ExternalMedia Reference Manual

1.0

Generated by Doxygen 1.5.1-p1

Wed Apr 25 10:25:56 2007

Contents

1 External Media HowTo **1**

1.1 Introduction 1

1.2 1. Architecture of the package 1

1.3 2. Using an already implemented external medium with Dymola 2

1.4 3. Developing a new external medium model 3

2 ExternalMedia Hierarchical Index **5**

2.1 ExternalMedia Class Hierarchy 5

3 ExternalMedia Class Index **7**

3.1 ExternalMedia Class List 7

4 ExternalMedia File Index **9**

4.1 ExternalMedia File List 9

5 ExternalMedia Class Documentation **11**

5.1 BaseSolver Class Reference 11

5.2 BaseTwoPhaseMedium Class Reference 18

5.3 FluidConstants Struct Reference 27

5.4 MediumMap Class Reference 28

5.5 SolverMap Class Reference 32

5.6 TestSolver Class Reference 34

5.7 TwoPhaseMedium Class Reference 38

5.8 TwoPhaseMediumProperties Class Reference 40

6 ExternalMedia File Documentation **43**

6.1 errorhandling.h File Reference 43

6.2 externaltwophasemedium.cpp File Reference 45

6.3 externaltwophasemedium.h File Reference 55

6.4 include.h File Reference 65

Chapter 1

External Media HowTo

1.1 Introduction

Welcome to the ExternalMediaPackage HowTo guide. This package is meant to help you interfacing your favourite external fluid computation code(s) to Modelica, using the Modelica.Media interface. Currently, the ExternalMedia library only supports two-phase media, using Dymola 6 and Microsoft Visual Studio. In the future, we might add support for single-phase pure and mixture media, as well as for OpenModelica and gcc. Please contact the authors if you're interested in such developments.

The ExternalMedia package has been implemented and is maintained by Francesco Casella (`casella@elet.polimi.it`) and Christoph Richter (`ch.richter@tu-bs.de`). The code is released under the Modelica license.

1.2 1. Architecture of the package

At the top level there is a Modelica package (ExternalMedia), which contains all the basic infrastructure needed to use external fluid properties computation software through a Modelica.Media compliant interface. In particular, the ExternalMedia.Media.ExternalTwoPhaseMedium package is a full-fledged implementation of a two-phase medium model, compliant with the Modelica.Media.Interfaces.PartialTwoPhaseMedium interface. The ExternalTwoPhaseMedium package can be used with any external fluid property computation software; the specific software to be used is specified by changing the libraryName package constant, which is then handled by the underlying C code to select the appropriate external code to use.

The functions within ExternalTwoPhaseMedium talk to a C/C++ interface layer (called externaltwophasemedium_lib.cpp) via external functions calls. This layer takes care of initializing the external packages, and maintains a collection of C++ objects (stored within a **MediumMap** (p. 28) C++ object), corresponding to different instances of the medium model. More specifically, each instance of ExternalTwoPhaseMedium.BaseProperties corresponds to an instance of the C++ object **TwoPhaseMedium** (p. 38), identified by an integer uniqueID; each **TwoPhaseMedium** (p. 38) object also contains a pointer to a C++ object inheriting from **BaseSolver** (p. 11); this is a wrapper object around the actual external code (or solver) which has to be called to compute the fluid properties. It is possible to use multiple instances of possibly different solvers at the same time; the collection of active solver is managed by the **SolverMap** (p. 32) C++ object.

The default implementation of the external medium works as follows. At initialization, each instance of the BaseProperties model calls the **createMedium_()** (p. 49) function of the interface layer. This triggers the allocation within the **MediumMap** (p. 28) object of a new **TwoPhaseMedium** (p. 38) object, which will contain all the medium properties, and a pointer to the appropriate solver. This object will be stored in the **MediumMap** (p. 28), and identified with a uniqueID, which is returned by the createMedium() function and stored within the BaseProperties model. The solver object is created by the **SolverMap** (p. 32) object, based on the mediumName, libraryName and substanceName constants of the Modelica medium package

- only one solver object is created for multiple instances of the same external medium.

When one of the setState functions within BaseProperties is called, the interface layer will call the corresponding setState function of the **TwoPhaseMedium** (p. 38) object, which will in turn invoke the setState function of the appropriat solver object, to actually compute the fluid properties. These will be stored within the **TwoPhaseMedium** (p. 38) object, for later retrieval. The setState function then returns a ThermodynamicState record, which only contains integer uniqueID, indentifying the **TwoPhaseMedium** (p. 38) object in the **MediumMap** (p. 28). Subsequent calls to compute any other functions of that state will just fetch the corresponding values from the **TwoPhaseMedium** (p. 38) object with the given uniqueID - the **TwoPhaseMedium** (p. 38) object acts like a cache for already computed properties. The default implementation assumes that all available properties are computed at once by the setState functions, so that subsequent calls to any property function will just result in a retrieval from the cache - more sophisticated implementations could be devised, where the setState function only computes some properties at first, and flags are provided within the **TwoPhaseMedium** (p. 38) object to keep into account what has already been computed and stored in the cache, and what has still to be computed if need.

With the current implementation, it is not possible to use setState functions outside the BaseProperties model, unless one explicitly manages and sets the uniqueID integers - this is not possible if the user model has to be compatible with other Modelica.Media fluid models, which do not have the uniqueID. Support for calls to setState functions with automatic management of the uniqueID might be added in the future.

1.3 2. Using an already implemented external medium with Dymola

If somebody else has already implemented the external medium, you only need a few things to use it on your computer:

- the ExternalMedia Modelica package
- the ExternalTwoPhaseMedium.lib (and possibly the ExternalTwoPhaseMedium.dll) files, containing the compiled C/C++ code
- the **externaltwophasemedium.h** (p. 55) C header file

Copy the ExternalTwoPhaseMedium.lib file within the Dymola/bin/lib directory, the ExternalTwoPhaseMedium.dll file in the Windows/System32 directory (only if the library has been compiled to a DLL), and the **externaltwophasemedium.h** (p. 55) file in the Dymola/Source directory.

You can check that the compiler setup is working by running the TestMedium model in the ExternalMedia.Test package - this will simulate a dummy external fluid model, roughly corresponding to liquid water at low pressure and temperature.

You can now use the external media provided in the ExternalMedia.Media package. There are two options here: the first is to use ExternalTwoPhaseMedium directly, using modifiers to change libraryName and substanceNames to suit your needs; the other is to use or modify one of the pre-packaged media. The libraryName string will be decoded by the **SolverMap** (p. 32) object - for further details on the syntax of the string, ask to the implementor of the interface to the code you'll actually use.

1.4 3. Developing a new external medium model

The ExternalMedia package has been designed to ease your task, so that you will only have to write the mimum amount of code which is strictly specific to your external code - everything else is already provided. The following instructions apply if you want to develop an external medium model which include a (sub)set of the functions defined in Modelica.Media.Interfaces.PartialTwoPhaseMedium, and you want to compute all the properties at once when you call the setState function (this is a reasonable choice, since the majority of the time is usually spent to run preliminary computations (e.g. compute the Helmholtz function), and then computing all the properties only adds a small overhead on top of that time).

First of all, you have to write you own solver object code; you can look at the code of the TestMedium and FluidPropMedium code as examples. Inherit from the **BaseSolver** (p. 11) object, which provides default implementations for most of the required functions, and then just add your own implementation for the following functions: object constructor, object destructor, setMediumConstants(), setSat_p(), setSat_T(), setState_ph(), setState_pT(), setState_ps(), setState_dT(). Note that the setState functions need to compute and fill in all the fields of the& properties record, otherwise the corresponding functions will not work. For example, if your setState functions do not set the value of properties, then you won't be able to later call thermalConductivity(medium.state) in your Modelica models. On the other hand, you don't necessarily need to implement all of four setState functions: if you know in advance that your models will only use certain combinations of variable as inputs (e.g. p, h), then you might omit implementing the setState functions corresponding to the other ones.

Then you must modify the SolverMap::addSolver() function, so that it will instantiate your new solver when it is called with the appropriate libraryName string. You are free to invent you own syntax for the libraryName string, in case you'd like to be able to set up the external medium with some additional configuration data from within Modelica - it's up to you to decode that syntax within the addSolver() function, and within the constructor of your solver object.

Finally, add the .cpp and .h files to the C/C++ project, recompile to a static library (or to a DLL), and then follow the instructions of Section 2. of this document.

Chapter 2

ExternalMedia Hierarchical Index

2.1 ExternalMedia Class Hierarchy

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

BaseSolver 11
  TestSolver 34
BaseTwoPhaseMedium 18
  TwoPhaseMedium 38
FluidConstants 27
MediumMap 28
SolverMap 32
TwoPhaseMediumProperties 40

Chapter 3

ExternalMedia Class Index

3.1 ExternalMedia Class List

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

BaseSolver (Base solver class) 11
BaseTwoPhaseMedium (Base two-phase medium class) 18
FluidConstants (Fluid constants struct) 27
MediumMap (Medium map) 28
SolverMap (Solver map) 32
TestSolver (Test solver class) 34
TwoPhaseMedium (Two phase medium) 38
TwoPhaseMediumProperties (Two phase medium property class) 40

Chapter 4

ExternalMedia File Index

4.1 ExternalMedia File List

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

errorhandling.h (Error handling for external library) 43
externaltwophasemedium.cpp (Interface layer) 45
externaltwophasemedium.h (Header file to be included in Dymola) 55
include.h (Generic include file) 65

Chapter 5

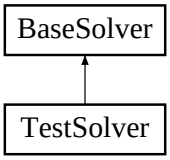
ExternalMedia Class Documentation

5.1 BaseSolver Class Reference

Base solver class.

`#include <basesolver.h>`

Inheritance diagram for BaseSolver::



Public Member Functions

- **BaseSolver** (const string &**mediumName**, const string &**libraryName**, const string &**substanceName**)

  Constructor.

- virtual ~**BaseSolver** ()

  Destructor.

- virtual void **setFluidConstants** ()

  Set fluid constants.

- virtual void **setSat_p** (double &p, **TwoPhaseMediumProperties** *const properties)

  Set saturation properties from p.

- virtual void **setSat_T** (double &T, **TwoPhaseMediumProperties** *const properties)

  Set saturation properties from T.

- virtual void **setSat_p_state** (**TwoPhaseMediumProperties** *const properties)

  Set saturation properties from within BaseProperties model.

- virtual void **setState_dT** (double &d, double &T, int &phase, **TwoPhaseMediumProperties** *const properties)

  Set state from d, T, and phase.

- virtual void **setState_ph** (double &p, double &h, int &phase, **TwoPhaseMediumProperties** *const properties)

  Set state from p, h, and phase.

