculation of centered A_1 gradient, setting it to zero.
 Missing grid.dLocalGridOld[grid.nU][i+1][j][k] and grid.dLocalGridOld[grid.nDM][nICen+1][0][0] in calculation of upwind gradient, when moving inward. Using centered gradient instead.

6.12.2.45 void calNewU_R_LES (Grid & grid, Parameters & parameters, Time & time, ProcTop & procTop)

Boundary Conditions

Missing grid.dLocalGridOld[grid.nD][nICen+1][j][k] in calculation of $\rho_{i+1/2}$, setting it to 0.0
 missing grid.dLocalGridOld[grid.nU][i+1][j][k] using velocity at i
 Assuming eddy viscosity outside model is zero.
 Missing grid.dLocalGridOld[grid.nP][nICen+1][j][k] in calculation of S_1 , setting it to $-1.0 \times \text{grid.dLocalGridOld[grid.nP][nICen][j][k]}$.
 Missing grid.dLocalGridOld[grid.nDM][nICen+1][0][0] in calculation of centered A_1 gradient, setting it to zero.
 Missing grid.dLocalGridOld[grid.nU][i+1][j][k] and grid.dLocalGridOld[grid.nDM][nICen+1][0][0] in calculation of upwind gradient, when moving inward. Using centered gradient instead.

6.12.2.46 void calNewU_RT (Grid & grid, Parameters & parameters, Time & time, ProcTop & procTop)

Boundary Conditions

Missing grid.dLocalGridOld[grid.nD][nICen+1][j][k] in calculation of $\rho_{i+1/2,j,k}$, setting it to zero.
 assuming theta velocity is constant across surface
 Missing grid.dLocalGridOld[grid.nDenAve][nICen+1][0][0] in calculation of $\langle \rho \rangle_{i+1/2}$, setting it to zero.
 Missing grid.dLocalGridOld[grid.nP][nICen+1][j][k] in calculation of S_1 , setting it to $-1.0 \times \text{dP_ijk_n}$.
 Missing grid.dLocalGridOld[grid.nDM][nICen+1][0][0] in calculation of centered A_1 gradient, setting it to zero.
 Missing grid.dLocalGridOld[grid.nU][i+1][j][k] and grid.dLocalGridOld[grid.nDM][nICen+1][0][0] in calculation of upwind gradient, when moving inward. Using centered gradient instead.
 Missing grid.dLocalGridOld[grid.nDM][i+1][0][0] in calculation of S_1 using `Parameters::dAlpha` $\times \text{grid.dLocalGridOld[grid.nDM][nICen][0][0]}$ instead.

6.12.2.47 void calNewU_RT_LES (**Grid** & *grid*, **Parameters** & *parameters*, **Time** & *time*, **ProcTop** & *procTop*)

Boundary Conditions

Missing `grid.dLocalGridOld[grid.nDM][i+1][0][0]` in calculation of S_1 using `Parameters::dAlpha * grid.dLocalGridOld[grid.nDM][nICen][0][0]` instead.

Missing density outside of surface, setting it to zero.

Assuming theta velocities are constant across surface.

Missing density outside model, setting it to zero.

Missing pressure outside surface setting it equal to negative pressure in the center of the first cell so that it will be zero at surface.

eddy viscosity outside the star is zero.

Missing mass outside model, setting it to zero.

Missing `grid.dLocalGridOld[grid.nU][i+1][j][k]` and `grid.dLocalGridOld[grid.nDM][nICen+1][0][0]` in calculation of upwind gradient, when moving inward. Using centered gradient instead.

6.12.2.48 void calNewU_RTP (**Grid** & *grid*, **Parameters** & *parameters*, **Time** & *time*, **ProcTop** & *procTop*)

Boundary Conditions

missing `grid.dLocalGridOld[grid.nU][i+1][j][k]` in calculation of $u_{i+1,j,k}$ setting $u_{i+1,j,k} = u_{i+1/2,j,k}$.

Missing `grid.dLocalGridOld[grid.nD][i+1][j][k]` in calculation of $\rho_{i+1/2,j,k}$, setting it to zero.

assuming theta velocity is constant across the surface.

assuming phi velocity is constant across the surface.

Missing `grid.dLocalGridOld[grid.nDenAve][nICen+1][0][0]` in calculation of $\langle \rho \rangle_{i+1/2}$ setting it to zero.

Missing `grid.dLocalGridOld[grid.nDM][nICen+1][0][0]` in calculation of centered A_1 gradient, setting it equal to `Parameters::dAlpha * grid.dLocalGridOld[grid.nDM][nICen][0][0]`.

Missing `grid.dLocalGridOld[grid.nU][i+1][j][k]` and `grid.dLocalGridOld[grid.nDM][nICen+1][0][0]` in calculation of upwind gradient, when moving inward. Using centered gradient instead.

Missing `grid.dLocalGridOld[grid.nDM][i+1][0][0]` in calculation of S_1 using `Parameters::dAlpha * grid.dLocalGridOld[grid.nDM][nICen][0][0]` instead.

6.12.2.49 void calNewU_RTP_LES (**Grid** & *grid*, **Parameters** & *parameters*, **Time** & *time*, **ProcTop** & *procTop*)

Boundary Conditions

assuming theta and phi velocity same outside star as inside.

assuming that $\$v\$$ at $\$i+1\$$ is equal to $\$v\$$ at $\$i\$$.

Missing pressure outside surface setting it equal to negative pressure in the center of the first cell so that it will be zero at surface.

eddy viscosity outside the star is zero.

Missing mass outside model, setting it to zero.

Missing `grid.dLocalGridOld[grid.nU][i+1][j][k]` and `grid.dLocalGridOld[grid.nDM][nICen+1][0][0]` in calculation of upwind gradient, when moving inward. Using centered gradient instead.

6.12.2.50 void calNewV_RT (**Grid** & *grid*, **Parameters** & *parameters*, **Time** & *time*, **ProcTop** & *procTop*)

Boundary Conditions

grid.dLocalGridOld[grid.nV][i+1][j+1][k] is missing
missing upwind gradient, using centred gradient instead

6.12.2.51 void calNewV_RT_LES (**Grid** & *grid*, **Parameters** & *parameters*, **Time** & *time*, **ProcTop** & *procTop*)

Boundary Conditions

Assuming theta is constant across surface.
Assuming eddy viscosity is zero at surface.

6.12.2.52 void calNewV_RTP (**Grid** & *grid*, **Parameters** & *parameters*, **Time** & *time*, **ProcTop** & *procTop*)

Boundary Conditions

Assuming theta and phi velocities are the same at the surface of the star as just inside the star.
using cetnered gradient for upwind gradient outside star at surface.

6.12.2.53 void calNewV_RTP_LES (**Grid** & *grid*, **Parameters** & *parameters*, **Time** & *time*, **ProcTop** & *procTop*)

Boundary Conditions

Assuming density outside star is zero

6.12.2.54 void calNewVelocities_R (**Grid** & *grid*, **Parameters** & *parameters*, **Time** & *time*, **ProcTop** & *procTop*)

This function simply calls a function that calculate the radial velocity. Calls the function `calNewU_R` to calculate radial velocity, including only radial terms.

Parameters:

- ↔ **grid** contains the local grid data and supplies the needed data to calculate the new velocities as well as holding the new velocities.
- ← **parameters** contains parameters used in the calculation of the new velocities.
- ← **time** contains time step information, current time step, and current time
- ← **procTop** contains processor topology information

6.12.2.55 void calNewVelocities_R_LES (**Grid** & *grid*, **Parameters** & *parameters*, **Time** & *time*, **ProcTop** & *procTop*)

This function simply calls a function that calculate the radial velocity. Calls the function [calNewU_R](#) to calculate radial velocity, including only radial terms.

Parameters:

- ↔ *grid* contains the local grid data and supplies the needed data to calculate the new velocities as well as holding the new velocities.
- ← *parameters* contains parameters used in the calculation of the new velocities.
- ← *time* contains time step information, current time step, and current time
- ← *procTop* contains processor topology information

6.12.2.56 void calNewVelocities_RT (**Grid** & *grid*, **Parameters** & *parameters*, **Time** & *time*, **ProcTop** & *procTop*)

This function simply calls two other functions that calculate the radial and theta velocities. Calls the two functions [calNewU_RT](#) and [calNewV_RT](#) to calculate radial and theta velocities, including both radial and theta terms.

Parameters:

- ↔ *grid* contains the local grid data and supplies the needed data to calculate the new velocities as well as holding the new velocities.
- ← *parameters* contains parameters used in the calculation of the new velocities.
- ← *time* contains time step information, current time step, and current time
- ← *procTop* contains processor topology information

6.12.2.57 void calNewVelocities_RT_LES (**Grid** & *grid*, **Parameters** & *parameters*, **Time** & *time*, **ProcTop** & *procTop*)

This function simply calls two other functions that calculate the radial and theta velocities. Calls the two functions [calNewU_RT](#) and [calNewV_RT](#) to calculate radial and theta velocities, including both radial and theta terms.

Parameters:

- ↔ *grid* contains the local grid data and supplies the needed data to calculate the new velocities as well as holding the new velocities.
- ← *parameters* contains parameters used in the calculation of the new velocities.
- ← *time* contains time step information, current time step, and current time
- ← *procTop* contains processor topology information

6.12.2.58 void calNewVelocities_RTP (**Grid** & *grid*, **Parameters** & *parameters*, **Time** & *time*, **ProcTop** & *procTop*)

This function simply calls three other functions that calculate the radial, theta and phi velocities. Calls the two functions `calNewU_RTP`, `calNewV_RTP` and `calNewW_RTP` to calculate radial, theta, and phi velocities, including radial, theta, and phi terms.

Parameters:

- ↔ **grid** contains the local grid data and supplies the needed data to calculate the new velocities as well as holding the new velocities.
- ← **parameters** contains parameters used in the calculation of the new velocities.
- ← **time** contains time step information, current time step, and current time
- ← **procTop** contains processor topology information

6.12.2.59 void calNewVelocities_RTP_LES (**Grid** & *grid*, **Parameters** & *parameters*, **Time** & *time*, **ProcTop** & *procTop*)

This function simply calls three other functions that calculate the radial, theta and phi velocities. Calls the two functions `calNewU_RTP`, `calNewV_RTP` and `calNewW_RTP` to calculate radial, theta, and phi velocities, including radial, theta, and phi terms.

Parameters:

- ↔ **grid** contains the local grid data and supplies the needed data to calculate the new velocities as well as holding the new velocities.
- ← **parameters** contains parameters used in the calculation of the new velocities.
- ← **time** contains time step information, current time step, and current time
- ← **procTop** contains processor topology information

6.12.2.60 void calNewW_RTP (**Grid** & *grid*, **Parameters** & *parameters*, **Time** & *time*, **ProcTop** & *procTop*)

Boundary Conditions

missing `grid.dLocalGridOld[grid.nW][i+1][j][k]` assuming that the phi velocity at the outer most interface is the same as the phi velocity in the center of the zone.

missing `grid.dLocalGridOld[grid.nW][i+1][j][k]` in outer most zone. This is needed to calculate the upwind gradient for donor cell. The centered gradient is used instead when moving in the negative direction.

6.12.2.61 void calNewW_RTP_LES (**Grid** & *grid*, **Parameters** & *parameters*, **Time** & *time*, **ProcTop** & *procTop*)

Boundary Conditions

assume theta and phi velocities are constant across surface

assume eddy viscosity is zero at surface

assume upwind gradient is the same as centered gradient across surface

6.12.2.62 void calOldDenave_None (**Grid** & *grid*)

This function is a dummy function, and doesn't do anything. In the case of a 1D calculation the average density is undefined, and only the density is used. This is different from the case where the 1D region exists on the rank 0 processor, but the grid as a whole is really 2D or 3D. In which case [calOldDenave_R](#) should be used instead.

6.12.2.63 void calOldDenave_R (**Grid** & *grid*)

This function does nothing as the averaged density is not needed in 1D calculations.

Parameters:

↔ *grid* supplies the information needed to calculate the horizontal density average, it also stores the calculated horizontally averaged density.

6.12.2.64 void calOldDenave_RT (**Grid** & *grid*)

This function calculates the horizontal average density in a 2D region. This function differs from [calNewDenave_RT](#) in that it calculates the average density from the old grid density and stores the result in the old grid. While [calNewDenave_RT](#) calculates the average density from the new grid density and places the result in the new grid.

Parameters:

↔ *grid* supplies the information needed to calculate the horizontal density average, it also stores the calculated horizontally averaged density.

