

FE2D Reference Manual

Generated by Doxygen 1.2.14

Sun Sep 15 07:55:37 2002

Contents

1	FE2D Hierarchical Index	1
1.1	FE2D Class Hierarchy	1
2	FE2D Compound Index	3
2.1	FE2D Compound List	3
3	FE2D File Index	5
3.1	FE2D File List	5
4	FE2D Class Documentation	7
4.1	AdvectionDiffusion Class Reference	7
4.2	Banner Class Reference	12
4.3	Beam2 Struct Reference	13
4.4	Conduction Class Reference	14
4.5	Control Class Reference	20
4.6	ConvectionBC Class Reference	24
4.7	DataContainer Class Reference	27
4.8	ElSet Struct Reference	31
4.9	fileIO Class Reference	32
4.10	FlowPhysics Class Reference	40
4.11	HeatFluxBC Class Reference	45
4.12	Hex8 Struct Reference	47
4.13	Material Class Reference	48
4.14	Mesh Class Reference	52
4.15	NavierStokes Class Reference	64
4.16	NodeBC Struct Reference	79
4.17	Nodeset Struct Reference	80
4.18	Physics Class Reference	81
4.19	Quad4 Struct Reference	91

4.20	SideBC Class Reference	92
4.21	Sideset Struct Reference	96
5	FE2D File Documentation	99
5.1	advection_diffusion.h File Reference	99
5.2	banner.h File Reference	101
5.3	bcs.h File Reference	103
5.4	conduction.h File Reference	104
5.5	control.h File Reference	105
5.6	cvdate.h File Reference	110
5.7	data_container.h File Reference	111
5.8	epsilon.h File Reference	113
5.9	fileio.h File Reference	114
5.10	flow_physics.h File Reference	116
5.11	letters.h File Reference	117
5.12	list.h File Reference	119
5.13	material.h File Reference	122
5.14	mesh.h File Reference	123
5.15	navier_stokes.h File Reference	127
5.16	options.h File Reference	132
5.17	physics.h File Reference	134
5.18	types.h File Reference	138

Chapter 1

FE2D Hierarchical Index

1.1 FE2D Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Banner	12
Beam2	13
Control	20
DataContainer	27
Mesh	52
ElSet	31
fileIO	32
Hex8	47
Material	48
NodeBC	79
Nodeset	80
Physics	81
Conduction	14
FlowPhysics	40
AdvectionDiffusion	7
NavierStokes	64
Quad4	91
SideBC	92
ConvectionBC	24
HeatFluxBC	45
Sideset	96

Chapter 2

FE2D Compound Index

2.1 FE2D Compound List

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

AdvectionDiffusion (Advection-Diffusion class)	7
Banner	12
Beam2	13
Conduction (Conduction heat transfer class)	14
Control (Holds all the relevant analysis options and parameters)	20
ConvectionBC (Convection boundary condition based on sidesets)	24
DataContainer (DataContainer is the basic data storage/managements object in FE2D)	27
ElSet	31
fileIO (The fileIO is used to handle all I/O functions – or at least most)	32
FlowPhysics	40
HeatFluxBC (Heat flux boundary condition for sidesets)	45
Hex8	47
Material	48
Mesh (Mesh object – holds basic mesh information, sidesets, nodesets, and data)	52
NavierStokes	64
NodeBC	79
Nodeset	80
Physics (Base class used to construct the solution algorithms for a given physics)	81
Quad4	91
SideBC (This is the base class for all sideset-based boundary conditions)	92
Sideset	96

Chapter 3

FE2D File Index

3.1 FE2D File List

Here is a list of all files with brief descriptions:

advection_diffusion.h (The AdvectionDiffusion (p.7) class is used only for scalar advection-diffusion)	99
banner.h (The Banner (p.12) class is used to print out the code name in 'banner' format)	101
bcs.h (The SideBC (p.92) class is used to derive physics-specific sideset based BC's) .	103
conduction.h (The Conduction (p.14) class provides the thermal conduction specific solver)	104
control.h (The Control (p.20) class holds all the relevant analysis options)	105
cvodate.h (The cvodate is used to keep track of the code version according to CVS) . .	110
data_container.h (The DataContainer (p.27) class is the base data-storage class used for all variables)	111
epsilon.h (Machine limits for the C-G solver (dscgnd.F))	113
fileio.h (The fileIO (p.32) class manages all file-based I/O, e.g., opens files, etc)	114
flow_physics.h (The FlowPhysics (p.40) class is used to construct the physics specific solvers)	116
letters.h (The ASCII letters used by the Banner (p.12) class)	117
list.h (Provides linked-list functions used for computing the PPE connectivity)	119
material.h (The Material (p.48) class provides a simplified way to store/access material properties)	122
mesh.h (The Mesh (p.52) object, Beam2 (p.13), Quad4 (p.91) and Hex8 (p.47) elements, Nodeset (p.80), Sideset (p.96) are defined here)	123
navier_stokes.h (The NavierStokes (p.64) class is used for solving the time-dependent incompressible Navier-Stokes equations)	127
options.h (Preformatted list of analysis options for data-echo purposes)	132
physics.h (The Physics (p.81) class is used to develop a physics-specific solver)	134
types.h (Defines all parameters and types for the FE2D code, e.g., Real is double) . . .	138

Chapter 4

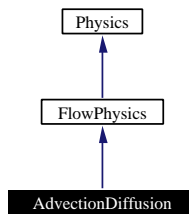
FE2D Class Documentation

4.1 AdvectionDiffusion Class Reference

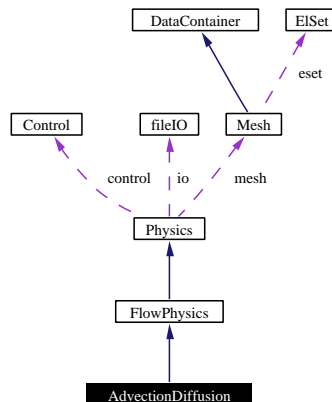
Advection-Diffusion class.

```
#include <advection_diffusion.h>
```

Inheritance diagram for AdvectionDiffusion:



Collaboration diagram for AdvectionDiffusion:



Public Methods

- void **formMK** ()
Form the M+K and M-K operators.

Constructor/Destructor

- **AdvectionDiffusion** ()
- virtual **~AdvectionDiffusion** ()

Data Registration

Each physics that is implemented will require its own storage and specific variables. The data registration function is implemented for each physics class to enable custom definition of variables for each algorithm that's implemented.

- virtual void **registerData** ()

Virtual Physics Functions

These functions are pure virtual functions that need to be implemented for each specific type of physics being solved with the framework.

- virtual void **setup** (**Mesh *mesh**, **Control *control**, **fileIO *io**)
Virtual setup function – implemented for each specific physics.
- void **solve** ()
Virtual solver – implemented for advection-diffusion physics.
- virtual void **setICs** ()
Virtual function to setup initial conditions for advection-diffusion.

Virtual Functions for I/O

These functions are implemented for each specific physics to permit custom options to be coded for reading and writing restart files, time-history data, bc's, etc.

- virtual void **writeRestart** (ofstream &dumpf)
- virtual void **readRestart** (ifstream &restf)
- virtual void **writeTHhead** (**Mesh *mesh**, **fileIO *io**)
- virtual void **writeTHdata** (**Mesh *mesh**, **fileIO *io**, **Real time**)
- virtual void **writeSolving** ()
- virtual void **echoBCs** (**Mesh *mesh**, ostream &ofs)

Protected Attributes

Data registered for advection-diffusion

*Only temperature and heat source term need to be registered for advection-diffusion – **Flow-Physics** (p.40) takes care of all the other flow parameters.*

- **DataIndex TEMPERATURE**
- **DataIndex Q**

Private Methods

- **AdvectionDiffusion** (const AdvectionDiffusion &)
- AdvectionDiffusion & **operator=** (const AdvectionDiffusion &)

Private Attributes

Element operators

- **Real Me** [4][4]
Element consistent mass matrix.
- **Real Mle** [4]
Element lumped mass matrix.
- **Real Ke** [4][4]
Element conductivity matrix.
- **Real C** [2][4]
Element gradient (Cx,Cy) operators.

4.1.1 Detailed Description

Advection-Diffusion class.

Definition at line 12 of file advection_diffusion.h.

4.1.2 Constructor & Destructor Documentation

4.1.2.1 AdvectionDiffusion::AdvectionDiffusion () [inline]

Definition at line 18 of file advection_diffusion.h.

```
18 {}
```

4.1.2.2 virtual AdvectionDiffusion::~~AdvectionDiffusion () [inline, virtual]

Definition at line 19 of file advection_diffusion.h.

```
19 {}
```

4.1.2.3 AdvectionDiffusion::AdvectionDiffusion (const AdvectionDiffusion &) [private]

4.1.3 Member Function Documentation

4.1.3.1 virtual void AdvectionDiffusion::echoBCs (Mesh * *mesh*, ostream & *ofs*) [virtual]

Implements **Physics** (p. 85).

4.1.3.2 void AdvectionDiffusion::formMK ()

Form the M+K and M-K operators.

4.1.3.3 AdvectionDiffusion& AdvectionDiffusion::operator= (const AdvectionDiffusion &) [private]

4.1.3.4 virtual void AdvectionDiffusion::readRestart (ifstream & *restf*) [virtual]

Reimplemented from **FlowPhysics** (p. 42).

4.1.3.5 virtual void AdvectionDiffusion::registerData () [virtual]

Reimplemented from **FlowPhysics** (p. 42).

4.1.3.6 virtual void AdvectionDiffusion::setICs () [virtual]

Virtual function to setup initial conditions for advection-diffusion.

Reimplemented from **Physics** (p. 86).

4.1.3.7 virtual void AdvectionDiffusion::setup (Mesh * *mesh*, Control * *control*, fileIO * *io*) [virtual]

Virtual setup function – implemented for each specific physics.

Reimplemented from **FlowPhysics** (p. 43).

4.1.3.8 void AdvectionDiffusion::solve () [virtual]

Virtual solver – implemented for advection-diffusion physics.

Reimplemented from **Physics** (p. 86).

4.1.3.9 virtual void AdvectionDiffusion::writeRestart (ofstream & *dumpf*) [virtual]

Reimplemented from **FlowPhysics** (p. 43).

4.1.3.10 virtual void AdvectionDiffusion::writeSolving () [virtual]

Implements **Physics** (p. 86).

4.1.3.11 virtual void AdvectionDiffusion::writeTHdata (Mesh * *mesh*, fileIO * *io*, Real *time*) [virtual]

Implements **Physics** (p. 87).

4.1.3.12 virtual void AdvectionDiffusion::writeTHhead (Mesh * *mesh*, fileIO * *io*) [virtual]

Implements **Physics** (p. 87).

4.1.4 Member Data Documentation

4.1.4.1 Real AdvectionDiffusion::C[2][4] [private]

Element gradient (Cx,Cy) operators.

Definition at line 95 of file advection_diffusion.h.

4.1.4.2 Real AdvectionDiffusion::Ke[4][4] [private]

Element conductivity matrix.

Definition at line 94 of file advection_diffusion.h.

4.1.4.3 Real AdvectionDiffusion::Me[4][4] [private]

Element consistent mass matrix.

Definition at line 92 of file advection_diffusion.h.

4.1.4.4 Real AdvectionDiffusion::Mle[4] [private]

Element lumped mass matrix.

Definition at line 93 of file advection_diffusion.h.

4.1.4.5 dataIndex AdvectionDiffusion::Q [protected]

Definition at line 80 of file advection_diffusion.h.

4.1.4.6 dataIndex AdvectionDiffusion::TEMPERATURE [protected]

Definition at line 79 of file advection_diffusion.h.

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

- **advection_diffusion.h**

4.2 Banner Class Reference

```
#include <banner.h>
```

Public Methods

- **Banner** (const char *s="", int sz=MEDIUM_FONT, int fmt=CENTER_JUSTIFIED)
- **~Banner** ()

Private Attributes

- int **font_size**
- int **format**
- char * **name**

Friends

- ostream & **operator<<** (ostream &, const Banner &)

4.2.1 Constructor & Destructor Documentation

4.2.1.1 **Banner::Banner** (const char * s = "", int sz = MEDIUM_FONT, int fmt = CENTER_JUSTIFIED)

4.2.1.2 **Banner::~~Banner** ()

4.2.2 Friends And Related Function Documentation

4.2.2.1 **ostream& operator<<** (ostream &, const Banner &) [friend]

4.2.3 Member Data Documentation

4.2.3.1 **int Banner::font_size** [private]

Definition at line 24 of file banner.h.

4.2.3.2 **int Banner::format** [private]

Definition at line 25 of file banner.h.

4.2.3.3 **char* Banner::name** [private]

Definition at line 26 of file banner.h.

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

- **banner.h**

4.3 Beam2 Struct Reference

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

Public Attributes

2-node 1-D element (for testing purposes)

This structure is used to hold all the book-keeping data for the element.

- **int mat**
Material (p. 48) *id.*
- **int ix** [NNPE_BEAM2]
Node-to-element connectivity.

4.3.1 Member Data Documentation

4.3.1.1 **int Beam2::ix**[NNPE_BEAM2]

Node-to-element connectivity.

Definition at line 85 of file mesh.h.

4.3.1.2 **int Beam2::mat**

Material (p. 48) *id.*

Definition at line 84 of file mesh.h.

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

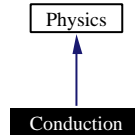
- **mesh.h**

4.4 Conduction Class Reference

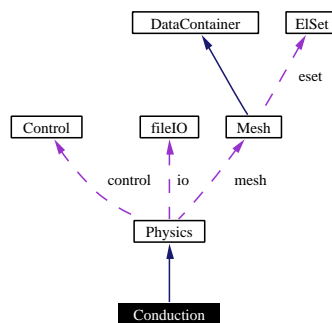
Conduction heat transfer class.

```
#include <conduction.h>
```

Inheritance diagram for Conduction:



Collaboration diagram for Conduction:



Public Methods

- void **formMK** ()
Form the M+K and M-K operators.

Constructor/Destructor

- **Conduction** ()
- virtual **~Conduction** ()

Data Registration

Each physics that is implemented will require its own storage and specific variables. The data registration function is implemented for each physics class to enable custom definition of variables for each algorithm that's implemented.

- virtual void **registerData** ()

Virtual Physics Functions

These functions are pure virtual functions that need to be implemented for each specific type of physics being solved with the framework.

- virtual void **setup** (**Mesh *mesh**, **Control *control**, **fileIO *io**)

Virtual setup function – implemented for each specific physics.

- void **solve** ()
Virtual solver – implemented for conduction physics.
- virtual void **setICs** ()
Virtual function to setup initial conditions for conduction.

Virtual Functions for I/O

These functions are implemented for each specific physics to permit custom options to be coded for reading and writing restart files, time-history data, bc's, etc.

- virtual void **writeRestart** (ofstream &dumpf)
- virtual void **readRestart** (ifstream &restf)
- virtual void **writeTHhead** (Mesh *mesh, fileIO *io)
- virtual void **writeTHdata** (Mesh *mesh, fileIO *io, Real time)
- virtual void **writeSolving** ()
- virtual void **echoBCs** (Mesh *mesh, ostream &ofs)

Conduction heat transfer specific boundary conditions

These functions are specific to conduction and are only implemented for this physics class.

- void **addHeatFluxBC** ()
Add a heat flux boundary condition to list of heat flux BC's.
- **HeatFluxBC** * **getHeatFluxBC** (int i)
Load a heat flux BC given an id 'i'.
- int **numHeatFluxBC** ()
How many heat flux bc's are loaded.
- void **addConvectionBC** ()
Add a convection boundary condition.
- **ConvectionBC** * **getConvectionBC** (int i)
Load a convection boundary condtion.
- int **numConvectionBC** ()
How many convection bc's are loaded.

Protected Attributes

Data registered for advection-diffusion

Only temperature and heat source term need to be registered for conduction.

- **DataIndex** TEMPERATURE
- **DataIndex** Q

Private Methods

- **Conduction** (const Conduction &)
- Conduction & **operator=** (const Conduction &)

Private Attributes

Element operators

- **Real Me** [4][4]
Element consistent mass matrix.
- **Real Mle** [4]
Element lumped mass matrix.
- **Real Ke** [4][4]
Element conductivity matrix.

STL vectors of heat flux BC objects

- vector< **HeatFluxBC** * > **heatflux**
- vector< **ConvectionBC** * > **convection**

4.4.1 Detailed Description

Conduction heat transfer class.

Definition at line 15 of file conduction.h.

4.4.2 Constructor & Destructor Documentation

4.4.2.1 **Conduction::Conduction** () [inline]

Definition at line 21 of file conduction.h.

```
21 {}
```

4.4.2.2 **virtual Conduction::~~Conduction** () [inline, virtual]

Definition at line 22 of file conduction.h.

```
22 {}
```

4.4.2.3 **Conduction::Conduction** (const **Conduction** &) [private]

4.4.3 Member Function Documentation

4.4.3.1 **void Conduction::addConvectionBC** () [inline]

Add a convection boundary condition.

Definition at line 89 of file conduction.h.

References convection.

```
89 {convection.push_back(new ConvectionBC);}
```

4.4.3.2 void Conduction::addHeatFluxBC () [inline]

Add a heat flux boundary condition to list of heat flux BC's.

Definition at line 80 of file conduction.h.

References heatflux.

```
80 {heatflux.push_back(new HeatFluxBC);}
```

4.4.3.3 virtual void Conduction::echoBCs (Mesh * *mesh*, ostream & *ofs*) [virtual]

Implements **Physics** (p. 85).

4.4.3.4 void Conduction::formMK ()

Form the M+K and M-K operators.

4.4.3.5 ConvectionBC* Conduction::getConvectionBC (int *i*) [inline]

Load a convection boundary condition.

Definition at line 92 of file conduction.h.

References convection.

```
92 {return convection[i];}
```

4.4.3.6 HeatFluxBC* Conduction::getHeatFluxBC (int *i*) [inline]

Load a heat flux BC given an id 'i'.

Definition at line 83 of file conduction.h.

References heatflux.

```
83 {return heatflux[i];}
```

4.4.3.7 int Conduction::numConvectionBC () [inline]

How many convection bc's are loaded.

Definition at line 95 of file conduction.h.

References convection.

```
95 {return convection.size();}
```

4.4.3.8 `int Conduction::numHeatFluxBC ()` [inline]

How many heat flux bc's are loaded.

Definition at line 86 of file `conduction.h`.

References `heatflux`.

```
86 {return heatflux.size();}
```

4.4.3.9 `Conduction& Conduction::operator= (const Conduction &)` [private]**4.4.3.10** `virtual void Conduction::readRestart (ifstream & restf)` [virtual]

Implements **Physics** (p. 85).

4.4.3.11 `virtual void Conduction::registerData ()` [virtual]

Reimplemented from **Physics** (p. 85).

4.4.3.12 `virtual void Conduction::setICs ()` [virtual]

Virtual function to setup initial conditions for conduction.

Reimplemented from **Physics** (p. 86).

4.4.3.13 `virtual void Conduction::setup (Mesh * mesh, Control * control, fileIO * io)` [virtual]

Virtual setup function – implemented for each specific physics.

Reimplemented from **Physics** (p. 86).

4.4.3.14 `void Conduction::solve ()` [virtual]

Virtual solver – implemented for conduction physics.

Reimplemented from **Physics** (p. 86).

4.4.3.15 `virtual void Conduction::writeRestart (ofstream & dumpf)` [virtual]

Implements **Physics** (p. 86).

4.4.3.16 `virtual void Conduction::writeSolving ()` [virtual]

Implements **Physics** (p. 86).

4.4.3.17 `virtual void Conduction::writeTHdata (Mesh * mesh, fileIO * io, Real time)` [virtual]

Implements **Physics** (p. 87).

4.4.3.18 virtual void Conduction::writeTHhead (Mesh * *mesh*, fileIO * *io*) [virtual]

Implements **Physics** (p. 87).

4.4.4 Member Data Documentation

4.4.4.1 vector<ConvectionBC*> Conduction::convection [private]

Definition at line 125 of file conduction.h.

Referenced by addConvectionBC, getConvectionBC, and numConvectionBC.

4.4.4.2 vector<HeatFluxBC*> Conduction::heatflux [private]

Definition at line 124 of file conduction.h.

Referenced by addHeatFluxBC, getHeatFluxBC, and numHeatFluxBC.

4.4.4.3 Real Conduction::Ke[4][4] [private]

Element conductivity matrix.

Definition at line 119 of file conduction.h.

4.4.4.4 Real Conduction::Me[4][4] [private]

Element consistent mass matrix.

Definition at line 117 of file conduction.h.

4.4.4.5 Real Conduction::Mle[4] [private]

Element lumped mass matrix.

Definition at line 118 of file conduction.h.

4.4.4.6 DataIndex Conduction::Q [protected]

Definition at line 105 of file conduction.h.

4.4.4.7 DataIndex Conduction::TEMPERATURE [protected]

Definition at line 104 of file conduction.h.

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

- **conduction.h**

4.5 Control Class Reference

The Control class holds all the relevant analysis options and parameters.

```
#include <control.h>
```

Public Methods

- **Control** ()
- virtual **~Control** ()

Generic Access functions

There is a set of generic and simple access functions to access all the options and control parameters for the code. These functions use the AnalysisOpt and AnalysisPrm enum's listed in this header file.

- string **getTitle** ()
- void **setTitle** (char *cbuf)
- void **setOption** (int val, **AnalysisOpt** opt)
- void **setParam** (**Real** val, **AnalysisPrm** opt)
- int **getOption** (**AnalysisOpt** opt)
- **Real** **getParam** (**AnalysisPrm** opt)
- int * **getOptions** ()
- **Real** * **getParams** ()

Restart functions

*The write/read restart functions for the **Control** (p. 20) object*

- void **writeRestart** (ofstream &dumpf)
- void **readRestart** (ifstream &restf)

Private Methods

- **Control** (const **Control** &)
- **Control** & **operator=** (const **Control** &)

Private Attributes

- string **title**
Analysis title from cntl file goes here.
- int **icntl** [NUM_ICNTL]
Integer control parameters for analysis.
- **Real** **rcntl** [NUM_RCNTL]
Real control parameters for analysis.

4.5.1 Detailed Description

The Control class holds all the relevant analysis options and parameters.

Definition at line 81 of file control.h.

4.5.2 Constructor & Destructor Documentation

4.5.2.1 Control::Control ()

4.5.2.2 virtual Control::~~Control () [inline, virtual]

Definition at line 85 of file control.h.

```
85 {}
```

4.5.2.3 Control::Control (const Control &) [private]

4.5.3 Member Function Documentation

4.5.3.1 int Control::getOption (AnalysisOpt *opt*) [inline]

Definition at line 102 of file control.h.

References AnalysisOpt, and icntl.

```
102 {return icntl[opt];}
```

4.5.3.2 int* Control::getOptions () [inline]

Definition at line 106 of file control.h.

References icntl.

```
106 {return &icntl[0];}
```

4.5.3.3 Real Control::getParam (AnalysisPrm *opt*) [inline]

Definition at line 104 of file control.h.

References AnalysisPrm, rcntl, and Real.

```
104 {return rcntl[opt];}
```

4.5.3.4 Real* Control::getParams () [inline]

Definition at line 108 of file control.h.

References rcntl, and Real.

```
108 {return &rcntl[0];}
```

4.5.3.5 `string Control::getTitle ()` [inline]

Definition at line 94 of file control.h.

References title.

```
94 {return title;}
```

4.5.3.6 `Control& Control::operator= (const Control &)` [private]**4.5.3.7** `void Control::readRestart (ifstream & restf)`**4.5.3.8** `void Control::setOption (int val, AnalysisOpt opt)` [inline]

Definition at line 98 of file control.h.

References AnalysisOpt, and icntl.

```
98 {icntl[opt] = val; }
```

4.5.3.9 `void Control::setParam (Real val, AnalysisPrm opt)` [inline]

Definition at line 100 of file control.h.

References AnalysisPrm, rcntl, and Real.

```
100 {rcntl[opt] = val; }
```

4.5.3.10 `void Control::setTitle (char * cbuf)` [inline]

Definition at line 96 of file control.h.

References title.

```
96 {title = cbuf;}
```

4.5.3.11 `void Control::writeRestart (ofstream & dumpf)`**4.5.4** Member Data Documentation**4.5.4.1** `int Control::icntl[NUM_ICNTL]` [private]

Integer control parameters for analysis.