- virtual void **setState_ps** (double &p, double &s, int &phase, **TwoPhaseMediumProperties** *const properties)

  Set state from p, s, and phase.

- virtual void **setState_pT** (double &p, double &T, **TwoPhaseMediumProperties** *const properties)

  Set state from p and T.

- virtual void **setBubbleState** (int phase, **TwoPhaseMediumProperties** *const properties, **TwoPhaseMediumProperties** *const bubbleProperties)

  Set bubble state.

- virtual void **setDewState** (int phase, **TwoPhaseMediumProperties** *const properties, **TwoPhaseMediumProperties** *const bubbleProperties)

  Set dew state.

- virtual bool **computeDerivatives** (**TwoPhaseMediumProperties** *const properties)

  Compute derivatives.

- virtual double **isentropicEnthalpy** (double &p, **TwoPhaseMediumProperties** *const properties)

  Compute isentropic enthalpy.

- double **molarMass** () const

  Return molar mass (Default implementation provided).

- double **criticalTemperature** () const

  Return temperature at critical point (Default implementation provided).

- double **criticalPressure** () const

  Return pressure at critical point (Default implementation provided).

- double **criticalDensity** () const

  Return density at critical point (Default implementation provided).

- double **criticalMolarVolume** () const

  Return molar volume at critical point (Default implementation provided).

- double **criticalEnthalpy** () const

  Return specific enthalpy at critical point (Default implementation provided).

- double **criticalEntropy** () const

  Return specific entropy at critical point (Default implementation provided).

Public Attributes

- string **mediumName**

  Medium name.

- string **libraryName**

  Library name.

- string **substanceName**

  Substance name.

Protected Attributes

- **FluidConstants _fluidConstants**

  Fluid constants.

5.1.1 Detailed Description

Base solver class.

This is the base class for all external solver objects (e.g. **TestSolver** (p. 34), FluidPropSolver).

Christoph Richter, Francesco Casella, Sep 2006

5.1.2 Constructor & Destructor Documentation

5.1.2.1 BaseSolver::BaseSolver (const string & *mediumName*, const string & *libraryName*, const string & *substanceName*)

Constructor.

The constructor is copying the medium name, library name and substance name to the locally defined variables.

Parameters:

mediumName Arbitrary medium name

libraryName Name of the external fluid property library

substanceName Substance name

5.1.2.2 BaseSolver::~BaseSolver () `[virtual]`

Destructor.

The destructor for the base solver if currently not doing anything.

5.1.3 Member Function Documentation

5.1.3.1 void BaseSolver::setFluidConstants () [virtual]

Set fluid constants.

This function sets the fluid constants which are defined in the **FluidConstants** (p. 27) record in Modelica. It should be called when a new solver is created.

Must be re-implemented in the specific solver

Reimplemented in **TestSolver** (p. 35).

5.1.3.2 void BaseSolver::setSat_p (double & *p*, TwoPhaseMediumProperties *const *properties*) [virtual]

Set saturation properties from p.

This function sets the saturation properties for the given pressure p. The computed values are written to the two phase medium propery struct.

Must be re-implemented in the specific solver

Parameters:

p Pressure

properties Two phase medium property record

Reimplemented in **TestSolver** (p. 35).

5.1.3.3 void BaseSolver::setSat_T (double & *T*, TwoPhaseMediumProperties *const *properties*) [virtual]

Set saturation properties from T.

This function sets the saturation properties for the given temperature T. The computed values are written to the two phase medium propery struct.

Must be re-implemented in the specific solver

Parameters:

T Temperature

properties Two phase medium property record

Reimplemented in **TestSolver** (p. 35).

5.1.3.4 void BaseSolver::setSat_p_state (TwoPhaseMediumProperties *const *properties*) [virtual]

Set saturation properties from within BaseProperties model.

This function sets the saturation properties for the given pressure p and is desined to be used from within the BaseProperties model in Modelica. The computed values are written to the two phase medium propery struct.

Must be re-implemented in the specific solver

Parameters:

properties Two phase medium property record

Reimplemented in **TestSolver** (p. 36).

5.1.3.5 void BaseSolver::setState_dT (double & *d*, double & *T*, int & *phase*, TwoPhaseMediumProperties *const *properties*) [virtual]

Set state from d, T, and phase.

This function sets the thermodynamic state record for the given density d, the temperature T and the specified phase. The computed values are written to the two phase medium property struct.

Must be re-implemented in the specific solver

Parameters:

d Density

T Temperature

phase Phase (2 for two-phase, 1 for one-phase, 0 if not known)

properties Two phase medium property record

Reimplemented in **TestSolver** (p. 36).

5.1.3.6 void BaseSolver::setState_ph (double & *p*, double & *h*, int & *phase*, TwoPhaseMediumProperties *const *properties*) [virtual]

Set state from p, h, and phase.

This function sets the thermodynamic state record for the given pressure p, the specific enthalpy h and the specified phase. The computed values are written to the two phase medium property struct.

Must be re-implemented in the specific solver

Parameters:

p Pressure

h Specific enthalpy

phase Phase (2 for two-phase, 1 for one-phase, 0 if not known)

properties Two phase medium property record

Reimplemented in **TestSolver** (p. 36).

5.1.3.7 void BaseSolver::setState_ps (double & *p*, double & *s*, int & *phase*, TwoPhaseMediumProperties *const *properties*) [virtual]

Set state from p, s, and phase.

This function sets the thermodynamic state record for the given pressure p, the specific entropy s and the specified phase. The computed values are written to the two phase medium property struct.

Must be re-implemented in the specific solver

Parameters:

p Pressure

s Specific entropy

phase Phase (2 for two-phase, 1 for one-phase, 0 if not known)

properties Two phase medium property record

Reimplemented in **TestSolver** (p. 37).

5.1.3.8 void BaseSolver::setState_pT (double & *p*, double & *T*, TwoPhaseMediumProperties *const *properties*) [virtual]

Set state from p and T.

This function sets the thermodynamic state record for the given pressure p and the temperature T. The computed values are written to the two phase medium property struct.

Must be re-implemented in the specific solver

Parameters:

p Pressure

T Temperature

properties Two phase medium property record

Reimplemented in **TestSolver** (p. 37).

5.1.3.9 void BaseSolver::setBubbleState (int *phase*, TwoPhaseMediumProperties *const *properties*, TwoPhaseMediumProperties *const *bubbleProperties*) [virtual]

Set bubble state.

This function sets the bubble state record bubbleProperties corresponding to the saturation data contained in the properties record.

The default implementation of the setBubbleState function is relying on the correct behaviour of setState_ph with respect to the state input. Can be overridden in the specific solver code to get more efficient or correct handling of this situation.

Parameters:

phase Phase (1: one-phase, 2: two-phase)

properties Two phase medium property record with saturation properties data

bubbleProperties Two phase medium property record where to write the bubble point properties

5.1.3.10 void BaseSolver::setDewState (int *phase*, TwoPhaseMediumProperties *const *properties*, TwoPhaseMediumProperties *const *dewProperties*) [virtual]

Set dew state.

This function sets the dew state record dewProperties corresponding to the saturation data contained in the properties record.

The default implementation of the setDewState function is relying on the correct behaviour of setState_ph with respect to the state input. Can be overridden in the specific solver code to get more efficient or correct handling of this situation.

Parameters:

phase Phase (1: one-phase, 2: two-phase)

properties Two phase medium property record with saturation properties data

dewProperties Two phase medium property record where to write the dew point properties

5.1.3.11 bool BaseSolver::computeDerivatives (TwoPhaseMediumProperties *const *properties*) [virtual]

Compute derivatives.

This function computes the derivatives according to the Bridgman's table. The computed values are written to the two phase medium property struct. This function can be called from within the setState_XX routines when implementing a new solver. Please be aware that cp, beta and kappa have to be provided to allow the computation of the derivatives. It returns false if the computation failed.

Default implementation provided.

Parameters:

properties Two phase medium property record

5.1.3.12 double BaseSolver::isentropicEnthalpy (double & *p*, TwoPhaseMediumProperties *const *properties*) [virtual]

Compute isentropic enthalpy.

This function returns the enthalpy at pressure p after an isentropic transformation from the state specified by the properties input

Must be re-implemented in the specific solver

Parameters:

p New pressure

properties Two phase medium property record corresponding to current state

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

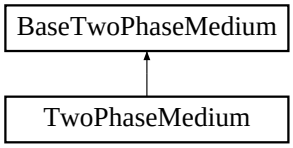
- basesolver.h
- basesolver.cpp

5.2 BaseTwoPhaseMedium Class Reference

Base two-phase medium class.

`#include <basetwophasemedium.h>`

Inheritance diagram for BaseTwoPhaseMedium::



Public Member Functions

- **BaseTwoPhaseMedium** (const string &mediumName, const string &libraryName, const string &substanceName, **BaseSolver** *const solver, const int &uniqueID)

  Constructor.

- virtual ~**BaseTwoPhaseMedium** ()

  Destructor.

- void **reinitMedium** (const string &mediumName, const string &libraryName, const string &substanceName, **BaseSolver** *const solver, const int &uniqueID)

  Reinit medium constants.

- int **uniqueID** () const

  Return unique ID number.

- string **mediumName** () const

  Return medium name.

- string **libraryName** () const

  Return library name.

- string **substanceName** () const

  Return substance name.

- **TwoPhaseMediumProperties** * **properties** () const

  Return pointer to properties.

- **BaseSolver** * **solver** () const

  Return pointer to solver.

- virtual void **setSolver** (**BaseSolver** *const solver)

  Set solver.

- int **phase** () const

  Return phase (error handling included).

- double **beta** () const

  Return isothermal expansion coefficient (error handling included).

- double **cp** () const

  Return specific heat capacity cp (error handling included).

- double **cv** () const

  Return specific heat capacity cv (error handling included).

- double **d** () const

  Return density (error handling included).

- double **dd_dp_h** () const

  Return derivative of density wrt pressure at constant specific enthalpy (error handling included).

- double **dd_dh_p** () const

  Return derivative of density wrt specific enthalpy at constant pressure (error handling included).

- double **h** () const

  Return specific enthalpy (error handling included).

- double **kappa** () const

  Return compressibility (error handling included).

- double **p** () const

  Return pressure (error handling included).

- double **s** () const

  Return specific entropy (error handling included).

- double **T** () const

  Return temperature (error handling included).

- double **h_iso** (double &p) const

  Return the enthalpy at pressure p after an isentropic transformation form the specified medium state.

- double **dT_dp_h** () const

  Return derivative of temperature wrt pressure at constant specific enthalpy (error handling included).

- double **dT_dh_p** () const

  Return derivative of temperature wrt specific enthalpy at constant pressure (error handling included).

- double **ps** () const

  Return saturation pressure (error handling included).