6.12.2.65 void calOldDenave_RTP (**Grid** & *grid*)

This function calculates the horizontal average density in a 3D region. This function differs from [calNewDenave_RTP](#) in that it calculates the average density from the old grid density and stores the result in the old grid. While [calNewDenave_RTP](#) calculates the average density from the new grid density and places the result in the new grid.

Parameters:

↔ *grid* supplies the information needed to calculate the horizontal density average, it also stores the calculated horizontally averaged density.

6.12.2.66 void calOldEddyVisc_R_CN (**Grid** & *grid*, **Parameters** & *parameters*)

Calculates the eddy viscosity using a constant times the zoning including only the radial terms. It puts the result into the old grid. This function is used to initialize the eddy viscosity when the code begins execution.

6.12.2.67 void calOldEddyVisc_R_SM (Grid & grid, Parameters & parameters)

Calculates the eddy viscosity including only the radial terms. It puts the result into the old grid. This function is used to initialize the eddy viscosity when the code begins execution. It uses the Smagorinsky model for calculating the eddy viscosity.

Parameters:

- ↔ *grid* supplies the input for calculating the eddy viscosity.
- ← *parameters* contains parameters used in calculating the eddy viscosity.

6.12.2.68 void calOldEddyVisc_RT_CN (Grid & grid, Parameters & parameters)

Calculates the eddy viscosity using a constant times the zoning including only the radial and theta terms. It puts the result into the old grid. This function is used to initialize the eddy viscosity when the code begins execution.

Parameters:

- ↔ *grid* supplies the input for calculating the eddy viscosity.
- ← *parameters* contains parameters used in calculating the eddy viscosity.

6.12.2.69 void calOldEddyVisc_RT_SM (Grid & grid, Parameters & parameters)

Calculates the eddy viscosity including only the radial and theta terms. It puts the result into the old grid. This function is used to initialize the eddy viscosity when the code begins execution. It uses the Smagorinsky model for calculating the eddy viscosity.

Parameters:

- ↔ *grid* supplies the input for calculating the eddy viscosity.
- ← *parameters* contains parameters used in calculating the eddy viscosity.

6.12.2.70 void calOldEddyVisc_RTP_CN (Grid & grid, Parameters & parameters)

Calculates the eddy viscosity using a constant times the zoning including the radial, theta, and phi terms. It puts the result into the old grid. This function is used to initialize the eddy viscosity when the code begins execution.

Parameters:

- ↔ *grid* supplies the input for calculating the eddy viscosity.
- ← *parameters* contains parameters used in calculating the eddy viscosity.

6.12.2.71 void calOldEddyVisc_RTP_SM (**Grid** & *grid*, **Parameters** & *parameters*)

Boundary Conditions

assuming that theta velocity is constant across surface
 assume phi velocity is constant across surface

6.12.2.72 void calOldP_GL (**Grid** & *grid*, **Parameters** & *parameters*)

This function calculates the pressure using a gamma law gas, calculate by **dEOS_GL**.

Parameters:

- ↔ **grid** supplies the input for calculating the pressure and also accepts the results of the pressure calculations
- ← **parameters** contains parameters used in calculating the pressure, namely the value of the adiabatic gamma

6.12.2.73 void calOldPEKappaGamma_TEOS (**Grid** & *grid*, **Parameters** & *parameters*)

This function calculates the pressure, energy, opacity, and adiabatic index of a cell. It calculates it using the old vaules of quantities and places the result in the old grid. This function is used to initialize the internal variables pressure, energy and kappa, and is suitable for both 1D and 3D calculations.

Parameters:

- ↔ **grid** supplies the input for calculating the pressure and also accepts the result of the pressure calculation
- ← **parameters** contains parameters used in calculating the pressure.

6.12.2.74 void calOldQ0_R_GL (**Grid** & *grid*, **Parameters** & *parameters*)

This function calculates the artificial viscosity of a cell. It calculates it using the old vaules of quantities and places the result in the old grid. It does this for the radial component of the viscosity only. This function is used when using a gamma law gas equation of state.

Parameters:

- ↔ **grid** supplies the input for calculating the artificial viscosity and also accepts the result of the artificial viscosity calculation
- ← **parameters** contains parameters used in calculating the artificial viscosity.

6.12.2.75 void calOldQ0_R_TEOS (Grid & *grid*, Parameters & *parameters*)

This function calculates the artificial viscosity of a cell. It calculates it using the old vaules of quantities and places the result in the old grid. It does this for 1D viscosity only.

Parameters:

- ↔ *grid* supplies the input for calculating the pressure and also accepts the result of the pressure calculation
- ← *parameters* contains parameters used in calculating the artificial viscosity.

6.12.2.76 void calOldQ0Q1_RT_GL (Grid & *grid*, Parameters & *parameters*)

This function calculates the artificial viscosity of a cell. It calculates it using the old vaules of quantities and places the result in the old grid. It does this for the two components of the viscosity. This function is used when using a gamma law gas equation of state.

Parameters:

- ↔ *grid* supplies the input for calculating the artificial viscosity and also accepts the result of the artificial viscosity calculation
- ← *parameters* contains parameters used in calculating the artificial viscosity.

6.12.2.77 void calOldQ0Q1_RT_TEOS (Grid & *grid*, Parameters & *parameters*)

This function calculates the artificial viscosity of a cell. It calculates it using the old vaules of quantities and places the result in the old grid. It does this for two components of the viscosity. This function is used when using a tabulated equation of state.

Parameters:

- ↔ *grid* supplies the input for calculating the artificial viscosity and also accepts the result of the artificial viscosity calculation
- ← *parameters* contains parameters used in calculating the artificial viscosity.

6.12.2.78 void calOldQ0Q1Q2_RTP_GL (Grid & *grid*, Parameters & *parameters*)

This function calculates the artificial viscosity of a cell. It calculates it using the old vaules of quantities and places the result in the old grid. It does this for the three components of the viscosity. This function is used when using a gamma law gas equation of state.

Parameters:

- ↔ *grid* supplies the input for calculating the artificial viscosity and also accepts the result of the artificial viscosity calculation
- ← *parameters* contains parameters used in calculating the artificial viscosity.

6.12.2.79 void calOldQ0Q1Q2_RTP_TEOS (Grid & *grid*, Parameters & *parameters*)

This function calculates the artificial viscosity of a cell. It calculates it using the old values of quantities and places the result in the old grid. It does this for the three components of the viscosity. This function is used when using a tabulated equation of state.

Parameters:

- ↔ *grid* supplies the input for calculating the artificial viscosity and also accepts the result of the artificial viscosity calculation
- ← *parameters* contains parameters used in calculating the artificial viscosity.

6.12.2.80 double dEOS_GL (double *dRho*, double *dE*, Parameters *parameters*)

Calculates the pressure from the energy and density using a γ -law gas.

Parameters:

- ← *dRho* the density of a cell
- ← *dE* the energy of a cell
- ← *parameters* contains various parameters, including γ needed to calculate the pressure.

Returns:

the pressure

This version of *dEOS_GL* uses the same value of γ through out the model. The equation of state is given by $\rho(\gamma - 1)E$.

6.12.2.81 double dImplicitEnergyFunction_None (Grid & *grid*, Parameters & *parameters*, Time & *time*, double *dTemps*[], int *i*, int *j*, int *k*)

This is an empty function, that isn't even called when no implicit solution is needed. This safe guards against future addition which may need to call an empty function when no implicit solve is being done.

6.12.2.82 double dImplicitEnergyFunction_R (Grid & *grid*, Parameters & *parameters*, Time & *time*, double *dTemps*[], int *i*, int *j*, int *k*)

This function is used to determine the agreement of the updated values at $n + 1$, with each other in the non-adiabatic energy equation. The *_R* version of the function contains only the radial terms, and should be used for purely radial calculations. This function can also be used for calculating numerical derivatives by varying the input temperatures.

Parameters:

- ← *grid*
- ← *parameters*
- ← *time*
- ← *dTemps* *dTemps*[0]=*dT_ijk_np1* is the temperature at radial position (i, j, k) and time $n + 1$, *dTemps*[1]=*dT_ip1jk_np1* is the temperature at radial position $(i + 1, j, k)$ and time $n + 1$, *dTemps*[2]=*dT_im1jk_np1* is the temperature at radial position $(i - 1, j, k)$ and time $n + 1$.

- ← *i* is the radial index to evaluate the function at.
- ← *j* is the theta index to evaluate the function at.
- ← *k* is the phi index to evaluate the function at.

6.12.2.83 double dImplicitEnergyFunction_R_LES (Grid & grid, Parameters & parameters, Time & time, double dTemps[], int i, int j, int k)

This function is used to determine the agreement of the updated values at $n + 1$, with each other in the non-adiabatic energy equation. The _R version of the function contains only the radial terms, and should be used for purely radial calculations. This function can also be used for calculating numerical derivatives by varying the input temperatures.

Parameters:

- ← *grid*
- ← *parameters*
- ← *time*
- ← *dTemps* dTemps[0]=dT_ijk_np1 is the temperature at radial position (i, j, k) and time $n + 1$, dTemps[1]=dT_ip1jk_np1 is the temperature at radial position $(i + 1, j, k)$ and time $n + 1$, dTemps[2]=dT_im1jk_np1 is the temperature at radial position $(i - 1, j, k)$ and time $n + 1$.
- ← *i* is the radial index to evaluate the function at.
- ← *j* is the theta index to evaluate the function at.
- ← *k* is the phi index to evaluate the function at.

6.12.2.84 double dImplicitEnergyFunction_R_LES_SB (Grid & grid, Parameters & parameters, Time & time, double dTemps[], int i, int j, int k)

Boundary Conditions

Missing grid.dLocalGridOld[grid.nE][i+1][j][k] in calculation of $E_{i+1/2,j,k}$ setting it equal to value at i.
 grid.dLocalGridOld[grid.nDM][i+1][0][0] and grid.dLocalGridOld[grid.nE][i+1][j][k] missing in the calculation of upwind gradient in dA1. Using the centered gradient instead.
 Missing grid.dLocalGridOld[grid.nT][i+1][0][0]

6.12.2.85 double dImplicitEnergyFunction_R_SB (Grid & grid, Parameters & parameters, Time & time, double dTemps[], int i, int j, int k)

Boundary Conditions

Missing grid.dLocalGridOld[grid.nE][i+1][j][k] in calculation of $E_{i+1/2,j,k}$ setting it equal to value at i.
 grid.dLocalGridOld[grid.nDM][i+1][0][0] and grid.dLocalGridOld[grid.nE][i+1][j][k] missing in the calculation of upwind gradient in dA1. Using the centered gradient instead.
 Missing grid.dLocalGridOld[grid.nT][i+1][0][0]

6.12.2.86 **double dImplicitEnergyFunction_RT** (**Grid** & *grid*, **Parameters** & *parameters*, **Time** & *time*, double *dTemps*[], int *i*, int *j*, int *k*)

This function is used to determine the agreement of the updated values at $n + 1$, with each other in the non-adiabatic energy equation. The `_RT` version of the function contains only the radial and theta terms, and should be used for radial-theta calculations. This function can also be used for calculating numerical derivatives by varying the input temperatures.

Parameters:

← *grid*

← *parameters*

← *time*

← *dTemps* *dTemps*[0]=*dT_ijk_np1* is the temperature at radial position (i, j, k) and time $n + 1$, *dTemps*[1]=*dT_ip1jk_np1* is the temperature at radial position $(i + 1, j, k)$ and time $n + 1$, *dTemps*[2]=*dT_im1jk_np1* is the temperature at radial position $(i - 1, j, k)$ and time $n + 1$, *dTemps*[3]=*dT_ijp1k_np1* is the temperature at radial position $(i, j + 1, k)$ and time $n + 1$, *dTemps*[4]=*dT_ijm1k_np1* is the temperature at radial position $(i, j - 1, k)$ and time $n + 1$.

← *i* is the radial index to evaluate the function at.

← *j* is the theta index to evaluate the function at.

← *k* is the phi index to evaluate the function at.

6.12.2.87 **double dImplicitEnergyFunction_RT_LES** (**Grid** & *grid*, **Parameters** & *parameters*, **Time** & *time*, double *dTemps*[], int *i*, int *j*, int *k*)

This function is used to determine the agreement of the updated values at $n + 1$, with each other in the non-adiabatic energy equation. The `_RT` version of the function contains only the radial and theta terms, and should be used for radial-theta calculations. This function can also be used for calculating numerical derivatives by varying the input temperatures.