Definition at line 128 of file control.h.

Referenced by `getOption`, `getOptions`, and `setOption`.

4.5.4.2 Real Control::rcntl[NUM_RCNTL] [private]

Real control parameters for analysis.

Definition at line 129 of file control.h.

Referenced by getParam, getParams, and setParam.

4.5.4.3 string Control::title [private]

Analysis title from cntl file goes here.

Definition at line 127 of file control.h.

Referenced by getTitle, and setTitle.

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

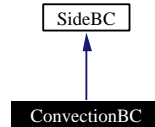
- **control.h**

4.6 ConvectionBC Class Reference

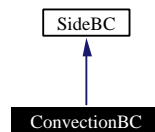
Convection boundary condition based on sidesets.

```
#include <bcs.h>
```

Inheritance diagram for ConvectionBC:



Collaboration diagram for ConvectionBC:



Public Methods

- void **applyBC** (**Mesh** *mesh, **Real** *q, **Real** dt)
Integrate convective heat flux over the surface and assemble into {Q}.
- void **setTamb** (**Real** val)
Set the ambient temperature, T_infinity.
- **Real** **getTamb** ()
Get the ambient temperature, T_infinity.

Constructor/Destructor

- **ConvectionBC** ()
- virtual **~ConvectionBC** ()

Private Methods

- **ConvectionBC** (const **ConvectionBC** &)
- **ConvectionBC** & **operator=** (const **ConvectionBC** &)

Private Attributes

- **Real** **Tamb**

4.6.1 Detailed Description

Convection boundary condition based on sidesets.

Definition at line 99 of file bcs.h.

4.6.2 Constructor & Destructor Documentation

4.6.2.1 ConvectionBC::ConvectionBC () [inline]

Definition at line 104 of file bcs.h.

```
104 {}
```

4.6.2.2 virtual ConvectionBC::~~ConvectionBC () [inline, virtual]

Definition at line 105 of file bcs.h.

```
105 {}
```

4.6.2.3 ConvectionBC::ConvectionBC (const ConvectionBC &) [private]

4.6.3 Member Function Documentation

4.6.3.1 void ConvectionBC::applyBC (Mesh * *mesh*, Real * *q*, Real *dt*)

Integrate convective heat flux over the surface and assemble into {Q}.

4.6.3.2 Real ConvectionBC::getTamb () [inline]

Get the ambient temperature, T_infinity.

Definition at line 115 of file bcs.h.

References Real, and Tamb.

```
115 {return Tamb;}
```

4.6.3.3 ConvectionBC& ConvectionBC::operator= (const ConvectionBC &) [private]

4.6.3.4 void ConvectionBC::setTamb (Real *val*) [inline]

Set the ambient temperature, T_infinity.

Definition at line 112 of file bcs.h.

References Real, and Tamb.

```
112 {Tamb = val;}
```

4.6.4 Member Data Documentation

4.6.4.1 Real ConvectionBC::Tamb [private]

Definition at line 126 of file bcs.h.

Referenced by getTamb, and setTamb.

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

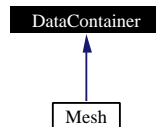
- **bcs.h**

4.7 DataContainer Class Reference

DataContainer is the basic data storage/managements object in FE2D.

```
#include <data_container.h>
```

Inheritance diagram for DataContainer:



Public Methods

- **DataContainer** ()
- virtual **~DataContainer** ()

DataContainer is the basic storage object in FE2D.

All objects and variables are registered in data containers using the registerVariable function.

The DataIndex is a simple handle for retrieving the data without carrying raw pointers around. The data register keeps track of the total memory usage in the code and can be queried from anywhere in the code.

- **DataIndex registerVariable** (int nobj, int size_t, const char *label)
Register data/objects using this function.
- void **freeVariable** (**DataIndex** vid)
Free data/objects.
- void **freeAllVariables** ()
Free ALL data/objects.
- void * **getVariable** (**DataIndex** vid)
Return a pointer given a DataIndex.
- void **reportMemory** ()
Report memory usage statistics.
- void **dumpVariables** ()
Dump a list of all registered data.

Private Methods

- **DataContainer** (const DataContainer &)
- DataContainer & **operator=** (const DataContainer &)

Private Attributes

- int **number** [MAX_MEMORY_ENTRIES]
Internal counter.
- int **registration** [MAX_MEMORY_ENTRIES]
Currently unused.
- int **size** [MAX_MEMORY_ENTRIES]
Object/Data sizes.
- string **name** [MAX_MEMORY_ENTRIES]
Object/Data names.
- void * **ptr** [MAX_MEMORY_ENTRIES]
Pointer to data/object.
- bool **inuse** [MAX_MEMORY_ENTRIES]
Currently inuse (or not).

4.7.1 Detailed Description

DataContainer is the basic data storage/managements object in FE2D.

Definition at line 16 of file data_container.h.

4.7.2 Constructor & Destructor Documentation

4.7.2.1 DataContainer::DataContainer ()

4.7.2.2 virtual DataContainer::~~DataContainer () [inline, virtual]

Definition at line 20 of file data_container.h.

```
20 {}
```

4.7.2.3 DataContainer::DataContainer (const DataContainer &) [private]

4.7.3 Member Function Documentation

4.7.3.1 void DataContainer::dumpVariables ()

Dump a list of all registered data.

4.7.3.2 void DataContainer::freeAllVariables ()

Free ALL data/objects.

4.7.3.3 void DataContainer::freeVariable (DataIndex *vid*)

Free data/objects.

4.7.3.4 void* DataContainer::getVariable (DataIndex *vid*)

Return a pointer given a DataIndex.

**4.7.3.5 DataContainer& DataContainer::operator= (const DataContainer &)
[private]****4.7.3.6 DataIndex DataContainer::registerVariable (int *nobj*, int *size_t*, const char
* *label*)**

Register data/objects using this function.

4.7.3.7 void DataContainer::reportMemory ()

Report memory usage statistics.

4.7.4 Member Data Documentation**4.7.4.1 bool DataContainer::inuse[MAX_MEMORY_ENTRIES] [private]**

Currently inuse (or not).

Definition at line 61 of file data_container.h.

4.7.4.2 string DataContainer::name[MAX_MEMORY_ENTRIES] [private]

Object/Data names.

Definition at line 59 of file data_container.h.

4.7.4.3 int DataContainer::number[MAX_MEMORY_ENTRIES] [private]

Internal counter.

Definition at line 56 of file data_container.h.

4.7.4.4 void* DataContainer::ptr[MAX_MEMORY_ENTRIES] [private]

Pointer to data/object.

Definition at line 60 of file data_container.h.

4.7.4.5 int DataContainer::registration[MAX_MEMORY_ENTRIES] [private]

Currently unused.

Definition at line 57 of file data_container.h.

4.7.4.6 int DataContainer::size[MAX_MEMORY_ENTRIES] [private]

Object/Data sizes.

Definition at line 58 of file data_container.h.

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

- **data_container.h**

4.8 ElSet Struct Reference

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

Public Attributes

Element set – used for setting hydrostatic pressures

The ElSet (p.31) structure holds the list of elements where the pressure is fixed.

- **int Nel_set**
No. of elements in the element set.
- **DataIndex EL_LIST**
List of elements.
- **DataIndex EL_VAL**
List of values associated with elements.

4.8.1 Member Data Documentation

4.8.1.1 DataIndex ElSet::EL_LIST

List of elements.

Definition at line 136 of file mesh.h.

4.8.1.2 DataIndex ElSet::EL_VAL

List of values associated with elements.

Definition at line 137 of file mesh.h.

4.8.1.3 int ElSet::Nel_set

No. of elements in the element set.

Definition at line 135 of file mesh.h.

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

- **mesh.h**

4.9 fileIO Class Reference

The fileIO is used to handle all I/O functions – or at least most.

```
#include <fileio.h>
```

Public Methods

- `fileIO ()`
- `virtual ~fileIO ()`

File and Filename management

- `string * getFnames ()`
- `void setFname (FileType type, char *name)`
- `void echoFnames (ostream &ofs)`
- `void openFiles ()`
- `void closeFiles (Mesh *mesh)`

Restart file management (for reading)

- `ifstream & openRestart ()`
- `ifstream & getRestart ()`
- `void closeRestart ()`

Dump file management (for writing)

- `ofstream & openDump ()`
- `void closeDump ()`

Global data file – where the kinetic energy, $\text{div}(\mathbf{U})$ is written

- `ofstream & openGlob (int append)`
- `void closeGlob ()`

Generic file manipulation

- `ifstream & inputStream (FileType type)`
- `ofstream & getOut ()`
- `ofstream & outputStream (FileType type)`

Time-history file management

- `ofstream & histStream (int id)`

Mesh file operations

- `void readMesh (Mesh *mesh)`
- `void readHeader (Mesh *mesh)`
- `void readConn (Mesh *mesh)`
- `void readCoord (Mesh *mesh)`
- `void readNodeset (Mesh *mesh)`
- `void readSideset (Mesh *mesh)`
- `void echoHeader (Mesh *mesh, ostream &ofs)`

Control file operations

- void **parseControl** (**Mesh** *mesh, **Control** *control)
- void **echoControl** (**Control** *control, ostream &ofs)
- void **parseAnalysis** (**Control** *control)
- void **parseElSet** (**Mesh** *mesh, int Nel)
- void **parseNodalBC** (**Mesh** *mesh, **DOFmap** dof)
- void **parseHeatFluxBC** (**Mesh** *mesh)
- void **parseConvectionBC** (**Mesh** *mesh)
- void **parseNodeSolver** (**Control** *control)
- void **parseMaterial** (**Mesh** *mesh)
- void **echoMaterials** (**Mesh** *mesh, ostream &outf)
- void **parseHistory** (**Mesh** *mesh)
- void **openHistoryFiles** (**Mesh** *mesh, **Control** *control)
- void **echoHistory** (**Mesh** *mesh, ostream &outf)
- char * **getToken** (char *line, char *token)
- int **iscomment** (char *cdat)

Banner functions

- void **printBanner** (const string **cvdate**, const string version, const string delimiter, const string date, ostream &ofs)
- void **echoRestart** (ostream &ofs)

GMV ASCII plot file output methods

- void **writeHeader** (int step, int cycle, **Real** time, **Mesh** *mesh, **Control** *control)
- void **writeFooter** ()
- void **writeField** (**Mesh** *mesh, **DataIndex** index, const char *name, int type)
- void **writeVelocity** (**Mesh** *mesh, **DataIndex** VELX, **DataIndex** VELY)
- void **writeVars** (bool head)

Private Types

- enum **FileFormat** { **ASCII** = 0, **EXODUS**, **NUM_FILEFORMATS** }

Private Methods

- **fileIO** (const fileIO &)
- fileIO & **operator=** (const fileIO &)

Private Attributes

Private data for file management

- string **fnames** [NUM_FILENAMES]
- ifstream **meshf**
Mesh (p. 52) *file*.
- ifstream **cntlif**
Control (p. 20) *file*.
- ifstream **restf**
Restart file.

- ofstream **outf**
Output file.
- ofstream **plotf**
State plot file.
- ofstream **globf**
Global data file.
- ofstream **dumpf**
Dump file.
- ofstream * **histf**
Array of Time-history files.

4.9.1 Detailed Description

The fileIO is used to handle all I/O functions – or at least most.

Definition at line 38 of file fileio.h.

4.9.2 Member Enumeration Documentation

4.9.2.1 enum fileIO::FileFormat [private]

Enumeration values:

ASCII

EXODUS

NUM_FILEFORMATS

Definition at line 40 of file fileio.h.

```

40             {ASCII=0,           //!< ASCII file format
41             EXODUS,           //!< EXODUS file format
42             NUM_FILEFORMATS   //!< No. of possible file formats
43     };

```

4.9.3 Constructor & Destructor Documentation

4.9.3.1 fileIO::fileIO ()

4.9.3.2 virtual fileIO::~~fileIO () [virtual]

4.9.3.3 fileIO::fileIO (const fileIO &) [private]

4.9.4 Member Function Documentation

4.9.4.1 void fileIO::closeDump () [inline]

Definition at line 75 of file fileio.h.

References dumpf.

```
75 {dumpf.close();}
```

4.9.4.2 void fileIO::closeFiles (Mesh * *mesh*)

4.9.4.3 void fileIO::closeGlob () [inline]

Definition at line 83 of file fileio.h.

References globf.

```
83 {globf.close();}
```

4.9.4.4 void fileIO::closeRestart () [inline]

Definition at line 68 of file fileio.h.

References restf.

```
68 {restf.close();}
```

4.9.4.5 void fileIO::echoControl (Control * *control*, ostream & *ofs*)

4.9.4.6 void fileIO::echoFnames (ostream & *ofs*)

4.9.4.7 void fileIO::echoHeader (Mesh * *mesh*, ostream & *ofs*)

4.9.4.8 void fileIO::echoHistory (Mesh * *mesh*, ostream & *outf*)

4.9.4.9 void fileIO::echoMaterials (Mesh * *mesh*, ostream & *outf*)

4.9.4.10 void fileIO::echoRestart (ostream & *ofs*)

4.9.4.11 string* fileIO::getFnames () [inline]

Definition at line 51 of file fileio.h.

References fnames.

```
51 {return fnames;}
```

4.9.4.12 ofstream& fileIO::getOut () [inline]

Definition at line 90 of file fileio.h.

References outf.

```
90 {return outf;}
```

4.9.4.13 `ifstream& fileIO::getRestart ()` [inline]

Definition at line 66 of file `fileio.h`.

References `restf`.

```
66 {return restf;}
```

4.9.4.14 `char* fileIO::getToken (char * line, char * token)`

4.9.4.15 `ofstream& fileIO::histStream (int id)` [inline]

Definition at line 97 of file `fileio.h`.

References `histf`.

```
97 {return histf[id];}
```

- 4.9.4.16 `ifstream& fileIO::inputStream (FileType type)`
- 4.9.4.17 `int fileIO::iscomment (char * cdat)`
- 4.9.4.18 `ofstream& fileIO::openDump ()`
- 4.9.4.19 `void fileIO::openFiles ()`
- 4.9.4.20 `ofstream& fileIO::openGlob (int append)`
- 4.9.4.21 `void fileIO::openHistoryFiles (Mesh * mesh, Control * control)`
- 4.9.4.22 `ifstream& fileIO::openRestart ()`
- 4.9.4.23 `fileIO& fileIO::operator= (const fileIO &) [private]`
- 4.9.4.24 `ofstream& fileIO::outputStream (FileType type)`
- 4.9.4.25 `void fileIO::parseAnalysis (Control * control)`
- 4.9.4.26 `void fileIO::parseControl (Mesh * mesh, Control * control)`
- 4.9.4.27 `void fileIO::parseConvectionBC (Mesh * mesh)`
- 4.9.4.28 `void fileIO::parseElSet (Mesh * mesh, int Nel)`
- 4.9.4.29 `void fileIO::parseHeatFluxBC (Mesh * mesh)`
- 4.9.4.30 `void fileIO::parseHistory (Mesh * mesh)`
- 4.9.4.31 `void fileIO::parseMaterial (Mesh * mesh)`
- 4.9.4.32 `void fileIO::parseNodalBC (Mesh * mesh, DOFmap dof)`
- 4.9.4.33 `void fileIO::parseNodeSolver (Control * control)`
- 4.9.4.34 `void fileIO::printBanner (const string cvdate, const string version, const string delimiter, const string date, ostream & ofs)`
- 4.9.4.35 `void fileIO::readConn (Mesh * mesh)`
- 4.9.4.36 `void fileIO::readCoord (Mesh * mesh)`
- 4.9.4.37 `void fileIO::readHeader (Mesh * mesh)`
- 4.9.4.38 `void fileIO::readMesh (Mesh * mesh)`
- 4.9.4.39 `void fileIO::readNodeset (Mesh * mesh)`
- 4.9.4.40 `void fileIO::readSideset (Mesh * mesh)`
- 4.9.4.41 `void fileIO::setFname (FileType type, char * name) [inline]`

Definition at line 53 of file fileio.h.

References FileType, and fnames.

```
53 {fnames[type] = name;}
```

4.9.4.42 void fileIO::writeField (Mesh * *mesh*, DataIndex *index*, const char * *name*, int *type*)

4.9.4.43 void fileIO::writeFooter ()

4.9.4.44 void fileIO::writeHeader (int *step*, int *cycle*, Real *time*, Mesh * *mesh*, Control * *control*)

4.9.4.45 void fileIO::writeVars (bool *head*)

4.9.4.46 void fileIO::writeVelocity (Mesh * *mesh*, DataIndex *VELX*, DataIndex *VELY*)

4.9.5 Member Data Documentation

4.9.5.1 ifstream fileIO::cntlf [private]

Control (p. 20) file.

Definition at line 186 of file fileio.h.

4.9.5.2 ofstream fileIO::dumpf [private]

Dump file.

Definition at line 192 of file fileio.h.

Referenced by closeDump.

4.9.5.3 string fileIO::fnames[NUM_FILENAMES] [private]

Definition at line 183 of file fileio.h.

Referenced by getFnames, and setFname.

4.9.5.4 ofstream fileIO::globf [private]

Global data file.

Definition at line 191 of file fileio.h.

Referenced by closeGlob.

4.9.5.5 ofstream* fileIO::histf [private]

Array of Time-history files.

Definition at line 194 of file fileio.h.

Referenced by histStream.

4.9.5.6 ifstream fileIO::meshf [private]

Mesh (p. 52) file.

Definition at line 185 of file fileio.h.

4.9.5.7 ofstream fileIO::outf [private]

Output file.

Definition at line 189 of file fileio.h.

Referenced by getOut.

4.9.5.8 ofstream fileIO::plotf [private]

State plot file.

Definition at line 190 of file fileio.h.

4.9.5.9 ifstream fileIO::restf [private]

Restart file.

Definition at line 187 of file fileio.h.

Referenced by closeRestart, and getRestart.

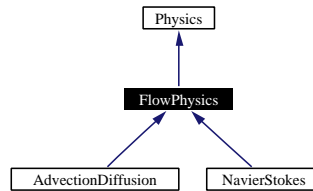
The documentation for this class was generated from the following file:

- fileio.h

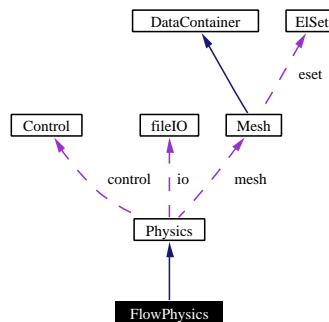
4.10 FlowPhysics Class Reference

```
#include <flow_physics.h>
```

Inheritance diagram for FlowPhysics:



Collaboration diagram for FlowPhysics:



Public Methods

Constructor/Destructor

- **FlowPhysics** ()
- virtual **~FlowPhysics** ()

Data Registration

Each physics that is implemented will require its own storage and specific variables. The data registration function is implemented for each physics class to enable custom definition of variables for each algorithm that's implemented.

- virtual void **registerData** ()

Virtual Physics Functions

These functions are pure virtual functions that need to be implemented for each specific type of physics being solved with the framework.

- virtual void **setup** (**Mesh *mesh**, **Control *control**, **fileIO *io**)
Virtual setup function – implemented for each specific physics.

Stream function & vorticity

These functions are used to compute the stream-function and vorticity for graphics output.

- void **calcEdge2D** ()
Compute the edge order for the stream function.
- void **calcStreamFunc** ()
Traverse the edges to compute the stream func.
- void **calcVorticity** ()
Compute the vorticity.

Virtual Functions for I/O

These functions are implemented for each specific physics to permit custom options to be coded for reading and writing restart files, time-history data, bc's, etc.

- virtual void **writeRestart** (ofstream &dumpf)
- virtual void **readRestart** (ifstream &restf)

Protected Attributes

Data registered for flow physics.

All of the basic variables for flow problems such as advection-diffusion and Navier-Stokes is registered here.

- **DataIndex VELX**
Y-velocity component.
- **DataIndex VELY**
X-velocity component.
- **DataIndex PSI**
Stream function.
- **DataIndex PSI_EL**
Element, edge for stream function computation.
- **DataIndex PSI_EDGE**
Edges for stream function (PSI).
- **DataIndex VORTICITY**
Vorticity (z-component).

Private Methods

- **FlowPhysics** (const FlowPhysics &)
- FlowPhysics & **operator=** (const FlowPhysics &)

Private Attributes

- int **Npsi**
No. of 2-D elements for stream function.

4.10.1 Detailed Description

Virtual base class used to construct the solution algorithms for a given flow physics, e.g., advection-diffusion, Navier-Stokes

Definition at line 17 of file `flow_physics.h`.

4.10.2 Constructor & Destructor Documentation

4.10.2.1 `FlowPhysics::FlowPhysics ()` [inline]

Definition at line 24 of file `flow_physics.h`.

```
24 {}
```

4.10.2.2 `virtual FlowPhysics::~~FlowPhysics ()` [inline, virtual]

Definition at line 25 of file `flow_physics.h`.

```
25 {}
```

4.10.2.3 `FlowPhysics::FlowPhysics (const FlowPhysics &)` [private]

4.10.3 Member Function Documentation

4.10.3.1 `void FlowPhysics::calcEdge2D ()`

Compute the edge order for the stream function.

4.10.3.2 `void FlowPhysics::calcStreamFunc ()`

Traverse the edges to compute the stream func.

4.10.3.3 `void FlowPhysics::calcVorticity ()`

Compute the vorticity.

4.10.3.4 `FlowPhysics& FlowPhysics::operator= (const FlowPhysics &)` [private]

4.10.3.5 `virtual void FlowPhysics::readRestart (ifstream & restf)` [virtual]

Implements `Physics` (p. 85).

Reimplemented in `AdvectionDiffusion` (p. 10).

4.10.3.6 `virtual void FlowPhysics::registerData ()` [virtual]

Reimplemented from `Physics` (p. 85).

Reimplemented in `AdvectionDiffusion` (p. 10).

4.10.3.7 virtual void FlowPhysics::setup (Mesh * *mesh*, Control * *control*, fileIO * *io*) [virtual]

Virtual setup function – implemented for each specific physics.

Reimplemented from **Physics** (p. 86).

Reimplemented in **AdvectionDiffusion** (p. 10).

4.10.3.8 virtual void FlowPhysics::writeRestart (ofstream & *dumpf*) [virtual]

Implements **Physics** (p. 86).

Reimplemented in **AdvectionDiffusion** (p. 10).

4.10.4 Member Data Documentation

4.10.4.1 int FlowPhysics::Npsi [private]

No. of 2-D elements for stream function.

Definition at line 94 of file flow_physics.h.

4.10.4.2 DataIndex FlowPhysics::PSI [protected]

Stream function.

Definition at line 80 of file flow_physics.h.

4.10.4.3 DataIndex FlowPhysics::PSI_EDGE [protected]

Edges for stream function (PSI).

Definition at line 82 of file flow_physics.h.

4.10.4.4 DataIndex FlowPhysics::PSI_EL [protected]

Element, edge for stream function computation.

Definition at line 81 of file flow_physics.h.

4.10.4.5 DataIndex FlowPhysics::VELX [protected]

Y-velocity component.

Definition at line 78 of file flow_physics.h.

4.10.4.6 DataIndex FlowPhysics::VELY [protected]

X-velocity component.

Definition at line 79 of file flow_physics.h.

4.10.4.7 `DataIndex FlowPhysics::VORTICITY` [protected]

Vorticity (z-component).

Definition at line 83 of file `flow_physics.h`.

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

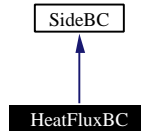
- `flow_physics.h`

4.11 HeatFluxBC Class Reference

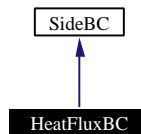
Heat flux boundary condition for sidesets.

```
#include <bcs.h>
```

Inheritance diagram for HeatFluxBC:



Collaboration diagram for HeatFluxBC:



Public Methods

- `void applyBC (Mesh *mesh, Real *q, Real dt)`
Integrate the heat flux over the surface and assemble into {Q}.

Constructor/Destructor

- `HeatFluxBC ()`
- `virtual ~HeatFluxBC ()`

Private Methods

- `HeatFluxBC (const HeatFluxBC &)`
- `HeatFluxBC & operator= (const HeatFluxBC &)`

4.11.1 Detailed Description

Heat flux boundary condition for sidesets.