- double **Ts** () const

Return saturation temperature (error handling included).

- double **dl** () const

  Return bubble density (error handling included).

- double **dv** () const

  Return dew density (error handling included).

- double **hl** () const

  Return bubble specific enthalpy (error handling included).

- double **hv** () const

  Return dew specific enthalpy (error handling included).

- double **sl** () const

  Return bubble specific entropy (error handling included).

- double **sv** () const

  Return dew specific entropy (error handling included).

- double **d_Ts_dp** () const

  Return derivative of saturation temperature wrt pressure (error handling included).

- double **d_dl_dp** () const

  Return derivative of bubble density wrt pressure (error handling included).

- double **d_dv_dp** () const

  Return derivative of dew density wrt pressure (error handling included).

- double **d_hl_dp** () const

  Return derivative of bubble specific enthalpy wrt pressure (error handling included).

- double **d_hv_dp** () const

  Return derivative of dew specific enthalpy wrt pressure (error handling included).

- double **eta** () const

  Return dynamic viscosity (error handling included).

- double **lambda** () const

  Return thermal conductivity (error handling included).

- double **Pr** () const

  Return Prandtl number (error handling included).

- double **sigma** () const

  Return surface tension (error handling included).

- virtual void **setSat_p** (double &p)

  Set saturation properties from p.

- virtual void **setSat_T** (double &T)
  Set saturation properties from T.
- virtual void **setSat_p_state** ()
  Set saturation properties.
- virtual void **setState_dT** (double &d, double &T, int &phase)
  Set properties from d, T, and phase.
- virtual void **setState_ph** (double &p, double &h, int &phase)
  Set properties from p, h, and phase.
- virtual void **setState_ps** (double &p, double &s, int &phase)
  Set properties from p, s, and phase.
- virtual void **setState_pT** (double &p, double &T)
  Set properties from p and T.
- virtual int **getBubbleUniqueID** (int phase)
  Get bubble unique ID.
- virtual int **getDewUniqueID** (int phase)
  Get dew unique ID.
- virtual void **setDewState** (int phase)
  Set dew state.
- virtual void **setBubbleState** (int phase)
  Set bubble state.

Protected Attributes

- **TwoPhaseMediumProperties** * **_properties**
  Medium property record.
- **BaseSolver** * **_solver**
  Solver.
- int **_uniqueID**
  Unique ID.
- string **_mediumName**
  Medium name.
- string **_libraryName**
  Library name.
- string **_substanceName**

Substance name.

- int **_dewUniqueIDOnePhase**

  Unique ID of corresponding 1-phase dew state medium object, set by setDewState.

- int **_dewUniqueIDTwoPhase**

  Unique ID of corresponding 2-phase dew state medium object, set by setDewState.

- int **_bubbleUniqueIDOnePhase**

  Unique ID of corresponding 1-phase bubble state medium object, set by setBubbleState.

- int **_bubbleUniqueIDTwoPhase**

  Unique ID of corresponding 2-phase bubble state medium object, set by setBubbleState.

5.2.1 Detailed Description

Base two-phase medium class.

This class defines all the variables and member functions which are needed to use external Modelica medium models extending from PartialExternalTwoPhaseMedium.

The functions defined here are not fluid-specific, thus need not be adapted to your own specific fluid property computation code.

Francesco Casella, Christoph Richter Sep 2006

5.2.2 Constructor & Destructor Documentation

5.2.2.1 BaseTwoPhaseMedium::BaseTwoPhaseMedium (const string & *mediumName*, const string & *libraryName*, const string & *substanceName*, BaseSolver *const *solver*, const int & *uniqueID*)

Constructor.

The constructor initializes the following variables with the specified values:

- _mediumName = mediumName
- _libraryName = libraryName
- _substanceName = substanceName
- _solver = solver
- _uniqueID = uniqueID
- _dewUniqueIDOnePhase = 0
- _dewUniqueIDTwoPhase = 0
- _bubbleUniqueIDOnePhase = 0
- _bubbleUniqueIDTwoPhase = 0

Parameters:

mediumName Medium name

libraryName Library name

substanceName Substance name

solver Solver

uniqueID Unique ID number

5.2.2.2 BaseTwoPhaseMedium::~BaseTwoPhaseMedium () [virtual]

Destructor.

The destructor for the base solver if currently not doing anything.

5.2.3 Member Function Documentation

5.2.3.1 void BaseTwoPhaseMedium::reinitMedium (const string & *mediumName*, const string & *libraryName*, const string & *substanceName*, BaseSolver *const *solver*, const int & *uniqueID*)

Reinit medium constants.

This function reinites the medium constans to the inputs and the properties to their default values.

Parameters:

mediumName Medium name

libraryName Library name

substanceName Substance name

solver Solver

uniqueID Unique ID number

See also:

BaseTwoPhaseMedium() (p. 22)

5.2.3.2 void BaseTwoPhaseMedium::setSolver (BaseSolver *const *solver*) [virtual]

Set solver.

This functions ets the solver.

Parameters:

solver New solver

5.2.3.3 void BaseTwoPhaseMedium::setSat_p (double & *p*) [virtual]

Set saturation properties from p.

This function calls the **setSat_p()** (p. 24) function of the solver.

Parameters:

p Pressure

5.2.3.4 void BaseTwoPhaseMedium::setSat_T (double & *T*) [virtual]

Set saturation properties from T.

This function calls the **setSat_T()** (p. 24) function of the solver.

Parameters:

T Temperature

5.2.3.5 void BaseTwoPhaseMedium::setSat_p_state () [virtual]

Set saturation properties.

This function calls the **setSat_p_state()** (p. 24) function of the solver and is designed to be used for function calls from within BaseProperties.

5.2.3.6 void BaseTwoPhaseMedium::setState_dT (double & *d*, double & *T*, int & *phase*) [virtual]

Set properties from d, T, and phase.

This function calls the **setState_dT()** (p. 24) function of the solver.

Parameters:

d Density

T Temperature

phase Phase (2 for two-phase, 1 for one-phase, 0 if not known)

5.2.3.7 void BaseTwoPhaseMedium::setState_ph (double & *p*, double & *h*, int & *phase*) [virtual]

Set properties from p, h, and phase.

This function calls the **setState_ph()** (p. 24) function of the solver.

Parameters:

p Pressure

h Specific enthalpy

phase Phase (2 for two-phase, 1 for one-phase, 0 if not known)

5.2.3.8 void BaseTwoPhaseMedium::setState_ps (double & *p*, double & *s*, int & *phase*) [virtual]

Set properties from p, s, and phase.

This function calls the **setState_ps()** (p. 25) function of the solver.

Parameters:

p Pressure

s Specific entropy

phase Phase (2 for two-phase, 1 for one-phase, 0 if not known)

5.2.3.9 void BaseTwoPhaseMedium::setState_pT (double & *p*, double & *T*) [virtual]

Set properties from p and T.

This function calls the **setState_pT()** (p. 25) function of the solver.

Parameters:

p Pressure

T Temperature

5.2.3.10 int BaseTwoPhaseMedium::getBubbleUniqueID (int *phase*) [virtual]

Get bubble unique ID.

Get the unique ID of the bubble point medium object corresponding to psat, Tsat and the phase input, possibly allocating a medium object on the medium map if needed.

Parameters:

phase Phase (2 for two-phase, 1 for one-phase, 0 if not known)

5.2.3.11 int BaseTwoPhaseMedium::getDewUniqueID (int *phase*) [virtual]

Get dew unique ID.

Get the unique ID of the dew point medium object corresponding to psat, Tsat and the phase input, possibly allocating a medium object on the medium map if needed.

Parameters:

phase Phase (2 for two-phase, 1 for one-phase, 0 if not known)

5.2.3.12 void BaseTwoPhaseMedium::setDewState (int *phase*) [virtual]

Set dew state.

Sets the properties of the dew medium depending on the specified phase.

Parameters:

phase Phase (2 for two-phase, 1 for one-phase, 0 if not known)

5.2.3.13 void BaseTwoPhaseMedium::setBubbleState (int *phase*) [virtual]

Set bubble state.

Sets the properties of the bubble medium depending on the specified phase.

Parameters:

phase Phase (2 for two-phase, 1 for one-phase, 0 if not known)

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

- basetwophasemedium.h
- basetwophasemedium.cpp

5.3 FluidConstants Struct Reference

Fluid constants struct.

`#include <fluidconstants.h>`

Public Attributes

- double **MM**

  Molar mass.

- double **pc**

  Pressure at critical point.

- double **Tc**

  Temperature at critical point.

- double **dc**

  Density at critical point.

- double **hc**

  Specific enthalpy at critical point.

- double **sc**

  Specific entropy at critical point.

5.3.1 Detailed Description

Fluid constants struct.

The fluid constants struct contains all the constant fluid properties that are returned by the external solver.

Francesco Casella, Christoph Richter Nov 2006

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

- fluidconstants.h

5.4 MediumMap Class Reference

Medium map.

`#include <mediummap.h>`

Static Public Member Functions

- static int **addMedium** (const string &mediumName, const string &libraryName, const string &substanceName)
  Add a new medium object to the medium map.
- static int **addTransientMedium** (const string &mediumName, const string &libraryName, const string &substanceName)
  Add a new transient medium object to the medium map.
- static void **addSolverMedium** (const string &solverKey, **BaseSolver** *const solver)
  Add the default medium object for the solver to the default solver media map.
- static void **changeMedium** (const string &mediumName, const string &libraryName, const string &substanceName, const int &uniqueID)
  Change a medium.
- static void **deleteMedium** (const int &uniqueID)
  Delete a medium from the map.
- static **BaseTwoPhaseMedium** * **medium** (const int &uniqueID)
  Return a pointer to a medium object.
- static **BaseTwoPhaseMedium** * **solverMedium** (**BaseSolver** *const solver)
  Return a pointer to the default medium for the solver.
- static **BaseTwoPhaseMedium** * **solverMedium** (const string &mediumName, const string &libraryName, const string &substanceName)
  Return a pointer to the default medium for the solver.

Static Protected Attributes

- static int **_uniqueID**
  Static counter for the positive unique ID number used by permanent medium objects.
- static int **_transientUniqueID**
  Static counter for the negative unique ID number used by transient medium objects.
- static map< int, **BaseTwoPhaseMedium** * > **_mediums**
  Map for mediums with unique ID as identifier.
- static map< string, **BaseTwoPhaseMedium** * > **_solverMediums**
  *Map for mediums that are used for function calls without specified unique ID such as **saturationTemperature_ ()** (p. 48) and **saturationPressure_ ()** (p. 48).*

5.4.1 Detailed Description

Medium map.