Parameters:

← *grid*

← *parameters*

← *time*

← *dTemps* *dTemps*[0]=*dT_ijk_np1* is the temperature at radial position (i, j, k) and time $n + 1$, *dTemps*[1]=*dT_ip1jk_np1* is the temperature at radial position $(i + 1, j, k)$ and time $n + 1$, *dTemps*[2]=*dT_im1jk_np1* is the temperature at radial position $(i - 1, j, k)$ and time $n + 1$, *dTemps*[3]=*dT_ijp1k_np1* is the temperature at radial position $(i, j + 1, k)$ and time $n + 1$, *dTemps*[4]=*dT_ijm1k_np1* is the temperature at radial position $(i, j - 1, k)$ and time $n + 1$.

← *i* is the radial index to evaluate the function at.

← *j* is the theta index to evaluate the function at.

← *k* is the phi index to evaluate the function at.

6.12.2.88 `double dImplicitEnergyFunction_RT_LES_SB` (**Grid** & *grid*, **Parameters** & *parameters*, **Time** & *time*, `double dTemps[]`, `int i`, `int j`, `int k`)

Boundary Conditions

Using centered gradient for upwind gradient when motion is into the star at the surface
Missing `grid.dLocalGridOld[grid.nT][i+1][0][0]` using flux equals $2\sigma T^4$ at surface.

6.12.2.89 `double dImplicitEnergyFunction_RT_SB` (**Grid** & *grid*, **Parameters** & *parameters*, **Time** & *time*, `double dTemps[]`, `int i`, `int j`, `int k`)

Boundary Conditions

Using centered gradient for upwind gradient when motion is into the star at the surface
Missing `grid.dLocalGridOld[grid.nT][i+1][0][0]` using flux equals $2\sigma T^4$ at surface.

6.12.2.90 `double dImplicitEnergyFunction_RTP` (**Grid** & *grid*, **Parameters** & *parameters*, **Time** & *time*, `double dTemps[]`, `int i`, `int j`, `int k`)

This function is used to determine the agreement of the updated values at $n + 1$, with each other in the non-adiabatic energy equation. The `_RTP` version of the function contains terms for all three directions, and should be used for calculations involving all three directions. This function can also be used for calculating numerical derivatives by varying the input temperatures. This function differs from the version without the `"_SB"` suffix (`dImplicitEnergyFunction_RT`) in that it is tailored to the surface boundary region.

Parameters:

← *grid*

← *parameters*

← *time*

← *dTemps*, *dTemps[0]=dT_ijk_np1* is the temperature at radial position (i, j, k) and time $n + 1$, *dTemps[1]=dT_ip1jk_np1* is the temperature at radial position $(i + 1, j, k)$ and time $n + 1$, *dTemps[2]=dT_im1jk_np1* is the temperature at radial position $(i - 1, j, k)$ and time $n + 1$, *dTemps[3]=dT_ijp1k_np1* is the temperature at radial position $(i, j + 1, k)$ and time $n + 1$, *dTemps[4]=dT_ijm1k_np1* is the temperature at radial position $(i, j - 1, k)$ and time $n + 1$, *dTemps[5]=dT_ijkp1_np1* is the temperature at radial position $(i, j, k + 1)$ and time $n + 1$, *dTemps[6]=dT_ijkm1_np1* is the temperature at radial position $(i, j, k - 1)$ and time $n + 1$.

← *i* is the radial index to evaluate the function at.

← *j* is the theta index to evaluate the function at.

← *k* is the phi index to evaluate the function at.

6.12.2.91 `double dImplicitEnergyFunction_RTP_LES` (**Grid** & *grid*, **Parameters** & *parameters*, **Time** & *time*, `double dTemps[]`, `int i`, `int j`, `int k`)

This function is used to determine the agreement of the updated values at $n + 1$, with each other in the non-adiabatic energy equation. The `_RTP` version of the function contains terms for all three directions, and

should be used for calculations involving all three directions. This function can also be used for calculating numerical derivatives by varying the input temperatures. This function differs from the version without the "_SB" suffix (`dImplicitEnergyFunction_RT`) in that it is tailored to the surface boundary region.

Parameters:

← *grid*

← *parameters*

← *time*

← *dTemps*, *dTemps[0]=dT_ijk_np1* is the temperature at radial position (i, j, k) and time $n + 1$, *dTemps[1]=dT_ip1jk_np1* is the temperature at radial position $(i + 1, j, k)$ and time $n + 1$, *dTemps[2]=dT_im1jk_np1* is the temperature at radial position $(i - 1, j, k)$ and time $n + 1$, *dTemps[3]=dT_ijp1k_np1* is the temperature at radial position $(i, j + 1, k)$ and time $n + 1$, *dTemps[4]=dT_ijm1k_np1* is the temperature at radial position $(i, j - 1, k)$ and time $n + 1$, *dTemps[5]=dT_ijkp1_np1* is the temperature at radial position $(i, j, k + 1)$ and time $n + 1$, *dTemps[6]=dT_ijkm1_np1* is the temperature at radial position $(i, j, k - 1)$ and time $n + 1$.

← *i* is the radial index to evaluate the function at.

← *j* is the theta index to evaluate the function at.

← *k* is the phi index to evaluate the function at.

6.12.2.92 double dImplicitEnergyFunction_RTP_LES_SB (*Grid* & *grid*, *Parameters* & *parameters*, *Time* & *time*, double *dTemps*[], int *i*, int *j*, int *k*)

Boundary Conditions

assuming V at $ip1half$ is the same as V at i

assuming W at $ip1half$ is the same as W at i

Using $E_{i,j,k}^{n+1/2}$ for $E_{i+1/2,j,k}^{n+1/2}$

Using centered gradient for upwind gradient when motion is into the star at the surface

Missing `grid.dLocalGridOld[grid.nT][i+1][0][0]` using flux equals $2\sigma T^4$ at surface.

6.12.2.93 double dImplicitEnergyFunction_RTP_SB (*Grid* & *grid*, *Parameters* & *parameters*, *Time* & *time*, double *dTemps*[], int *i*, int *j*, int *k*)

Boundary Conditions

Using $E_{i,j,k}^{n+1/2}$ for $E_{i+1/2,j,k}^{n+1/2}$

Using centered gradient for upwind gradient when motion is into the star at the surface

Missing `grid.dLocalGridOld[grid.nT][i+1][0][0]` using flux equals $2\sigma T^4$ at surface.

6.12.2.94 void implicitSolve_None (*Grid* & *grid*, *Implicit* & *implicit*, *Parameters* & *parameters*, *Time* & *time*, *ProcTop* & *procTop*, *MessPass* & *messPass*, *Functions* & *functions*)

This is an empty function, to be called when no implicit solution is needed. This allows the same code in the main program to be executed whether or not an implicit solution is being preformed by setting the function pointer to this function if there is no implicit solution required.

6.12.2.95 `void implicitSolve_R (Grid & grid, Implicit & implicit, Parameters & parameters, Time & time, ProcTop & procTop, MessPass & messPass, Functions & functions)`

This function solves for temperature corrections based on derivatives of the radial non-adiabatic energy equation with respect to the new temperature. It then uses these derivatives as entries in the coefficient matrix. The discrepancy in the balance of the energy equation with the new temperature, energy, pressure, and opacity are included as the right hand side of the system of equations. Solving this system of equations provides the corrections needed for the new temperature. This process is then repeated until the corrections are small. At this point the new temperature is used to update the energy, pressure, and opacity in the new grid via the equation of state.

6.12.2.96 `void implicitSolve_RT (Grid & grid, Implicit & implicit, Parameters & parameters, Time & time, ProcTop & procTop, MessPass & messPass, Functions & functions)`

This function solves for temperature corrections based on derivatives of the radial-theta non-adiabatic energy equation with respect to the new temperature. It then uses these derivatives as entries in the coefficient matrix. The discrepancy in the balance of the energy equation with the new temperature, energy, pressure, and opacity are included as the right hand side of the system of equations. Solving this system of equations provides the corrections needed for the new temperature. This process is then repeated until the corrections are small. At this point the new temperature is used to update the energy, pressure, and opacity in the new grid via the equation of state.

6.12.2.97 `void implicitSolve_RTP (Grid & grid, Implicit & implicit, Parameters & parameters, Time & time, ProcTop & procTop, MessPass & messPass, Functions & functions)`

This function solves for temperature corrections based on derivatives of the radial-theta-phi non-adiabatic energy equation with respect to the new temperature. It then uses these derivatives as entries in the coefficient matrix. The discrepancy in the balance of the energy equation with the new temperature, energy, pressure, and opacity are included as the right hand side of the system of equations. Solving this system of equations provides the corrections needed for the new temperature. This process is then repeated until the corrections are small. At this point the new temperature is used to update the energy, pressure, and opacity in the new grid via the equation of state.

6.12.2.98 `void initDonorFracAndMaxConVel_R_GL (Grid & grid, Parameters & parameters)`

Initializes the donor fraction, and the maximum convective velocity when starting a calculation. The donor fraction is used to determine the amount of upwinded donor cell to use in advection terms. The maximum convective velocity is used for calculation of constant eddy viscosity parameter. This version of the function is for 1D, gamma law calculations.

6.12.2.99 `void initDonorFracAndMaxConVel_R_TEOS (Grid & grid, Parameters & parameters)`

Initializes the donor fraction, and the maximum convective velocity when starting a calculation. The donor fraction is used to determine the amount of upwinded donor cell to use in advection terms. The maximum convective velocity is used for calculation of constant eddy viscosity parameter. This version of the function is for 1D, tabulated equation of state calculations.

6.12.2.100 void initDonorFracAndMaxConVel_RT_GL (Grid & grid, Parameters & parameters)

Initializes the donor fraction, and the maximum convective velocity when starting a calculation. The donor fraction is used to determine the amount of upwinded donor cell to use in advection terms. The maximum convective velocity is used for calculation of constant eddy viscosity parameter. This version of the function is for 2D, gamma law calculations.

6.12.2.101 void initDonorFracAndMaxConVel_RT_TEOS (Grid & grid, Parameters & parameters)

Initializes the donor fraction, and the maximum convective velocity when starting a calculation. The donor fraction is used to determine the amount of upwinded donor cell to use in advection terms. The maximum convective velocity is used for calculation of constant eddy viscosity parameter. This version of the function is for 2D, tabulated equation of state calculations.

6.12.2.102 void initDonorFracAndMaxConVel_RTP_GL (Grid & grid, Parameters & parameters)

Initializes the donor fraction, and the maximum convective velocity when starting a calculation. The donor fraction is used to determine the amount of upwinded donor cell to use in advection terms. The maximum convective velocity is used for calculation of constant eddy viscosity parameter. This version of the function is for 3D, gamma law calculations.

6.12.2.103 void initDonorFracAndMaxConVel_RTP_TEOS (Grid & grid, Parameters & parameters)

Initializes the donor fraction, and the maximum convective velocity when starting a calculation. The donor fraction is used to determine the amount of upwinded donor cell to use in advection terms. The maximum convective velocity is used for calculation of constant eddy viscosity parameter. This version of the function is for 3D, tabulated equation of state calculations.

6.12.2.104 void initInternalVars (Grid & grid, ProcTop & procTop, Parameters & parameters)**Warning:**

$\Delta\theta$, $\Delta\phi$, $\sin\theta_{i,j,k}$, $\Delta\cos\theta_{i,j,k}$, all don't have the first zone calculated. At the moment this is a ghost cell that doesn't matter, but it may become a problem if calculations require this quantity. This is an issue for quantities that aren't updated in time, as those that are will have boundary cells updated with periodic boundary conditions.

6.12.2.105 void setInternalVarInf (Grid & grid, Parameters & parameters)

This function sets the information for internal variables. While external verabile information is derived from the starting model, internal variables infos are set in this function. In other words this function sets the values of [Grid::nVariables](#).

Parameters:

↔ *grid* supplies the information needed to calculate the horizontal density average, it also stores the calculated horizontally averaged density.

← *parameters* is used when setting variable infos, since one needs to know if the code is calculating using a gamma law gas, or a tabulated equation of state.

6.12.2.106 void setMainFunctions (**Functions** & *functions*, **ProcTop** & *procTop*, **Parameters** & *parameters*, **Grid** & *grid*, **Time** & *time*, **Implicit** & *implicit*)

Used to set the functions that `main()` uses to evolve the input model.

Parameters:

- *functions* is of class **Functions** and is used to specify the functions called to calculate the evolution of the input model.
- ← *procTop* is of type **ProcTop**. **ProcTop::nRank** is used to set different functions based on processor rank. For instance processor rank 1 requires 1D versions of the equations.
- ← *parameters* is of class **Parameters**. It holds various constants and runtime parameters.
- ← *grid* of type **Grid**. This function requires the number of dimensions, specified by **Grid::nNumDims**.
- ← *time* of type **Time**. This function requires knowledge of the type of time setp being used, specified by **Time::bVariableTimeStep**.
- ← *implicit* of type **Implicit**. This function needs to know if there is an implicit region, specified when **Implicit::nNumImplicitZones**>0.