Definition at line 75 of file bcs.h.

4.11.2 Constructor & Destructor Documentation

4.11.2.1 HeatFluxBC::HeatFluxBC () [inline]

Definition at line 80 of file bcs.h.

80 {}

4.11.2.2 virtual HeatFluxBC::~~HeatFluxBC () [inline, virtual]

Definition at line 81 of file bcs.h.

81 {}

4.11.2.3 HeatFluxBC::HeatFluxBC (const HeatFluxBC &) [private]

4.11.3 Member Function Documentation

4.11.3.1 void HeatFluxBC::applyBC (Mesh * *mesh*, Real * *q*, Real *dt*)

Integrate the heat flux over the surface and assemble into {Q}.

4.11.3.2 HeatFluxBC& HeatFluxBC::operator= (const HeatFluxBC &) [private]

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

- **bcs.h**

4.12 Hex8 Struct Reference

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

8-node hex element

This structure is used to hold all the book-keeping data for the element.

- int **mat**
Material (p. 48) *id*.
- int **ix** [NNPE_HEX8]
Node-to-element connectivity.
- int **faces** [NUM_HEX8_FACES][NNPE_HEX8_FACES]

4.12.1 Member Data Documentation

4.12.1.1 int Hex8::faces[NUM_HEX8_FACES][NNPE_HEX8_FACES] [static]

Definition at line 53 of file mesh.h.

4.12.1.2 int Hex8::ix[NNPE_HEX8]

Node-to-element connectivity.

Definition at line 52 of file mesh.h.

4.12.1.3 int Hex8::mat

Material (p. 48) *id*.

Definition at line 51 of file mesh.h.

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

- **mesh.h**

4.13 Material Class Reference

```
#include <material.h>
```

Public Methods

- **Material** ()
- **Material** (int id, Real rho, Real Cp, Real k11, Real k12, Real k22, Real mu)
- virtual **~Material** ()

Material model interface

- int **MatId** ()
Return the material id.
- **Real Density** ()
- **Real SpecificHeat** ()
- **Real Viscosity** ()
- **Real Conductivity** (Real &k.11, Real &k.12, Real &k.22)

Private Methods

- **Material** (const Material &)
- **Material & operator=** (const Material &)

Private Attributes

- int **id**
Material id.
- **Real rho**
Density.
- **Real Cp**
Specific heat.
- **Real k11**
 k_{xx} term in symmetric tensor for thermal conductivity.
- **Real k12**
 k_{xy} term in symmetric tensor for thermal conductivity.
- **Real k22**
 k_{yy} term in symmetric tensor for thermal conductivity.
- **Real mu**
dynamics viscosity.

4.13.1 Constructor & Destructor Documentation

4.13.1.1 `Material::Material ()` [inline]

Definition at line 21 of file material.h.

```
21 {}
```

4.13.1.2 `Material::Material (int id, Real rho, Real Cp, Real k11, Real k12, Real k22, Real mu)`

4.13.1.3 `virtual Material::~~Material ()` [inline, virtual]

Definition at line 24 of file material.h.

```
24 {}
```

4.13.1.4 `Material::Material (const Material &)` [private]

4.13.2 Member Function Documentation

4.13.2.1 `Real Material::Conductivity (Real & k_11, Real & k_12, Real & k_22)` [inline]

Definition at line 32 of file material.h.

References `k11`, `k12`, `k22`, and `Real`.

```
33     {k_11 = k11; k_12 = k12; k_22 = k22;}
```

4.13.2.2 `Real Material::Density ()` [inline]

Definition at line 29 of file material.h.

References `Real`, and `rho`.

```
29 {return rho;}
```

4.13.2.3 `int Material::MatId ()` [inline]

Return the material id.

Definition at line 28 of file material.h.

References `id`.

4.13.2.4 `Material& Material::operator= (const Material &) [private]`

4.13.2.5 `Real Material::SpecificHeat () [inline]`

Definition at line 30 of file material.h.

References `Cp`, and `Real`.

```
30 {return Cp;}
```

4.13.2.6 `Real Material::Viscosity () [inline]`

Definition at line 31 of file material.h.

References `mu`, and `Real`.

```
31 {return mu;}
```

4.13.3 Member Data Documentation

4.13.3.1 `Real Material::Cp [private]`

Specific heat.

Definition at line 48 of file material.h.

Referenced by `SpecificHeat`.

4.13.3.2 `int Material::id [private]`

Material id.

Definition at line 46 of file material.h.

Referenced by `MatId`.

4.13.3.3 `Real Material::k11 [private]`

`kxx` term in symmetric tensor for thermal conductivity.

Definition at line 49 of file material.h.

Referenced by `Conductivity`.

4.13.3.4 `Real Material::k12 [private]`

`kxy` term in symmetric tensor for thermal conductivity.

Definition at line 50 of file material.h.

Referenced by `Conductivity`.

4.13.3.5 Real Material::k22 [private]

k_{yy} term in symmetric tensor for thermal conductivity.

Definition at line 51 of file material.h.

Referenced by Conductivity.

4.13.3.6 Real Material::mu [private]

dynamics viscosity.

Definition at line 52 of file material.h.

Referenced by Viscosity.

4.13.3.7 Real Material::rho [private]

Density.

Definition at line 47 of file material.h.

Referenced by Density.

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

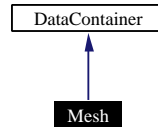
- **material.h**

4.14 Mesh Class Reference

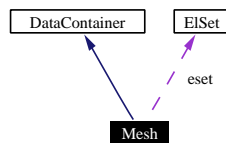
Mesh object – holds basic mesh information, sidesets, nodesets, and data.

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

Inheritance diagram for Mesh:



Collaboration diagram for Mesh:



Public Methods

- **Mesh** ()
- virtual **~Mesh** ()
- void **registerData** ()
Register all of the variables/objects for the mesh.
- void **Print** ()
Print out the element connectivity.
- void **echoMeshParms** (void)
Print out the mesh parameters.
- void **setTitle** (char *cbuf)
Store the mesh title.
- const char * **getTitle** ()
Retrieve a character string for the mesh title.
- string **Title** ()
Retrieve the C++ string for that holds the mesh title.
- void **setMeshParm** (int Ndim, int Nnp, int Nel, int Nnpe, int Nmat, int Nnd_sets, int Nsd_sets, int Nblock)
Set all the relevant mesh parameters.

Mesh Parameter Functions

These functions are used to access the basic parameters that describe the mesh, e.g., the number of nodes, number of elements, etc.

- `const int getNdim ()`
- `const int getNnp ()`
- `const int getNel ()`
- `const int getNmat ()`
- `const int getNnpe ()`
- `void setNmat (const int nmat)`
- `const int getNndsets ()`
- `const int getNsdsets ()`

Nodeset and Sideset Functions

These functions are used to access the nodeset and sideset data for the application of BC's.

- `Nodeset * getNodesets ()`
- `NodeBC * getNodeBCs ()`
- `int * getNbc ()`
- `Sideset * getSidesets ()`
- `ElSet * getElSet ()`

DataIndex variables registered by the datamesh

These functions provide access to the nodal coordinates, and volume

- `Real * getX ()`
- `Real * getY ()`
- `Real * getZ ()`
- `Real * getVolume ()`

Elements based on dimensionality or element type

These functions provide access to the arrays of elements

- `Beam2 * getBeam2 ()`
- `Quad4 * getQuad4 ()`
- `Hex8 * getHex8 ()`

Material model interface

These functions provide access to the material parameters for fluid/thermal problems. More sophisticated functions may be implemented, e.g., for temperature-dependent properties.

- `void addMaterial (int matid, Real rho, Real Cp, Real k11, Real k12, Real k22, Real mu)`
- `int numMaterials ()`
- `Material * getMaterial (int matid)`
- `void checkMaterials ()`
- `void echoMaterials (ostream &ofs)`

Nodal Time-history interface

- `void addTHnode (const string fname, int node)`
- `void echoTHnode (ostream &ofs)`
- `int numTHnode ()`
- `const string getTHname (int i)`
- `int getTHnode (int i)`

Private Types

- typedef map< int, **Material** * > **matmap**
Use a typedef map for fast index to materials.

Private Methods

- **Mesh** (const Mesh &)
- Mesh & **operator=** (const Mesh &)

Private Attributes

- **DataIndex ELEMENTS**
Elements – use one DataIndex for all element types.
- **DataIndex VOLUME**
Volume [Nel].
- **DataIndex SIDESSETS**
Sidesets & bc's.
- **ElSet eset**
Element set used for hydrostatic pressure.
- **matmap materials**
matmap is the real map used to lookup materials by id.

Mesh parameters

- string **title**
Mesh title.
- int **meshtype**
Mesh type.
- int **Ndim**
Dimension of mesh.
- int **Nnp**
Number of nodes.
- int **Nel**
Number of element.
- int **Nnpe**
Number of nodes per element.
- int **Nmat**
Number of materials.

- int **Nface**
Number of faces.
- int **Nedge**
Number of edges.
- int **Nblock**
Number of element blocks (for Exodus).
- int **Nnd_sets**
Number of nodesets.
- int **Nsd_sets**
Number of sidesets.

Coordinates – dimensions are [Nnp]

- **DataIndex XCOOR**
- **DataIndex YCOOR**
- **DataIndex ZCOOR**

Nodesets & nodal bc's

- **DataIndex NODESETS**
- **DataIndex NODEBCS**
- int **nbc** [MAXDOF]

Time-history nodes

- vector< int > **th_nodes**
- vector< string > **th_names**

4.14.1 Detailed Description

Mesh object – holds basic mesh information, sidesets, nodesets, and data.

Definition at line 147 of file mesh.h.

4.14.2 Member Typedef Documentation

4.14.2.1 typedef map<int, Material*> Mesh::matmap [private]

Use a typedef map for fast index to materials.

Definition at line 302 of file mesh.h.

4.14.3 Constructor & Destructor Documentation

4.14.3.1 `Mesh::Mesh ()`

4.14.3.2 `virtual Mesh::~~Mesh () [virtual]`

4.14.3.3 `Mesh::Mesh (const Mesh &) [private]`

4.14.4 Member Function Documentation

4.14.4.1 `void Mesh::addMaterial (int matid, Real rho, Real Cp, Real k11, Real k12, Real k22, Real mu)`

4.14.4.2 `void Mesh::addTHnode (const string fname, int node)`

4.14.4.3 `void Mesh::checkMaterials ()`

4.14.4.4 `void Mesh::echoMaterials (ostream & ofs)`

4.14.4.5 `void Mesh::echoMeshParms (void)`

Print out the mesh parameters.

4.14.4.6 `void Mesh::echoTHnode (ostream & ofs)`

4.14.4.7 `Beam2* Mesh::getBeam2 () [inline]`

Definition at line 222 of file mesh.h.

```
222 {return (Beam2*) getVariable(ELEMENTS);}
```

4.14.4.8 `ElSet* Mesh::getElSet () [inline]`

Definition at line 205 of file mesh.h.

```
205 {return &eset;}
```

4.14.4.9 `Hex8* Mesh::getHex8 () [inline]`

Definition at line 224 of file mesh.h.

```
224 {return (Hex8* ) getVariable(ELEMENTS);}
```

4.14.4.10 `Material* Mesh::getMaterial (int matid) [inline]`

Definition at line 236 of file mesh.h.

```
236 {return materials[matid];}
```

4.14.4.11 `int* Mesh::getNbc ()` [inline]

Definition at line 201 of file mesh.h.

```
201 {return &nbc[0];}
```

4.14.4.12 `const int Mesh::getNdim ()` [inline]

Definition at line 182 of file mesh.h.

```
182 {return Ndim;}
```

4.14.4.13 `const int Mesh::getNel ()` [inline]

Definition at line 184 of file mesh.h.

```
184 {return Nel;}
```

4.14.4.14 `const int Mesh::getNmat ()` [inline]

Definition at line 185 of file mesh.h.

```
185 {return Nmat;}
```

4.14.4.15 `const int Mesh::getNndsets ()` [inline]

Definition at line 190 of file mesh.h.

```
190 {return Nnd_sets;}
```

4.14.4.16 `const int Mesh::getNnp ()` [inline]

Definition at line 183 of file mesh.h.

```
183 {return Nnp; }
```

4.14.4.17 `const int Mesh::getNnpe ()` [inline]

Definition at line 186 of file mesh.h.

```
186 {return Nnpe;}
```

4.14.4.18 NodeBC* Mesh::getNodeBCs () [inline]

Definition at line 200 of file mesh.h.

```
200 {return (NodeBC*)  getVariable(NODEBCS );}
```

4.14.4.19 Nodeset* Mesh::getNodeSets () [inline]

Definition at line 199 of file mesh.h.

```
199 {return (Nodeset*)  getVariable(NODESETS);}
```

4.14.4.20 const int Mesh::getNsdsets () [inline]

Definition at line 191 of file mesh.h.

```
191 {return Nsd_sets;}
```

4.14.4.21 Quad4* Mesh::getQuad4 () [inline]

Definition at line 223 of file mesh.h.

```
223 {return (Quad4*)  getVariable(ELEMENTS);}
```

4.14.4.22 Sideset* Mesh::getSidesets () [inline]

Definition at line 203 of file mesh.h.

```
203 {return (Sideset*)  getVariable(SIDSETS);}
```

4.14.4.23 const string Mesh::getTHname (int *i*) [inline]

Definition at line 247 of file mesh.h.

```
247 {return th_names[i];  }
```

4.14.4.24 int Mesh::getTHnode (int *i*) [inline]

Definition at line 248 of file mesh.h.

```
248 {return th_nodes[i];  }
```

4.14.4.25 `const char* Mesh::getTitle () [inline]`

Retrieve a character string for the mesh title.

Definition at line 167 of file mesh.h.

```
167 {return title.c_str();}
```

4.14.4.26 `Real* Mesh::getVolume () [inline]`

Definition at line 215 of file mesh.h.

References Real.

```
215 {return (Real*) getVariable(VOLUME);}
```

4.14.4.27 `Real* Mesh::getX () [inline]`

Definition at line 212 of file mesh.h.

References Real.

```
212 {return (Real*) getVariable(XCOORD); }
```

4.14.4.28 `Real* Mesh::getY () [inline]`

Definition at line 213 of file mesh.h.

References Real.

```
213 {return (Real*) getVariable(YCOORD); }
```

4.14.4.29 `Real* Mesh::getZ () [inline]`

Definition at line 214 of file mesh.h.

References Real.

```
214 {return (Real*) getVariable(ZCOORD); }
```

4.14.4.30 `int Mesh::numMaterials () [inline]`

Definition at line 235 of file mesh.h.

```
235 {return materials.size();}
```

4.14.4.31 `int Mesh::numTHnode () [inline]`

Definition at line 246 of file mesh.h.

```
246 {return th_nodes.size();}
```

4.14.4.32 `Mesh& Mesh::operator= (const Mesh &) [private]`**4.14.4.33** `void Mesh::Print ()`

Print out the element connectivity.

4.14.4.34 `void Mesh::registerData ()`

Register all of the variables/objects for the mesh.

4.14.4.35 `void Mesh::setMeshParm (int Ndim, int Nnp, int Nel, int Nnpe, int Nmat, int Nnd_sets, int Nsd_sets, int Nblock)`

Set all the relevant mesh parameters.

4.14.4.36 `void Mesh::setNmat (const int nmat) [inline]`

Definition at line 188 of file mesh.h.

```
188 {Nmat = nmat;}
```

4.14.4.37 `void Mesh::setTitle (char * cbuf) [inline]`

Store the mesh title.

Definition at line 164 of file mesh.h.

```
164 {title = cbuf;}
```

4.14.4.38 `string Mesh::Title () [inline]`

Retrieve the C++ string for that holds the mesh title.

Definition at line 170 of file mesh.h.

```
170 {return title;}
```

4.14.5 Member Data Documentation**4.14.5.1** `DataIndex Mesh::ELEMENTS [private]`

Elements – use one DataIndex for all element types.

Definition at line 275 of file mesh.h.

4.14.5.2 ElSet Mesh::eset [private]

Element set used for hydrostatic pressure.

Definition at line 299 of file mesh.h.

4.14.5.3 matmap Mesh::materials [private]

matmap is the real map used to lookup materials by id.

Definition at line 304 of file mesh.h.

4.14.5.4 int Mesh::meshtype [private]

Mesh type.

Definition at line 261 of file mesh.h.

4.14.5.5 int Mesh::nbc[MAXDOF] [private]

Definition at line 292 of file mesh.h.

4.14.5.6 int Mesh::Ndim [private]

Dimension of mesh.

Definition at line 262 of file mesh.h.

4.14.5.7 int Mesh::Neblock [private]

Number of element blocks (for Exodus).

Definition at line 269 of file mesh.h.

4.14.5.8 int Mesh::Nedge [private]

Number of edges.

Definition at line 268 of file mesh.h.

4.14.5.9 int Mesh::Nel [private]

Number of element.

Definition at line 264 of file mesh.h.

4.14.5.10 int Mesh::Nface [private]

Number of faces.

Definition at line 267 of file mesh.h.

4.14.5.11 `int Mesh::Nmat` [private]

Number of materials.

Definition at line 266 of file mesh.h.

4.14.5.12 `int Mesh::Nnd_sets` [private]

Number of nodesets.

Definition at line 270 of file mesh.h.

4.14.5.13 `int Mesh::Nnp` [private]

Number of nodes.

Definition at line 263 of file mesh.h.

4.14.5.14 `int Mesh::Nnpe` [private]

Number of nodes per element.

Definition at line 265 of file mesh.h.

4.14.5.15 `DataIndex Mesh::NODEBCS` [private]

Definition at line 290 of file mesh.h.

4.14.5.16 `DataIndex Mesh::NODESETS` [private]

Definition at line 289 of file mesh.h.

4.14.5.17 `int Mesh::Nsd_sets` [private]

Number of sidesets.

Definition at line 271 of file mesh.h.

4.14.5.18 `DataIndex Mesh::SIDESETS` [private]

Sidesets & bc's.

Definition at line 296 of file mesh.h.

4.14.5.19 `vector<string> Mesh::th_names` [private]

Definition at line 309 of file mesh.h.

4.14.5.20 `vector<int> Mesh::th_nodes` [private]

Definition at line 308 of file mesh.h.

4.14.5.21 `string Mesh::title` [private]

Mesh title.

Definition at line 260 of file mesh.h.

4.14.5.22 `DataIndex Mesh::VOLUME` [private]

Volume [Nel].

Definition at line 285 of file mesh.h.

4.14.5.23 `DataIndex Mesh::XCOOR` [private]

Definition at line 279 of file mesh.h.

4.14.5.24 `DataIndex Mesh::YCOOR` [private]

Definition at line 280 of file mesh.h.

4.14.5.25 `DataIndex Mesh::ZCOOR` [private]

Definition at line 281 of file mesh.h.

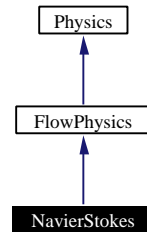
The documentation for this class was generated from the following file:

- **mesh.h**

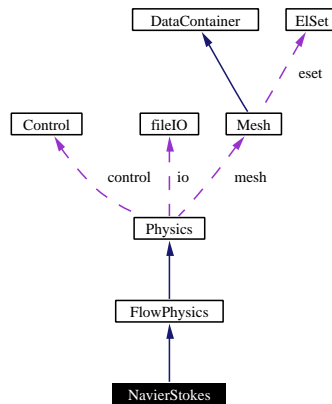
4.15 NavierStokes Class Reference

```
#include <navier_stokes.h>
```

Inheritance diagram for NavierStokes:



Collaboration diagram for NavierStokes:



Public Methods

Constructor/Destructor

- `NavierStokes ()`
- `virtual ~NavierStokes ()`

Data Registration

Each physics that is implemented will require its own storage and specific variables. The data registration function is implemented for each physics class to enable custom definition of variables for each algorithm that's implemented.

- `virtual void registerData ()`

Virtual Physics Functions

These functions are pure virtual functions that need to be implemented for each specific type of physics being solved with the framework.

- `virtual void setup (Mesh *mesh, Control *control, fileIO *io)`

Virtual setup function – implemented for each specific physics.

- void **solve** ()
Virtual solver – implemented for advection-diffusion physics.
- virtual void **setICs** ()
Virtual function to setup initial conditions for Navier-Stokes.

Pressure-Poisson Equation (PPE)

- void **setupPPE** ()
register variables, compute bandwidth, etc.
- void **genElCon** ()
calculate the element-to-element connectivity.
- void **formPPE** ()
Form the PPE operator.
- void **solvePPE** ()
driver to solve the PPE problem.
- void **reorderP** ()
reorder the pressure for minimum bandwidth.
- void **reorderRHS** ()
reorder the PPE rhs for minimum bandwidth.
- void **nodalPressure** ()
compute nodal pressures for graphics output.

Basic operators and forces

- void **formMpK** ()
Form the M+K and M-K operators.
- void **formCMK** ()
Form the C, M and K operators for startup.
- void **calcForces** ()
Compute the force terms for the momentum equations.

Mass matrix & BC operations for the Projection algorithm

- void **setupInverseMass** ()
setup inverse mass + EBC's.
- void **applyInverseMassBCs** (**DOFmap** dof, **DataIndex** index)
set zero's in inverse mass according to EBC's.

Projection machinery & initial pressure calculation

- void **divFree** ()
perform the startup projection.
- void **divU** (**DataIndex** U, **DataIndex** V)
compute the div for a given velocity field.
- **Real** **rmsDiv** ()
calculate the current RMS divergence.
- void **projectVelocity** (**Real** dt)
L2 projection onto div-free subspace.
- void **initPressure** ()
initialize the pressure field at t=0.

Global utility functions

- **Real** **calcKE** (**DataIndex** U, **DataIndex** V, **DataIndex** ML)
Compute the global kinetic energy.

Virtual Functions for I/O

These functions are implemented for each specific physics to permit custom options to be coded for reading and writing restart files, time-history data, bc's, etc.

- virtual void **writeRestart** (ofstream &dumpf)
- virtual void **readRestart** (ifstream &restf)
- virtual void **writeTHhead** (**Mesh** *mesh, **fileIO** *io)
- virtual void **writeTHdata** (**Mesh** *mesh, **fileIO** *io, **Real** time)
- virtual void **writeSolving** ()
- virtual void **echoBCs** (**Mesh** *mesh, ostream &ofs)

I/O functions specific to this flow solver

- void **echoElSet** (**Mesh** *mesh, ostream &ofs)
Write out any element sets used to peg pressure hydrostats.
- void **echoPPEStats** (ostream &ofs)
Write out the PPE statistics: bandwidth, etc.
- void **globHeader** (ofstream &globf)
Write the header for the global time-history data file.
- void **writeGlobal** (ofstream &globf, **Real** time, **Real** divu, **Real** ke)
Write the global time history data at each step.

Protected Attributes

Data required for addition of advection-diffusion – not used

- **DataIndex** TEMPERATURE
- **DataIndex** Q

Nodal data for P2 solver stored in the datamesh.

Allocate vectors as $[Nnp]$

- **DataIndex VELXN**
X-velocity at $n+1$.
- **DataIndex VELYN**
Y-velocity at $n+1$.
- **DataIndex FX**
X-force at n .
- **DataIndex FY**
Y-force at n .
- **DataIndex FXN**
X-force at $n+1$ (not used).
- **DataIndex FYN**
X-force at $n+1$ (not used).
- **DataIndex RIMX**
X-inverse lumped mass w. EBC's imposed.
- **DataIndex RIMY**
Y-inverse lumped mass w. EBC's imposed.
- **DataIndex CONST_MASS**
Consistent mass matrix.
- **DataIndex LUMPED_MASS**
Lumped mass matrix.
- **DataIndex DM**
Temporary to hold diagonal of $M+K$.
- **DataIndex V1**
Scratch vector.
- **DataIndex V2**
Scratch vector.

Element operators for P2 solver stored in the datamesh.

Allocate as $[Nel]$

- **DataIndex IEN**
New element number in terms of old (not used).
- **DataIndex IEO**
Old element number in terms of new.
- **DataIndex P**
Pressure at n .

- **DataIndex PN**
Pressure at $n+1$.
- **DataIndex RHS**
PPE right-hand-side.
- **DataIndex CX**
X-gradient operator.
- **DataIndex CY**
Y-gradient operator.