This class is used to hold a static medium map. It stores instances of **TwoPhaseMedium** (p. 38) to avoid the time-consuming recomputation of fluid properties. It uses positive as well as negative unique ID number to uniquely identify the instances. The positive unique ID numbers are used for instances of the Modelica model BaseProperties. They have to be created from within Modelica when the Modelica models are initialized. The negative unique ID numbers are also called transient unique IDs and are used when the BaseProperties record is not used in Modelica. The maximum number of transient unique IDs is called MAX_TRANSIENT_MEDIUM and can be set in **include.h** (p. 65).

Christoph Richter, Francesco Casella, Sep 2006

5.4.2 Member Function Documentation

5.4.2.1 int MediumMap::addMedium (const string & *mediumName*, const string & *libraryName*, const string & *substanceName*) [static]

Add a new medium object to the medium map.

This function adds a new medium object to the medium map and returns its (positive) uniqueID.

Parameters:

mediumName Medium name

libraryName Library name

substanceName Substance name

5.4.2.2 int MediumMap::addTransientMedium (const string & *mediumName*, const string & *libraryName*, const string & *substanceName*) [static]

Add a new transient medium object to the medium map.

This function adds a new transient medium object to the medium map and returns its (negative) uniqueID.

Parameters:

mediumName Medium name

libraryName Library name

substanceName Substance name

5.4.2.3 void MediumMap::addSolverMedium (const string & *solverKey*, BaseSolver *const *solver*) [static]

Add the default medium object for the solver to the default solver media map.

This function adds the default medium object for the specified solver to the default solver media map. The default medium object is used for function calls without a uniqueID (i.e. all functions returning fluid constants, saturationPressure(), saturationTemperature(), etc.).

Parameters:

solverKey Solver key

solver Pointer to solver

5.4.2.4 void MediumMap::changeMedium (const string & *mediumName*, const string & *libraryName*, const string & *substanceName*, const int & *uniqueID*) [static]

Change a medium.

This function replaces the existing pointer to a solver that is part of the medium object with a pointer to another solver.

Parameters:

mediumName Medium name

libraryName Library name

substanceName Substance name

uniqueID uniqueID of the medium object for which the solver should be changed

5.4.2.5 void MediumMap::deleteMedium (const int & *uniqueID*) [static]

Delete a medium from the map.

This function deletes a medium object from the medium map. It can be called when the Modelica simulation is terminated.

Parameters:

uniqueID UniqueID of the medium to be deleted

5.4.2.6 BaseTwoPhaseMedium * MediumMap::medium (const int & *uniqueID*) [static]

Return a pointer to a medium object.

This function returns a pointer to a medium object in the medium map. The advantage of using this function instead of using the medium map directly is that an error reporting mechanism can be implemented that avoids crashed due to calls with an invalid uniqueID.

Parameters:

uniqueID UniqueID of the medium object

5.4.2.7 BaseTwoPhaseMedium * MediumMap::solverMedium (BaseSolver *const *solver*) [static]

Return a pointer to the default medium for the solver.

This function returns a pointer to a medium object in the solver medium map.

Parameters:

solver Pointer to solver

5.4.2.8 BaseTwoPhaseMedium * MediumMap::solverMedium (const string & *mediumName*, const string & *libraryName*, const string & *substanceName*) [static]

Return a pointer to the default medium for the solver.

This function returns a pointer to a medium object in the solver medium map. It is a convenience function if the pointer to the solver is not available. It creates a new solver medium if it does not exist.

Parameters:

mediumName Medium name

libraryName Library name

substanceName Substance name

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

- mediummap.h
- mediummap.cpp

5.5 SolverMap Class Reference

Solver map.

`#include <solvermap.h>`

Static Public Member Functions

- static **BaseSolver** * **getSolver** (const string &mediumName, const string &libraryName, const string &substanceName)

  Get a specific solver.

- static string **solverKey** (const string &libraryName, const string &substanceName)

  Generate a unique solver key.

Static Protected Attributes

- static map< string, **BaseSolver** * > **_solvers**

  Map for all solver instances identified by the SolverKey.

5.5.1 Detailed Description

Solver map.

This class manages the map of all solvers. A solver is a class that inherits from **BaseSolver** (p. 11) and that interfaces the external fluid property computation code. Only one instance is created for each external library.

Christoph Richter, Francesco Casella, Sep 2006

5.5.2 Member Function Documentation

5.5.2.1 BaseSolver * SolverMap::getSolver (const string & *mediumName*, const string & *libraryName*, const string & *substanceName*) [static]

Get a specific solver.

This function returns the solver for the specified library name, substance name and possibly medium name. It creates a new solver if the solver does not already exist. When implementing new solvers, one has to add the newly created solvers to this function. An error message is generated if the specific library is not supported by the interface library.

Parameters:

mediumName Medium name

libraryName Library name

substanceName Substance name

5.5.2.2 string SolverMap::solverKey (const string & *libraryName*, const string & *substanceName*) [static]

Generate a unique solver key.

This function generates a unique solver key based on the library name and substance name.

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

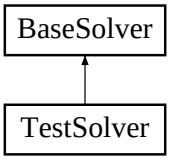
- solvermap.h
- solvermap.cpp

5.6 TestSolver Class Reference

Test solver class.

`#include <testsolver.h>`

Inheritance diagram for TestSolver::



Public Member Functions

- virtual void **setFluidConstants** ()

  Set fluid constants.

- virtual void **setSat_p** (double &p, **TwoPhaseMediumProperties** *const properties)

  Set saturation properties from p.

- virtual void **setSat_T** (double &T, **TwoPhaseMediumProperties** *const properties)

  Set saturation properties from T.

- virtual void **setSat_p_state** (**TwoPhaseMediumProperties** *const properties)

  Set saturation properties from within BaseProperties model.

- virtual void **setState_dT** (double &d, double &T, int &phase, **TwoPhaseMediumProperties** *const properties)

  Set state from d, T, and phase.

- virtual void **setState_ph** (double &p, double &h, int &phase, **TwoPhaseMediumProperties** *const properties)

  Set state from p, h, and phase.

- virtual void **setState_ps** (double &p, double &s, int &phase, **TwoPhaseMediumProperties** *const properties)

  Set state from p, s, and phase.

- virtual void **setState_pT** (double &p, double &T, **TwoPhaseMediumProperties** *const properties)

  Set state from p and T.

5.6.1 Detailed Description

Test solver class.

This class defines a dummy solver object, computing properties of a fluid roughly resembling warm water at low pressure, without the need of any further external code. The class is useful for

debugging purposes, to test whether the C compiler and the Modelica tools are set up correctly before tackling problems with the actual - usually way more complex - external code. It is *not* meant to be used as an actual fluid model for any real application.

To keep complexity down to the absolute medium, the current version of the solver can only compute the fluid properties in the liquid phase region: 1e5 Pa < p < 2e5 Pa 300 K < T < 350 K ; results returned with inputs outside that range (possibly corresponding to two-phase or vapour points) are not reliable. Saturation properies are computed in the range 1e5 Pa < psat < 2e5 Pa ; results obtained outside that range might be unrealistic.

To instantiate this solver, it is necessary to set the library name package constant in Modelica as follows:

libraryName = "TestMedium";

Francesco Casella, Christoph Richter, Oct 2006

5.6.2 Member Function Documentation

5.6.2.1 void TestSolver::setFluidConstants () [virtual]

Set fluid constants.

This function sets the fluid constants which are defined in the **FluidConstants** (p. 27) record in Modelica. It should be called when a new solver is created.

Must be re-implemented in the specific solver

Reimplemented from **BaseSolver** (p. 14).

5.6.2.2 void TestSolver::setSat_p (double & *p*, TwoPhaseMediumProperties *const *properties*) [virtual]

Set saturation properties from p.

This function sets the saturation properties for the given pressure p. The computed values are written to the two phase medium propery struct.

Must be re-implemented in the specific solver

Parameters:

p Pressure

properties Two phase medium property record

Reimplemented from **BaseSolver** (p. 14).

5.6.2.3 void TestSolver::setSat_T (double & *T*, TwoPhaseMediumProperties *const *properties*) [virtual]

Set saturation properties from T.

This function sets the saturation properties for the given temperature T. The computed values are written to the two phase medium propery struct.

Must be re-implemented in the specific solver

Parameters:

T Temperature

properties Two phase medium property record

Reimplemented from **BaseSolver** (p. 14).

5.6.2.4 void TestSolver::setSat_p_state (TwoPhaseMediumProperties *const *properties*) [virtual]

Set saturation properties from within BaseProperties model.

This function sets the saturation properties for the given pressure p and is desined to be used from within the BaseProperties model in Modelica. The computed values are written to the two phase medium propery struct.

Must be re-implemented in the specific solver

Parameters:

properties Two phase medium property record

Reimplemented from **BaseSolver** (p. 14).

5.6.2.5 void TestSolver::setState_dT (double & *d*, double & *T*, int & *phase*, TwoPhaseMediumProperties *const *properties*) [virtual]

Set state from d, T, and phase.

This function sets the thermodynamic state record for the given density d, the temperature T and the specified phase. The computed values are written to the two phase medium property struct.

Must be re-implemented in the specific solver

Parameters:

d Density

T Temperature

phase Phase (2 for two-phase, 1 for one-phase, 0 if not known)

properties Two phase medium property record

Reimplemented from **BaseSolver** (p. 15).

5.6.2.6 void TestSolver::setState_ph (double & *p*, double & *h*, int & *phase*, TwoPhaseMediumProperties *const *properties*) [virtual]

Set state from p, h, and phase.

This function sets the thermodynamic state record for the given pressure p, the specific enthalpy h and the specified phase. The computed values are written to the two phase medium property struct.

Must be re-implemented in the specific solver

Parameters:

p Pressure

h Specific enthalpy

phase Phase (2 for two-phase, 1 for one-phase, 0 if not known)

properties Two phase medium property record

Reimplemented from **BaseSolver** (p. 15).

5.6.2.7 void TestSolver::setState_ps (double & *p*, double & *s*, int & *phase*, TwoPhaseMediumProperties *const *properties*) [virtual]

Set state from p, s, and phase.

This function sets the thermodynamic state record for the given pressure p, the specific entropy s and the specified phase. The computed values are written to the two phase medium property struct.

Must be re-implemented in the specific solver

Parameters:

p Pressure

s Specific entropy

phase Phase (2 for two-phase, 1 for one-phase, 0 if not known)

properties Two phase medium property record

Reimplemented from **BaseSolver** (p. 15).

5.6.2.8 void TestSolver::setState_pT (double & *p*, double & *T*, TwoPhaseMediumProperties *const *properties*) [virtual]

Set state from p and T.

This function sets the thermodynamic state record for the given pressure p and the temperature T. The computed values are written to the two phase medium property struct.

Must be re-implemented in the specific solver

Parameters:

p Pressure

T Temperature

properties Two phase medium property record

Reimplemented from **BaseSolver** (p. 16).