The functions are picked based on model geometry, and the physics requested or required by the input model, and the configuration file. The specific functions pointers that are set are described in the **Functions** class.

6.13 userguide.h File Reference

6.13.1 Detailed Description

Contains the text for the "Using and Modifying SPHERIS" section of this manual.

6.14 watchzone.cpp File Reference

```
#include "watchzone.h"  
#include "exception2.h"  
#include <sstream>
```

6.14.1 Detailed Description

This file holds the implementation of the watchzone class.

6.15 watchzone.h File Reference

```
#include <string>
#include <fstream>
```

Classes

- class [WatchZone](#)

6.15.1 Detailed Description

This file holds the definition of the watchzone class.

Chapter 7

SPHERLS Page Documentation

7.1 Todo List

Member `initImplicitCalculation(Implicit &implicit, Grid &grid, ProcTop &procTop, int nNumArgs, char *cArgs[])`
isFrom, isTo, matCoeff, vecTCorrections, vecTCorrections, vecRHS, vecTCorrectionsLocal, ksp-Context, vecscatTCorrections all need to be destroyed before program finishes.

Member `modelRead(std::string sFileName, ProcTop &procTop, Grid &grid, Time &time, Parameters ¶meters)`
At some point should get it working with only 1 processor

Member `updateLocalBoundaries(ProcTop &procTop, MessPass &messPass, Grid &grid)`
Shouldn't need `MPI::COMM_WORLD.Barrier()` may want to test out removing this at some point as it might produce a bit of a speed up.

Member `updateLocalBoundariesNewGrid(int nVar, ProcTop &procTop, MessPass &messPass, Grid &grid)`
May want to do some waiting on this message at some point before the end of the timestep, but it doesn't need to be done in this function. It might also be that this is built into the code by waiting at some other point. This is something that should be checked out at somepoint, perhaps once the preformance starts to be analyzed. I would think that if the send buffer was being modified before the send was completed, that there would be some errors popping up that would likely kill the program.

Member `calNewU0_R(Grid &grid, Parameters ¶meters, Time &time, ProcTop &procTop, MessPass &messPass)`
At some point I will likely want to make this functon compatiabile with a 3D domain decomposition instead of a purely radial domain decomposition.

Member `calNewU0_RT(Grid &grid, Parameters ¶meters, Time &time, ProcTop &procTop, MessPass &messPass)`

At some point I will likely want to make this function compatible with a 3D domain decomposition instead of a purely radial domain decomposition.

Member `calNewU0_RTP(Grid &grid, Parameters ¶meters, Time &time, ProcTop &procTop, MessPass &messPass)`

At some point I will likely want to make this function compatible with a 3D domain decomposition instead of a purely radial domain decomposition.

7.2 Boundary Conditions

Member `calNewD_RT(Grid &grid, Parameters ¶meters, Time &time, ProcTop &procTop)`
 doesn't include flux through outer interface

Member `calNewD_RTP(Grid &grid, Parameters ¶meters, Time &time, ProcTop &procTop)`
 doesn't allow mass flux through outer interface

Member `calNewE_R_NA(Grid &grid, Parameters ¶meters, Time &time, ProcTop &procTop)`
 Missing `grid.dLocalGridOld[grid.nE][i+1][j][k]` in calculation of $E_{i+1/2,j,k}$ setting it equal to value at i .
`grid.dLocalGridOld[grid.nDM][i+1][0][0]` and `grid.dLocalGridOld[grid.nE][i+1][j][k]` missing in the calculation of upwind gradient in `dA1`. Using the centered gradient instead.
 Missing `grid.dLocalGridOld[grid.nT][i+1][0][0]`

Member `calNewE_R_NA_LES(Grid &grid, Parameters ¶meters, Time &time, ProcTop &procTop)`
 Missing `grid.dLocalGridOld[grid.nE][i+1][j][k]` in calculation of $E_{i+1/2,j,k}$ setting it equal to value at i .
`grid.dLocalGridOld[grid.nDM][i+1][0][0]` and `grid.dLocalGridOld[grid.nE][i+1][j][k]` missing in the calculation of upwind gradient in `dA1`. Using the centered gradient instead.
 Missing `grid.dLocalGridOld[grid.nT][i+1][0][0]`

Member `calNewE_RT_AD(Grid &grid, Parameters ¶meters, Time &time, ProcTop &procTop)`
`grid.dLocalGridOld[grid.nE][i+1][j][k]` is missing
`grid.dLocalGridOld[grid.nDM][i+1][0][0]` and `grid.dLocalGridOld[grid.nE][i+1][j][k]` missing using inner gradient for both

Member `calNewE_RT_NA(Grid &grid, Parameters ¶meters, Time &time, ProcTop &procTop)`
 Missing `grid.dLocalGridOld[grid.nE][i+1][j][k]` in calculation of $E_{i+1/2,j,k}$ setting it equal to value at i .
`grid.dLocalGridOld[grid.nDM][i+1][0][0]` and `grid.dLocalGridOld[grid.nE][i+1][j][k]` missing in the calculation of upwind gradient in `dA1`. Using the centered gradient instead.
 Missing `grid.dLocalGridOld[grid.nT][i+1][0][0]` using flux equals $2\sigma T^4$ at surface.

Member calNewE_RT_NA_LES(Grid &grid, Parameters ¶meters, Time &time, ProcTop &procTop)

Setting energy at surface equal to energy in last zone.

Missing grid.dLocalGridOld[grid.nT][i+1][0][0] using flux equals $2\sigma T^4$ at surface.

Member calNewE_RTP_AD(Grid &grid, Parameters ¶meters, Time &time, ProcTop &procTop)

Missing grid.dLocalGridOld[grid.nE][i+1][j][k] in calculation of $E_{i+1/2,j,k}$ setting it equal to zero.

grid.dLocalGridOld[grid.nDM][i+1][0][0] and grid.dLocalGridOld[grid.nE][i+1][j][k] missing in the calculation of upwind gradient in dA1. Using the centered gradient.

Member calNewE_RTP_NA(Grid &grid, Parameters ¶meters, Time &time, ProcTop &procTop)

Missing grid.dLocalGridOld[grid.nE][i+1][j][k] in calculation of $E_{i+1/2,j,k}$ setting it equal to the value at i.

grid.dLocalGridOld[grid.nDM][i+1][0][0] and grid.dLocalGridOld[grid.nE][i+1][j][k] missing in the calculation of upwind gradient in dA1. Using the centered gradient instead.

Missing grid.dLocalGridOld[grid.nT][i+1][0][0]

Member calNewE_RTP_NA_LES(Grid &grid, Parameters ¶meters, Time &time, ProcTop &procTop)

Missing W at i+1, assuming the same as at i

Missing grid.dLocalGridOld[grid.nE][i+1][j][k] in calculation of $E_{i+1/2,j,k}$ setting it equal to the value at i.

grid.dLocalGridOld[grid.nDM][i+1][0][0] and grid.dLocalGridOld[grid.nE][i+1][j][k] missing in the calculation of upwind gradient in dA1. Using the centered gradient instead.

Missing grid.dLocalGridOld[grid.nT][i+1][0][0]

Member calNewEddyVisc_RTP_SM(Grid &grid, Parameters ¶meters) assuming that theta velocity is constant across surface

assume phi velocity is constant across surface

Member calNewU0_RT(Grid &grid, Parameters ¶meters, Time &time, ProcTop &procTop, MessPass &messPass)

grid.dLocalGridOld[grid.nD][i+1][j][k] is missing

Member calNewU0_RTP(Grid &grid, Parameters ¶meters, Time &time, ProcTop &procTop, MessPass &messPass)

grid.dLocalGridOld[grid.nD][i+1][j][k] is missing

Member calNewU_R(Grid &grid, Parameters ¶meters, Time &time, ProcTop &procTop)

Missing grid.dLocalGridOld[grid.nD][nICen+1][j][k] in calculation of $\rho_{i+1/2}$, setting it to 0.0

Missing grid.dLocalGridOld[grid.nP][nICen+1][j][k] in calculation of S_1 , setting it to $-1.0 \times \text{grid.dLocalGridOld}[\text{grid.nP}][\text{nICen}][j][k]$.

Missing grid.dLocalGridOld[grid.nDM][nICen+1][0][0] in calculation of centered A_1 gradient, setting it to zero.

Missing grid.dLocalGridOld[grid.nU][i+1][j][k] and grid.dLocalGridOld[grid.nDM][nICen+1][0][0] in calculation of upwind gradient, when moving inward. Using centered gradient instead.

Member calNewU_R_LES(Grid &grid, Parameters ¶meters, Time &time, ProcTop &procTop)

Missing grid.dLocalGridOld[grid.nD][nICen+1][j][k] in calculation of $\rho_{i+1/2}$, setting it to 0.0

missing grid.dLocalGridOld[grid.nU][i+1][j][k] using velocity at i

Assuming eddy viscosity outside model is zero.

Missing grid.dLocalGridOld[grid.nP][nICen+1][j][k] in calculation of S_1 , setting it to $-1.0 \times \text{grid.dLocalGridOld}[\text{grid.nP}][\text{nICen}][j][k]$.

Missing grid.dLocalGridOld[grid.nDM][nICen+1][0][0] in calculation of centered A_1 gradient, setting it to zero.

Missing grid.dLocalGridOld[grid.nU][i+1][j][k] and grid.dLocalGridOld[grid.nDM][nICen+1][0][0] in calculation of upwind gradient, when moving inward. Using centered gradient instead.

Member calNewU_RT(Grid &grid, Parameters ¶meters, Time &time, ProcTop &procTop)

Missing grid.dLocalGridOld[grid.nD][nICen+1][j][k] in calculation of $\rho_{i+1/2,j,k}$, setting it to zero.

assuming theta velocity is constant across surface

Missing grid.dLocalGridOld[grid.nDenAve][nICen+1][0][0] in calculation of $\langle \rho \rangle_{i+1/2}$, setting it to zero.

Missing grid.dLocalGridOld[grid.nP][nICen+1][j][k] in calculation of S_1 , setting it to $-1.0 \times \text{dP_ijk_n}$.

Missing grid.dLocalGridOld[grid.nDM][nICen+1][0][0] in calculation of centered A_1 gradient, setting it to zero.

Missing grid.dLocalGridOld[grid.nU][i+1][j][k] and grid.dLocalGridOld[grid.nDM][nICen+1][0][0] in calculation of upwind gradient, when moving inward. Using centered gradient instead.

Missing grid.dLocalGridOld[grid.nDM][i+1][0][0] in calculation of S_1 using [Parameters::dAlpha](#) * grid.dLocalGridOld[grid.nDM][nICen][0][0] instead.

Member calNewU_RT_LES(Grid &grid, Parameters ¶meters, Time &time, ProcTop &procTop)

Missing grid.dLocalGridOld[grid.nDM][i+1][0][0] in calculation of S_1 using [Parameters::dAlpha](#) * grid.dLocalGridOld[grid.nDM][nICen][0][0] instead.

Missing density outside of surface, setting it to zero.

Assuming theta velocities are constant across surface.

Missing density outside model, setting it to zero.

Missing pressure outside surface setting it equal to negative pressure in the center of the first cell so that it will be zero at surface.

eddy viscosity outside the star is zero.

Missing mass outside model, setting it to zero.

Missing `grid.dLocalGridOld[grid.nU][i+1][j][k]` and `grid.dLocalGridOld[grid.nDM][nICen+1][0][0]` in calculation of upwind gradient, when moving inward. Using centered gradient instead.

Member `calNewU_RTP(Grid &grid, Parameters ¶meters, Time &time, ProcTop &procTop)`

missing `grid.dLocalGridOld[grid.nU][i+1][j][k]` in calculation of $u_{i+1,j,k}$ setting $u_{i+1,j,k} = u_{i+1/2,j,k}$.

Missing `grid.dLocalGridOld[grid.nD][i+1][j][k]` in calculation of $\rho_{i+1/2,j,k}$, setting it to zero.

assuming theta velocity is constant across the surface.

assuming phi velocity is constant across the surface.

Missing `grid.dLocalGridOld[grid.nDenAve][nICen+1][0][0]` in calculation of $\langle \rho \rangle_{i+1/2}$ setting it to zero.

Missing `grid.dLocalGridOld[grid.nDM][nICen+1][0][0]` in calculation of centered A_1 gradient, setting it equal to `Parameters::dAlpha` `grid.dLocalGridOld[grid.nDM][nICen][0][0]`.