PPE storage for the PVS solver & bandwidth minimization.

- **DataIndex ROWL**
Row lengths.
- **DataIndex COLH**
Column lengths.
- **DataIndex MAXA**
Maximum row in the matrix.
- **DataIndex PPE**
Coefficient matrix for the PPE.
- **DataIndex IDEG**
Degree array for bandwidth minimization.
- **DataIndex IRST**
Working storage for bandwidth minimization.
- **DataIndex IERV**
Reorder vector for bandwidth minimization.
- **DataIndex IECN**
Element-to-element connectivity.
- **DataIndex MKJE**
Column pointers for PPE operator.

Private Methods

- **NavierStokes** (const NavierStokes &)
- **NavierStokes & operator=** (const NavierStokes &)

Private Attributes

- **int Npe**
No. of pressure elements.

- int **Elbwi**
initial 1/2-bandwidth.
- int **Elbwf**
final 1/2-bandwidth.
- int **Ndwe**
No. of non-zero row entries for PPE.
- int **Npre**
profile.
- int **ppe_na**
size of ppe for PVS.
- int **luflag**
LU-flag for PVS.

Element operators

- **Real Me** [4][4]
Element consistent mass matrix.
- **Real Mle** [4]
Element lumped mass matrix.
- **Real Ke** [4][4]
Element conductivity matrix.
- **Real C** [2][4]
Element gradient (Cx,Cy) operators.

4.15.1 Detailed Description

The NavierStokes class is used for solving the time-dependent incompressible Navier-Stokes equations.

Definition at line 40 of file `navier_stokes.h`.

4.15.2 Constructor & Destructor Documentation

4.15.2.1 NavierStokes::NavierStokes () [inline]

Definition at line 46 of file `navier_stokes.h`.

46 {}

4.15.2.2 `virtual NavierStokes::~~NavierStokes ()` [inline, virtual]

Definition at line 47 of file `navier_stokes.h`.

47 }

4.15.2.3 `NavierStokes::NavierStokes (const NavierStokes &)` [private]

4.15.3 Member Function Documentation

4.15.3.1 `void NavierStokes::applyInverseMassBCs (DOFmap dof, DataIndex index)`

set zero's in inverse mass according to EBC's.

4.15.3.2 `void NavierStokes::calcForces ()`

Compute the force terms for the momentum equations.

4.15.3.3 `Real NavierStokes::calcKE (DataIndex U, DataIndex V, DataIndex ML)`

Compute the global kinetic energy.

4.15.3.4 `void NavierStokes::divFree ()`

perform the startup projection.

4.15.3.5 `void NavierStokes::divU (DataIndex U, DataIndex V)`

compute the div for a given velocity field.

4.15.3.6 `virtual void NavierStokes::echoBCs (Mesh * mesh, ostream & ofs)`
[virtual]

Implements `Physics` (p. 85).

4.15.3.7 `void NavierStokes::echoElSet (Mesh * mesh, ostream & ofs)`

Write out any element sets used to peg pressure hydrostats.

4.15.3.8 `void NavierStokes::echoPPEStats (ostream & ofs)`

Write out the PPE statistics: bandwidth, etc.

4.15.3.9 `void NavierStokes::formCMK ()`

Form the C, M and K operators for startup.

4.15.3.10 void NavierStokes::formMpK ()

Form the M+K and M-K operators.

4.15.3.11 void NavierStokes::formPPE ()

Form the PPE operator.

4.15.3.12 void NavierStokes::genElCon ()

calculate the element-to-element connectivity.

4.15.3.13 void NavierStokes::globHeader (ofstream & *globf*)

Write the header for the global time-history data file.

4.15.3.14 void NavierStokes::initPressure ()

initialize the pressure field at t=0.

4.15.3.15 void NavierStokes::nodalPressure ()

compute nodal pressures for graphics output.

**4.15.3.16 NavierStokes& NavierStokes::operator= (const NavierStokes &)
[private]****4.15.3.17 void NavierStokes::projectVelocity (Real *dt*)**

L2 projection onto div-free subspace.

4.15.3.18 virtual void NavierStokes::readRestart (ifstream & *restf*) [virtual]

Reimplemented from **FlowPhysics** (p. 42).

4.15.3.19 virtual void NavierStokes::registerData () [virtual]

Reimplemented from **FlowPhysics** (p. 42).

4.15.3.20 void NavierStokes::reorderP ()

reorder the pressure for minimum bandwidth.

4.15.3.21 void NavierStokes::reorderRHS ()

reorder the PPE rhs for minimum bandwidth.

4.15.3.22 Real NavierStokes::rmsDiv ()

calculate the current RMS divergence.

4.15.3.23 virtual void NavierStokes::setICs () [virtual]

Virtual function to setup initial conditions for Navier-Stokes.

Reimplemented from **Physics** (p. 86).

4.15.3.24 virtual void NavierStokes::setup (Mesh * *mesh*, Control * *control*, fileIO * *io*) [virtual]

Virtual setup function – implemented for each specific physics.

Reimplemented from **FlowPhysics** (p. 43).

4.15.3.25 void NavierStokes::setupInverseMass ()

setup inverse mass + EBC's.

4.15.3.26 void NavierStokes::setupPPE ()

register variables, compute bandwidth, etc.

4.15.3.27 void NavierStokes::solve () [virtual]

Virtual solver – implemented for advection-diffusion physics.

Reimplemented from **Physics** (p. 86).

4.15.3.28 void NavierStokes::solvePPE ()

driver to solve the PPE problem.

4.15.3.29 void NavierStokes::writeGlobal (ofstream & *globf*, Real *time*, Real *divu*, Real *ke*)

Write the global time history data at each step.

4.15.3.30 virtual void NavierStokes::writeRestart (ofstream & *dumpf*) [virtual]

Reimplemented from **FlowPhysics** (p. 43).

4.15.3.31 virtual void NavierStokes::writeSolving () [virtual]

Implements **Physics** (p. 86).

4.15.3.32 `virtual void NavierStokes::writeTHdata (Mesh * mesh, fileIO * io, Real time)` [virtual]

Implements **Physics** (p. 87).

4.15.3.33 `virtual void NavierStokes::writeTHhead (Mesh * mesh, fileIO * io)` [virtual]

Implements **Physics** (p. 87).

4.15.4 Member Data Documentation

4.15.4.1 `Real NavierStokes::C[2][4]` [private]

Element gradient (Cx,Cy) operators.

Definition at line 250 of file `navier_stokes.h`.

4.15.4.2 `DataIndex NavierStokes::COLH` [protected]

Column lengths.

Definition at line 216 of file `navier_stokes.h`.

4.15.4.3 `DataIndex NavierStokes::CONST_MASS` [protected]

Consistent mass matrix.

Definition at line 192 of file `navier_stokes.h`.

4.15.4.4 `DataIndex NavierStokes::CX` [protected]

X-gradient operator.

Definition at line 208 of file `navier_stokes.h`.

4.15.4.5 `DataIndex NavierStokes::CY` [protected]

Y-gradient operator.

Definition at line 209 of file `navier_stokes.h`.

4.15.4.6 `DataIndex NavierStokes::DM` [protected]

Temporary to hold diagonal of M+K.

Definition at line 194 of file `navier_stokes.h`.

4.15.4.7 `int NavierStokes::Elbwf` [private]

final 1/2-bandwidth.

Definition at line 238 of file `navier_stokes.h`.

4.15.4.8 `int NavierStokes::Elbwi` [private]

initial 1/2-bandwidth.

Definition at line 237 of file `navier_stokes.h`.

4.15.4.9 `DataIndex NavierStokes::FX` [protected]

X-force at n .

Definition at line 186 of file `navier_stokes.h`.

4.15.4.10 `DataIndex NavierStokes::FXN` [protected]

X-force at $n+1$ (not used).

Definition at line 188 of file `navier_stokes.h`.

4.15.4.11 `DataIndex NavierStokes::FY` [protected]

Y-force at n .

Definition at line 187 of file `navier_stokes.h`.

4.15.4.12 `DataIndex NavierStokes::FYN` [protected]

X-force at $n+1$ (not used).

Definition at line 189 of file `navier_stokes.h`.

4.15.4.13 `DataIndex NavierStokes::IDEG` [protected]

Degree array for bandwidth minimization.

Definition at line 219 of file `navier_stokes.h`.

4.15.4.14 `DataIndex NavierStokes::IECN` [protected]

Element-to-element connectivity.

Definition at line 222 of file `navier_stokes.h`.

4.15.4.15 `DataIndex NavierStokes::IEN` [protected]

New element number in terms of old (not used).

Definition at line 203 of file `navier_stokes.h`.

4.15.4.16 DataIndex NavierStokes::IEO [protected]

Old element number in terms of new.

Definition at line 204 of file navier_stokes.h.

4.15.4.17 DataIndex NavierStokes::IERV [protected]

Reorder vector for bandwidth minimization.

Definition at line 221 of file navier_stokes.h.

4.15.4.18 DataIndex NavierStokes::IRST [protected]

Working storage for bandwidth minimization.

Definition at line 220 of file navier_stokes.h.

4.15.4.19 Real NavierStokes::Ke[4][4] [private]

Element conductivity matrix.

Definition at line 249 of file navier_stokes.h.

4.15.4.20 int NavierStokes::luflag [private]

LU-flag for PVS.

Definition at line 243 of file navier_stokes.h.

4.15.4.21 DataIndex NavierStokes::LUMPED_MASS [protected]

Lumped mass matrix.

Definition at line 193 of file navier_stokes.h.

4.15.4.22 DataIndex NavierStokes::MAXA [protected]

Maximum row in the matrix.

Definition at line 217 of file navier_stokes.h.

4.15.4.23 Real NavierStokes::Me[4][4] [private]

Element consistent mass matrix.

Definition at line 247 of file navier_stokes.h.

4.15.4.24 DataIndex NavierStokes::MKJE [protected]

Column pointers for PPE operator.

Definition at line 223 of file navier_stokes.h.

4.15.4.25 Real NavierStokes::Mle[4] [private]

Element lumped mass matrix.

Definition at line 248 of file navier_stokes.h.

4.15.4.26 int NavierStokes::Ndwe [private]

No. of non-zero row entries for PPE.

Definition at line 239 of file navier_stokes.h.

4.15.4.27 int NavierStokes::Npe [private]

No. of pressure elements.

Definition at line 236 of file navier_stokes.h.

4.15.4.28 int NavierStokes::Npre [private]

profile.

Definition at line 240 of file navier_stokes.h.

4.15.4.29 DataIndex NavierStokes::P [protected]

Pressure at n.

Definition at line 205 of file navier_stokes.h.

4.15.4.30 DataIndex NavierStokes::PN [protected]

Pressure at n+1.

Definition at line 206 of file navier_stokes.h.

4.15.4.31 DataIndex NavierStokes::PPE [protected]

Coefficient matrix for the PPE.

Definition at line 218 of file navier_stokes.h.

4.15.4.32 int NavierStokes::ppe_na [private]

size of ppe for PVS.

Definition at line 241 of file navier_stokes.h.

4.15.4.33 DataIndex NavierStokes::Q [protected]

Definition at line 176 of file navier_stokes.h.

4.15.4.34 DataIndex NavierStokes::RHS [protected]

PPE right-hand-side.

Definition at line 207 of file navier_stokes.h.

4.15.4.35 DataIndex NavierStokes::RIMX [protected]

X-inverse lumped mass w. EBC's imposed.

Definition at line 190 of file navier_stokes.h.

4.15.4.36 DataIndex NavierStokes::RIMY [protected]

Y-inverse lumped mass w. EBC's imposed.

Definition at line 191 of file navier_stokes.h.

4.15.4.37 DataIndex NavierStokes::ROWL [protected]

Row lengths.

Definition at line 215 of file navier_stokes.h.

4.15.4.38 DataIndex NavierStokes::TEMPERATURE [protected]

Definition at line 175 of file navier_stokes.h.

4.15.4.39 DataIndex NavierStokes::V1 [protected]

Scratch vector.

Definition at line 195 of file navier_stokes.h.

4.15.4.40 DataIndex NavierStokes::V2 [protected]

Scratch vector.

Definition at line 196 of file navier_stokes.h.

4.15.4.41 DataIndex NavierStokes::VELXN [protected]

X-velocity at n+1.

Definition at line 184 of file navier_stokes.h.

4.15.4.42 DataIndex NavierStokes::VELYN [protected]

Y-velocity at n+1.

Definition at line 185 of file navier_stokes.h.

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

- `navier_stokes.h`

4.16 NodeBC Struct Reference

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

Public Attributes

Nodal BC structures

The **NodeBC** (p. 79) structure holds nodal BC data from the control file.

- **int nsid** [MAXDOF]
internal Nodeset (p. 80) id.
- **int nset** [MAXDOF]
internal nodeset number.
- **int lcid** [MAXDOF]
Load curve id.
- **Real amp** [MAXDOF]
Amplitude – used for nodal BC values.

4.16.1 Member Data Documentation

4.16.1.1 Real NodeBC::amp[MAXDOF]

Amplitude – used for nodal BC values.

Definition at line 110 of file mesh.h.

4.16.1.2 int NodeBC::lcid[MAXDOF]

Load curve id.

Definition at line 109 of file mesh.h.

4.16.1.3 int NodeBC::nset[MAXDOF]

internal nodeset number.

Definition at line 108 of file mesh.h.

4.16.1.4 int NodeBC::nsid[MAXDOF]

internal **Nodeset** (p. 80) id.

Definition at line 107 of file mesh.h.

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

- **mesh.h**

4.17 Nodeset Struct Reference

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

Public Attributes

Nodeset data structure

The **Nodeset** (p. 80) data structure is used to hold input nodeset data for the mesh.

- **int id**
Nodeset id.
- **int Nnp**
Number of nodes in nodeset.
- **DataIndex NODE_LIST**
Node list for nodeset.

4.17.1 Member Data Documentation

4.17.1.1 int Nodeset::id

Nodeset id.

Definition at line 95 of file mesh.h.

4.17.1.2 int Nodeset::Nnp

Number of nodes in nodeset.

Definition at line 96 of file mesh.h.

4.17.1.3 DataIndex Nodeset::NODE_LIST

Node list for nodeset.

Definition at line 97 of file mesh.h.

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

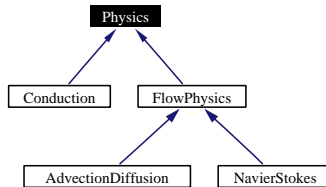
- **mesh.h**

4.18 Physics Class Reference

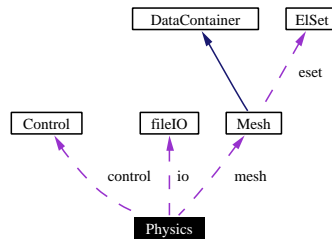
Base class used to construct the solution algorithms for a given physics.

```
#include <physics.h>
```

Inheritance diagram for Physics:



Collaboration diagram for Physics:



Public Methods

- void **calcVolume** ()
Calculate the element volumes.
- void **checkVolume** ()
Check element volumes to be sure that there are no malformed elements.
- void **echoNodalBCs** (Mesh *mesh, ostream &ofs)
- void **FieldLimits** (DataIndex index, Real &fmin, Real &fmax)

Constructor/Destructor

- **Physics** ()
Constructor for Physics class.
- virtual **~Physics** ()
Destructor for Physics class.

Virtual Physics Functions

These functions are pure virtual functions that need to be implemented for each specific type of physics being solved with the framework.

- virtual void **setup** (**Mesh** *cur_mesh, **Control** *cur_control, **fileIO** *cur_io)
Virtual setup function – implemented for each specific physics.
- virtual void **solve** ()
Virtual solver – implemented for each specific physics.
- virtual void **setICs** ()
Virtual function to setup initial conditions – one per physics.

Nodal Boundary Conditions

These functions implement the specification of essential BC's using the penalty method.

- void **applyNodalBC** (**DOFmap** dof, **DataIndex** index, **Real** scale)
- void **applyPenaltyLHS** (**Real** *lhs, **DOFmap** dof)
- void **applyPenaltyRHS** (**Real** *lhs, **Real** *rhs, **DOFmap** dof, **Real** scale)

Data Registration

Each physics that is implemented will require its own storage and specific variables. The data registration function is implemented for each physics class to enable custom definition of variables for each algorithm that's implemented.

- virtual void **registerData** ()

Virtual Functions for I/O

These functions are implemented for each specific physics to permit custom options to be coded for reading and writing restart files, time-history data, bc's, etc.

- virtual void **writeRestart** (ofstream &dumpf)=0
- virtual void **readRestart** (ifstream &restf)=0
- virtual void **writeTHhead** (**Mesh** *mesh, **fileIO** *io)=0
- virtual void **writeTHdata** (**Mesh** *mesh, **fileIO** *io, **Real** time)=0
- virtual void **writeSolving** ()=0
- virtual void **echoBCs** (**Mesh** *mesh, ostream &ofs)=0

Linear Algebra Functions

These utility functions are used to setup the storage for the nodal linear algebra and for the FEM assembly process.

- void **calcNodalBandwidth** ()
- void **echoNodalStats** (ostream &ofs)
- void **setColumnIds** (int *mkj, int *ix, int Nnp, int Nnpe)
- void **EdgeToNode** ()
- void **setupNodeIndex** ()

Gaussian quadrature & element shape functions

These functions setup the Gaussian quadrature and the pre-computed shape functions and their derivatives.

- void **initQuadrature** ()
- void **setupQuad4** (int Nnpe)
- void **setupHex8** (int Nnpe)
- **Real** sf4 (**Real** xi, **Real** eta, int node)
- **Real** sf4xi (**Real** xi, **Real** eta, int node)
- **Real** sf4eta (**Real** xi, **Real** eta, int node)

Protected Attributes

Nodal statistics (assumes 1-DOF per node)

These variables are used to keep track of topology related aspects of the nodes in a mesh

- **int Nodebw**
Nodal 1/2-bandwidth.
- **int mxNeln**
Max. elements connected to a node.
- **int mxNedn**
Max. edges connected to a node.
- **int mxNd**
Node where max. occurs.
- **int mxNdrow**
Maximum nodal row size.

Data required for linear algebra

- **DataIndex MPK**
DataIndex for the M+K operator.
- **DataIndex MMK**
DataIndex for the M-K operator.
- **DataIndex MKJ**
DataIndex for the compressed row storage column pointers.

Temporary storage for required for the C-G algorithm

These DataIndices are used for the temporary storage required for the C-G solution algorithm for nodal variables. Note that TMP is simply a 'temporary' variable.

- **DataIndex PP**
Search direction p.
- **DataIndex AP**
For the Ap product.
- **DataIndex W**
The W scaling matrix – really the diagonal of A.
- **DataIndex R**
The residual vector r.
- **DataIndex Z**
The pseudo-residual in PCCG.
- **DataIndex TMP**
Just a temporary vector.

FEM assembly data and penalty parameter

- **Real lambda**
The Penalty parameter.
- **int col_id [4][4]**
Data for assembly procedure.

Data for quadrature rules

- **int Nqpt**
Total number of quadrature points.
- **Real sf [8][8][4]**
The sf array ala class notes.
- **Real qpt [8][3]**
The quadrature data.
- **Real ssf [8][8][3]**
The surface shape functions.
- **Real sqpt [4][3]**
The surface quadrature data.

Objects used - NOT owned

- **Mesh * mesh**
The finite element mesh.
- **Control * control**
*The **Control** (p. 20) object holds all of the control info.*
- **fileIO * io**
*The **fileIO** (p. 32) object is used for all I/O operations.*

Private Methods

- **Physics** (const Physics &)
- **Physics & operator=** (const Physics &)

4.18.1 Detailed Description

Base class used to construct the solution algorithms for a given physics.

Definition at line 29 of file physics.h.

4.18.2 Constructor & Destructor Documentation

4.18.2.1 Physics::Physics ()

Constructor for Physics class.

4.18.2.2 virtual Physics::~Physics () [inline, virtual]

Destructor for Physics class.

Definition at line 36 of file physics.h.

4.18.2.3 Physics::Physics (const Physics &) [private]**4.18.3 Member Function Documentation****4.18.3.1 void Physics::applyNodalBC (DOFmap *dof*, DataIndex *index*, Real *scale*)****4.18.3.2 void Physics::applyPenaltyLHS (Real * *lhs*, DOFmap *dof*)****4.18.3.3 void Physics::applyPenaltyRHS (Real * *lhs*, Real * *rhs*, DOFmap *dof*, Real *scale*)****4.18.3.4 void Physics::calcNodalBandwidth ()****4.18.3.5 void Physics::calcVolume ()**

Calculate the element volumes.

4.18.3.6 void Physics::checkVolume ()

Check element volumes to be sure that there are no malformed elements.

4.18.3.7 virtual void Physics::echoBCs (Mesh * *mesh*, ostream & *ofs*) [pure virtual]

Implemented in **AdvectionDiffusion** (p. 9).

4.18.3.8 void Physics::echoNodalBCs (Mesh * *mesh*, ostream & *ofs*)**4.18.3.9 void Physics::echoNodalStats (ostream & *ofs*)****4.18.3.10 void Physics::EdgeToNode ()****4.18.3.11 void Physics::FieldLimits (DataIndex *index*, Real & *fmin*, Real & *fmax*)****4.18.3.12 void Physics::initQuadrature ()****4.18.3.13 Physics& Physics::operator= (const Physics &)** [private]**4.18.3.14 virtual void Physics::readRestart (ifstream & *restf*)** [pure virtual]

Implemented in **AdvectionDiffusion** (p. 10).

4.18.3.15 virtual void Physics::registerData () [virtual]

Reimplemented in **AdvectionDiffusion** (p. 10).

4.18.3.16 void Physics::setColumnIds (int * *mkj*, int * *ix*, int *Nnp*, int *Nnpe*)

4.18.3.17 virtual void Physics::setICs () [inline, virtual]

Virtual function to setup initial conditions – one per physics.

Reimplemented in **AdvectionDiffusion** (p. 10).

Definition at line 58 of file physics.h.

```
58 {cout << "bogus IC's" << endl;}
```

4.18.3.18 virtual void Physics::setup (Mesh * *cur_mesh*, Control * *cur_control*, fileIO * *cur_io*) [virtual]

Virtual setup function – implemented for each specific physics.

Reimplemented in **AdvectionDiffusion** (p. 10).

4.18.3.19 void Physics::setupHex8 (int *Nnpe*) [inline]

Definition at line 132 of file physics.h.

```
133     {cout << "Hex8 not implemented yet !!! " << endl;}
```

4.18.3.20 void Physics::setupNodeIndex ()

4.18.3.21 void Physics::setupQuad4 (int *Nnpe*)

4.18.3.22 Real Physics::sf4 (Real *xi*, Real *eta*, int *node*)

4.18.3.23 Real Physics::sf4eta (Real *xi*, Real *eta*, int *node*)

4.18.3.24 Real Physics::sf4xi (Real *xi*, Real *eta*, int *node*)

4.18.3.25 virtual void Physics::solve () [inline, virtual]

Virtual solver – implemented for each specific physics.

Reimplemented in **AdvectionDiffusion** (p. 10).

Definition at line 55 of file physics.h.

```
55 {cout << "!!! No default physics !!!" << endl;}
```

4.18.3.26 virtual void Physics::writeRestart (ofstream & *dumpf*) [pure virtual]

Implemented in **AdvectionDiffusion** (p. 10).

4.18.3.27 virtual void Physics::writeSolving () [pure virtual]

Implemented in **AdvectionDiffusion** (p. 10).

4.18.3.28 virtual void **Physics::writeTHdata** (Mesh * *mesh*, fileIO * *io*, Real *time*)
[pure virtual]

Implemented in **AdvectionDiffusion** (p. 10).

4.18.3.29 virtual void **Physics::writeTHhead** (Mesh * *mesh*, fileIO * *io*) [pure virtual]

Implemented in **AdvectionDiffusion** (p. 11).

4.18.4 Member Data Documentation

4.18.4.1 **DataIndex Physics::AP** [protected]

For the Ap product.

Definition at line 171 of file physics.h.

4.18.4.2 **int Physics::col_id[4][4]** [protected]

Data for assembly procedure.

Definition at line 185 of file physics.h.

4.18.4.3 **Control* Physics::control** [protected]

The **Control** (p. 20) object holds all of the control info.