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

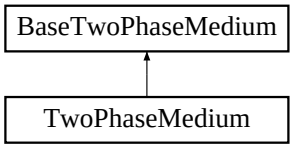
- testsolver.h
- testsolver.cpp

5.7 TwoPhaseMedium Class Reference

Two phase medium.

`#include <twophasemedium.h>`

Inheritance diagram for TwoPhaseMedium::



Public Member Functions

- **TwoPhaseMedium** (const string &mediumName, const string &libraryName, const string &substanceName, **BaseSolver** *const solver, const int &uniqueID)

  Constructor.

- ~**TwoPhaseMedium** ()

  Destructor.

5.7.1 Detailed Description

Two phase medium.

This class is the default object embedding the fluid property computations at a given point of the plant.

Christoph Richter, Francesco Casella, Sep 2006

5.7.2 Constructor & Destructor Documentation

5.7.2.1 TwoPhaseMedium::TwoPhaseMedium (const string & *mediumName*, const string & *libraryName*, const string & *substanceName*, BaseSolver *const *solver*, const int & *uniqueID*)

Constructor.

This constructor is passing the argument to the constructor of the base class and is creating a the property container.

Parameters:

mediumName Medium name

libraryName Library name

substanceName Substance name

solver Solver

uniqueID Unique ID number

5.7.2.2 TwoPhaseMedium::~TwoPhaseMedium ()

Destructor.

The destructor is deleting the property container.

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

- twophasemedium.h
- twophasemedium.cpp

5.8 TwoPhaseMediumProperties Class Reference

Two phase medium property class.

`#include <twophasemediumproperties.h>`

Public Member Functions

- **TwoPhaseMediumProperties** ()
- void **initializeFields** ()

Public Attributes

- int **phase**

  Phase.

- double **beta**

  Isothermal expansion coefficient.

- double **cp**

  Specific heat capacity cp.

- double **cv**

  Specific heat capacity cv.

- double **d**

  Density.

- double **dd_dp_h**

  Derivative of density wrt pressure at constant enthalpy.

- double **dd_dh_p**

  Derivative of density wrt enthalpy at constant pressure.

- double **h**

  Specific enthalpy.

- double **kappa**

  Compressibility.

- double **p**

  Pressure.

- double **s**

  Specific entropy.

- double **T**

  Temperature.

- double **dT_dp_h**

Derivative of temperature wrt pressure at constant enthalpy.

- double **dT_dh_p**

  Derivative of temperature wrt enthalpy at constant pressure.

- double **ps**

  Saturation pressure.

- double **Ts**

  Saturation temperature.

- double **dl**

  Density at bubble line (for pressure ps).

- double **dv**

  Density at dew line (for pressure ps).

- double **hl**

  Specific enthalpy at bubble line (for pressure ps).

- double **hv**

  Specific enthalpy at dew line (for pressure ps).

- double **sl**

  Specific entropy at bubble line (for pressure ps).

- double **sv**

  Specific entropy at dew line (for pressure ps).

- double **eta**

  Dynamic viscosity.

- double **lambda**

  Thermal conductivity.

- double **Pr**

  Prandtl number.

- double **sigma**

  Surface tension.

- double **d_Ts_dp**

  Derivative of Ts wrt pressure.

- double **d_dl_dp**

  Derivative of dls wrt pressure.

- double **d_dv_dp**

  Derivative of dvs wrt pressure.

- double **d_hl_dp**

  Derivative of hls wrt pressure.

- double **d_hv_dp**

  Derivative of hvs wrt pressure.

5.8.1 Detailed Description

Two phase medium property class.

The two phase medium propery struct defines all the properties that are computed by external Modelica medium models extending from PartialExternalTwoPhaseMedium. It also contains the saturation properties for the dew and the bubble line which makes it a real two phase medium property struct.

Francesco Casella, Christoph Richter Sep 2006

5.8.2 Constructor & Destructor Documentation

5.8.2.1 TwoPhaseMediumProperties::TwoPhaseMediumProperties ()

The constructor is calling initializeFields.

5.8.3 Member Function Documentation

5.8.3.1 void TwoPhaseMediumProperties::initializeFields ()

This function initializes all fields of **TwoPhaseMediumProperties** (p. 40) to the NAN value defined in **include.h** (p. 65). The phase is initialized to zero.

5.8.4 Member Data Documentation

5.8.4.1 int TwoPhaseMediumProperties::phase

Phase.

This phase flag is defined according to the phase flag in Modelica.Media: 2 for two-phase, 1 for one-phase.

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

- twophasemediumproperties.h
- twophasemediumproperties.cpp

Chapter 6

ExternalMedia File Documentation

6.1 errorhandling.h File Reference

Error handling for external library.

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

```
#include "ModelicaUtilities.h"
```

Functions

- void **errorMessage** (char *errorMessage)
  Function to display error message.
- void **warningMessage** (char *warningMessage)
  Function to display warning message.

6.1.1 Detailed Description

Error handling for external library.

Errors in the external fluid property library have to be reported to the Modelica layer. This class defines the required interface functions.

Francesco Casella, Christoph Richter Nov 2006

6.1.2 Function Documentation

6.1.2.1 void errorMessage (char * *errorMessage*)

Function to display error message.

Calling this function will display the specified error message and will terminate the simulation.

Parameters:

errorMessage Error message to be displayed

6.1.2.2 void warningMessage (char * *warningMessage*)

Function to display warning message.

Calling this function will display the specified warning message.

Parameters:

warningMessage Warning message to be displayed

6.2 externaltwophasemedium.cpp File Reference

Interface layer.

```
#include "externaltwophasemedium.h"
#include "mediummap.h"
#include "twophasemedium.h"
```

Functions

- int **createMedium_** (const char *mediumName, const char *libraryName, const char *substanceName, int oldUniqueID)

  Create medium.

- double **getMolarMass_** (const char *mediumName, const char *libraryName, const char *substanceName)

  Get molar mass.

- double **getCriticalTemperature_** (const char *mediumName, const char *libraryName, const char *substanceName)

  Get critical temperature.

- double **getCriticalPressure_** (const char *mediumName, const char *libraryName, const char *substanceName)

  Get critical pressure.

- double **getCriticalMolarVolume_** (const char *mediumName, const char *libraryName, const char *substanceName)

  Get critical molar volume.

- void **setState_dT_** (double d, double T, int phase, int uniqueID, int *state_uniqueID, int *state_phase, double *state_d, double *state_h, double *state_p, double *state_s, double *state_T, const char *mediumName, const char *libraryName, const char *substanceName)

  Compute properties from d, T, and phase.

- void **setState_ph_** (double p, double h, int phase, int uniqueID, int *state_uniqueID, int *state_phase, double *state_d, double *state_h, double *state_p, double *state_s, double *state_T, const char *mediumName, const char *libraryName, const char *substanceName)

  Compute properties from p, h, and phase.

- void **setState_ps_** (double p, double s, int phase, int uniqueID, int *state_uniqueID, int *state_phase, double *state_d, double *state_h, double *state_p, double *state_s, double *state_T, const char *mediumName, const char *libraryName, const char *substanceName)

  Compute properties from p, s, and phase.

- void **setState_pT_** (double p, double T, int phase, int uniqueID, int *state_uniqueID, int *state_phase, double *state_d, double *state_h, double *state_p, double *state_s, double *state_T, const char *mediumName, const char *libraryName, const char *substanceName)

  Compute properties from p, T, and phase.

- void **setSat_p_** (double p, int uniqueID, double *sat_psat, double *sat_Tsat, int *sat_uniqueID, const char *mediumName, const char *libraryName, const char *substanceName)

  Compute saturation properties from p.

- void **setSat_T_** (double T, int uniqueID, double *sat_psat, double *sat_Tsat, int *sat_uniqueID, const char *mediumName, const char *libraryName, const char *substanceName)

  Compute saturation properties from T.

- void **setSat_p_state_** (int uniqueID, double *sat_psat, double *sat_Tsat, int *sat_uniqueID)

  Compute saturation properties from within BaseProperties.

- void **setDewState_** (int uniqueID, int phase, int *state_uniqueID, int *state_phase, const char *mediumName, const char *libraryName, const char *substanceName)

  Compute dew state.

- void **setBubbleState_** (int uniqueID, int phase, int *state_uniqueID, int *state_phase, const char *mediumName, const char *libraryName, const char *substanceName)

  Compute bubble state.

- double **density_** (int uniqueID)

  Return density of specified medium.

- double **density_derp_h_** (int uniqueID)

  Return derivative of density wrt pressure at constant specific enthalpy of specified medium.

- double **density_derh_p_** (int uniqueID)

  Return derivative of density wrt specific enthalpy at constant pressure of specified medium.

- double **density_ph_der_** (int uniqueID, double p_der, double h_der)

  Return derivative of density wrt pressure and specific enthalpy of specified medium.

- double **pressure_** (int uniqueID)

  Return pressure of specified medium.

- double **specificEnthalpy_** (int uniqueID)

  Return specific enthalpy of specified medium.

- double **specificEntropy_** (int uniqueID)

  Return specific entropy of specified medium.

- double **temperature_** (int uniqueID)

  Return temperature of specified medium.

- double **temperature_ph_der_** (int uniqueID, double p_der, double h_der)

  Return derivative of temperature wrt pressure and specific enthalpy of specified medium.

- void **isentropicEnthalpy_** (double p, int uniqueID, double *h_is)

  Return the enthalpy at pressure p after an isentropic transformation form the specified medium state.

- double **saturationTemperature_derp_sat_** (double psat, double Tsat, int uniqueID, const char *mediumName, const char *libraryName, const char *substanceName)

  Return derivative of saturation temperature of specified medium from saturation properties.

- double **bubbleDensity_** (double psat, double Tsat, int uniqueID, const char *mediumName, const char *libraryName, const char *substanceName)

  Return bubble density of specified medium from saturation properties.

- double **dewDensity_** (double psat, double Tsat, int uniqueID, const char *mediumName, const char *libraryName, const char *substanceName)

  Return dew density of specified medium from saturation properties.

- double **bubbleEnthalpy_** (double psat, double Tsat, int uniqueID, const char *mediumName, const char *libraryName, const char *substanceName)

  Return bubble specific enthalpy of specified medium from saturation properties.

- double **dewEnthalpy_** (double psat, double Tsat, int uniqueID, const char *mediumName, const char *libraryName, const char *substanceName)

  Return dew specific enthalpy of specified medium from saturation properties.

- double **bubbleEntropy_** (double psat, double Tsat, int uniqueID, const char *mediumName, const char *libraryName, const char *substanceName)

  Return bubble specific entropy of specified medium from saturation properties.