Missing `grid.dLocalGridOld[grid.nU][i+1][j][k]` and `grid.dLocalGridOld[grid.nDM][nICen+1][0][0]` in calculation of upwind gradient, when moving inward. Using centered gradient instead.

Missing `grid.dLocalGridOld[grid.nDM][i+1][0][0]` in calculation of S_1 using `Parameters::dAlpha` `*grid.dLocalGridOld[grid.nDM][nICen][0][0]` instead.

Member `calNewU_RTP_LES(Grid &grid, Parameters ¶meters, Time &time, ProcTop &procTop)`

assuming theta and phi velocity same outside star as inside.

assuming that V_i at $i+1$ is equal to v_i at i .

Missing pressure outside surface setting it equal to negative pressure in the center of the first cell so that it will be zero at surface.

eddy viscosity outside the star is zero.

Missing mass outside model, setting it to zero.

Missing `grid.dLocalGridOld[grid.nU][i+1][j][k]` and `grid.dLocalGridOld[grid.nDM][nICen+1][0][0]` in calculation of upwind gradient, when moving inward. Using centered gradient instead.

Member `calNewV_RT(Grid &grid, Parameters ¶meters, Time &time, ProcTop &procTop)`

`grid.dLocalGridOld[grid.nV][i+1][j+1][k]` is missing

missing upwind gradient, using centred gradient instead

Member `calNewV_RT_LES(Grid &grid, Parameters ¶meters, Time &time, ProcTop &procTop)`

Assuming theta is constant across surface.

Assuming eddy viscosity is zero at surface.

Member `calNewV_RTP(Grid &grid, Parameters ¶meters, Time &time, ProcTop &procTop)`

Assuming theta and phi velocities are the same at the surface of the star as just inside the star.

Assuming centered gradient for upwind gradient outside star at surface.

Member `calNewV_RTP_LES(Grid &grid, Parameters ¶meters, Time &time, ProcTop &procTop)`

Assuming density outside star is zero

Member `calNewW_RTP(Grid &grid, Parameters ¶meters, Time &time, ProcTop &procTop)`

missing `grid.dLocalGridOld[grid.nW][i+1][j][k]` assuming that the phi velocity at the outer most interface is the same as the phi velocity in the center of the zone.

missing `grid.dLocalGridOld[grid.nW][i+1][j][k]` in outer most zone. This is needed to calculate the upwind gradient for donor cell. The centered gradient is used instead when moving in the negative direction.

Member `calNewW_RTP_LES(Grid &grid, Parameters ¶meters, Time &time, ProcTop &procTop)`

assume theta and phi velocities are constant across surface

assume eddy viscosity is zero at surface

assume upwind gradient is the same as centered gradient across surface

Member `calOldEddyVisc_RTP_SM(Grid &grid, Parameters ¶meters)` assuming that theta ve-

locity is constant across surface

assume phi velocity is constant across surface

Member `dImplicitEnergyFunction_R_LES_SB(Grid &grid, Parameters ¶meters, Time &time, double dTemps[], int i)`

Missing `grid.dLocalGridOld[grid.nE][i+1][j][k]` in calculation of $E_{i+1/2,j,k}$ setting it equal to value at i.

`grid.dLocalGridOld[grid.nDM][i+1][0][0]` and `grid.dLocalGridOld[grid.nE][i+1][j][k]` missing in the calculation of upwind gradient in dA1. Using the centered gradient instead.

Missing `grid.dLocalGridOld[grid.nT][i+1][0][0]`

Member `dImplicitEnergyFunction_R_SB(Grid &grid, Parameters ¶meters, Time &time, double dTemps[], int i, int j,`

Missing `grid.dLocalGridOld[grid.nE][i+1][j][k]` in calculation of $E_{i+1/2,j,k}$ setting it equal to value at `i`.

`grid.dLocalGridOld[grid.nDM][i+1][0][0]` and `grid.dLocalGridOld[grid.nE][i+1][j][k]` missing in the calculation of upwind gradient in `dA1`. Using the centered gradient instead.

Missing `grid.dLocalGridOld[grid.nT][i+1][0][0]`

Member `dImplicitEnergyFunction_RT_LES_SB(Grid &grid, Parameters ¶meters, Time &time, double dTemps[], int`

Using centered gradient for upwind gradient when motion is into the star at the surface

Missing `grid.dLocalGridOld[grid.nT][i+1][0][0]` using flux equals $2\sigma T^4$ at surface.

Member `dImplicitEnergyFunction_RT_SB(Grid &grid, Parameters ¶meters, Time &time, double dTemps[], int i, int`

Using centered gradient for upwind gradient when motion is into the star at the surface

Missing `grid.dLocalGridOld[grid.nT][i+1][0][0]` using flux equals $2\sigma T^4$ at surface.

Member `dImplicitEnergyFunction_RTP_LES_SB(Grid &grid, Parameters ¶meters, Time &time, double dTemps[], int`

assuming `V` at `ip1half` is the same as `V` at `i`

assuming `W` at `ip1half` is the same as `W` at `i`

Using $E_{i,j,k}^{n+1/2}$ for $E_{i+1/2,j,k}^{n+1/2}$

Using centered gradient for upwind gradient when motion is into the star at the surface

Missing `grid.dLocalGridOld[grid.nT][i+1][0][0]` using flux equals $2\sigma T^4$ at surface.

Member `dImplicitEnergyFunction_RTP_SB(Grid &grid, Parameters ¶meters, Time &time, double dTemps[], int i, int`

Using $E_{i,j,k}^{n+1/2}$ for $E_{i+1/2,j,k}^{n+1/2}$

Using centered gradient for upwind gradient when motion is into the star at the surface

Missing `grid.dLocalGridOld[grid.nT][i+1][0][0]` using flux equals $2\sigma T^4$ at surface.

Index

/home/cgeroux/SPHERLS/src/eos.cpp, [57](#)

/home/cgeroux/SPHERLS/src/eos.h, [58](#)

~eos

 eos, [14](#)

average3DTo1DBoundariesNew

 dataManipulation.cpp, [60](#)

 dataManipulation.h, [65](#)

average3DTo1DBoundariesOld

 dataManipulation.cpp, [60](#)

 dataManipulation.h, [66](#)

bAdiabatic

 Parameters, [45](#)

bDump

 Output, [42](#)

bEOSGammaLaw

 Parameters, [46](#)

bFileExists

 dataMonitoring.cpp, [72](#)

 dataMonitoring.h, [75](#)

bPrint

 Output, [42](#)

bVariableTimeStep

 Time, [52](#)

calDelt_CONST

 physEquations.cpp, [89](#)

 physEquations.h, [118](#)

calDelt_R_GL

 physEquations.cpp, [89](#)

 physEquations.h, [119](#)

calDelt_R_TEOS

 physEquations.cpp, [89](#)

 physEquations.h, [119](#)

calDelt_RT_GL

 physEquations.cpp, [89](#)

 physEquations.h, [119](#)

calDelt_RT_TEOS

 physEquations.cpp, [90](#)

 physEquations.h, [119](#)

calDelt_RTP_GL

 physEquations.cpp, [90](#)

 physEquations.h, [120](#)

calDelt_RTP_TEOS

 physEquations.cpp, [90](#)

 physEquations.h, [120](#)

calNewD_R

 physEquations.cpp, [91](#)

 physEquations.h, [120](#)

calNewD_RT

 physEquations.cpp, [91](#)

 physEquations.h, [121](#)

calNewD_RTP

 physEquations.cpp, [91](#)

 physEquations.h, [121](#)

calNewDenave_None

 physEquations.cpp, [91](#)

 physEquations.h, [121](#)

calNewDenave_R

 physEquations.cpp, [92](#)

 physEquations.h, [121](#)

calNewDenave_RT

 physEquations.cpp, [92](#)

 physEquations.h, [122](#)

calNewDenave_RTP

 physEquations.cpp, [92](#)

 physEquations.h, [122](#)

calNewE_R_AD

 physEquations.cpp, [92](#)

 physEquations.h, [122](#)

calNewE_R_NA

 physEquations.cpp, [93](#)

 physEquations.h, [122](#)

calNewE_R_NA_LES

 physEquations.cpp, [93](#)

 physEquations.h, [122](#)

calNewE_RT_AD

 physEquations.cpp, [93](#)

 physEquations.h, [123](#)

calNewE_RT_NA

 physEquations.cpp, [93](#)

 physEquations.h, [123](#)

calNewE_RT_NA_LES

 physEquations.cpp, [94](#)

 physEquations.h, [123](#)

calNewE_RTP_AD

 physEquations.cpp, [94](#)

 physEquations.h, [123](#)

calNewE_RTP_NA

- physEquations.cpp, 94
- physEquations.h, 123
- calNewE_RTP_NA_LES
 - physEquations.cpp, 94
 - physEquations.h, 124
- calNewEddyVisc_None
 - physEquations.cpp, 95
 - physEquations.h, 124
- calNewEddyVisc_R_CN
 - physEquations.cpp, 95
 - physEquations.h, 124
- calNewEddyVisc_R_SM
 - physEquations.cpp, 95
 - physEquations.h, 124
- calNewEddyVisc_RT_CN
 - physEquations.cpp, 95
 - physEquations.h, 125
- calNewEddyVisc_RT_SM
 - physEquations.cpp, 95
 - physEquations.h, 125
- calNewEddyVisc_RTP_CN
 - physEquations.cpp, 96
 - physEquations.h, 125
- calNewEddyVisc_RTP_SM
 - physEquations.cpp, 96
 - physEquations.h, 125
- calNewP_GL
 - physEquations.cpp, 96
 - physEquations.h, 125
- calNewPEKappaGamma_TEOS
 - physEquations.cpp, 96
 - physEquations.h, 126
- calNewQ0_R_GL
 - physEquations.cpp, 97
 - physEquations.h, 126
- calNewQ0_R_TEOS
 - physEquations.cpp, 97
 - physEquations.h, 126
- calNewQ0Q1_RT_GL
 - physEquations.cpp, 97
 - physEquations.h, 126
- calNewQ0Q1_RT_TEOS
 - physEquations.cpp, 97
 - physEquations.h, 127
- calNewQ0Q1Q2_RTP_GL
 - physEquations.cpp, 98
 - physEquations.h, 127
- calNewQ0Q1Q2_RTP_TEOS
 - physEquations.cpp, 98
 - physEquations.h, 127
- calNewR
 - physEquations.cpp, 98
 - physEquations.h, 127
- calNewTPKappaGamma_TEOS
 - physEquations.cpp, 98
 - physEquations.h, 128
- calNewU0_R
 - physEquations.cpp, 99
 - physEquations.h, 128
- calNewU0_RT
 - physEquations.cpp, 99
 - physEquations.h, 128
- calNewU0_RTP
 - physEquations.cpp, 99
 - physEquations.h, 128
- calNewU_R
 - physEquations.cpp, 99
 - physEquations.h, 129
- calNewU_R_LES
 - physEquations.cpp, 99
 - physEquations.h, 129
- calNewU_RT
 - physEquations.cpp, 100
 - physEquations.h, 129
- calNewU_RT_LES
 - physEquations.cpp, 100
 - physEquations.h, 129
- calNewU_RTP
 - physEquations.cpp, 100
 - physEquations.h, 130
- calNewU_RTP_LES
 - physEquations.cpp, 101
 - physEquations.h, 130
- calNewV_RT
 - physEquations.cpp, 101
 - physEquations.h, 130
- calNewV_RT_LES
 - physEquations.cpp, 101
 - physEquations.h, 131
- calNewV_RTP
 - physEquations.cpp, 101
 - physEquations.h, 131
- calNewV_RTP_LES
 - physEquations.cpp, 102
 - physEquations.h, 131
- calNewVelocities_R
 - physEquations.cpp, 102
 - physEquations.h, 131
- calNewVelocities_R_LES
 - physEquations.cpp, 102
 - physEquations.h, 131
- calNewVelocities_RT
 - physEquations.cpp, 102
 - physEquations.h, 132
- calNewVelocities_RT_LES
 - physEquations.cpp, 103
 - physEquations.h, 132
- calNewVelocities_RTP