Definition at line 200 of file physics.h.

4.18.4.4 **fileIO* Physics::io** [protected]

The **fileIO** (p. 32) object is used for all I/O operations.

Definition at line 201 of file physics.h.

4.18.4.5 **Real Physics::lambda** [protected]

The Penalty parameter.

Definition at line 182 of file physics.h.

4.18.4.6 **Mesh* Physics::mesh** [protected]

The finite element mesh.

Definition at line 199 of file physics.h.

4.18.4.7 **DataIndex Physics::MKJ** [protected]

DataIndex for the compressed row storage column pointers.

Definition at line 161 of file physics.h.

4.18.4.8 DataIndex Physics::MMK [protected]

DataIndex for the M-K operator.

Definition at line 160 of file physics.h.

4.18.4.9 DataIndex Physics::MPK [protected]

DataIndex for the M+K operator.

Definition at line 159 of file physics.h.

4.18.4.10 int Physics::mxNd [protected]

Node where max. occurs.

Definition at line 153 of file physics.h.

4.18.4.11 int Physics::mxNdrow [protected]

Maximum nodal row size.

Definition at line 154 of file physics.h.

4.18.4.12 int Physics::mxNedn [protected]

Max. edges connected to a node.

Definition at line 152 of file physics.h.

4.18.4.13 int Physics::mxNeln [protected]

Max. elements connected to a node.

Definition at line 151 of file physics.h.

4.18.4.14 int Physics::Nodebw [protected]

Nodal 1/2-bandwidth.

Definition at line 150 of file physics.h.

4.18.4.15 int Physics::Nqpt [protected]

Total number of quadrature points.

Definition at line 190 of file physics.h.

4.18.4.16 DataIndex Physics::PP [protected]

Search direction p.

Definition at line 170 of file physics.h.

4.18.4.17 Real Physics::qpt[8][3] [protected]

The quadrature data.

Definition at line 192 of file physics.h.

4.18.4.18 DataIndex Physics::R [protected]

The residual vector r.

Definition at line 173 of file physics.h.

4.18.4.19 Real Physics::sf[8][8][4] [protected]

The sf array ala class notes.

Definition at line 191 of file physics.h.

4.18.4.20 Real Physics::sqpt[4][3] [protected]

The surface quadrature data.

Definition at line 194 of file physics.h.

4.18.4.21 Real Physics::ssf[8][8][3] [protected]

The surface shape functions.

Definition at line 193 of file physics.h.

4.18.4.22 DataIndex Physics::TMP [protected]

Just a temporary vector.

Definition at line 175 of file physics.h.

4.18.4.23 DataIndex Physics::W [protected]

The W scaling matrix – really the diagonal of A.

Definition at line 172 of file physics.h.

4.18.4.24 DataIndex Physics::Z [protected]

The pseudo-residual in PCCG.

Definition at line 174 of file physics.h.

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

- `physics.h`

4.19 Quad4 Struct Reference

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

4-node quadrilateral element

This structure is used to hold all the book-keeping data for the element.

- int **mat**
Material (p. 48) *id*.
- int **ix** [NNPE_QUAD4]
Node-to-element connectivity.
- int **faces** [NUM_QUAD4_FACES][NNPE_QUAD4_FACES]

4.19.1 Member Data Documentation

4.19.1.1 int Quad4::faces[NUM_QUAD4_FACES][NNPE_QUAD4_FACES] [static]

Definition at line 71 of file mesh.h.

4.19.1.2 int Quad4::ix[NNPE_QUAD4]

Node-to-element connectivity.

Definition at line 70 of file mesh.h.

4.19.1.3 int Quad4::mat

Material (p. 48) *id*.

Definition at line 69 of file mesh.h.

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

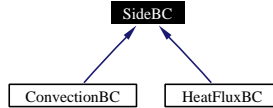
- **mesh.h**

4.20 SideBC Class Reference

This is the base class for all sideset-based boundary conditions.

```
#include <bcs.h>
```

Inheritance diagram for SideBC:



Public Methods

Constructor/Destructor

- **SideBC** ()
- virtual **~SideBC** ()

SideBC access functions

The base **SideBC** (p.92) class is setup to use a sideset id, a time (or parameter based) load curve identified by a load curve id and with an associated amplitude. The amplitude is generic and is used as an amplitude multiplier on all load curves constant or otherwise.

In FE2D, only 'constant' load curves are currently implemented.

- void **setID** (int id)
Set the sideset id.
- void **setLcurveID** (int id)
Set the load curve id.
- void **setAmp** (**Real** val)
Set the load curve amplitude.
- void **setNumber** (int num)
Set the internal id.
- int **getID** ()
Get the sideset id.
- int **getSet** ()
Get the internal set id.
- int **getLcurveID** ()
Get the load curve id.
- **Real** **getAmp** ()
Get the load curve amplitude.

Protected Attributes

- **int ssid**
internal sideset id.
- **int set**
internal sideset number.
- **int lcid**
load curve id.
- **Real amp**
amplitude.

Private Methods

SideBC data

*This data is used in the inherited physics-specific boundary conditions. So, the amplitude is interpreted in each of the physics-specific cases accordingly. For example, in the **HeatFluxBC** (p. 45), the amplitude (*amp*) represents the heat flux amplitude or *q_n*. In the convection *bc*, it represents the convective heat transfer coefficient.*

- **SideBC** (const SideBC &)
- **SideBC & operator=** (const SideBC &)

4.20.1 Detailed Description

This is the base class for all sideset-based boundary conditions.

Definition at line 21 of file `bcs.h`.

4.20.2 Constructor & Destructor Documentation

4.20.2.1 SideBC::SideBC () [inline]

Definition at line 26 of file `bcs.h`.

```
26 {}
```

4.20.2.2 virtual SideBC::~~SideBC () [inline, virtual]

Definition at line 27 of file `bcs.h`.

```
27 {}
```

4.20.2.3 `SideBC::SideBC (const SideBC &) [private]`

4.20.3 Member Function Documentation

4.20.3.1 `Real SideBC::getAmp () [inline]`

Get the load curve amplitude.

Definition at line 48 of file bcs.h.

References amp, and Real.

4.20.3.2 `int SideBC::getID () [inline]`

Get the sideset id.

Definition at line 45 of file bcs.h.

References ssid.

4.20.3.3 `int SideBC::getLcurveID () [inline]`

Get the load curve id.

Definition at line 47 of file bcs.h.

References lcid.

4.20.3.4 `int SideBC::getSet () [inline]`

Get the internal set id.

Definition at line 46 of file bcs.h.

References set.

4.20.3.5 `SideBC& SideBC::operator= (const SideBC &) [private]`

4.20.3.6 `void SideBC::setAmp (Real val) [inline]`

Set the load curve amplitude.

Definition at line 42 of file bcs.h.

References amp, and Real.

4.20.3.7 `void SideBC::setID (int id) [inline]`

Set the sideset id.

Definition at line 40 of file bcs.h.

References ssid.

4.20.3.8 void SideBC::setLcurveID (int *id*) [inline]

Set the load curve id.

Definition at line 41 of file bcs.h.

References lcid.

4.20.3.9 void SideBC::setNumber (int *num*) [inline]

Set the internal id.

Definition at line 43 of file bcs.h.

References set.

4.20.4 Member Data Documentation

4.20.4.1 Real SideBC::amp [protected]

amplitude.

Definition at line 63 of file bcs.h.

Referenced by getAmp, and setAmp.

4.20.4.2 int SideBC::lcid [protected]

load curve id.

Definition at line 62 of file bcs.h.

Referenced by getLcurveID, and setLcurveID.

4.20.4.3 int SideBC::set [protected]

internal sideset number.

Definition at line 61 of file bcs.h.

Referenced by getSet, and setNumber.

4.20.4.4 int SideBC::ssid [protected]

internal sideset id.

Definition at line 60 of file bcs.h.

Referenced by getID, and setID.

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

- **bcs.h**

4.21 Sideset Struct Reference

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

Public Attributes

Sideset data structure

The **Sideset** (p. 96) data structure is used to hold input sideset data for the mesh.

- **int id**
Sodeset id.
- **int Nel**
Number of elements in sideset.
- **int Nnp**
Number of nodes in sideset.
- **int Ndf**
Number of distribution factors.
- **DataIndex EL_LIST**
List of elements in the sideset.
- **DataIndex SIDE_LIST**
List of sides for each element in the sideset.

4.21.1 Member Data Documentation

4.21.1.1 DataIndex Sideset::EL_LIST

List of elements in the sideset.

Definition at line 124 of file mesh.h.

4.21.1.2 int Sideset::id

Sodeset id.

Definition at line 120 of file mesh.h.

4.21.1.3 int Sideset::Ndf

Number of distribution factors.

Definition at line 123 of file mesh.h.

4.21.1.4 int Sideset::Nel

Number of elements in sideset.

Definition at line 121 of file mesh.h.

4.21.1.5 int Sideset::Nnp

Number of nodes in sideset.

Definition at line 122 of file mesh.h.

4.21.1.6 DataIndex Sideset::SIDE_LIST

List of sides for each element in the sideset.

Definition at line 125 of file mesh.h.

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

- **mesh.h**

Chapter 5

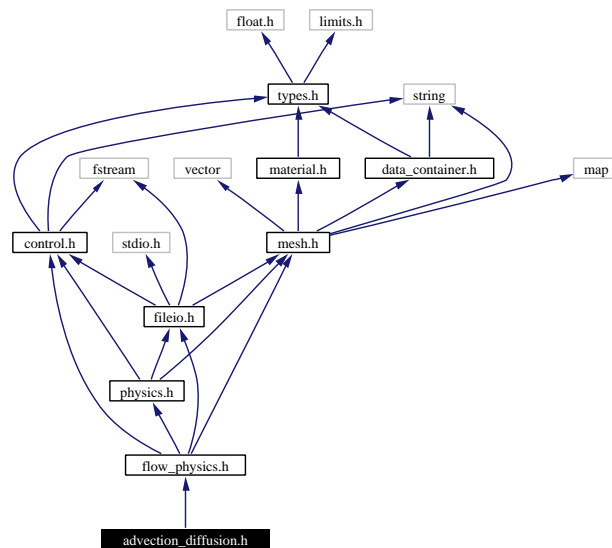
FE2D File Documentation

5.1 advection_diffusion.h File Reference

The **AdvectionDiffusion** (p. 7) class is used only for scalar advection-diffusion.

```
#include "flow_physics.h"
```

Include dependency graph for advection_diffusion.h:



Compounds

- class **AdvectionDiffusion**

Advection-Diffusion class.

5.1.1 Detailed Description

The **AdvectionDiffusion** (p. 7) class is used only for scalar advection-diffusion.

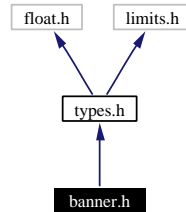
Definition in file **advection_diffusion.h**.

5.2 banner.h File Reference

The **Banner** (p. 12) class is used to print out the code name in 'banner' format.

```
#include "types.h"
```

Include dependency graph for banner.h:



Compounds

- class **Banner**

Defines

- `#define SMALL_FONT 1`
- `#define MEDIUM_FONT 2`
- `#define LARGE_FONT 3`
- `#define LEFT_JUSTIFIED 1`
- `#define CENTER_JUSTIFIED 2`
- `#define RIGHT_JUSTIFIED 3`

5.2.1 Detailed Description

The **Banner** (p. 12) class is used to print out the code name in 'banner' format.

Definition in file **banner.h**.

5.2.2 Define Documentation

5.2.2.1 `#define CENTER_JUSTIFIED 2`

Definition at line 18 of file banner.h.

5.2.2.2 `#define LARGE_FONT 3`

Definition at line 15 of file banner.h.

5.2.2.3 `#define LEFT_JUSTIFIED 1`

Definition at line 17 of file banner.h.

5.2.2.4 #define MEDIUM_FONT 2

Definition at line 14 of file banner.h.

5.2.2.5 #define RIGHT_JUSTIFIED 3

Definition at line 19 of file banner.h.

5.2.2.6 #define SMALL_FONT 1

Definition at line 13 of file banner.h.

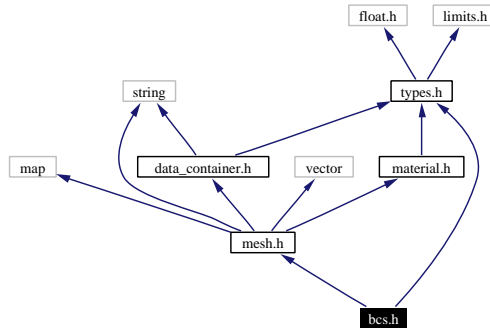
5.3 bcs.h File Reference

The **SideBC** (p. 92) class is used to derive physics-specific sideset based BC's.

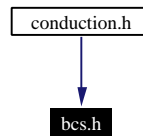
```
#include "types.h"
```

```
#include "mesh.h"
```

Include dependency graph for bcs.h:



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



Compounds

- class **ConvectionBC**
Convection boundary condition based on sidesets.
- class **HeatFluxBC**
Heat flux boundary condition for sidesets.
- class **SideBC**
This is the base class for all sideset-based boundary conditions.

5.3.1 Detailed Description

The **SideBC** (p. 92) class is used to derive physics-specific sideset based BC's.

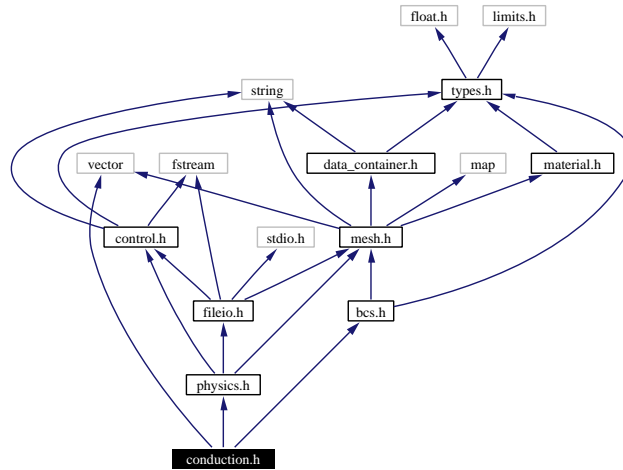
Definition in file **bcs.h**.

5.4 conduction.h File Reference

The **Conduction** (p. 14) class provides the thermal conduction specific solver.

```
#include <vector>
#include "physics.h"
#include "bcs.h"
```

Include dependency graph for conduction.h:



Compounds

- class **Conduction**
Conduction heat transfer class.

5.4.1 Detailed Description

The **Conduction** (p. 14) class provides the thermal conduction specific solver.

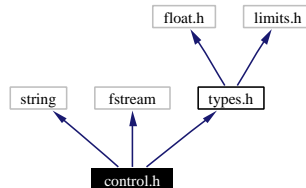
Definition in file **conduction.h**.

5.5 control.h File Reference

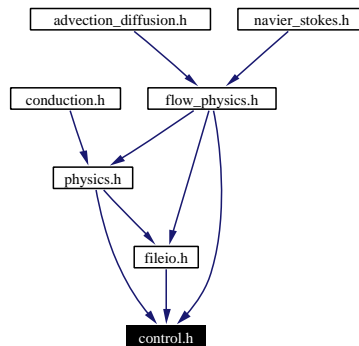
The **Control** (p. 20) class holds all the relevant analysis options.

```
#include <string>
#include <fstream>
#include "types.h"
```

Include dependency graph for control.h:



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



Compounds

- class **Control**

The Control class holds all the relevant analysis options and parameters.

5.5.1 Detailed Description

The **Control** (p. 20) class holds all the relevant analysis options.

Definition in file **control.h**.

5.5.2 Enumeration Type Documentation

5.5.2.1 enum AnalysisOpt

Enumeration values:

SOLTYP physics selection.
NSTEPS number of timesteps.
IDTCHK interval to check stable time step size.
PRTLEV 0 - parm's only, 1 - results, 2 - verbose.
PRTI print interval.
PLTYPE plot output type (GMV for now).
PLTI plot interval.
TTYI Interval for screen reports of min./max.
DUMP write restart file 1=yes, 0=no.
DORESTART perform restart using dump file.
MASS mass matrix - lumped, consistent, high-order.
BTD BTD on or off.
UPWIND 0 - no spatial upwind, 1 - spatial upwind.
FCT 0 - no flux correction, 1 - flux correction.
NODESOL nodal equation solver type.
NDITMAX nodal equation iteration limit.
NDITCHK nodal equation interval to check convergence.
NDWRT flag for writing cg information.
PPESOL PPE solver type.
PITMAX PPE equation iteration limit.
PITCHK PPE equation interval to check convergence.
PWRT PPE flag for writing cg information.
NUM_ECHO_OPT Parameters after this will not be echod.
PLNUM # of state plot file.
LUFLAG flag for udu factorization.
NUM_ICNTL Number of integer control variables.

Definition at line 29 of file control.h.

Referenced by Control::getOption, and Control::setOption.

```

29          {SOLTYP=0,    //!< physics selection
30            NSTEPS,    //!< number of timesteps
31            IDTCHK,    //!< interval to check stable time step size
32            PRTLEV,    //!< 0 - parm's only, 1 - results, 2 - verbose
33            PRTI,      //!< print interval
34            PLTYPE,    //!< plot output type (GMV for now)
35            PLTI,      //!< plot interval
36            TTYI,      //!< Interval for screen reports of min./max.
37            DUMP,      //!< write restart file 1=yes, 0=no
38            DORESTART, //!< perform restart using dump file
39            MASS,      //!< mass matrix - lumped, consistent, high-order
40            BTD,       //!< BTD on or off

```

```

41         UPWIND,      //!< 0 - no spatial upwind, 1 - spatial upwind
42         FCT,        //!< 0 - no flux correction, 1 - flux correction
43         NODESOL,    //!< nodal equation solver type
44         NDITMAX,    //!< nodal equation iteration limit
45         NDITCHK,    //!< nodal equation interval to check convergence
46         NDWRT,      //!< flag for writing cg information
47         PPESOL,     //!< PPE solver type
48         PITMAX,     //!< PPE equation iteration limit
49         PITCHK,     //!< PPE equation interval to check convergence
50         PWRT,       //!< PPE flag for writing cg information
51         NUM_ECHO_OPT, //!< Parameters after this will not be echod
52
53         PLNUM,      //!< # of state plot file
54         LUFLAG,     //!< flag for udu factorization
55         NUM_ICNTL   //!< Number of integer control variables
56 };

```

5.5.2.2 enum AnalysisPrm

Enumeration values:

DELTAT time step size.

TF termination time.

NDEPSCG C-G convergence limit.

DIVEPS div(u) tolerance.

DTEPS accuracy measure for sub-cyling.

DTSCALE scale factor for time step.

PEPS PPE CG convergence limit.

THETAK time weight for diffusion.

THETAB time weight for BTD.

THETAA time weight for advection.

THETAF time weight for BC's.

NUM_ECHO_PRM Parameters after this will not be echod.

TS total simulated time.

WTK (1-thetak).

WTB (1-thetab).

WTA (1-thetaa).

WTF (1-thetaf).

NUM_RCNTL Number of real control parameters.

Definition at line 58 of file control.h.

Referenced by Control::getParam, and Control::setParam.

```

58         {DELTAT,      //!< time step size
59         TF,          //!< termination time
60         NDEPSCG,    //!< C-G convergence limit
61         DIVEPS,     //!< div(u) tolerance
62         DTEPS,      //!< accuracy measure for sub-cyling
63         DTSCALE,    //!< scale factor for time step
64         PEPS,       //!< PPE CG convergence limit
65         THETAK,     //!< time weight for diffusion

```

```

66          THETAB,      //!< time weight for BTD
67          THETAA,      //!< time weight for advection
68          THETAF,      //!< time weight for BC's
69          NUM_ECHO_PRM, //!< Parameters after this will not be echod
70
71          TS,          //!< total simulated time
72          WTK,          //!< (1-thetak)
73          WTB,          //!< (1-thetab)
74          WTA,          //!< (1-thetaa)
75          WTF,          //!< (1-thetaf)
76          NUM_RCNTL    //!< Number of real control parameters
77 };

```

5.5.2.3 enum MassType

Enumeration values:

LUMPED

CONSISTENT

HIGH_ORDER

Definition at line 18 of file control.h.

```
18 {LUMPED=0, CONSISTENT, HIGH_ORDER};
```

5.5.2.4 enum NodeSolver

Enumeration values:

DSCG

UNKNOWN_SOLVER

Definition at line 24 of file control.h.

```
24 {DSCG=0, UNKNOWN_SOLVER};
```

5.5.2.5 enum PhysicsType

Enumeration values:

CONDUCTION

SI_ADV_DIFF

P2_NAV_STOKES

Definition at line 16 of file control.h.

```
16 {CONDUCTION=0, SI_ADV_DIFF, P2_NAV_STOKES};
```

5.5.2.6 enum PlotType

Enumeration values:

GMV_ASCII

GMV_BINARY

UNKNOWN_PLOT

Definition at line 22 of file control.h.

```
22 {GMV_ASCII=0, GMV_BINARY, UNKNOWN_PLOT};
```

5.5.2.7 enum PPEsolver

Enumeration values:

PVS

PDSCG

PSSORCG

Definition at line 26 of file control.h.

```
26 {PVS=0, PDSCG, PSSORCG};
```

5.5.2.8 enum PrintType

Enumeration values:

PARMS_ONLY

RESULTS

VERBOSE

Definition at line 20 of file control.h.

```
20 {PARMS_ONLY=0, RESULTS, VERBOSE};
```

5.6 cvsdate.h File Reference

The cvsdate is used to keep track of the code version according to CVS.

Variables

- string `cvsdate` = "Sat Jul 20 17:58:29 MDT 2002"

5.6.1 Detailed Description

The cvsdate is used to keep track of the code version according to CVS.

Definition in file `cvsdate.h`.

5.6.2 Variable Documentation

5.6.2.1 string `cvsdate` = "Sat Jul 20 17:58:29 MDT 2002"

Definition at line 5 of file `cvsdate.h`.

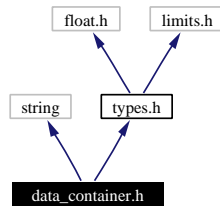
5.7 data_container.h File Reference

The **DataContainer** (p. 27) class is the base data-storage class used for all variables.

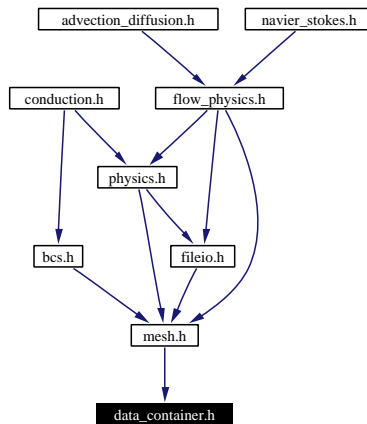
```
#include <string>
```

```
#include "types.h"
```

Include dependency graph for data_container.h:



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



Compounds

- class **DataContainer**

DataContainer is the basic data storage/managements object in FE2D.

Defines

- #define **MAX_MEMORY_ENTRIES** 1024

5.7.1 Detailed Description

The **DataContainer** (p. 27) class is the base data-storage class used for all variables.

Definition in file **data_container.h**.

5.7.2 Define Documentation

5.7.2.1 `#define MAX_MEMORY_ENTRIES 1024`

Definition at line 13 of file data_container.h.

5.8 epsilon.h File Reference

Machine limits for the C-G solver (dscgnd.F).

5.8.1 Detailed Description

Machine limits for the C-G solver (dscgnd.F).

C

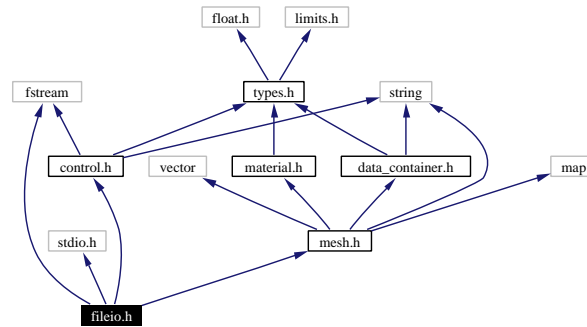
Definition in file **epsilon.h**.