- double **dewEntropy_** (double psat, double Tsat, int uniqueID, const char *mediumName, const char *libraryName, const char *substanceName)

  Return dew specific entropy of specified medium from saturation properties.

- double **dBubbleDensity_dPressure_** (double psat, double Tsat, int uniqueID, const char *mediumName, const char *libraryName, const char *substanceName)

  Return derivative of bubble density wrt pressure of specified medium from saturation properties.

- double **dDewDensity_dPressure_** (double psat, double Tsat, int uniqueID, const char *mediumName, const char *libraryName, const char *substanceName)

  Return derivative of dew density wrt pressure of specified medium from saturation properties.

- double **dBubbleEnthalpy_dPressure_** (double psat, double Tsat, int uniqueID, const char *mediumName, const char *libraryName, const char *substanceName)

  Return derivative of bubble specific enthalpy wrt pressure of specified medium from saturation properties.

- double **dDewEnthalpy_dPressure_** (double psat, double Tsat, int uniqueID, const char *mediumName, const char *libraryName, const char *substanceName)

Return derivative of dew specific enthalpy wrt pressure of specified medium from saturation properties.

- double **isobaricExpansionCoefficient_** (int uniqueID)
  Return isobaric expansion coefficient of specified medium.
- double **isothermalCompressibility_** (int uniqueID)
  Return isothermal compressibility of specified medium.
- double **specificHeatCapacityCp_** (int uniqueID)
  Return specific heat capacity cp of specified medium.
- double **specificHeatCapacityCv_** (int uniqueID)
  Return specific heat capacity cv of specified medium.
- double **dynamicViscosity_** (int uniqueID)
  Return dynamic viscosity of specified medium.
- double **thermalConductivity_** (int uniqueID)
  Return thermal conductivity of specified medium.
- double **prandtlNumber_** (int uniqueID)
  Return Prandtl number of specified medium.
- double **surfaceTension_** (double psat, double Tsat, int uniqueID)
  Return surface tension of specified medium.
- double **dDensity_dPressure_h_** (int uniqueID)
  Return derivative of density wrt pressure at constant specific enthalpy of specified medium.
- double **dDensity_dEnthalpy_p_** (int uniqueID)
  Return derivative of density wrt specific enthalpy at constant pressure of specified medium.
- double **saturationPressure_** (double T, const char *mediumName, const char *libraryName, const char *substanceName)
  Compute saturation pressure for specified medium and temperature.
- double **saturationTemperature_** (double p, const char *mediumName, const char *libraryName, const char *substanceName)
  Compute saturation temperature for specified medium and pressure.
- double **saturationTemperature_derp_** (double p, const char *mediumName, const char *libraryName, const char *substanceName)
  Compute derivative of saturation temperature for specified medium and pressure.

6.2.1 Detailed Description

Interface layer.

C/C++ layer for external medium models extending from PartialExternalTwoPhaseMedium.

Francesco Casella, Christoph Richter, Sep 2006

6.2.2 Function Documentation

6.2.2.1 int createMedium_ (const char * *mediumName*, const char * *libraryName*, const char * *substanceName*, int *oldUniqueID*)

Create medium.

This function creates a new medium with the specified medium name, library name, and substance name. The old unique ID is required to check whether a medium has already been created.

Parameters:

mediumName Medium name

libraryName Library name

substanceName Substance name

oldUniqueID Old unique ID number

6.2.2.2 double getCriticalMolarVolume_ (const char * *mediumName*, const char * *libraryName*, const char * *substanceName*)

Get critical molar volume.

This function returns the critical molar volume of the specified medium.

Parameters:

mediumName Medium name

libraryName Library name

substanceName Substance name

6.2.2.3 double getCriticalPressure_ (const char * *mediumName*, const char * *libraryName*, const char * *substanceName*)

Get critical pressure.

This function returns the critical pressure of the specified medium.

Parameters:

mediumName Medium name

libraryName Library name

substanceName Substance name

6.2.2.4 double getCriticalTemperature_ (const char * *mediumName*, const char * *libraryName*, const char * *substanceName*)

Get critical temperature.

This function returns the critical temperature of the specified medium.

Parameters:

mediumName Medium name

libraryName Library name

substanceName Substance name

6.2.2.5 double getMolarMass_ (const char * *mediumName*, const char * *libraryName*, const char * *substanceName*)

Get molar mass.

This function returns the molar mass of the specified medium.

Parameters:

mediumName Medium name

libraryName Library name

substanceName Substance name

6.2.2.6 void setBubbleState_ (int *uniqueID*, int *phase*, int * *state_uniqueID*, int * *state_phase*, const char * *mediumName*, const char * *libraryName*, const char * *substanceName*)

Compute bubble state.

This function computes the bubble state for the specified medium.

Parameters:

uniqueID Unique ID number

phase Phase (2 for two-phase, 1 for one-phase, 0 if not known)

state_uniqueID Pointer to return unique ID number for state record

state_phase Pointer to return phase for state record

mediumName Medium name

libraryName Library name

substanceName Substance name

6.2.2.7 void setDewState_ (int *uniqueID*, int *phase*, int * *state_uniqueID*, int * *state_phase*, const char * *mediumName*, const char * *libraryName*, const char * *substanceName*)

Compute dew state.

This function computes the dew state for the specified medium.

Parameters:

uniqueID Unique ID number

phase Phase (2 for two-phase, 1 for one-phase, 0 if not known)

state_uniqueID Pointer to return unique ID number for state record

state_phase Pointer to return phase for state record

mediumName Medium name

libraryName Library name

substanceName Substance name

6.2.2.8 void setSat_p_ (double *p*, int *uniqueID*, double * *sat_psat*, double * *sat_Tsat*, int * *sat_uniqueID*, const char * *mediumName*, const char * *libraryName*, const char * *substanceName*)

Compute saturation properties from p.

This function computes the saturation properties for the specified inputs. If the function is called with uniqueID == 0 a new transient unique ID is assigned and the function is called again with this new transient unique ID number.

Parameters:

p Pressure

uniqueID Unique ID number

sat_psat Pointer to return pressure for saturation record

sat_Tsat Pointer to return temperature for saturation record

sat_uniqueID Pointer to return unique ID number for saturation record

mediumName Medium name

libraryName Library name

substanceName Substance name

6.2.2.9 void setSat_p_state_ (int *uniqueID*, double * *sat_psat*, double * *sat_Tsat*, int * *sat_uniqueID*)

Compute saturation properties from within BaseProperties.

This function computes the saturation properties and is designed to be called from within the BaseProperties model. The saturation properties are set according to the medium pressure

Parameters:

uniqueID Unique ID number

sat_psat Pointer to return pressure for saturation record

sat_Tsat Pointer to return temperature for saturation record

sat_uniqueID Pointer to return unique ID number for saturation record

6.2.2.10 void setSat_T_ (double *T*, int *uniqueID*, double * *sat_psat*, double * *sat_Tsat*, int * *sat_uniqueID*, const char * *mediumName*, const char * *libraryName*, const char * *substanceName*)

Compute saturation properties from T.

This function computes the saturation properties for the specified inputs. If the function is called with uniqueID == 0 a new transient unique ID is assigned and the function is called again with this new transient unique ID number.

Parameters:

T Temperature

uniqueID Unique ID number

sat_psat Pointer to return pressure for saturation record

sat_ Tsat Pointer to return temperature for saturation record

sat_ uniqueID Pointer to return unique ID number for saturation record

mediumName Medium name

libraryName Library name

substanceName Substance name

6.2.2.11 void setState_dT_ (double *d*, double *T*, int *phase*, int *uniqueID*, int * *state_ uniqueID*, int * *state_ phase*, double * *state_ d*, double * *state_ h*, double * *state_ p*, double * *state_ s*, double * *state_ T*, const char * *mediumName*, const char * *libraryName*, const char * *substanceName*)

Compute properties from d, T, and phase.

This function computes the properties for the specified inputs. If the function is called with uniqueID == 0 a new transient unique ID is assigned and the function is called again with this new transient unique ID number.

Parameters:

d Density

T Temperature

phase Phase (2 for two-phase, 1 for one-phase, 0 if not known)

uniqueID Unique ID number

state_ uniqueID Pointer to return unique ID number for state record

state_ phase Pointer to return phase for state record

state_ d Pointer to return density for state record

state_ h Pointer to return specific enthalpy for state record

state_ p Pointer to return pressure for state record

state_ s Pointer to return specific entropy for state record

state_ T Pointer to return temperature for state record

mediumName Medium name

libraryName Library name

substanceName Substance name

6.2.2.12 void setState_ph_ (double *p*, double *h*, int *phase*, int *uniqueID*, int * *state_ uniqueID*, int * *state_ phase*, double * *state_ d*, double * *state_ h*, double * *state_ p*, double * *state_ s*, double * *state_ T*, const char * *mediumName*, const char * *libraryName*, const char * *substanceName*)

Compute properties from p, h, and phase.

This function computes the properties for the specified inputs. If the function is called with uniqueID == 0 a new transient unique ID is assigned and the function is called again with this new transient unique ID number.

Parameters:

p Pressure

h Specific enthalpy

phase Phase (2 for two-phase, 1 for one-phase, 0 if not known)

uniqueID Unique ID number

state_uniqueID Pointer to return unique ID number for state record

state_phase Pointer to return phase for state record

state_d Pointer to return density for state record

state_h Pointer to return specific enthalpy for state record

state_p Pointer to return pressure for state record

state_s Pointer to return specific entropy for state record

state_T Pointer to return temperature for state record

mediumName Medium name

libraryName Library name

substanceName Substance name

6.2.2.13 void setState_ps_ (double *p*, double *s*, int *phase*, int *uniqueID*, int * *state_uniqueID*, int * *state_phase*, double * *state_d*, double * *state_h*, double * *state_p*, double * *state_s*, double * *state_T*, const char * *mediumName*, const char * *libraryName*, const char * *substanceName*)

Compute properties from p, s, and phase.

This function computes the properties for the specified inputs. If the function is called with uniqueID == 0 a new transient unique ID is assigned and the function is called again with this new transient unique ID number.

Parameters:

p Pressure

s Specific entropy

phase Phase (2 for two-phase, 1 for one-phase, 0 if not known)

uniqueID Unique ID number

state_uniqueID Pointer to return unique ID number for state record

state_phase Pointer to return phase for state record

state_d Pointer to return density for state record

state_h Pointer to return specific enthalpy for state record

state_p Pointer to return pressure for state record

state_s Pointer to return specific entropy for state record

state_T Pointer to return temperature for state record

mediumName Medium name

libraryName Library name

substanceName Substance name

6.2.2.14 void setState_pT_ (double *p*, double *T*, int *phase*, int *uniqueID*, int * *state_uniqueID*, int * *state_phase*, double * *state_d*, double * *state_h*, double * *state_p*, double * *state_s*, double * *state_T*, const char * *mediumName*, const char * *libraryName*, const char * *substanceName*)

Compute properties from p, T, and phase.