- physEquations.cpp, [103](#)
- physEquations.h, [132](#)
- calNewVelocities_RTP_LES
 - physEquations.cpp, [103](#)
 - physEquations.h, [133](#)
- calNewW_RTP
 - physEquations.cpp, [104](#)
 - physEquations.h, [133](#)
- calNewW_RTP_LES
 - physEquations.cpp, [104](#)
 - physEquations.h, [133](#)
- calOldDenave_None
 - physEquations.cpp, [104](#)
 - physEquations.h, [134](#)
- calOldDenave_R
 - physEquations.cpp, [104](#)
 - physEquations.h, [134](#)
- calOldDenave_RT
 - physEquations.cpp, [105](#)
 - physEquations.h, [134](#)
- calOldDenave_RTP
 - physEquations.cpp, [105](#)
 - physEquations.h, [134](#)
- calOldEddyVisc_R_CN
 - physEquations.cpp, [105](#)
 - physEquations.h, [134](#)
- calOldEddyVisc_R_SM
 - physEquations.cpp, [105](#)
 - physEquations.h, [134](#)
- calOldEddyVisc_RT_CN
 - physEquations.cpp, [105](#)
 - physEquations.h, [135](#)
- calOldEddyVisc_RT_SM
 - physEquations.cpp, [106](#)
 - physEquations.h, [135](#)
- calOldEddyVisc_RTP_CN
 - physEquations.cpp, [106](#)
 - physEquations.h, [135](#)
- calOldEddyVisc_RTP_SM
 - physEquations.cpp, [106](#)
 - physEquations.h, [135](#)
- calOldP_GL
 - physEquations.cpp, [106](#)
 - physEquations.h, [136](#)
- calOldPEKappaGamma_TEOS
 - physEquations.cpp, [107](#)
 - physEquations.h, [136](#)
- calOldQ0_R_GL
 - physEquations.cpp, [107](#)
 - physEquations.h, [136](#)
- calOldQ0_R_TEOS
 - physEquations.cpp, [107](#)
 - physEquations.h, [136](#)
- calOldQ0Q1_RT_GL
 - physEquations.cpp, [107](#)
 - physEquations.h, [137](#)
- calOldQ0Q1_RT_TEOS
 - physEquations.cpp, [108](#)
 - physEquations.h, [137](#)
- calOldQ0Q1Q2_RTP_GL
 - physEquations.cpp, [108](#)
 - physEquations.h, [137](#)
- calOldQ0Q1Q2_RTP_TEOS
 - physEquations.cpp, [108](#)
 - physEquations.h, [137](#)
- dA
 - Parameters, [46](#)
- dAlpha
 - Parameters, [46](#)
- dataManipulation.cpp, [59](#)
- dataManipulation.cpp
 - average3DTo1DBoundariesNew, [60](#)
 - average3DTo1DBoundariesOld, [60](#)
 - fin, [60](#)
 - init, [61](#)
 - initImplicitCalculation, [61](#)
 - initUpdateLocalBoundaries, [62](#)
 - modelRead, [62](#)
 - modelWrite_GL, [62](#)
 - modelWrite_TEOS, [63](#)
 - setupLocalGrid, [63](#)
 - updateLocalBoundaries, [63](#)
 - updateLocalBoundariesNewGrid, [63](#)
 - updateLocalBoundaryVelocitiesNewGrid_-R, [64](#)
 - updateLocalBoundaryVelocitiesNewGrid_-RT, [64](#)
 - updateLocalBoundaryVelocitiesNewGrid_-RTP, [64](#)
 - updateNewGridWithOld, [64](#)
 - updateOldGrid, [64](#)
- dataManipulation.h, [65](#)
- dataManipulation.h
 - average3DTo1DBoundariesNew, [65](#)
 - average3DTo1DBoundariesOld, [66](#)
 - fin, [66](#)
 - init, [66](#)
 - initImplicitCalculation, [67](#)
 - initUpdateLocalBoundaries, [67](#)
 - modelRead, [68](#)
 - modelWrite_GL, [68](#)
 - modelWrite_TEOS, [68](#)
 - setupLocalGrid, [69](#)
 - updateLocalBoundaries, [69](#)
 - updateLocalBoundariesNewGrid, [69](#)
 - updateLocalBoundaryVelocitiesNewGrid_-R, [69](#)

- updateLocalBoundaryVelocitiesNewGrid_-RT, 70
- updateLocalBoundaryVelocitiesNewGrid_-RTP, 70
- updateNewGridWithOld, 70
- updateOldGrid, 70
- dataMonitoring.cpp, 71
- dataMonitoring.h
 - bFileExists, 72
 - finWatchZones, 72
 - initWatchZones, 72
 - writeWatchZones_R_GL, 72
 - writeWatchZones_R_TEOS, 73
 - writeWatchZones_RT_GL, 73
 - writeWatchZones_RT_TEOS, 73
 - writeWatchZones_RTP_GL, 74
 - writeWatchZones_RTP_TEOS, 74
- dataMonitoring.h, 75
- dataMonitoring.h
 - bFileExists, 75
 - finWatchZones, 75
 - initWatchZones, 76
 - writeWatchZones_R_GL, 76
 - writeWatchZones_R_TEOS, 76
 - writeWatchZones_RT_GL, 77
 - writeWatchZones_RT_TEOS, 77
 - writeWatchZones_RTP_GL, 77
 - writeWatchZones_RTP_TEOS, 78
- dAverageRHS
 - Implicit, 37
- dAVThreshold
 - Parameters, 46
- dConstTimeStep
 - Time, 52
- dCurrentRelTErr
 - Implicit, 37
- dDelE_t_E_max
 - Time, 53
- dDelRho_t_Rho_max
 - Time, 53
- dDelT_t_T_max
 - Time, 53
- dDeltat_n
 - Time, 53
- dDeltat_nm1half
 - Time, 53
- dDeltat_np1half
 - Time, 53
- dDerivativeStepFraction
 - Implicit, 37
- dDonorFrac
 - Parameters, 46
- dDRhoDP
 - eos, 14
- dDumpFrequencyTime
 - Output, 43
- dEddyViscosity
 - Parameters, 46
- dEndTime
 - Time, 53
- dEndTimer
 - Performance, 48
- dEOS_GL
 - physEquations.cpp, 108
 - physEquations.h, 138
- dG
 - Parameters, 46
- dGamma
 - Parameters, 46
- dGetEnergy
 - eos, 15
- dGetOpacity
 - eos, 15
- dGetPressure
 - eos, 15
- dImplicitEnergyFunction_None
 - physEquations.cpp, 109
 - physEquations.h, 138
- dImplicitEnergyFunction_R
 - physEquations.cpp, 109
 - physEquations.h, 138
- dImplicitEnergyFunction_R_LES
 - physEquations.cpp, 109
 - physEquations.h, 139
- dImplicitEnergyFunction_R_LES_SB
 - physEquations.cpp, 110
 - physEquations.h, 139
- dImplicitEnergyFunction_R_SB
 - physEquations.cpp, 110
 - physEquations.h, 139
- dImplicitEnergyFunction_RT
 - physEquations.cpp, 110
 - physEquations.h, 139
- dImplicitEnergyFunction_RT_LES
 - physEquations.cpp, 111
 - physEquations.h, 140
- dImplicitEnergyFunction_RT_LES_SB
 - physEquations.cpp, 111
 - physEquations.h, 140
- dImplicitEnergyFunction_RT_SB
 - physEquations.cpp, 111
 - physEquations.h, 141
- dImplicitEnergyFunction_RTP
 - physEquations.cpp, 111
 - physEquations.h, 141
- dImplicitEnergyFunction_RTP_LES
 - physEquations.cpp, 112
 - physEquations.h, 141

- dImplicitEnergyFunction_RTP_LES_SB
 - physEquations.cpp, [112](#)
 - physEquations.h, [142](#)
- dImplicitEnergyFunction_RTP_SB
 - physEquations.cpp, [113](#)
 - physEquations.h, [142](#)
- dLocalGridNew
 - Grid, [29](#)
- dLocalGridOld
 - Grid, [30](#)
- dLogE
 - eos, [19](#)
- dLogKappa
 - eos, [19](#)
- dLogP
 - eos, [19](#)
- dLogRhoDelta
 - eos, [19](#)
- dLogRhoMin
 - eos, [19](#)
- dLogTDelta
 - eos, [19](#)
- dLogTMin
 - eos, [19](#)
- dMaxConvectiveVelocity
 - Parameters, [46](#)
- dMaxConvectiveVelocity_c
 - Parameters, [46](#)
- dMaxErrorInRHS
 - Implicit, [37](#)
- dPerChange
 - Time, [53](#)
- dPi
 - Parameters, [47](#)
- dPrintFrequencyTime
 - Output, [43](#)
- dSigma
 - Parameters, [47](#)
- dSoundSpeed
 - eos, [15](#)
- dStartTimer
 - Performance, [48](#)
- dt
 - Time, [53](#)
- dTimeLastDump
 - Output, [43](#)
- dTimeLastPrint
 - Output, [43](#)
- dTimeStepFactor
 - Time, [53](#)
- dTolerance
 - Implicit, [37](#)
 - Parameters, [47](#)
- DUMP_VERSION
 - global.h, [80](#)
- dXMassFrac
 - eos, [20](#)
- dYMassFrac
 - eos, [20](#)
- eos, [13](#)
 - ~eos, [14](#)
 - dDRhoDP, [14](#)
 - dGetEnergy, [15](#)
 - dGetOpacity, [15](#)
 - dGetPressure, [15](#)
 - dLogE, [19](#)
 - dLogKappa, [19](#)
 - dLogP, [19](#)
 - dLogRhoDelta, [19](#)
 - dLogRhoMin, [19](#)
 - dLogTDelta, [19](#)
 - dLogTMin, [19](#)
 - dSoundSpeed, [15](#)
 - dXMassFrac, [20](#)
 - dYMassFrac, [20](#)
 - eos, [14](#)
 - gamma1DelAdC_v, [16](#)
 - getEAndDTDE, [16](#)
 - getEKappa, [16](#)
 - getPAndDRhoDP, [16](#)
 - getPEKappa, [17](#)
 - getPEKappaGamma, [17](#)
 - getPKappaGamma, [17](#)
 - nNumRho, [20](#)
 - nNumT, [20](#)
 - operator=, [18](#)
 - readAscii, [18](#)
 - readBin, [18](#)
 - readBobsAscii, [18](#)
 - writeAscii, [18](#)
 - writeBin, [19](#)
- eosTable
 - Parameters, [47](#)
- fin
 - dataManipulation.cpp, [60](#)
 - dataManipulation.h, [66](#)
- finWatchZones
 - dataMonitoring.cpp, [72](#)
 - dataMonitoring.h, [75](#)
- fpCalculateAveDensities
 - Functions, [21](#)
- fpCalculateDeltat
 - Functions, [22](#)
- fpCalculateNewAV
 - Functions, [22](#)
- fpCalculateNewDensities

- Functions, 22
- fpCalculateNewEddyVisc
 - Functions, 22
- fpCalculateNewEnergies
 - Functions, 22
- fpCalculateNewEOSVars
 - Functions, 22
- fpCalculateNewGridVelocities
 - Functions, 22
- fpCalculateNewRadii
 - Functions, 22
- fpCalculateNewVelocities
 - Functions, 22
- fpImplicitSolve
 - Functions, 22
- fpModelWrite
 - Functions, 22
- fpUpdateLocalBoundaryVelocitiesNewGrid
 - Functions, 23
- fpWriteWatchZones
 - Functions, 23
- Functions, 21
 - fpCalculateAveDensities, 21
 - fpCalculateDeltat, 22
 - fpCalculateNewAV, 22
 - fpCalculateNewDensities, 22
 - fpCalculateNewEddyVisc, 22
 - fpCalculateNewEnergies, 22
 - fpCalculateNewEOSVars, 22
 - fpCalculateNewGridVelocities, 22
 - fpCalculateNewRadii, 22
 - fpCalculateNewVelocities, 22
 - fpImplicitSolve, 22
 - fpModelWrite, 22
 - fpUpdateLocalBoundaryVelocitiesNew-Grid, 23
 - fpWriteWatchZones, 23
- Functions, 21
- functions
 - Global, 24
- gamma1DelAdC_v
 - eos, 16
- getEAndDTDE
 - eos, 16
- getEKappa
 - eos, 16
- getPAndDRhoDP
 - eos, 16
- getPEKappa
 - eos, 17
- getPEKappaGamma
 - eos, 17
- getPKappaGamma