5.9 fileio.h File Reference

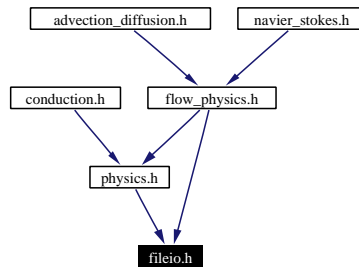
The `fileIO` (p. 32) class manages all file-based I/O, e.g., opens files, etc.

```
#include <fstream>
#include <stdio.h>
#include "control.h"
#include "mesh.h"
```

Include dependency graph for `fileio.h`:



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



Compounds

- class `fileIO`

The `fileIO` is used to handle all I/O functions – or at least most.

Enumerations

- enum `FileType` { `EXE` = 0, `MESH`, `CONTROL`, `OUTPUT`, `PLOT`, `HISTORY`, `GLOBAL`, `RESTARTW`, `RESTARTR`, `NUM_FILENAMES` }

5.9.1 Detailed Description

The `fileIO` (p. 32) class manages all file-based I/O, e.g., opens files, etc.

Definition in file `fileio.h`.

5.9.2 Enumeration Type Documentation

5.9.2.1 enum FileType

Enumeration values:

- EXE** Executable name.
- MESH** Input mesh file.
- CONTROL** Input control file.
- OUTPUT** File for ASCII data echo.
- PLOT** File for plot data.
- HISTORY** File for nodal time-history data.
- GLOBAL** File for global time-history data.
- RESTARTW** File to write restarts to.
- RESTARTR** File to read restarts from.
- NUM_FILENAMES** Number of filenames.

Definition at line 25 of file `fileio.h`.

Referenced by `fileIO::setFname`.

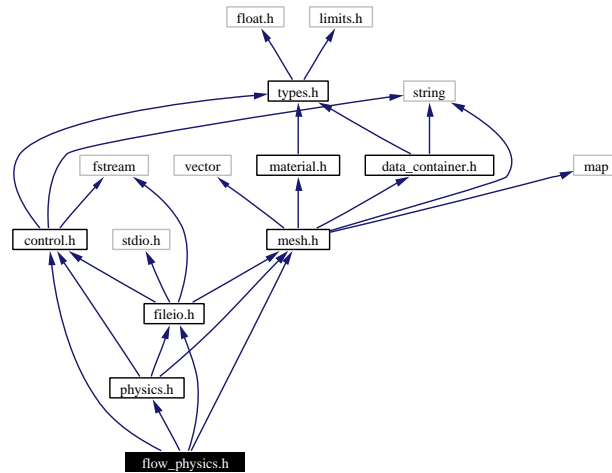
```
25         {EXE=0,          //!< Executable name
26           MESH,          //!< Input mesh file
27           CONTROL,       //!< Input control file
28           OUTPUT,        //!< File for ASCII data echo
29           PLOT,          //!< File for plot data
30           HISTORY,       //!< File for nodal time-history data
31           GLOBAL,        //!< File for global time-history data
32           RESTARTW,      //!< File to write restarts to
33           RESTARTR,      //!< File to read restarts from
34           NUM_FILENAMES  //!< Number of filenames
35 };
```

5.10 flow_physics.h File Reference

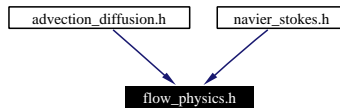
The **FlowPhysics** (p. 40) class is used to construct the physics specific solvers.

```
#include "control.h"
#include "fileio.h"
#include "mesh.h"
#include "physics.h"
```

Include dependency graph for flow_physics.h:



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



Compounds

- class **FlowPhysics**

5.10.1 Detailed Description

The **FlowPhysics** (p. 40) class is used to construct the physics specific solvers.

Definition in file **flow_physics.h**.

5.11 letters.h File Reference

The ASCII letters used by the **Banner** (p. 12) class.

Variables

- int **letters_width** [3] = {1, 8, 10}
- int **letters_length** [3] = {1, 7, 9}
- char **letters** [97]
- char **small_font** [96][1][1]
- char **medium_font** [96][7][9]

5.11.1 Detailed Description

The ASCII letters used by the **Banner** (p. 12) class.

Definition in file **letters.h**.

5.11.2 Variable Documentation

5.11.2.1 char letters[97]

Initial value:

```
{'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j',
    'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't',
    'u', 'v', 'w', 'x', 'y', 'z', 'A', 'B', 'C', 'D',
    'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N',
    'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X',
    'Y', 'Z', '1', '2', '3', '4', '5', '6', '7', '8',
    '9', '0', '!', '@', '#', '$', '%', '^', '&', '*',
    '(', ')', '-', '_', '+', '=', '\\', '|', '[', '{',
    ']', '}', ':', ';', '\'', '"', '<', '~', ',', '.',
    '<', '>', '/', '?', ' ', '\n', '\0'}
```

Definition at line 11 of file **letters.h**.

5.11.2.2 int letters_length[3] = {1, 7, 9}

Definition at line 9 of file **letters.h**.

5.11.2.3 int letters_width[3] = {1, 8, 10}

Definition at line 8 of file **letters.h**.

5.11.2.4 char medium_font[96][7][9]

Definition at line 33 of file **letters.h**.

5.11.2.5 char small_font[96][1][1]

Initial value:

```
{'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j',
  'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't',
  'u', 'v', 'w', 'x', 'y', 'z', 'A', 'B', 'C', 'D',
  'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N',
  'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X',
  'Y', 'Z', '1', '2', '3', '4', '5', '6', '7', '8',
  '9', '0', '!', '@', '#', '$', '%', '^', '&', '*',
  '(', ')', '-', '_', '+', '=', '\\', '|', '[', '{',
  '}', ']', ';', ':', '\'', '"', ',', '~', ' ', '.',
  '<', '>', '/', '?', ' ', '\n'}
```

Definition at line 22 of file letters.h.

5.12 list.h File Reference

Provides linked-list functions used for computing the PPE connectivity.

Defines

- #define **NEXT**(elem) ((elem) = (elem) → next)
- #define **PREV**(elem) ((elem) = (elem) → prev)
- #define **INIT_PTRS**(elem) { (elem) → next = NULL; (elem) → prev = NULL; }
- #define **INSERT**(elem, list)
- #define **INSERT_BEFORE**(new_elem, elem)
- #define **INSERT_AFTER**(new_elem, elem)
- #define **APPEND**(elem, list)
- #define **UNLINK**(elem, list)
- #define **DELETE**(elem, list)
- #define **DELETE_LIST**(list)

5.12.1 Detailed Description

Provides linked-list functions used for computing the PPE connectivity.

Definition in file `list.h`.

5.12.2 Define Documentation

5.12.2.1 #define APPEND(elem, list)

Value:

```
{ if ( (list) == NULL ) (list) = (elem); \
    else { (elem)->prev = (list); \
          while( (elem)->prev->next != NULL ) \
            (elem)->prev = (elem)->prev->next; \
          (elem)->prev->next = (elem); } }
```

Definition at line 94 of file `list.h`.

5.12.2.2 #define DELETE(elem, list)

Value:

```
{ if ( (list) == (elem) ) \
    (list) = (elem)->next; \
    else \
    (elem)->prev->next = (elem)->next; \
    if ( (elem)->next != NULL ) \
    (elem)->next->prev = (elem)->prev; \
    (elem)->next = NULL; (elem)->prev = NULL; }
```

Definition at line 131 of file `list.h`.

5.12.2.3 #define DELETE_LIST(list)**Value:**

```
{ if ( (list) != NULL ) { \
    while ( (list)->next != NULL ) \
        { (list) = (list)->next; free( (list)->prev ); } \
    free( (list) ); (list) = NULL; }
```

Definition at line 145 of file list.h.

5.12.2.4 #define INIT_PTRS(elem) { (elem) → next = NULL; (elem) → prev = NULL; }

Definition at line 53 of file list.h.

5.12.2.5 #define INSERT(elem, list)**Value:**

```
{ if ( (list) != NULL ) \
    { (elem)->next = (list); \
      (list)->prev = (elem); } \
  (list) = (elem); }
```

Definition at line 61 of file list.h.

5.12.2.6 #define INSERT_AFTER(new_elem, elem)**Value:**

```
{ if ( (elem)->next != NULL ) { \
    (elem)->next->prev = (new_elem); \
    (new_elem)->next = (elem)->next; } \
  (new_elem)->prev = (elem); (elem)->next = (new_elem); }
```

Definition at line 83 of file list.h.

5.12.2.7 #define INSERT_BEFORE(new_elem, elem)**Value:**

```
{ if ( (elem)->prev != NULL ) { \
    (elem)->prev->next = (new_elem); \
    (new_elem)->prev = (elem)->prev; } \
  (new_elem)->next = (elem); (elem)->prev = (new_elem); }
```

Definition at line 72 of file list.h.

5.12.2.8 #define NEXT(elem) ((elem) → next)

Definition at line 37 of file list.h.

5.12.2.9 #define PREV(elem) ((elem) = (elem) → prev)

Definition at line 45 of file list.h.

5.12.2.10 #define UNLINK(elem, list)

Value:

```
{ if ( (list) == (elem) ) \  
    (list) = (elem)->next; \  
  else \  
    (elem)->prev->next = (elem)->next; \  
  if ( (elem)->next != NULL ) \  
    (elem)->next->prev = (elem)->prev; \  
  (elem)->next = NULL; (elem)->prev = NULL; }
```

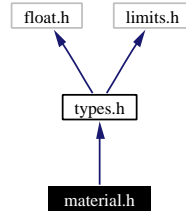
Definition at line 115 of file list.h.

5.13 material.h File Reference

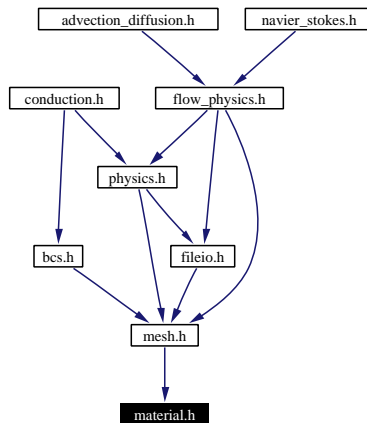
The **Material** (p. 48) class provides a simplified way to store/access material properties.

```
#include "types.h"
```

Include dependency graph for material.h:



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



Compounds

- class **Material**

5.13.1 Detailed Description

The **Material** (p. 48) class provides a simplified way to store/access material properties.

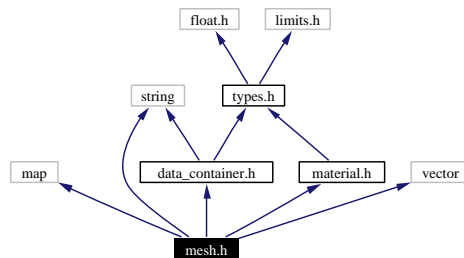
Definition in file **material.h**.

5.14 mesh.h File Reference

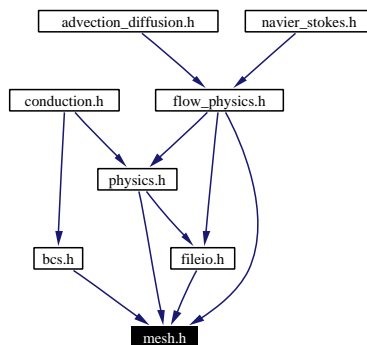
The **Mesh** (p. 52) object, **Beam2** (p. 13), **Quad4** (p. 91) and **Hex8** (p. 47) elements, **Nodeset** (p. 80), **Sideset** (p. 96) are defined here.

```
#include <map>
#include <string>
#include <vector>
#include "data_container.h"
#include "material.h"
```

Include dependency graph for mesh.h:



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



Compounds

- struct **Beam2**
- struct **ElSet**
- struct **Hex8**
- class **Mesh**

Mesh object – holds basic mesh information, sidesets, nodesets, and data.

- struct **NodeBC**
- struct **Nodeset**
- struct **Quad4**
- struct **Sideset**

Defines

- `#define NNPE_HEX8 8`
No. of nodes per element.
- `#define NUM_HEX8_FACES 6`
No. of faces per element.
- `#define NNPE_HEX8_FACES 4`
No. of nodes per hex8 face.
- `#define NEPE_HEX8 8`
No. of edges per hex8 element.
- `#define NNPE_QUAD4 4`
No. of nodes per element.
- `#define NEPE_QUAD4 4`
No. of edges per element.
- `#define NUM_QUAD4_FACES 4`
No. of faces per element.
- `#define NNPE_QUAD4_FACES 2`
No. of nodes per quad4 face.
- `#define NNPE_BEAM2 2`
No. of nodes per 2-node beam element.

Enumerations

- `enum Eltype { TRI3, QUAD4, QUAD9, HEX8, HEX20, TET4, TET10, NUM_ELEMENT_TYPES }`
- `enum Meshtype { ASCILMESH, EXODUS_MESH, NUM_MESH_TYPES }`
- `enum DOFmap { XVELOCITY = 0, YVELOCITY, ZVELOCITY, TEMP, MAXDOF }`

5.14.1 Detailed Description

The **Mesh** (p. 52) object, **Beam2** (p. 13), **Quad4** (p. 91) and **Hex8** (p. 47) elements, **Nodeset** (p. 80), **Sideset** (p. 96) are defined here.

Definition in file `mesh.h`.

5.14.2 Define Documentation

5.14.2.1 `#define NEPE_HEX8 8`

No. of edges per hex8 element.

Definition at line 44 of file `mesh.h`.

5.14.2.2 #define NEPE_QUAD4 4

No. of edges per element.

Definition at line 60 of file mesh.h.

5.14.2.3 #define NNPE_BEAM2 2

No. of nodes per 2-node beam element.

Definition at line 77 of file mesh.h.

5.14.2.4 #define NNPE_HEX8 8

No. of nodes per element.

Definition at line 41 of file mesh.h.

5.14.2.5 #define NNPE_HEX8_FACES 4

No. of nodes per hex8 face.

Definition at line 43 of file mesh.h.

5.14.2.6 #define NNPE_QUAD4 4

No. of nodes per element.

Definition at line 59 of file mesh.h.

5.14.2.7 #define NNPE_QUAD4_FACES 2

No. of nodes per quad4 face.

Definition at line 62 of file mesh.h.

5.14.2.8 #define NUM_HEX8_FACES 6

No. of faces per element.

Definition at line 42 of file mesh.h.

5.14.2.9 #define NUM_QUAD4_FACES 4

No. of faces per element.

Definition at line 61 of file mesh.h.

5.14.3 Enumeration Type Documentation

5.14.3.1 enum DOFmap

Enumeration values:

XVELOCITY
YVELOCITY
ZVELOCITY
TEMP
MAXDOF

Definition at line 37 of file mesh.h.

```
37 {XVELOCITY=0, YVELOCITY, ZVELOCITY, TEMP, MAXDOF};
```

5.14.3.2 enum Eltype

Enumeration values:

TRI3 3-node triangle.
QUAD4 4-node quadrilateral.
QUAD9 9-node quadrilateral.
HEX8 8-node hex.
HEX20 20-node hex.
TET4 4-node tetrahedra.
TET10 10-node tetrahedra.
NUM_ELEMENT_TYPES # of element types.

Definition at line 24 of file mesh.h.

```
24      {TRI3,          //!< 3-node triangle
25      QUAD4,         //!< 4-node quadrilateral
26      QUAD9,         //!< 9-node quadrilateral
27      HEX8,          //!< 8-node hex
28      HEX20,         //!< 20-node hex
29      TET4,          //!< 4-node tetrahedra
30      TET10,         //!< 10-node tetrahedra
31      NUM_ELEMENT_TYPES //!< # of element types
32 };
```

5.14.3.3 enum Meshtype

Enumeration values:

ASCII_MESH
EXODUS_MESH
NUM_MESH_TYPES

Definition at line 34 of file mesh.h.

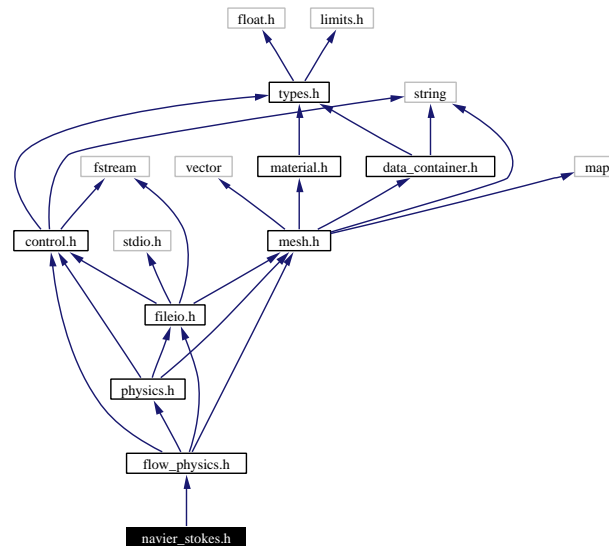
```
34 {ASCII_MESH, EXODUS_MESH, NUM_MESH_TYPES};
```

5.15 navier_stokes.h File Reference

The `NavierStokes` (p.64) class is used for solving the time-dependent incompressible Navier-Stokes equations.

```
#include "flow_physics.h"
```

Include dependency graph for `navier_stokes.h`:



Compounds

- class `NavierStokes`

Functions

- void `FORTRAN` (elmkj)(int *iecn
- void `FORTRAN` (rzpmkj)(int *mkj
- void `FORTRAN` (gpskca)(int &N
- void `FORTRAN` (pvsrow)(int *ideg

Variables

- void int * `mkj`
- void int int * `icnt`
- void int int int * `ielst`
- void int int int int & `nnp`
- void int int int int int & `nel`
- void int int int int int int & `npe`
- void int int int int int int int & `mxe`
- void int int int int int int int int & `ndw`
- void int int int int int int int int int & `iebw`
- void int int int int int int int int int int & `luo`

- void int * **icon**
- void int int int int & **na**
- void int * **DEGREE**
- void int int * **RSTART**
- void int int int * **CONNEC**
- void int int int int & **OPT**
- void int int int int int & **WRKLEN**
- void int int int int int int * **PERMUT**
- void int int int int int int int * **WORK**
- void int int int int int int int int & **BANDWD**
- void int int int int int int int int int & **PROFIL**
- void int int int int int int int int int int & **ERROR**
- void int int int int int int int int int int int & **SPACE**
- void int * **irst**
- void int int * **ierv**
- void int int int int * **irowl**
- void int int int int int * **icolh**
- void int int int int int int * **maxa**
- void int int int int int int int & **neq**
- void int int int int int int int int int & **lev**
- void **Real** * **B**
- void **Real** int * **MAXA**
- void **Real** int int * **IROWL**
- void **Real** int int int * **ICOLH**
- void **Real** int int int int int & **NEQ**
- void **Real** int int int int int int & **IFLAG**

5.15.1 Detailed Description

The **NavierStokes** (p.64) class is used for solving the time-dependent incompressible Navier-Stokes equations.

Definition in file **navier_stokes.h**.

5.15.2 Function Documentation

5.15.2.1 void FORTRAN (pvsrow)

5.15.2.2 void FORTRAN (gpskca)

5.15.2.3 void FORTRAN (rzpmkj)

5.15.2.4 void FORTRAN (elmkj)

5.15.3 Variable Documentation

5.15.3.1 void **Real*** **B**

Definition at line 31 of file **navier_stokes.h**.

5.15.3.2 void int int int int int int int int& BANDWD

Definition at line 22 of file navier_stokes.h.

5.15.3.3 void int int int* CONNEC

Definition at line 22 of file navier_stokes.h.

5.15.3.4 void int* DEGREE

Definition at line 22 of file navier_stokes.h.

5.15.3.5 void int int int int int int int int int& ERROR

Definition at line 22 of file navier_stokes.h.

5.15.3.6 void int int* icnt

Definition at line 14 of file navier_stokes.h.

5.15.3.7 void Real int int int* ICOLH

Definition at line 31 of file navier_stokes.h.

5.15.3.8 void int int int int int* icolh

Definition at line 27 of file navier_stokes.h.

5.15.3.9 void int* icon

Definition at line 19 of file navier_stokes.h.

5.15.3.10 void int int int int int int int int int& iebw

Definition at line 14 of file navier_stokes.h.

5.15.3.11 void int int int* ielst

Definition at line 14 of file navier_stokes.h.

5.15.3.12 void int int* ierv

Definition at line 27 of file navier_stokes.h.

5.15.3.13 void Real int int int int int int& IFLAG

Definition at line 31 of file navier_stokes.h.

5.15.3.14 void Real int int* IROWL

Definition at line 31 of file navier_stokes.h.

5.15.3.15 void int int int int* irowl

Definition at line 27 of file navier_stokes.h.

5.15.3.16 void int* irst

Definition at line 27 of file navier_stokes.h.

5.15.3.17 void int int int int int int int int int& lev

Definition at line 27 of file navier_stokes.h.

5.15.3.18 void int int int int int int int int int int& luo

Definition at line 14 of file navier_stokes.h.

5.15.3.19 void Real int* MAXA

Definition at line 31 of file navier_stokes.h.

5.15.3.20 void int int int int int int* maxa

Definition at line 27 of file navier_stokes.h.

5.15.3.21 void int int int * mkj

Definition at line 27 of file navier_stokes.h.

5.15.3.22 void int int int int int int int& mxe

Definition at line 14 of file navier_stokes.h.

5.15.3.23 void Real int int int int & na

Definition at line 31 of file navier_stokes.h.

5.15.3.24 void Real Real int int int & ndw

Definition at line 24 of file physics.h.

5.15.3.25 void int int & nel

Definition at line 19 of file navier_stokes.h.

5.15.3.26 void Real int int int int int& NEQ

Definition at line 31 of file navier_stokes.h.

5.15.3.27 void Real Real int int & neq

Definition at line 24 of file physics.h.

5.15.3.28 void int int int int& nnp

Definition at line 14 of file navier_stokes.h.

5.15.3.29 void int int int int int int& npe

Definition at line 14 of file navier_stokes.h.

5.15.3.30 void int int int int& OPT

Definition at line 22 of file navier_stokes.h.

5.15.3.31 void int int int int int int* PERMUT

Definition at line 22 of file navier_stokes.h.

5.15.3.32 void int int int int int int int int int& PROFIL

Definition at line 22 of file navier_stokes.h.

5.15.3.33 void int int* RSTART

Definition at line 22 of file navier_stokes.h.

5.15.3.34 void int int int int int int int int int int& SPACE

Definition at line 22 of file navier_stokes.h.

5.15.3.35 void int int int int int int int* WORK

Definition at line 22 of file navier_stokes.h.

5.15.3.36 void int int int int int& WRKLEN

Definition at line 22 of file navier_stokes.h.

5.16 options.h File Reference

Preformatted list of analysis options for data-echo purposes.

Variables

- char * **analysis_opt** []
- char * **analysis_prm** []

5.16.1 Detailed Description

Preformatted list of analysis options for data-echo purposes.

Definition in file **options.h**.

5.16.2 Variable Documentation

5.16.2.1 char* analysis_opt[]

Initial value:

```
{
  "\tPhysics option ..... ",
  "\tNumber of time steps ..... ",
  "\tInterval to check stable time step ..... ",
  "\tPrint Level ..... ",
  "\tPrint interval ..... ",
  "\tPlot file type ..... ",
  "\tPlot interval ..... ",
  "\tScreen interval ..... ",
  "\tWrite a restart dump file ..... ",
  "\tPerform restart using the dump ..... ",
  "\tMass matrix option ..... ",
  "\tBalancing tensor diffusivity ..... ",
  "\tSpatial upwinding ..... ",
  "\tFlux-correction ..... ",
  "\tNodal equation solver ..... ",
  "\t  Iteration limit ..... ",
  "\t  Interval to check convergence ..... ",
  "\t  Write diagnostic information ..... ",
  "\tPPE equation solver ..... ",
  "\t  Iteration limit ..... ",
  "\t  Interval to check convergence ..... ",
  "\t  Write diagnostic information ....."
}
```

Definition at line 7 of file options.h.