This function computes the properties for the specified inputs. If the function is called with uniqueID == 0 a new transient unique ID is assigned and the function is called again with this new transient unique ID number.

Attention: The phase input is ignored for this function!

Parameters:

p Pressure

T Temperature

phase Phase (2 for two-phase, 1 for one-phase, 0 if not known)

uniqueID Unique ID number

state_uniqueID Pointer to return unique ID number for state record

state_phase Pointer to return phase for state record

state_d Pointer to return density for state record

state_h Pointer to return specific enthalpy for state record

state_p Pointer to return pressure for state record

state_s Pointer to return specific entropy for state record

state_T Pointer to return temperature for state record

mediumName Medium name

libraryName Library name

substanceName Substance name

6.3 externaltwophasemedium.h File Reference

Header file to be included in Dymola.

Functions

- EXPORT int **createMedium_** (const char *mediumName, const char *libraryName, const char *substanceName, int oldUniqueID)

  Create medium.

- EXPORT double **getMolarMass_** (const char *mediumName, const char *libraryName, const char *substanceName)

  Get molar mass.

- EXPORT double **getCriticalTemperature_** (const char *mediumName, const char *libraryName, const char *substanceName)

  Get critical temperature.

- EXPORT double **getCriticalPressure_** (const char *mediumName, const char *libraryName, const char *substanceName)

  Get critical pressure.

- EXPORT double **getCriticalMolarVolume_** (const char *mediumName, const char *libraryName, const char *substanceName)

  Get critical molar volume.

- EXPORT void **setSat_p_** (double p, int uniqueID, double *sat_psat, double *sat_Tsat, int *sat_uniqueID, const char *mediumName, const char *libraryName, const char *substanceName)

  Compute saturation properties from p.

- EXPORT void **setSat_T_** (double T, int uniqueID, double *sat_psat, double *sat_Tsat, int *sat_uniqueID, const char *mediumName, const char *libraryName, const char *substanceName)

  Compute saturation properties from T.

- EXPORT void **setSat_p_state_** (int uniqueID, double *sat_psat, double *sat_Tsat, int *sat_uniqueID)

  Compute saturation properties from within BaseProperties.

- EXPORT void **setDewState_** (int uniqueID, int phase, int *state_uniqueID, int *state_phase, const char *mediumName, const char *libraryName, const char *substanceName)

  Compute dew state.

- EXPORT void **setBubbleState_** (int uniqueID, int phase, int *state_uniqueID, int *state_phase, const char *mediumName, const char *libraryName, const char *substanceName)

  Compute bubble state.

- EXPORT double **saturationPressure_** (double T, const char *mediumName, const char *libraryName, const char *substanceName)

Compute saturation pressure for specified medium and temperature.

- EXPORT double **saturationTemperature_** (double p, const char *mediumName, const char *libraryName, const char *substanceName)

  Compute saturation temperature for specified medium and pressure.

- EXPORT double **saturationTemperature_derp_sat_** (double psat, double Tsat, int uniqueID, const char *mediumName, const char *libraryName, const char *substanceName)

  Return derivative of saturation temperature of specified medium from saturation properties.

- EXPORT double **bubbleDensity_** (double psat, double Tsat, int uniqueID, const char *mediumName, const char *libraryName, const char *substanceName)

  Return bubble density of specified medium from saturation properties.

- EXPORT double **dewDensity_** (double psat, double Tsat, int uniqueID, const char *mediumName, const char *libraryName, const char *substanceName)

  Return dew density of specified medium from saturation properties.

- EXPORT double **bubbleEnthalpy_** (double psat, double Tsat, int uniqueID, const char *mediumName, const char *libraryName, const char *substanceName)

  Return bubble specific enthalpy of specified medium from saturation properties.

- EXPORT double **dewEnthalpy_** (double psat, double Tsat, int uniqueID, const char *mediumName, const char *libraryName, const char *substanceName)

  Return dew specific enthalpy of specified medium from saturation properties.

- EXPORT double **bubbleEntropy_** (double psat, double Tsat, int uniqueID, const char *mediumName, const char *libraryName, const char *substanceName)

  Return bubble specific entropy of specified medium from saturation properties.

- EXPORT double **dewEntropy_** (double psat, double Tsat, int uniqueID, const char *mediumName, const char *libraryName, const char *substanceName)

  Return dew specific entropy of specified medium from saturation properties.

- EXPORT double **dBubbleDensity_dPressure_** (double psat, double Tsat, int uniqueID, const char *mediumName, const char *libraryName, const char *substanceName)

  Return derivative of bubble density wrt pressure of specified medium from saturation properties.

- EXPORT double **dDewDensity_dPressure_** (double psat, double Tsat, int uniqueID, const char *mediumName, const char *libraryName, const char *substanceName)

  Return derivative of dew density wrt pressure of specified medium from saturation properties.

- EXPORT double **dBubbleEnthalpy_dPressure_** (double psat, double Tsat, int uniqueID, const char *mediumName, const char *libraryName, const char *substanceName)

  Return derivative of bubble specific enthalpy wrt pressure of specified medium from saturation properties.

- EXPORT double **dDewEnthalpy_dPressure_** (double psat, double Tsat, int uniqueID, const char *mediumName, const char *libraryName, const char *substanceName)

  Return derivative of dew specific enthalpy wrt pressure of specified medium from saturation properties.

- EXPORT void **setState_dT_** (double d, double T, int phase, int uniqueID, int *state_uniqueID, int *state_phase, double *state_d, double *state_h, double *state_p, double *state_s, double *state_T, const char *mediumName, const char *libraryName, const char *substanceName)

  Compute properties from d, T, and phase.

- EXPORT void **setState_ph_** (double p, double h, int phase, int uniqueID, int *state_uniqueID, int *state_phase, double *state_d, double *state_h, double *state_p, double *state_s, double *state_T, const char *mediumName, const char *libraryName, const char *substanceName)

  Compute properties from p, h, and phase.

- EXPORT void **setState_ps_** (double p, double s, int phase, int uniqueID, int *state_uniqueID, int *state_phase, double *state_d, double *state_h, double *state_p, double *state_s, double *state_T, const char *mediumName, const char *libraryName, const char *substanceName)

  Compute properties from p, s, and phase.

- EXPORT void **setState_pT_** (double p, double T, int phase, int uniqueID, int *state_uniqueID, int *state_phase, double *state_d, double *state_h, double *state_p, double *state_s, double *state_T, const char *mediumName, const char *libraryName, const char *substanceName)

  Compute properties from p, T, and phase.

- EXPORT double **density_** (int uniqueID)

  Return density of specified medium.

- EXPORT double **density_ph_der_** (int uniqueID, double p_der, double h_der)

  Return derivative of density wrt pressure and specific enthalpy of specified medium.

- EXPORT double **density_derp_h_** (int uniqueID)

  Return derivative of density wrt pressure at constant specific enthalpy of specified medium.

- EXPORT double **density_derh_p_** (int uniqueID)

  Return derivative of density wrt specific enthalpy at constant pressure of specified medium.

- EXPORT double **pressure_** (int uniqueID)

  Return pressure of specified medium.

- EXPORT double **specificEnthalpy_** (int uniqueID)

  Return specific enthalpy of specified medium.

- EXPORT double **specificEntropy_** (int uniqueID)

  Return specific entropy of specified medium.

- EXPORT double **temperature_** (int uniqueID)

  Return temperature of specified medium.

- EXPORT double **isobaricExpansionCoefficient_** (int uniqueID)

  Return isobaric expansion coefficient of specified medium.

- EXPORT double **isothermalCompressibility_** (int uniqueID)
  Return isothermal compressibility of specified medium.
- EXPORT double **specificHeatCapacityCp_** (int uniqueID)
  Return specific heat capacity cp of specified medium.
- EXPORT double **specificHeatCapacityCv_** (int uniqueID)
  Return specific heat capacity cv of specified medium.
- EXPORT double **dynamicViscosity_** (int uniqueID)
  Return dynamic viscosity of specified medium.
- EXPORT double **thermalConductivity_** (int uniqueID)
  Return thermal conductivity of specified medium.
- EXPORT double **prandtlNumber_** (int uniqueID)
  Return Prandtl number of specified medium.
- EXPORT double **surfaceTension_** (double psat, double Tsat, int uniqueID)
  Return surface tension of specified medium.
- EXPORT double **dDensity_dPressure_h_** (int uniqueID)
  Return derivative of density wrt pressure at constant specific enthalpy of specified medium.
- EXPORT double **dDensity_dEnthalpy_p_** (int uniqueID)
  Return derivative of density wrt specific enthalpy at constant pressure of specified medium.
- EXPORT double **temperature_ph_der_** (int uniqueID, double p_der, double h_der)
  Return derivative of temperature wrt pressure and specific enthalpy of specified medium.

6.3.1 Detailed Description

Header file to be included in Dymola.

This is the header file to be included in the Dymola/Source directory. It proived function prototypes for all the external functions needed by medium models extending from PartialExternalTwoPhaseMedium.

Please be aware that other Modelica tools might require a slightly different header files depending on the compiler.

Francesco Casella, Christoph Richter Sep 2006

6.3.2 Function Documentation

6.3.2.1 EXPORT int createMedium_ (const char * *mediumName*, const char * *libraryName*, const char * *substanceName*, int *oldUniqueID*)

Create medium.

This function creates a new medium with the specified medium name, library name, and substance name. The old unique ID is required to check whether a medium has already been created.

Parameters:

mediumName Medium name

libraryName Library name

substanceName Substance name

oldUniqueID Old unique ID number

6.3.2.2 EXPORT double getCriticalMolarVolume_ (const char * *mediumName*, const char * *libraryName*, const char * *substanceName*)

Get critical molar volume.

This function returns the critical molar volume of the specified medium.

Parameters:

mediumName Medium name

libraryName Library name

substanceName Substance name

6.3.2.3 EXPORT double getCriticalPressure_ (const char * *mediumName*, const char * *libraryName*, const char * *substanceName*)

Get critical pressure.

This function returns the critical pressure of the specified medium.

Parameters:

mediumName Medium name

libraryName Library name

substanceName Substance name

6.3.2.4 EXPORT double getCriticalTemperature_ (const char * *mediumName*, const char * *libraryName*, const char * *substanceName*)

Get critical temperature.

This function returns the critical temperature of the specified medium.

Parameters:

mediumName Medium name

libraryName Library name

substanceName Substance name

6.3.2.5 EXPORT double getMolarMass_ (const char * *mediumName*, const char * *libraryName*, const char * *substanceName*)

Get molar mass.