- eos, 17
- Global, 24
 - functions, 24
- Global, 24
 - grid, 24
 - implicit, 24
 - messPass, 25
 - output, 25
 - parameters, 25
 - performance, 25
 - procTop, 25
 - time, 25
- global.cpp, 79
- global.h, 80
 - DUMP_VERSION, 80
 - SEDOV, 81
 - SIGNEGDN, 81
 - SIGNEGENG, 81
 - SIGNEGTEMP, 81
 - TRACKMAXSOLVERERROR, 81
 - VISCOUS_ENERGY_EQ, 81
- Grid, 26
 - dLocalGridNew, 29
 - dLocalGridOld, 30
 - Grid, 29
 - nCenIntOffset, 30
 - nCotThetaIJK, 30
 - nCotThetaIjp1halfK, 30
 - nD, 30
 - nDCosThetaIJK, 30
 - nDenAve, 30
 - nDM, 30
 - nDPhi, 31
 - nDTheta, 31
 - nE, 31
 - nEddyVisc, 31
 - nEndGhostUpdateExplicit, 31
 - nEndGhostUpdateImplicit, 31
 - nEndUpdateExplicit, 31
 - nEndUpdateImplicit, 31
 - nGamma, 32
 - nGlobalGridDims, 32
 - nGlobalGridPositionLocalGrid, 32
 - nKappa, 32
 - nLocalGridDims, 32
 - nM, 32
 - nNum1DZones, 32
 - nNumDims, 32
 - nNumGhostCells, 33
 - nNumIntVars, 33
 - nNumVars, 33
 - nP, 33
 - nPhi, 33
 - nQ0, 33

- nQ1, [33](#)
- nQ2, [33](#)
- nR, [34](#)
- nSinThetaIJK, [34](#)
- nSinThetaIjp1halfK, [34](#)
- nStartGhostUpdateExplicit, [34](#)
- nStartGhostUpdateImplicit, [34](#)
- nStartUpdateExplicit, [34](#)
- nStartUpdateImplicit, [34](#)
- nT, [34](#)
- nTheta, [35](#)
- nU, [35](#)
- nU0, [35](#)
- nV, [35](#)
- nVariables, [35](#)
- nW, [35](#)
- grid
 - Global, [24](#)
- Implicit, [36](#)
 - dAverageRHS, [37](#)
 - dCurrentRelTErr, [37](#)
 - dDerivativeStepFraction, [37](#)
 - dMaxErrorInRHS, [37](#)
 - dTolerance, [37](#)
 - Implicit, [36](#)
 - kspContext, [37](#)
 - matCoeff, [37](#)
 - nCurrentNumIterations, [37](#)
 - nLocDer, [37](#)
 - nLocFun, [37](#)
 - nMaxNumIterations, [38](#)
 - nMaxNumSolverIterations, [38](#)
 - nNumDerPerRow, [38](#)
 - nNumImplicitZones, [38](#)
 - nNumRowsALocal, [38](#)
 - nNumRowsALocalSB, [38](#)
 - nTypeDer, [38](#)
 - vecRHS, [38](#)
 - vecscatTCorrections, [38](#)
 - vecTCorrections, [39](#)
 - vecTCorrectionsLocal, [39](#)
- implicit
 - Global, [24](#)
- implicitSolve_None
 - physEquations.cpp, [113](#)
 - physEquations.h, [142](#)
- implicitSolve_R
 - physEquations.cpp, [113](#)
 - physEquations.h, [142](#)
- implicitSolve_RT
 - physEquations.cpp, [113](#)
 - physEquations.h, [143](#)
- implicitSolve_RTP
 - physEquations.cpp, [113](#)
 - physEquations.h, [143](#)
- init
 - dataManipulation.cpp, [61](#)
 - dataManipulation.h, [66](#)
- initDonorFracAndMaxConVel_R_GL
 - physEquations.cpp, [114](#)
 - physEquations.h, [143](#)
- initDonorFracAndMaxConVel_R_TEOS
 - physEquations.cpp, [114](#)
 - physEquations.h, [143](#)
- initDonorFracAndMaxConVel_RT_GL
 - physEquations.cpp, [114](#)
 - physEquations.h, [143](#)
- initDonorFracAndMaxConVel_RT_TEOS
 - physEquations.cpp, [114](#)
 - physEquations.h, [144](#)
- initDonorFracAndMaxConVel_RTP_GL
 - physEquations.cpp, [114](#)
 - physEquations.h, [144](#)
- initDonorFracAndMaxConVel_RTP_TEOS
 - physEquations.cpp, [114](#)
 - physEquations.h, [144](#)
- initImplicitCalculation
 - dataManipulation.cpp, [61](#)
 - dataManipulation.h, [67](#)
- initInternalVars
 - physEquations.cpp, [115](#)
 - physEquations.h, [144](#)
- initUpdateLocalBoundaries
 - dataManipulation.cpp, [62](#)
 - dataManipulation.h, [67](#)
- initWatchZones
 - dataMonitoring.cpp, [72](#)
 - dataMonitoring.h, [76](#)
- kspContext
 - Implicit, [37](#)
- main
 - main.cpp, [82](#)
 - main.h, [84](#)
- main.cpp, [82](#)
 - main, [82](#)
 - signalHandler, [83](#)
- main.h, [84](#)
 - main, [84](#)
 - signalHandler, [85](#)
- matCoeff
 - Implicit, [37](#)
- MessPass, [40](#)
 - MessPass, [40](#)
- MessPass
 - MessPass, [40](#)

- requestRecv, [40](#)
- requestSend, [40](#)
- statusRecv, [40](#)
- statusSend, [41](#)
- typeRecvNewVar, [41](#)
- typeRecvOldGrid, [41](#)
- typeSendNewGrid, [41](#)
- typeSendNewVar, [41](#)
- messPass
 - Global, [25](#)
- modelRead
 - dataManipulation.cpp, [62](#)
 - dataManipulation.h, [68](#)
- modelWrite_GL
 - dataManipulation.cpp, [62](#)
 - dataManipulation.h, [68](#)
- modelWrite_TEOS
 - dataManipulation.cpp, [63](#)
 - dataManipulation.h, [68](#)
- nCenIntOffset
 - Grid, [30](#)
- nCoords
 - ProcTop, [49](#)
- nCotThetaIJK
 - Grid, [30](#)
- nCotThetaIjp1halfK
 - Grid, [30](#)
- nCurrentNumIterations
 - Implicit, [37](#)
- nD
 - Grid, [30](#)
- nDCosThetaIJK
 - Grid, [30](#)
- nDenAve
 - Grid, [30](#)
- nDM
 - Grid, [30](#)
- nDPhi
 - Grid, [31](#)
- nDTheta
 - Grid, [31](#)
- nDumpFrequencyStep
 - Output, [43](#)
- nE
 - Grid, [31](#)
- nEddyVisc
 - Grid, [31](#)
- nEndGhostUpdateExplicit
 - Grid, [31](#)
- nEndGhostUpdateImplicit
 - Grid, [31](#)
- nEndUpdateExplicit
 - Grid, [31](#)
- nEndUpdateImplicit
 - Grid, [31](#)
- nGamma
 - Grid, [32](#)
- nGlobalGridDims
 - Grid, [32](#)
- nGlobalGridPositionLocalGrid
 - Grid, [32](#)
- nKappa
 - Grid, [32](#)
- nLocalGridDims
 - Grid, [32](#)
- nLocDer
 - Implicit, [37](#)
- nLocFun
 - Implicit, [37](#)
- nM
 - Grid, [32](#)
- nMaxIterations
 - Parameters, [47](#)
- nMaxNumIterations
 - Implicit, [38](#)
- nMaxNumSolverIterations
 - Implicit, [38](#)
- nNeighborRanks
 - ProcTop, [49](#)
- nNum1DZones
 - Grid, [32](#)
- nNumDerPerRow
 - Implicit, [38](#)
- nNumDims
 - Grid, [32](#)
- nNumGhostCells
 - Grid, [33](#)
- nNumImplicitZones
 - Implicit, [38](#)
- nNumIntVars
 - Grid, [33](#)
- nNumNeighbors
 - ProcTop, [50](#)
- nNumProcs
 - ProcTop, [50](#)
- nNumRadialNeighbors
 - ProcTop, [50](#)
- nNumRho
 - eos, [20](#)
- nNumRowsALocal
 - Implicit, [38](#)
- nNumRowsALocalSB
 - Implicit, [38](#)
- nNumT
 - eos, [20](#)
- nNumTimeStepsSinceLastPrint
 - Output, [43](#)

- nNumVars
 - Grid, [33](#)
- nP
 - Grid, [33](#)
- nPeriodic
 - ProcTop, [50](#)
- nPhi
 - Grid, [33](#)
- nPrintFrequencyStep
 - Output, [43](#)
- nPrintMode
 - Output, [43](#)
- nProcDims
 - ProcTop, [50](#)
- nQ0
 - Grid, [33](#)
- nQ1
 - Grid, [33](#)
- nQ2
 - Grid, [33](#)
- nR
 - Grid, [34](#)
- nRadialNeighborNeighborIDs
 - ProcTop, [50](#)
- nRadialNeighborRanks
 - ProcTop, [50](#)
- nRank
 - ProcTop, [50](#)
- nSinThetaJK
 - Grid, [34](#)
- nSinThetaJp1halfK
 - Grid, [34](#)
- nStartGhostUpdateExplicit
 - Grid, [34](#)
- nStartGhostUpdateImplicit
 - Grid, [34](#)
- nStartUpdateExplicit
 - Grid, [34](#)
- nStartUpdateImplicit
 - Grid, [34](#)
- nT
 - Grid, [34](#)
- nTheta
 - Grid, [35](#)
- nTimeStepIndex
 - Time, [54](#)
- nTypeDer
 - Implicit, [38](#)
- nTypeTurbulenceMod
 - Parameters, [47](#)
- nU
 - Grid, [35](#)
- nU0
 - Grid, [35](#)
- nV
 - Grid, [35](#)
- nVariables
 - Grid, [35](#)
- nW
 - Grid, [35](#)
- ofWatchZoneFiles
 - Output, [43](#)
- operator=
 - eos, [18](#)
- Output, [42](#)
 - bDump, [42](#)
 - bPrint, [42](#)
 - dDumpFrequencyTime, [43](#)
 - dPrintFrequencyTime, [43](#)
 - dTimeLastDump, [43](#)
 - dTimeLastPrint, [43](#)
 - nDumpFrequencyStep, [43](#)
 - nNumTimeStepsSinceLastPrint, [43](#)
 - nPrintFrequencyStep, [43](#)
 - nPrintMode, [43](#)
 - ofWatchZoneFiles, [43](#)
 - Output, [42](#)
 - sBaseOutputFileName, [43](#)
 - watchzoneList, [44](#)
- output
 - Global, [25](#)
- Parameters, [45](#)
 - bAdiabatic, [45](#)
 - bEOSGammaLaw, [46](#)
 - dA, [46](#)
 - dAlpha, [46](#)
 - dAVThreshold, [46](#)
 - dDonorFrac, [46](#)
 - dEddyViscosity, [46](#)
 - dG, [46](#)
 - dGamma, [46](#)
 - dMaxConvectiveVelocity, [46](#)
 - dMaxConvectiveVelocity_c, [46](#)
 - dPi, [47](#)
 - dSigma, [47](#)
 - dTolerance, [47](#)
 - eosTable, [47](#)
 - nMaxIterations, [47](#)
 - nTypeTurbulenceMod, [47](#)
 - Parameters, [45](#)
 - sEOSFileName, [47](#)
- parameters
 - Global, [25](#)
- Performance, [48](#)
 - dEndTimer, [48](#)
 - dStartTimer, [48](#)