5.16.2.2 char* analysis_prm[]

Initial value:

```
{
  "\tTime step ..... ",
  "\tTermination time ....."
}
```

```
"\tNodal convergence criteria ..... ",
"\tDivergence tolerance ..... ",
"\tSubcycling tolerance ..... ",
"\tTime step scale factor ..... ",
"\tPressure convergence criteria ..... ",
"\tViscous/Diffusion time weighting ..... ",
"\tBTD time weighting ..... ",
"\tAdvection time weighting ..... ",
"\tSource term time weighting ..... "
}
```

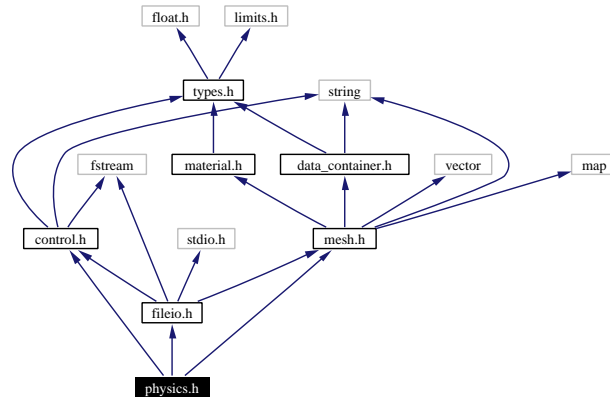
Definition at line 33 of file options.h.

5.17 physics.h File Reference

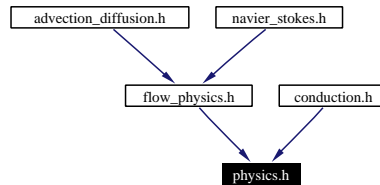
The **Physics** (p.81) class is used to develop a physics-specific solver.

```
#include "control.h"
#include "fileio.h"
#include "mesh.h"
```

Include dependency graph for physics.h:



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



Compounds

- class **Physics**

Base class used to construct the solution algorithms for a given physics.

Functions

- void **FORTTRAN** (dscgnd)(**Real** *a
- void **FORTTRAN** (smva)(**Real** *a

Variables

- void int * **ja**
- void int **Real** * **x**

- void int **Real Real * b**
- void int **Real Real Real * p**
- void int **Real Real Real Real * r**
- void int **Real Real Real Real Real * ap**
- void int **Real Real Real Real Real Real * z**
- void int **Real Real Real Real Real Real Real * rmi**
- void int **Real Real Real Real Real Real Real int & neq**
- void int **Real Real Real Real Real Real Real int int & ndw**
- void int **Real Real Real Real Real Real Real int int int & itchk**
- void int **Real Real Real Real Real Real Real int int int int & itmax**
- void int **Real Real Real Real Real Real Real int int int int Real & eps**
- void int **Real Real Real Real Real Real Real int int int int Real int & prt**
- void int **Real Real Real Real Real Real Real int int int int Real int int & lun**
- void int **Real Real Real Real Real Real Real int int int int Real int int int & itknt**
- void int **Real Real Real Real Real Real Real int int int int Real int int int Real & err**

5.17.1 Detailed Description

The **Physics** (p. 81) class is used to develop a physics-specific solver.

Definition in file **physics.h**.

5.17.2 Function Documentation

5.17.2.1 void FORTRAN (smva)

5.17.2.2 void FORTRAN (dscgnd)

5.17.3 Variable Documentation

5.17.3.1 void int **Real Real Real Real Real* ap**

Definition at line 15 of file **physics.h**.

5.17.3.2 void **Real Real * b**

Definition at line 24 of file **physics.h**.

5.17.3.3 void int **Real Real Real Real Real Real Real int int int int Real& eps**

Definition at line 15 of file **physics.h**.

5.17.3.4 void int **Real Real Real Real Real Real Real int int int int Real int int int Real& err**

Definition at line 15 of file **physics.h**.

5.17.3.5 void int **Real Real Real Real Real Real Real int int int& itchk**

Definition at line 15 of file **physics.h**.

**5.17.3.6 void int Real Real Real Real Real Real int int int int Real int int
int& itknt**

Definition at line 15 of file physics.h.

5.17.3.7 void int Real Real Real Real Real Real Real int int int int& itmax

Definition at line 15 of file physics.h.

5.17.3.8 void Real Real int * ja

Definition at line 24 of file physics.h.

**5.17.3.9 void int Real Real Real Real Real Real Real int int int int Real int int&
lun**

Definition at line 15 of file physics.h.

5.17.3.10 void Real Real int int int& ndw

Definition at line 24 of file physics.h.

5.17.3.11 void Real Real int int& neq

Definition at line 24 of file physics.h.

5.17.3.12 void int Real Real Real* p

Definition at line 15 of file physics.h.

5.17.3.13 void int Real Real Real Real Real Real Real int int int int Real int& prt

Definition at line 15 of file physics.h.

5.17.3.14 void int Real Real Real Real* r

Definition at line 15 of file physics.h.

5.17.3.15 void int Real Real Real Real Real Real Real* rmi

Definition at line 15 of file physics.h.

5.17.3.16 void Real * x

Definition at line 24 of file physics.h.

5.17.3.17 void int Real Real Real Real Real Real* z

Definition at line 15 of file physics.h.

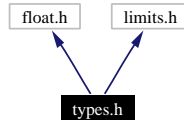
5.18 types.h File Reference

Defines all parameters and types for the FE2D code, e.g., Real is double.

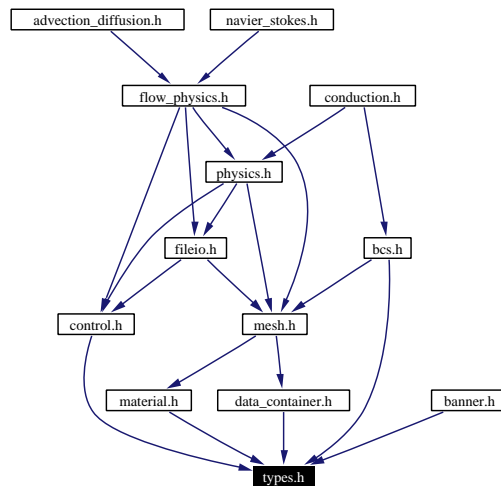
```
#include <float.h>
```

```
#include <limits.h>
```

Include dependency graph for types.h:



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



Defines

- #define **MAXCHR** 80
- #define **ONED** 1
- #define **TWOD** 2
- #define **THREED** 3
- #define **REAL_MAX** FLT_MAX
- #define **REAL_MIN** FLT_MIN
- #define **REAL_EPSILON** FLT_EPSILON

Typedefs

- typedef float **Real**
- typedef unsigned int **uint**
- typedef int **DataIndex**

5.18.1 Detailed Description

Defines all parameters and types for the FE2D code, e.g., Real is double.

Definition in file **types.h**.

5.18.2 Define Documentation

5.18.2.1 `#define MAXCHR 80`

Definition at line 25 of file types.h.

5.18.2.2 `#define ONED 1`

Definition at line 27 of file types.h.

5.18.2.3 `#define REAL_EPSILON FLT_EPSILON`

Definition at line 50 of file types.h.

5.18.2.4 `#define REAL_MAX FLT_MAX`

Definition at line 48 of file types.h.

5.18.2.5 `#define REAL_MIN FLT_MIN`

Definition at line 49 of file types.h.

5.18.2.6 `#define THREED 3`

Definition at line 29 of file types.h.

5.18.2.7 `#define TWOD 2`

Definition at line 28 of file types.h.

5.18.3 Typedef Documentation

5.18.3.1 `typedef int DataIndex`

Definition at line 41 of file types.h.

5.18.3.2 `typedef float Real`

Definition at line 37 of file types.h.

Referenced by `Material::Conductivity`, `Material::Density`, `SideBC::getAmp`, `Control::getParam`, `Control::getParams`, `ConvectionBC::getTamb`, `Mesh::getVolume`, `Mesh::getX`, `Mesh::getY`,

Mesh::getZ, SideBC::setAmp, Control::setParam, ConvectionBC::setTamb, Material::Specific-Heat, and Material::Viscosity.

5.18.3.3 typedef unsigned int uint

Definition at line 39 of file types.h.

Index

- ~AdvectionDiffusion
 - AdvectionDiffusion, 9
 - ~Banner
 - Banner, 12
 - ~Conduction
 - Conduction, 16
 - ~Control
 - Control, 21
 - ~ConvectionBC
 - ConvectionBC, 25
 - ~DataContainer
 - DataContainer, 28
 - ~FlowPhysics
 - FlowPhysics, 42
 - ~HeatFluxBC
 - HeatFluxBC, 46
 - ~Material
 - Material, 49
 - ~Mesh
 - Mesh, 56
 - ~NavierStokes
 - NavierStokes, 69
 - ~Physics
 - Physics, 84
 - ~SideBC
 - SideBC, 93
 - ~fileIO
 - fileIO, 34
 - addConvectionBC
 - Conduction, 16
 - addHeatFluxBC
 - Conduction, 16
 - addMaterial
 - Mesh, 56
 - addTHnode
 - Mesh, 56
 - advection_diffusion.h, 99
 - AdvectionDiffusion
 - AdvectionDiffusion, 9
 - AdvectionDiffusion, 7
 - ~AdvectionDiffusion, 9
 - AdvectionDiffusion, 9
 - C, 11
 - echoBCs, 9
 - formMK, 9
 - Ke, 11
 - Me, 11
 - Mle, 11
 - operator=, 10
 - Q, 11
 - readRestart, 10
 - registerData, 10
 - setICs, 10
 - setup, 10
 - solve, 10
 - TEMPERATURE, 11
 - writeRestart, 10
 - writeSolving, 10
 - writeTHdata, 10
 - writeTHhead, 10
 - amp
 - NodeBC, 79
 - SideBC, 95
 - analysis_opt
 - options.h, 132
 - analysis_prm
 - options.h, 132
 - AnalysisOpt
 - control.h, 106
 - AnalysisPrm
 - control.h, 107
 - AP
 - Physics, 87
 - ap
 - physics.h, 135
 - APPEND
 - list.h, 119
 - applyBC
 - ConvectionBC, 25
 - HeatFluxBC, 46
 - applyInverseMassBCs
 - NavierStokes, 70
 - applyNodalBC
 - Physics, 85
 - applyPenaltyLHS
 - Physics, 85
 - applyPenaltyRHS
 - Physics, 85
-

- ASCII
 - fileIO, 34
- ASCII_MESH
 - mesh.h, 126
- B
 - navier_stokes.h, 128
- b
 - physics.h, 135
- BANDWD
 - navier_stokes.h, 128
- Banner, 12
 - ~Banner, 12
 - Banner, 12
 - font_size, 12
 - format, 12
 - name, 12
 - operator<<, 12
- banner.h, 101
 - CENTER_JUSTIFIED, 101
 - LARGE_FONT, 101
 - LEFT_JUSTIFIED, 101
 - MEDIUM_FONT, 101
 - RIGHT_JUSTIFIED, 102
 - SMALL_FONT, 102
- bcs.h, 103
- Beam2, 13
 - ix, 13
 - mat, 13
- BTD
 - control.h, 106
- C
 - AdvectionDiffusion, 11
 - NavierStokes, 73
- calcEdge2D
 - FlowPhysics, 42
- calcForces
 - NavierStokes, 70
- calcKE
 - NavierStokes, 70
- calcNodalBandwidth
 - Physics, 85
- calcStreamFunc
 - FlowPhysics, 42
- calcVolume
 - Physics, 85
- calcVorticity
 - FlowPhysics, 42
- CENTER_JUSTIFIED
 - banner.h, 101
- checkMaterials
 - Mesh, 56
- checkVolume
 - Physics, 85
- closeDump
 - fileIO, 34
- closeFiles
 - fileIO, 35
- closeGlob
 - fileIO, 35
- closeRestart
 - fileIO, 35
- cntl
 - fileIO, 38
- col_id
 - Physics, 87
- COLH
 - NavierStokes, 73
- CONDUCTION
 - control.h, 108
- Conduction, 14
 - ~Conduction, 16
 - addConvectionBC, 16
 - addHeatFluxBC, 16
 - Conduction, 16
 - convection, 19
 - echoBCs, 17
 - formMK, 17
 - getConvectionBC, 17
 - getHeatFluxBC, 17
 - heatflux, 19
 - Ke, 19
 - Me, 19
 - Mle, 19
 - numConvectionBC, 17
 - numHeatFluxBC, 17
 - operator=, 18
 - Q, 19
 - readRestart, 18
 - registerData, 18
 - setICs, 18
 - setup, 18
 - solve, 18
 - TEMPERATURE, 19
 - writeRestart, 18
 - writeSolving, 18
 - writeTHdata, 18
 - writeTHhead, 18
- conduction.h, 104
- Conductivity
 - Material, 49
- CONNEC
 - navier_stokes.h, 129
- CONSISTENT
 - control.h, 108
- CONST_MASS
 - NavierStokes, 73

- CONTROL
 - fileio.h, 115
- Control, 20
 - ~Control, 21
 - Control, 21
 - getOption, 21
 - getOptions, 21
 - getParam, 21
 - getParams, 21
 - getTitle, 21
 - icntl, 22
 - operator=, 22
 - rcntl, 22
 - readRestart, 22
 - setOption, 22
 - setParam, 22
 - setTitle, 22
 - title, 23
 - writeRestart, 22
- control
 - Physics, 87
- control.h, 105
 - AnalysisOpt, 106
 - AnalysisPrm, 107
 - BTD, 106
 - CONDUCTION, 108
 - CONSISTENT, 108
 - DELTAT, 107
 - DIVEPS, 107
 - DORESTART, 106
 - DSCG, 108
 - DTEPS, 107
 - DTSCALE, 107
 - DUMP, 106
 - FCT, 106
 - GMV_ASCII, 109
 - GMV_BINARY, 109
 - HIGH_ORDER, 108
 - IDTCHK, 106
 - LUFLAG, 106
 - LUMPED, 108
 - MASS, 106
 - MassType, 108
 - NDEPSCG, 107
 - NDITCHK, 106
 - NDITMAX, 106
 - NDWRT, 106
 - NODESOL, 106
 - NodeSolver, 108
 - NSTEPS, 106
 - NUM_ECHO_OPT, 106
 - NUM_ECHO_PRM, 107
 - NUM_ICNTL, 106
 - NUM_RCNTL, 107
 - P2_NAV_STOKES, 108
 - PARAMS_ONLY, 109
 - PDSCG, 109
 - PEPS, 107
 - PhysicsType, 108
 - PITCHK, 106
 - PITMAX, 106
 - PLNUM, 106
 - PlotType, 108
 - PLTI, 106
 - PLTYPE, 106
 - PPESOL, 106
 - PPESolver, 109
 - PrintType, 109
 - PRTI, 106
 - PRTLEV, 106
 - PSSORCG, 109
 - PVS, 109
 - PWRT, 106
 - RESULTS, 109
 - SI_ADV_DIFF, 108
 - SOLTYP, 106
 - TF, 107
 - THETAA, 107
 - THETAB, 107
 - THETAF, 107
 - THETAK, 107
 - TS, 107
 - TTYI, 106
 - UNKNOWN_PLOT, 109
 - UNKNOWN_SOLVER, 108
 - UPWIND, 106
 - VERBOSE, 109
 - WTA, 107
 - WTB, 107
 - WTF, 107
 - WTK, 107
- convection
 - Conduction, 19
- ConvectionBC
 - ConvectionBC, 25
- ConvectionBC, 24
 - ~ConvectionBC, 25
 - applyBC, 25
 - ConvectionBC, 25
 - getTamb, 25
 - operator=, 25
 - setTamb, 25
 - Tamb, 26
- Cp
 - Material, 50
- cvsdate
 - cvsdate.h, 110
 - cvsdate.h, 110

- cvsdate, 110
- CX
 - NavierStokes, 73
- CY
 - NavierStokes, 73
- data_container.h, 111
 - MAX_MEMORY_ENTRIES, 112
- DataContainer
 - DataContainer, 28
- DataContainer, 27
 - ~DataContainer, 28
 - DataContainer, 28
 - dumpVariables, 28
 - freeAllVariables, 28
 - freeVariable, 28
 - getVariable, 29
 - inuse, 29
 - name, 29
 - number, 29
 - operator=, 29
 - ptr, 29
 - registerVariable, 29
 - registration, 29
 - reportMemory, 29
 - size, 29
- DataIndex
 - types.h, 139
- DEGREE
 - navier_stokes.h, 129
- DELETE
 - list.h, 119
- DELETE_LIST
 - list.h, 119
- DELTAT
 - control.h, 107
- Density
 - Material, 49
- DIVEPS
 - control.h, 107
- divFree
 - NavierStokes, 70
- divU
 - NavierStokes, 70
- DM
 - NavierStokes, 73
- DOFmap
 - mesh.h, 126
- DORESTART
 - control.h, 106
- DSCG
 - control.h, 108
- DTEPS
 - control.h, 107
- DTSCALE
 - control.h, 107
- DUMP
 - control.h, 106
- dumpf
 - fileIO, 38
- dumpVariables
 - DataContainer, 28
- echoBCs
 - AdvectionDiffusion, 9
 - Conduction, 17
 - NavierStokes, 70
 - Physics, 85
- echoControl
 - fileIO, 35
- echoElSet
 - NavierStokes, 70
- echoFnames
 - fileIO, 35
- echoHeader
 - fileIO, 35
- echoHistory
 - fileIO, 35
- echoMaterials
 - fileIO, 35
 - Mesh, 56
- echoMeshParms
 - Mesh, 56
- echoNodalBCs
 - Physics, 85
- echoNodalStats
 - Physics, 85
- echoPPEStats
 - NavierStokes, 70
- echoRestart
 - fileIO, 35
- echoTHnode
 - Mesh, 56
- EdgeToNode
 - Physics, 85
- EL_LIST
 - ElSet, 31
 - Sideset, 96
- EL_VAL
 - ElSet, 31
- Elbwf
 - NavierStokes, 73
- Elbwi
 - NavierStokes, 74
- ELEMENTS
 - Mesh, 60
- ElSet, 31
 - EL_LIST, 31

- EL_VAL, 31
- Nel_set, 31
- Eltype
 - mesh.h, 126
- eps
 - physics.h, 135
- epsilon.h, 113
- err
 - physics.h, 135
- ERROR
 - navier_stokes.h, 129
- eset
 - Mesh, 60
- EXE
 - fileio.h, 115
- EXODUS
 - fileIO, 34
- EXODUS_MESH
 - mesh.h, 126
- faces
 - Hex8, 47
 - Quad4, 91
- FCT
 - control.h, 106
- FieldLimits
 - Physics, 85
- FileFormat
 - fileIO, 34
- fileIO
 - ASCII, 34
 - EXODUS, 34
 - fileIO, 34
 - NUM_FILEFORMATS, 34
- fileIO, 32
 - ~fileIO, 34
 - closeDump, 34
 - closeFiles, 35
 - closeGlob, 35
 - closeRestart, 35
 - cntlF, 38
 - dumpf, 38
 - echoControl, 35
 - echoFnames, 35
 - echoHeader, 35
 - echoHistory, 35
 - echoMaterials, 35
 - echoRestart, 35
 - FileFormat, 34
 - fileIO, 34
 - fnames, 38
 - getFnames, 35
 - getOut, 35
 - getRestart, 35
 - getToken, 36
 - globf, 38
 - histf, 38
 - histStream, 36
 - inputStream, 36
 - iscomment, 37
 - meshf, 38
 - openDump, 37
 - openFiles, 37
 - openGlob, 37
 - openHistoryFiles, 37
 - openRestart, 37
 - operator=, 37
 - outf, 39
 - outputStream, 37
 - parseAnalysis, 37
 - parseControl, 37
 - parseConvectionBC, 37
 - parseElSet, 37
 - parseHeatFluxBC, 37
 - parseHistory, 37
 - parseMaterial, 37
 - parseNodalBC, 37
 - parseNodeSolver, 37
 - plotf, 39
 - printBanner, 37
 - readConn, 37
 - readCoord, 37
 - readHeader, 37
 - readMesh, 37
 - readNodeset, 37
 - readSideset, 37
 - restf, 39
 - setFname, 37
 - writeField, 38
 - writeFooter, 38
 - writeHeader, 38
 - writeVars, 38
 - writeVelocity, 38
- fileio.h, 114
 - CONTROL, 115
 - EXE, 115
 - FileType, 115
 - GLOBAL, 115
 - HISTORY, 115
 - MESH, 115
 - NUM_FILENAMES, 115
 - OUTPUT, 115
 - PLOT, 115
 - RESTARTR, 115
 - RESTARTW, 115
- FileType
 - fileio.h, 115
- flow_physics.h, 116

FlowPhysics
 FlowPhysics, 42
 FlowPhysics, 40
 ~FlowPhysics, 42
 calcEdge2D, 42
 calcStreamFunc, 42
 calcVorticity, 42
 FlowPhysics, 42
 Npsi, 43
 operator=, 42
 PSI, 43
 PSI_EDGE, 43
 PSI_LEL, 43
 readRestart, 42
 registerData, 42
 setup, 42
 VELX, 43
 VELY, 43
 VORTICITY, 43
 writeRestart, 43
 fnames
 fileIO, 38
 font_size
 Banner, 12
 format
 Banner, 12
 formCMK
 NavierStokes, 70
 formMK
 AdvectionDiffusion, 9
 Conduction, 17
 formMpK
 NavierStokes, 70
 formPPE
 NavierStokes, 71
 FORTRAN
 navier_stokes.h, 128
 physics.h, 135
 freeAllVariables
 DataContainer, 28
 freeVariable
 DataContainer, 28
 FX
 NavierStokes, 74
 FXN
 NavierStokes, 74
 FY
 NavierStokes, 74
 FYN
 NavierStokes, 74
 genElCon
 NavierStokes, 71
 getAmp
 SideBC, 94
 getBeam2
 Mesh, 56
 getConvectionBC
 Conduction, 17
 getElSet
 Mesh, 56
 getFnames
 fileIO, 35
 getHeatFluxBC
 Conduction, 17
 getHex8
 Mesh, 56
 getID
 SideBC, 94
 getLcurveID
 SideBC, 94
 getMaterial
 Mesh, 56
 getNbc
 Mesh, 56
 getNdim
 Mesh, 57
 getNel
 Mesh, 57
 getNmat
 Mesh, 57
 getNndsets
 Mesh, 57
 getNnp
 Mesh, 57
 getNnpe
 Mesh, 57
 getNodeBCs
 Mesh, 57
 getNodesets
 Mesh, 58
 getNsdsets
 Mesh, 58
 getOption
 Control, 21
 getOptions
 Control, 21
 getOut
 fileIO, 35
 getParam
 Control, 21
 getParams
 Control, 21
 getQuad4
 Mesh, 58
 getRestart
 fileIO, 35
 getSet

- SideBC, 94
- getSidesets
 - Mesh, 58
- getTamb
 - ConvectionBC, 25
- getTHname
 - Mesh, 58
- getTHnode
 - Mesh, 58
- getTitle
 - Control, 21
 - Mesh, 58
- getToken
 - fileIO, 36
- getVariable
 - DataContainer, 29
- getVolume
 - Mesh, 59
- getX
 - Mesh, 59
- getY
 - Mesh, 59
- getZ
 - Mesh, 59
- GLOBAL
 - fileio.h, 115
- globf
 - fileIO, 38
- globHeader
 - NavierStokes, 71
- GMV_ASCII
 - control.h, 109
- GMV_BINARY
 - control.h, 109
- heatflux
 - Conduction, 19
- HeatFluxBC
 - HeatFluxBC, 45, 46
- HeatFluxBC, 45
 - ~HeatFluxBC, 46
 - applyBC, 46
 - HeatFluxBC, 45, 46
 - operator=, 46
- HEX20
 - mesh.h, 126
- HEX8
 - mesh.h, 126
- Hex8, 47
 - faces, 47
 - ix, 47
 - mat, 47
- HIGH_ORDER
 - control.h, 108
- histf
 - fileIO, 38
- HISTORY
 - fileio.h, 115
- histStream
 - fileIO, 36
- icnt
 - navier_stokes.h, 129
- icntl
 - Control, 22
- ICOLH
 - navier_stokes.h, 129
- icolh
 - navier_stokes.h, 129
- icon
 - navier_stokes.h, 129
- id
 - Material, 50
 - Nodeset, 80
 - Sideset, 96
- IDEG
 - NavierStokes, 74
- IDTCHK
 - control.h, 106
- iebw
 - navier_stokes.h, 129
- IECN
 - NavierStokes, 74
- ielst
 - navier_stokes.h, 129
- IEN
 - NavierStokes, 74
- IEO
 - NavierStokes, 74
- IERV
 - NavierStokes, 75
- ierv
 - navier_stokes.h, 129
- IFLAG
 - navier_stokes.h, 129
- INIT_PTRS
 - list.h, 120
- initPressure
 - NavierStokes, 71
- initQuadrature
 - Physics, 85
- inputStream
 - fileIO, 36
- INSERT
 - list.h, 120
- INSERT_AFTER
 - list.h, 120
- INSERT_BEFORE