- Performance, 48
- performance
 - Global, 25
- physEquations.cpp, 86
- physEquations.cpp
 - calDelt_CONST, 89
 - calDelt_R_GL, 89
 - calDelt_R_TEOS, 89
 - calDelt_RT_GL, 89
 - calDelt_RT_TEOS, 90
 - calDelt_RTP_GL, 90
 - calDelt_RTP_TEOS, 90
 - calNewD_R, 91
 - calNewD_RT, 91
 - calNewD_RTP, 91
 - calNewDenave_None, 91
 - calNewDenave_R, 92
 - calNewDenave_RT, 92
 - calNewDenave_RTP, 92
 - calNewE_R_AD, 92
 - calNewE_R_NA, 93
 - calNewE_R_NA_LES, 93
 - calNewE_RT_AD, 93
 - calNewE_RT_NA, 93
 - calNewE_RT_NA_LES, 94
 - calNewE_RTP_AD, 94
 - calNewE_RTP_NA, 94
 - calNewE_RTP_NA_LES, 94
 - calNewEddyVisc_None, 95
 - calNewEddyVisc_R_CN, 95
 - calNewEddyVisc_R_SM, 95
 - calNewEddyVisc_RT_CN, 95
 - calNewEddyVisc_RT_SM, 95
 - calNewEddyVisc_RTP_CN, 96
 - calNewEddyVisc_RTP_SM, 96
 - calNewP_GL, 96
 - calNewPEKappaGamma_TEOS, 96
 - calNewQ0_R_GL, 97
 - calNewQ0_R_TEOS, 97
 - calNewQ0Q1_RT_GL, 97
 - calNewQ0Q1_RT_TEOS, 97
 - calNewQ0Q1Q2_RTP_GL, 98
 - calNewQ0Q1Q2_RTP_TEOS, 98
 - calNewR, 98
 - calNewTPKappaGamma_TEOS, 98
 - calNewU0_R, 99
 - calNewU0_RT, 99
 - calNewU0_RTP, 99
 - calNewU_R, 99
 - calNewU_R_LES, 99
 - calNewU_RT, 100
 - calNewU_RT_LES, 100
 - calNewU_RTP, 100
 - calNewU_RTP_LES, 101
 - calNewV_RT, 101
 - calNewV_RT_LES, 101
 - calNewV_RTP, 101
 - calNewV_RTP_LES, 102
 - calNewVelocities_R, 102
 - calNewVelocities_R_LES, 102
 - calNewVelocities_RT, 102
 - calNewVelocities_RT_LES, 103
 - calNewVelocities_RTP, 103
 - calNewVelocities_RTP_LES, 103
 - calNewW_RTP, 104
 - calNewW_RTP_LES, 104
 - calOldDenave_None, 104
 - calOldDenave_R, 104
 - calOldDenave_RT, 105
 - calOldDenave_RTP, 105
 - calOldEddyVisc_R_CN, 105
 - calOldEddyVisc_R_SM, 105
 - calOldEddyVisc_RT_CN, 105
 - calOldEddyVisc_RT_SM, 106
 - calOldEddyVisc_RTP_CN, 106
 - calOldEddyVisc_RTP_SM, 106
 - calOldP_GL, 106
 - calOldPEKappaGamma_TEOS, 107
 - calOldQ0_R_GL, 107
 - calOldQ0_R_TEOS, 107
 - calOldQ0Q1_RT_GL, 107
 - calOldQ0Q1_RT_TEOS, 108
 - calOldQ0Q1Q2_RTP_GL, 108
 - calOldQ0Q1Q2_RTP_TEOS, 108
 - dEOS_GL, 108
 - dImplicitEnergyFunction_None, 109
 - dImplicitEnergyFunction_R, 109
 - dImplicitEnergyFunction_R_LES, 109
 - dImplicitEnergyFunction_R_LES_SB, 110
 - dImplicitEnergyFunction_R_SB, 110
 - dImplicitEnergyFunction_RT, 110
 - dImplicitEnergyFunction_RT_LES, 111
 - dImplicitEnergyFunction_RT_LES_SB, 111
 - dImplicitEnergyFunction_RT_SB, 111
 - dImplicitEnergyFunction_RTP, 111
 - dImplicitEnergyFunction_RTP_LES, 112
 - dImplicitEnergyFunction_RTP_LES_SB, 112
 - dImplicitEnergyFunction_RTP_SB, 113
 - implicitSolve_None, 113
 - implicitSolve_R, 113
 - implicitSolve_RT, 113
 - implicitSolve_RTP, 113
 - initDonorFracAndMaxConVel_R_GL, 114
 - initDonorFracAndMaxConVel_R_TEOS, 114

- initDonorFracAndMaxConVel_RT_GL, 114
- initDonorFracAndMaxConVel_RT_TEOS, 114
- initDonorFracAndMaxConVel_RTP_GL, 114
- initDonorFracAndMaxConVel_RTP_-TEOS, 114
- initInternalVars, 115
- setInternalVarInf, 115
- setMainFunctions, 115
- physEquations.h, 116
- physEquations.h
 - calDelt_CONST, 118
 - calDelt_R_GL, 119
 - calDelt_R_TEOS, 119
 - calDelt_RT_GL, 119
 - calDelt_RT_TEOS, 119
 - calDelt_RTP_GL, 120
 - calDelt_RTP_TEOS, 120
 - calNewD_R, 120
 - calNewD_RT, 121
 - calNewD_RTP, 121
 - calNewDenave_None, 121
 - calNewDenave_R, 121
 - calNewDenave_RT, 122
 - calNewDenave_RTP, 122
 - calNewE_R_AD, 122
 - calNewE_R_NA, 122
 - calNewE_R_NA_LES, 122
 - calNewE_RT_AD, 123
 - calNewE_RT_NA, 123
 - calNewE_RT_NA_LES, 123
 - calNewE_RTP_AD, 123
 - calNewE_RTP_NA, 123
 - calNewE_RTP_NA_LES, 124
 - calNewEddyVisc_None, 124
 - calNewEddyVisc_R_CN, 124
 - calNewEddyVisc_R_SM, 124
 - calNewEddyVisc_RT_CN, 125
 - calNewEddyVisc_RT_SM, 125
 - calNewEddyVisc_RTP_CN, 125
 - calNewEddyVisc_RTP_SM, 125
 - calNewP_GL, 125
 - calNewPEKappaGamma_TEOS, 126
 - calNewQ0_R_GL, 126
 - calNewQ0_R_TEOS, 126
 - calNewQ0Q1_RT_GL, 126
 - calNewQ0Q1_RT_TEOS, 127
 - calNewQ0Q1Q2_RTP_GL, 127
 - calNewQ0Q1Q2_RTP_TEOS, 127
 - calNewR, 127
 - calNewTPKappaGamma_TEOS, 128
 - calNewU0_R, 128
 - calNewU0_RT, 128
 - calNewU0_RTP, 128
 - calNewU_R, 129
 - calNewU_R_LES, 129
 - calNewU_RT, 129
 - calNewU_RT_LES, 129
 - calNewU_RTP, 130
 - calNewU_RTP_LES, 130
 - calNewV_RT, 130
 - calNewV_RT_LES, 131
 - calNewV_RTP, 131
 - calNewV_RTP_LES, 131
 - calNewVelocities_R, 131
 - calNewVelocities_R_LES, 131
 - calNewVelocities_RT, 132
 - calNewVelocities_RT_LES, 132
 - calNewVelocities_RTP, 132
 - calNewVelocities_RTP_LES, 133
 - calNewW_RTP, 133
 - calNewW_RTP_LES, 133
 - calOldDenave_None, 134
 - calOldDenave_R, 134
 - calOldDenave_RT, 134
 - calOldDenave_RTP, 134
 - calOldEddyVisc_R_CN, 134
 - calOldEddyVisc_R_SM, 134
 - calOldEddyVisc_RT_CN, 135
 - calOldEddyVisc_RT_SM, 135
 - calOldEddyVisc_RTP_CN, 135
 - calOldEddyVisc_RTP_SM, 135
 - calOldP_GL, 136
 - calOldPEKappaGamma_TEOS, 136
 - calOldQ0_R_GL, 136
 - calOldQ0_R_TEOS, 136
 - calOldQ0Q1_RT_GL, 137
 - calOldQ0Q1_RT_TEOS, 137
 - calOldQ0Q1Q2_RTP_GL, 137
 - calOldQ0Q1Q2_RTP_TEOS, 137
 - dEOS_GL, 138
 - dImplicitEnergyFunction_None, 138
 - dImplicitEnergyFunction_R, 138
 - dImplicitEnergyFunction_R_LES, 139
 - dImplicitEnergyFunction_R_LES_SB, 139
 - dImplicitEnergyFunction_R_SB, 139
 - dImplicitEnergyFunction_RT, 139
 - dImplicitEnergyFunction_RT_LES, 140
 - dImplicitEnergyFunction_RT_LES_SB, 140
 - dImplicitEnergyFunction_RT_SB, 141
 - dImplicitEnergyFunction_RTP, 141
 - dImplicitEnergyFunction_RTP_LES, 141
 - dImplicitEnergyFunction_RTP_LES_SB, 142
 - dImplicitEnergyFunction_RTP_SB, 142

- implicitSolve_None, [142](#)
- implicitSolve_R, [142](#)
- implicitSolve_RT, [143](#)
- implicitSolve_RTP, [143](#)
- initDonorFracAndMaxConVel_R_GL, [143](#)
- initDonorFracAndMaxConVel_R_TEOS, [143](#)
- initDonorFracAndMaxConVel_RT_GL, [143](#)
- initDonorFracAndMaxConVel_RT_TEOS, [144](#)
- initDonorFracAndMaxConVel_RTP_GL, [144](#)
- initDonorFracAndMaxConVel_RTP_-TEOS, [144](#)
- initInternalVars, [144](#)
- setInternalVarInf, [144](#)
- setMainFunctions, [145](#)
- ProcTop, [49](#)
 - ProcTop, [49](#)
- ProcTop
 - nCoords, [49](#)
 - nNeighborRanks, [49](#)
 - nNumNeighbors, [50](#)
 - nNumProcs, [50](#)
 - nNumRadialNeighbors, [50](#)
 - nPeriodic, [50](#)
 - nProcDims, [50](#)
 - nRadialNeighborNeighborIDs, [50](#)
 - nRadialNeighborRanks, [50](#)
 - nRank, [50](#)
 - ProcTop, [49](#)
- procTop
 - Global, [25](#)
- readAscii
 - eos, [18](#)
- readBin
 - eos, [18](#)
- readBobsAscii
 - eos, [18](#)
- requestRecv
 - MessPass, [40](#)
- requestSend
 - MessPass, [40](#)
- sBaseOutputFileName
 - Output, [43](#)
- SED OV
 - global.h, [81](#)
- sEOSFileName
 - Parameters, [47](#)
- setInternalVarInf
 - physEquations.cpp, [115](#)
 - physEquations.h, [144](#)
- setMainFunctions
 - physEquations.cpp, [115](#)
 - physEquations.h, [145](#)
- setupLocalGrid
 - dataManipulation.cpp, [63](#)
 - dataManipulation.h, [69](#)
- signalHandler
 - main.cpp, [83](#)
 - main.h, [85](#)
- SIGNEG DEN
 - global.h, [81](#)
- SIGNEG ENG
 - global.h, [81](#)
- SIGNEG TEMP
 - global.h, [81](#)
- statusRecv
 - MessPass, [40](#)
- statusSend
 - MessPass, [41](#)
- Time, [52](#)
 - bVariableTimeStep, [52](#)
 - dConstTimeStep, [52](#)
 - dDelE_t_E_max, [53](#)
 - dDelRho_t_Rho_max, [53](#)
 - dDelT_t_T_max, [53](#)
 - dDeltat_n, [53](#)
 - dDeltat_nmlhalf, [53](#)
 - dDeltat_nplhalf, [53](#)
 - dEndTime, [53](#)
 - dPerChange, [53](#)
 - dt, [53](#)
 - dTimeStepFactor, [53](#)
 - nTimeStepIndex, [54](#)
 - Time, [52](#)
- time
 - Global, [25](#)
- TRACKMAXSOLVERERROR
 - global.h, [81](#)
- typeRecvNewVar
 - MessPass, [41](#)
- typeRecvOldGrid
 - MessPass, [41](#)
- typeSendNewGrid
 - MessPass, [41](#)
- typeSendNewVar
 - MessPass, [41](#)
- updateLocalBoundaries
 - dataManipulation.cpp, [63](#)
 - dataManipulation.h, [69](#)
- updateLocalBoundariesNewGrid
 - dataManipulation.cpp, [63](#)

dataManipulation.h, 69
updateLocalBoundaryVelocitiesNewGrid_R
dataManipulation.cpp, 64
dataManipulation.h, 69
updateLocalBoundaryVelocitiesNewGrid_RT
dataManipulation.cpp, 64
dataManipulation.h, 70
updateLocalBoundaryVelocitiesNewGrid_RTP
dataManipulation.cpp, 64
dataManipulation.h, 70
updateNewGridWithOld
dataManipulation.cpp, 64
dataManipulation.h, 70
updateOldGrid
dataManipulation.cpp, 64
dataManipulation.h, 70
userguide.h, 146

vecRHS
Implicit, 38
vecscatTCorrections
Implicit, 38
vecTCorrections
Implicit, 39
vecTCorrectionsLocal
Implicit, 39
VISCOUS_ENERGY_EQ
global.h, 81

WatchZone, 55
watchzone.cpp, 147
watchzone.h, 148
watchzoneList
Output, 44
writeAscii
eos, 18
writeBin
eos, 19
writeWatchZones_R_GL
dataMonitoring.cpp, 72
dataMonitoring.h, 76
writeWatchZones_R_TEOS
dataMonitoring.cpp, 73
dataMonitoring.h, 76
writeWatchZones_RT_GL
dataMonitoring.cpp, 73
dataMonitoring.h, 77
writeWatchZones_RT_TEOS
dataMonitoring.cpp, 73
dataMonitoring.h, 77
writeWatchZones_RTP_GL
dataMonitoring.cpp, 74
dataMonitoring.h, 77
writeWatchZones_RTP_TEOS