- list.h, 120
- inuse
 - DataContainer, 29
- io
 - Physics, 87
- IROWL
 - navier_stokes.h, 129
- irowl
 - navier_stokes.h, 130
- IRST
 - NavierStokes, 75
- irst
 - navier_stokes.h, 130
- iscomment
 - fileIO, 37
- itchk
 - physics.h, 135
- itknt
 - physics.h, 135
- itmax
 - physics.h, 136
- ix
 - Beam2, 13
 - Hex8, 47
 - Quad4, 91
- ja
 - physics.h, 136
- k11
 - Material, 50
- k12
 - Material, 50
- k22
 - Material, 50
- Ke
 - AdvectionDiffusion, 11
 - Conduction, 19
 - NavierStokes, 75
- lambda
 - Physics, 87
- LARGE_FONT
 - banner.h, 101
- lcid
 - NodeBC, 79
 - SideBC, 95
- LEFT_JUSTIFIED
 - banner.h, 101
- letters
 - letters.h, 117
- letters.h, 117
 - letters, 117
 - letters.length, 117
- letters_width, 117
 - medium_font, 117
 - small_font, 117
- letters_length
 - letters.h, 117
- letters_width
 - letters.h, 117
- lev
 - navier_stokes.h, 130
- list.h, 119
 - APPEND, 119
 - DELETE, 119
 - DELETE_LIST, 119
 - INIT_PTRS, 120
 - INSERT, 120
 - INSERT_AFTER, 120
 - INSERT_BEFORE, 120
 - NEXT, 120
 - PREV, 120
 - UNLINK, 121
- LUFLAG
 - control.h, 106
- luflag
 - NavierStokes, 75
- LUMPED
 - control.h, 108
- LUMPED_MASS
 - NavierStokes, 75
- lun
 - physics.h, 136
- luo
 - navier_stokes.h, 130
- MASS
 - control.h, 106
- MassType
 - control.h, 108
- mat
 - Beam2, 13
 - Hex8, 47
 - Quad4, 91
- Material, 48
 - ~Material, 49
 - Conductivity, 49
 - Cp, 50
 - Density, 49
 - id, 50
 - k11, 50
 - k12, 50
 - k22, 50
 - Material, 49
 - MatId, 49
 - mu, 51
 - operator=, 49

- rho, 51
- SpecificHeat, 50
- Viscosity, 50
- material.h, 122
- materials
 - Mesh, 61
- MatId
 - Material, 49
- matmap
 - Mesh, 55
- MAX_MEMORY_ENTRIES
 - data_container.h, 112
- MAXA
 - navier_stokes.h, 130
 - NavierStokes, 75
- maxa
 - navier_stokes.h, 130
- MAXCHR
 - types.h, 139
- MAXDOF
 - mesh.h, 126
- Me
 - AdvectionDiffusion, 11
 - Conduction, 19
 - NavierStokes, 75
- MEDIUM_FONT
 - banner.h, 101
- medium_font
 - letters.h, 117
- MESH
 - fileio.h, 115
- Mesh, 52
 - ~Mesh, 56
 - addMaterial, 56
 - addTHnode, 56
 - checkMaterials, 56
 - echoMaterials, 56
 - echoMeshParms, 56
 - echoTHnode, 56
 - ELEMENTS, 60
 - eset, 60
 - getBeam2, 56
 - getElSet, 56
 - getHex8, 56
 - getMaterial, 56
 - getNbc, 56
 - getNdim, 57
 - getNel, 57
 - getNmat, 57
 - getNndsets, 57
 - getNnp, 57
 - getNnpe, 57
 - getNodeBCs, 57
 - getNodeSets, 58
 - getNsdsets, 58
 - getQuad4, 58
 - getSidesets, 58
 - getTHname, 58
 - getTHnode, 58
 - getTitle, 58
 - getVolume, 59
 - getX, 59
 - getY, 59
 - getZ, 59
 - materials, 61
 - matmap, 55
 - Mesh, 56
 - meshtype, 61
 - nbc, 61
 - Ndim, 61
 - Neblock, 61
 - Nedge, 61
 - Nel, 61
 - Nface, 61
 - Nmat, 61
 - Nnd_sets, 62
 - Nnp, 62
 - Nnpe, 62
 - NODEBCS, 62
 - NODESETS, 62
 - Nsd_sets, 62
 - numMaterials, 59
 - numTHnode, 59
 - operator=, 60
 - Print, 60
 - registerData, 60
 - setMeshParm, 60
 - setNmat, 60
 - setTitle, 60
 - SIDASETS, 62
 - th_names, 62
 - th_nodes, 62
 - Title, 60
 - title, 62
 - VOLUME, 63
 - XCOOR, 63
 - YCOOR, 63
 - ZCOOR, 63
- mesh
 - Physics, 87
- mesh.h, 123
 - ASCILMESH, 126
 - DOFmap, 126
 - Eltype, 126
 - EXODUS_MESH, 126
 - HEX20, 126
 - HEX8, 126
 - MAXDOF, 126

- Meshtype, 126
- NEPE_HEX8, 124
- NEPE_QUAD4, 124
- NNPE_BEAM2, 125
- NNPE_HEX8, 125
- NNPE_HEX8_FACES, 125
- NNPE_QUAD4, 125
- NNPE_QUAD4_FACES, 125
- NUM_ELEMENT_TYPES, 126
- NUM_HEX8_FACES, 125
- NUM_MESH_TYPES, 126
- NUM_QUAD4_FACES, 125
- QUAD4, 126
- QUAD9, 126
- TEMP, 126
- TET10, 126
- TET4, 126
- TRI3, 126
- XVELOCITY, 126
- YVELOCITY, 126
- ZVELOCITY, 126
- meshf
 - fileIO, 38
- Meshtype
 - mesh.h, 126
- meshtype
 - Mesh, 61
- MKJ
 - Physics, 87
- mkj
 - navier_stokes.h, 130
- MKJE
 - NavierStokes, 75
- Mle
 - AdvectionDiffusion, 11
 - Conduction, 19
 - NavierStokes, 75
- MMK
 - Physics, 88
- MPK
 - Physics, 88
- mu
 - Material, 51
- mxe
 - navier_stokes.h, 130
- mxNd
 - Physics, 88
- mxNdrow
 - Physics, 88
- mxNedn
 - Physics, 88
- mxNeln
 - Physics, 88
- na
 - navier_stokes.h, 130
- name
 - Banner, 12
 - DataContainer, 29
- navier_stokes.h, 127
 - B, 128
 - BANDWD, 128
 - CONNEC, 129
 - DEGREE, 129
 - ERROR, 129
 - FORTRAN, 128
 - icnt, 129
 - ICOLH, 129
 - icolh, 129
 - icon, 129
 - iebw, 129
 - ielst, 129
 - ierv, 129
 - IFLAG, 129
 - IROWL, 129
 - irowl, 130
 - irst, 130
 - lev, 130
 - luo, 130
 - MAXA, 130
 - maxa, 130
 - mkj, 130
 - mxe, 130
 - na, 130
 - ndw, 130
 - nel, 130
 - NEQ, 130
 - neq, 131
 - nnp, 131
 - npe, 131
 - OPT, 131
 - PERMUT, 131
 - PROFIL, 131
 - RSTART, 131
 - SPACE, 131
 - WORK, 131
 - WRKLEN, 131
- NavierStokes
 - NavierStokes, 69, 70
- NavierStokes, 64
 - ~NavierStokes, 69
 - applyInverseMassBCs, 70
 - C, 73
 - calcForces, 70
 - calcKE, 70
 - COLH, 73
 - CONST_MASS, 73
 - CX, 73

CY, 73
divFree, 70
divU, 70
DM, 73
echoBCs, 70
echoElSet, 70
echoPPEStats, 70
Elbwf, 73
Elbwi, 74
formCMK, 70
formMpK, 70
formPPE, 71
FX, 74
FXN, 74
FY, 74
FYN, 74
genElCon, 71
globHeader, 71
IDEG, 74
IECN, 74
IEN, 74
IEO, 74
IERV, 75
initPressure, 71
IRST, 75
Ke, 75
luflag, 75
LUMPED_MASS, 75
MAXA, 75
Me, 75
MKJE, 75
Mle, 75
NavierStokes, 69, 70
Ndwe, 76
nodalPressure, 71
Npe, 76
Npre, 76
operator=, 71
P, 76
PN, 76
PPE, 76
ppe_na, 76
projectVelocity, 71
Q, 76
readRestart, 71
registerData, 71
reorderP, 71
reorderRHS, 71
RHS, 76
RIMX, 77
RIMY, 77
rmsDiv, 71
ROWL, 77
setICs, 72
setup, 72
setupInverseMass, 72
setupPPE, 72
solve, 72
solvePPE, 72
TEMPERATURE, 77
V1, 77
V2, 77
VELXN, 77
VELYN, 77
writeGlobal, 72
writeRestart, 72
writeSolving, 72
writeTHdata, 72
writeTHhead, 73
nbc
 Mesh, 61
NDEPSCG
 control.h, 107
Ndf
 Sideset, 96
Ndim
 Mesh, 61
NDITCHK
 control.h, 106
NDITMAX
 control.h, 106
ndw
 navier_stokes.h, 130
 physics.h, 136
Ndwe
 NavierStokes, 76
NDWRT
 control.h, 106
Neblock
 Mesh, 61
Nedge
 Mesh, 61
Nel
 Mesh, 61
 Sideset, 96
nel
 navier_stokes.h, 130
Nel_set
 ElSet, 31
NEPE_HEX8
 mesh.h, 124
NEPE_QUAD4
 mesh.h, 124
NEQ
 navier_stokes.h, 130
neq
 navier_stokes.h, 131
 physics.h, 136

- NEXT
 - list.h, 120
- Nface
 - Mesh, 61
- Nmat
 - Mesh, 61
- Nnd_sets
 - Mesh, 62
- Nnp
 - Mesh, 62
 - Nodeset, 80
 - Sideset, 96
- nnp
 - navier_stokes.h, 131
- Nnpe
 - Mesh, 62
- NNPE_BEAM2
 - mesh.h, 125
- NNPE_HEX8
 - mesh.h, 125
- NNPE_HEX8_FACES
 - mesh.h, 125
- NNPE_QUAD4
 - mesh.h, 125
- NNPE_QUAD4_FACES
 - mesh.h, 125
- nodalPressure
 - NavierStokes, 71
- NODE_LIST
 - Nodeset, 80
- NodeBC, 79
 - amp, 79
 - lcid, 79
 - nset, 79
 - nsid, 79
- NODEBCS
 - Mesh, 62
- Nodebw
 - Physics, 88
- Nodeset, 80
 - id, 80
 - Nnp, 80
 - NODE_LIST, 80
- NODESETS
 - Mesh, 62
- NODESOL
 - control.h, 106
- NodeSolver
 - control.h, 108
- Npe
 - NavierStokes, 76
- npe
 - navier_stokes.h, 131
- Npre
 - NavierStokes, 76
- Npsi
 - FlowPhysics, 43
- Nqpt
 - Physics, 88
- Nsd_sets
 - Mesh, 62
- nset
 - NodeBC, 79
- nsid
 - NodeBC, 79
- NSTEPS
 - control.h, 106
- NUM_ECHO_OPT
 - control.h, 106
- NUM_ECHO_PRM
 - control.h, 107
- NUM_ELEMENT_TYPES
 - mesh.h, 126
- NUM_FILEFORMATS
 - fileIO, 34
- NUM_FILENAMES
 - fileio.h, 115
- NUM_HEX8_FACES
 - mesh.h, 125
- NUM_ICNTL
 - control.h, 106
- NUM_MESH_TYPES
 - mesh.h, 126
- NUM_QUAD4_FACES
 - mesh.h, 125
- NUM_RCNTL
 - control.h, 107
- number
 - DataContainer, 29
- numConvectionBC
 - Conduction, 17
- numHeatFluxBC
 - Conduction, 17
- numMaterials
 - Mesh, 59
- numTHnode
 - Mesh, 59
- ONED
 - types.h, 139
- openDump
 - fileIO, 37
- openFiles
 - fileIO, 37
- openGlob
 - fileIO, 37
- openHistoryFiles
 - fileIO, 37

- openRestart
 - fileIO, 37
- operator<<
 - Banner, 12
- operator=
 - AdvectionDiffusion, 10
 - Conduction, 18
 - Control, 22
 - ConvectionBC, 25
 - DataContainer, 29
 - fileIO, 37
 - FlowPhysics, 42
 - HeatFluxBC, 46
 - Material, 49
 - Mesh, 60
 - NavierStokes, 71
 - Physics, 85
 - SideBC, 94
- OPT
 - navier_stokes.h, 131
- options.h, 132
 - analysis_opt, 132
 - analysis_prm, 132
- outf
 - fileIO, 39
- OUTPUT
 - fileio.h, 115
- outputStream
 - fileIO, 37
- P
 - NavierStokes, 76
- p
 - physics.h, 136
- P2_NAV_STOKES
 - control.h, 108
- PARMS_ONLY
 - control.h, 109
- parseAnalysis
 - fileIO, 37
- parseControl
 - fileIO, 37
- parseConvectionBC
 - fileIO, 37
- parseElSet
 - fileIO, 37
- parseHeatFluxBC
 - fileIO, 37
- parseHistory
 - fileIO, 37
- parseMaterial
 - fileIO, 37
- parseNodalBC
 - fileIO, 37
- parseNodeSolver
 - fileIO, 37
- PDSCG
 - control.h, 109
- PEPS
 - control.h, 107
- PERMUT
 - navier_stokes.h, 131
- Physics, 81
 - ~Physics, 84
 - AP, 87
 - applyNodalBC, 85
 - applyPenaltyLHS, 85
 - applyPenaltyRHS, 85
 - calcNodalBandwidth, 85
 - calcVolume, 85
 - checkVolume, 85
 - col_id, 87
 - control, 87
 - echoBCs, 85
 - echoNodalBCs, 85
 - echoNodalStats, 85
 - EdgeToNode, 85
 - FieldLimits, 85
 - initQuadrature, 85
 - io, 87
 - lambda, 87
 - mesh, 87
 - MKJ, 87
 - MMK, 88
 - MPK, 88
 - mxNd, 88
 - mxNdrow, 88
 - mxNedn, 88
 - mxNeln, 88
 - Nodebw, 88
 - Nqpt, 88
 - operator=, 85
 - Physics, 84, 85
 - PP, 88
 - qpt, 89
 - R, 89
 - readRestart, 85
 - registerData, 85
 - setColumnIds, 85
 - setICs, 86
 - setup, 86
 - setupHex8, 86
 - setupNodeIndex, 86
 - setupQuad4, 86
 - sf, 89
 - sf4, 86
 - sf4eta, 86
 - sf4xi, 86

- solve, 86
- sqpt, 89
- ssf, 89
- TMP, 89
- W, 89
- writeRestart, 86
- writeSolving, 86
- writeTHdata, 86
- writeTHhead, 87
- Z, 89
- physics.h, 134
 - ap, 135
 - b, 135
 - eps, 135
 - err, 135
 - FORTRAN, 135
 - itchk, 135
 - itknt, 135
 - itmax, 136
 - ja, 136
 - lun, 136
 - ndw, 136
 - neq, 136
 - p, 136
 - prt, 136
 - r, 136
 - rmi, 136
 - x, 136
 - z, 136
- PhysicsType
 - control.h, 108
- PITCHK
 - control.h, 106
- PITMAX
 - control.h, 106
- PLNUM
 - control.h, 106
- PLOT
 - fileio.h, 115
- plotf
 - fileIO, 39
- PlotType
 - control.h, 108
- PLTI
 - control.h, 106
- PLTYPE
 - control.h, 106
- PN
 - NavierStokes, 76
- PP
 - Physics, 88
- PPE
 - NavierStokes, 76
- ppe.na
 - NavierStokes, 76
- PPESOL
 - control.h, 106
- PPESolver
 - control.h, 109
- PREV
 - list.h, 120
- Print
 - Mesh, 60
- printBanner
 - fileIO, 37
- PrintType
 - control.h, 109
- PROFIL
 - navier_stokes.h, 131
- projectVelocity
 - NavierStokes, 71
- prt
 - physics.h, 136
- PRTI
 - control.h, 106
- PRTLEV
 - control.h, 106
- PSI
 - FlowPhysics, 43
- PSI.EDGE
 - FlowPhysics, 43
- PSI.EL
 - FlowPhysics, 43
- PSSORCG
 - control.h, 109
- ptr
 - DataContainer, 29
- PVS
 - control.h, 109
- PWRT
 - control.h, 106
- Q
 - AdvectionDiffusion, 11
 - Conduction, 19
 - NavierStokes, 76
- qpt
 - Physics, 89
- QUAD4
 - mesh.h, 126
- Quad4, 91
 - faces, 91
 - ix, 91
 - mat, 91
- QUAD9
 - mesh.h, 126
- R

- Physics, 89
- r
 - physics.h, 136
- rcntl
 - Control, 22
- readConn
 - fileIO, 37
- readCoord
 - fileIO, 37
- readHeader
 - fileIO, 37
- readMesh
 - fileIO, 37
- readNodeset
 - fileIO, 37
- readRestart
 - AdvectionDiffusion, 10
 - Conduction, 18
 - Control, 22
 - FlowPhysics, 42
 - NavierStokes, 71
 - Physics, 85
- readSideset
 - fileIO, 37
- Real
 - types.h, 139
- REAL_EPSILON
 - types.h, 139
- REAL_MAX
 - types.h, 139
- REAL_MIN
 - types.h, 139
- registerData
 - AdvectionDiffusion, 10
 - Conduction, 18
 - FlowPhysics, 42
 - Mesh, 60
 - NavierStokes, 71
 - Physics, 85
- registerVariable
 - DataContainer, 29
- registration
 - DataContainer, 29
- reorderP
 - NavierStokes, 71
- reorderRHS
 - NavierStokes, 71
- reportMemory
 - DataContainer, 29
- RESTARTR
 - fileio.h, 115
- RESTARTW
 - fileio.h, 115
- restf
 - fileIO, 39
- RESULTS
 - control.h, 109
- rho
 - Material, 51
- RHS
 - NavierStokes, 76
- RIGHT_JUSTIFIED
 - banner.h, 102
- RIMX
 - NavierStokes, 77
- RIMY
 - NavierStokes, 77
- rmi
 - physics.h, 136
- rmsDiv
 - NavierStokes, 71
- ROWL
 - NavierStokes, 77
- RSTART
 - navier_stokes.h, 131
- set
 - SideBC, 95
- setAmp
 - SideBC, 94
- setColumnIds
 - Physics, 85
- setFname
 - fileIO, 37
- setICs
 - AdvectionDiffusion, 10
 - Conduction, 18
 - NavierStokes, 72
 - Physics, 86
- setID
 - SideBC, 94
- setLcurveID
 - SideBC, 94
- setMeshParm
 - Mesh, 60
- setNmat
 - Mesh, 60
- setNumber
 - SideBC, 95
- setOption
 - Control, 22
- setParam
 - Control, 22
- setTamb
 - ConvectionBC, 25
- setTitle
 - Control, 22
 - Mesh, 60

- setup
 - AdvectionDiffusion, 10
 - Conduction, 18
 - FlowPhysics, 42
 - NavierStokes, 72
 - Physics, 86
- setupHex8
 - Physics, 86
- setupInverseMass
 - NavierStokes, 72
- setupNodeIndex
 - Physics, 86
- setupPPE
 - NavierStokes, 72
- setupQuad4
 - Physics, 86
- sf
 - Physics, 89
- sf4
 - Physics, 86
- sf4eta
 - Physics, 86
- sf4xi
 - Physics, 86
- SLADV_DIFF
 - control.h, 108
- SIDE_LIST
 - Sideset, 97
- SideBC
 - SideBC, 93
- SideBC, 92
 - ~SideBC, 93
 - amp, 95
 - getAmp, 94
 - getID, 94
 - getLcurveID, 94
 - getSet, 94
 - lcid, 95
 - operator=, 94
 - set, 95
 - setAmp, 94
 - setID, 94
 - setLcurveID, 94
 - setNumber, 95
 - SideBC, 93
 - ssid, 95
- Sideset, 96
 - EL_LIST, 96
 - id, 96
 - Ndf, 96
 - Nel, 96
 - Nnp, 96
 - SIDE_LIST, 97
- SIDSETS
 - Mesh, 62
- size
 - DataContainer, 29
- SMALL_FONT
 - banner.h, 102
- small_font
 - letters.h, 117
- SOLTYP
 - control.h, 106
- solve
 - AdvectionDiffusion, 10
 - Conduction, 18
 - NavierStokes, 72
 - Physics, 86
- solvePPE
 - NavierStokes, 72
- SPACE
 - navier_stokes.h, 131
- SpecificHeat
 - Material, 50
- sqpt
 - Physics, 89
- ssf
 - Physics, 89
- ssid
 - SideBC, 95
- Tamb
 - ConvectionBC, 26
- TEMP
 - mesh.h, 126
- TEMPERATURE
 - AdvectionDiffusion, 11
 - Conduction, 19
 - NavierStokes, 77
- TET10
 - mesh.h, 126
- TET4
 - mesh.h, 126
- TF
 - control.h, 107
- th_names
 - Mesh, 62
- th_nodes
 - Mesh, 62
- THETAA
 - control.h, 107
- THETAB
 - control.h, 107
- THETAF
 - control.h, 107
- THETAK
 - control.h, 107
- THREED

- types.h, 139
- Title
 - Mesh, 60
- title
 - Control, 23
 - Mesh, 62
- TMP
 - Physics, 89
- TR13
 - mesh.h, 126
- TS
 - control.h, 107
- TTYI
 - control.h, 106
- TWOD
 - types.h, 139
- types.h, 138
 - DataIndex, 139
 - MAXCHR, 139
 - ONED, 139
 - Real, 139
 - REAL_EPSILON, 139
 - REAL_MAX, 139
 - REAL_MIN, 139
 - THREED, 139
 - TWOD, 139
 - uint, 140
- uint
 - types.h, 140
- UNKNOWN_PLOT
 - control.h, 109
- UNKNOWN_SOLVER
 - control.h, 108
- UNLINK
 - list.h, 121
- UPWIND
 - control.h, 106
- V1
 - NavierStokes, 77
- V2
 - NavierStokes, 77
- VELX
 - FlowPhysics, 43
- VELXN
 - NavierStokes, 77
- VELY
 - FlowPhysics, 43
- VELYN
 - NavierStokes, 77
- VERBOSE
 - control.h, 109
- Viscosity
 - Material, 50
- VOLUME
 - Mesh, 63
- VORTICITY
 - FlowPhysics, 43
- W
 - Physics, 89
- WORK
 - navier_stokes.h, 131
- writeField
 - fileIO, 38
- writeFooter
 - fileIO, 38
- writeGlobal
 - NavierStokes, 72
- writeHeader
 - fileIO, 38
- writeRestart
 - AdvectionDiffusion, 10
 - Conduction, 18
 - Control, 22
 - FlowPhysics, 43
 - NavierStokes, 72
 - Physics, 86
- writeSolving
 - AdvectionDiffusion, 10
 - Conduction, 18
 - NavierStokes, 72
 - Physics, 86
- writeTHdata
 - AdvectionDiffusion, 10
 - Conduction, 18
 - NavierStokes, 72
 - Physics, 86
- writeTHhead
 - AdvectionDiffusion, 10
 - Conduction, 18
 - NavierStokes, 73
 - Physics, 87
- writeVars
 - fileIO, 38
- writeVelocity
 - fileIO, 38
- WRKLEN
 - navier_stokes.h, 131
- WTA
 - control.h, 107
- WTB
 - control.h, 107
- WTF
 - control.h, 107
- WTK
 - control.h, 107

x
 physics.h, 136
XCOOR
 Mesh, 63
XVELOCITY
 mesh.h, 126

YCOOR
 Mesh, 63
YVELOCITY
 mesh.h, 126

Z
 Physics, 89
z
 physics.h, 136
ZCOOR
 Mesh, 63
ZVELOCITY
 mesh.h, 126