This function returns the molar mass of the specified medium.

Parameters:

mediumName Medium name

libraryName Library name

substanceName Substance name

6.3.2.6 EXPORT void setBubbleState_ (int *uniqueID*, int *phase*, int * *state_uniqueID*, int * *state_phase*, const char * *mediumName*, const char * *libraryName*, const char * *substanceName*)

Compute bubble state.

This function computes the bubble state for the specified medium.

Parameters:

uniqueID Unique ID number

phase Phase (2 for two-phase, 1 for one-phase, 0 if not known)

state_uniqueID Pointer to return unique ID number for state record

state_phase Pointer to return phase for state record

mediumName Medium name

libraryName Library name

substanceName Substance name

6.3.2.7 EXPORT void setDewState_ (int *uniqueID*, int *phase*, int * *state_uniqueID*, int * *state_phase*, const char * *mediumName*, const char * *libraryName*, const char * *substanceName*)

Compute dew state.

This function computes the dew state for the specified medium.

Parameters:

uniqueID Unique ID number

phase Phase (2 for two-phase, 1 for one-phase, 0 if not known)

state_uniqueID Pointer to return unique ID number for state record

state_phase Pointer to return phase for state record

mediumName Medium name

libraryName Library name

substanceName Substance name

6.3.2.8 EXPORT void setSat_p_ (double *p*, int *uniqueID*, double * *sat_psat*, double * *sat_Tsat*, int * *sat_uniqueID*, const char * *mediumName*, const char * *libraryName*, const char * *substanceName*)

Compute saturation properties from p.

This function computes the saturation properties for the specified inputs. If the function is called with uniqueID == 0 a new transient unique ID is assigned and the function is called again with this new transient unique ID number.

Parameters:

p Pressure

uniqueID Unique ID number

sat_psat Pointer to return pressure for saturation record

sat_Tsat Pointer to return temperature for saturation record

sat_uniqueID Pointer to return unique ID number for saturation record

mediumName Medium name

libraryName Library name

substanceName Substance name

6.3.2.9 EXPORT void setSat_p_state_ (int *uniqueID*, double * *sat_psat*, double * *sat_Tsat*, int * *sat_uniqueID*)

Compute saturation properties from within BaseProperties.

This function computes the saturation properties and is designed to be called from within the BaseProperties model. The saturation properties are set according to the medium pressure

Parameters:

uniqueID Unique ID number

sat_psat Pointer to return pressure for saturation record

sat_Tsat Pointer to return temperature for saturation record

sat_uniqueID Pointer to return unique ID number for saturation record

6.3.2.10 EXPORT void setSat_T_ (double *T*, int *uniqueID*, double * *sat_psat*, double * *sat_Tsat*, int * *sat_uniqueID*, const char * *mediumName*, const char * *libraryName*, const char * *substanceName*)

Compute saturation properties from T.

This function computes the saturation properties for the specified inputs. If the function is called with uniqueID == 0 a new transient unique ID is assigned and the function is called again with this new transient unique ID number.

Parameters:

T Temperature

uniqueID Unique ID number

sat_psat Pointer to return pressure for saturation record

sat_ Tsat Pointer to return temperature for saturation record

sat_ uniqueID Pointer to return unique ID number for saturation record

mediumName Medium name

libraryName Library name

substanceName Substance name

6.3.2.11 EXPORT void setState_dT_ (double *d*, double *T*, int *phase*, int *uniqueID*, int * *state_ uniqueID*, int * *state_ phase*, double * *state_ d*, double * *state_ h*, double * *state_ p*, double * *state_ s*, double * *state_ T*, const char * *mediumName*, const char * *libraryName*, const char * *substanceName*)

Compute properties from d, T, and phase.

This function computes the properties for the specified inputs. If the function is called with uniqueID == 0 a new transient unique ID is assigned and the function is called again with this new transient unique ID number.

Parameters:

d Density

T Temperature

phase Phase (2 for two-phase, 1 for one-phase, 0 if not known)

uniqueID Unique ID number

state_ uniqueID Pointer to return unique ID number for state record

state_ phase Pointer to return phase for state record

state_ d Pointer to return density for state record

state_ h Pointer to return specific enthalpy for state record

state_ p Pointer to return pressure for state record

state_ s Pointer to return specific entropy for state record

state_ T Pointer to return temperature for state record

mediumName Medium name

libraryName Library name

substanceName Substance name

6.3.2.12 EXPORT void setState_ph_ (double *p*, double *h*, int *phase*, int *uniqueID*, int * *state_ uniqueID*, int * *state_ phase*, double * *state_ d*, double * *state_ h*, double * *state_ p*, double * *state_ s*, double * *state_ T*, const char * *mediumName*, const char * *libraryName*, const char * *substanceName*)

Compute properties from p, h, and phase.

This function computes the properties for the specified inputs. If the function is called with uniqueID == 0 a new transient unique ID is assigned and the function is called again with this new transient unique ID number.

Parameters:

p Pressure

h Specific enthalpy

phase Phase (2 for two-phase, 1 for one-phase, 0 if not known)

uniqueID Unique ID number

state_uniqueID Pointer to return unique ID number for state record

state_phase Pointer to return phase for state record

state_d Pointer to return density for state record

state_h Pointer to return specific enthalpy for state record

state_p Pointer to return pressure for state record

state_s Pointer to return specific entropy for state record

state_T Pointer to return temperature for state record

mediumName Medium name

libraryName Library name

substanceName Substance name

6.3.2.13 EXPORT void setState_ps_ (double *p*, double *s*, int *phase*, int *uniqueID*, int * *state_uniqueID*, int * *state_phase*, double * *state_d*, double * *state_h*, double * *state_p*, double * *state_s*, double * *state_T*, const char * *mediumName*, const char * *libraryName*, const char * *substanceName*)

Compute properties from p, s, and phase.

This function computes the properties for the specified inputs. If the function is called with uniqueID == 0 a new transient unique ID is assigned and the function is called again with this new transient unique ID number.

Parameters:

p Pressure

s Specific entropy

phase Phase (2 for two-phase, 1 for one-phase, 0 if not known)

uniqueID Unique ID number

state_uniqueID Pointer to return unique ID number for state record

state_phase Pointer to return phase for state record

state_d Pointer to return density for state record

state_h Pointer to return specific enthalpy for state record

state_p Pointer to return pressure for state record

state_s Pointer to return specific entropy for state record

state_T Pointer to return temperature for state record

mediumName Medium name

libraryName Library name

substanceName Substance name

6.3.2.14 EXPORT void setState_pT_ (double *p*, double *T*, int *phase*, int *uniqueID*, int * *state_uniqueID*, int * *state_phase*, double * *state_d*, double * *state_h*, double * *state_p*, double * *state_s*, double * *state_T*, const char * *mediumName*, const char * *libraryName*, const char * *substanceName*)

Compute properties from p, T, and phase.

This function computes the properties for the specified inputs. If the function is called with uniqueID == 0 a new transient unique ID is assigned and the function is called again with this new transient unique ID number.

Attention: The phase input is ignored for this function!

Parameters:

p Pressure

T Temperature

phase Phase (2 for two-phase, 1 for one-phase, 0 if not known)

uniqueID Unique ID number

state_uniqueID Pointer to return unique ID number for state record

state_phase Pointer to return phase for state record

state_d Pointer to return density for state record

state_h Pointer to return specific enthalpy for state record

state_p Pointer to return pressure for state record

state_s Pointer to return specific entropy for state record

state_T Pointer to return temperature for state record

mediumName Medium name

libraryName Library name

substanceName Substance name

6.4 include.h File Reference

generic include file

```
#include <map>
#include <string>
#include "errorhandling.h"
```

Defines

- #define **DYMOLA** 1
  Modelica compiler is Dymola.
- #define **OPEN_MODELICA** 0
  Modelica compiler is OpenModelica.
- #define **FLUIDPROP** 0
  FluidProp solver.
- #define **BUILD_DLL** 0
  Build project into a DLL.
- #define **BUILD_LIB** 1
  Build project into a LIB.
- #define **MAX_TRANSIENT_MEDIUM** 1000
  Maximum number of non-overlapping transient medium objects.
- #define **NAN** 0xffffffff
  Not a number.

6.4.1 Detailed Description

generic include file

This is a generic include for for the entire ExternalMediaPackage project. It defines some important preprocessor variables that might have to be changed by the user.

Uncomment the define directives as appropriate

Christoph Richter, Francesco Casella, Oct 2006

6.4.2 Define Documentation

6.4.2.1 #define BUILD_DLL 0

Build project into a DLL.

Set this preprocessor variable to 1 if the project is built into a dynamic link library. This setting influences the error reporting mechanism as well as the export statement.

See also:

BUILD_LIB (p. 66)

6.4.2.2 #define BUILD_LIB 1

Build project into a LIB.

Set this preprocessor variable to 1 if the project is built into a static library. This setting influences the error reporting mechanism as well as the export statement.

See also:

BUILD_DLL (p. 65)

6.4.2.3 #define DYMOLA 1

Modelica compiler is Dymola.

Set this preprocessor variable to 1 if Dymola is the Modelica compiler that is going to be used with the compiled library.

See also:

OPEN_MODELICA (p. 66)

6.4.2.4 #define FLUIDPROP 0

FluidProp solver.

Set this preprocessor variable to 1 to include the interface to the FluidProp solver developed and maintained by Francesco Casella.

6.4.2.5 #define MAX_TRANSIENT_MEDIUM 1000

Maximum number of non-overlapping transient medium objects.

Increase this number if you ran a large model with more than a thousand instances of ThermodynamicState and SaturationState outside BaseProperties records, and without explicit uniqueID handling

6.4.2.6 #define NAN 0xffffffff

Not a number.

This value is used as not a number value. It can be changed by the user if there is a more appropriate value.

6.4.2.7 #define OPEN_MODELICA 0

Modelica compiler is OpenModelica.

Set this preprocessor variable to 1 if OpanModelica is the Modelica compiler that is going to be used with the compiled library.

See also:

DYMOLA (p. 66)

Index

~BaseSolver
    BaseSolver, 13
~BaseTwoPhaseMedium
    BaseTwoPhaseMedium, 23
~TwoPhaseMedium
    TwoPhaseMedium, 38

addMedium
    MediumMap, 29
addSolverMedium
    MediumMap, 29
addTransientMedium
    MediumMap, 29

BaseSolver, 11
    BaseSolver, 13
BaseSolver
    ~BaseSolver, 13
    BaseSolver, 13
    computeDerivatives, 17
    isentropicEnthalpy, 17
    setBubbleState, 16
    setDewState, 16
    setFluidConstants, 14
    setSat_p, 14
    setSat_p_state, 14
    setSat_T, 14
    setState_dT, 15
    setState_ph, 15
    setState_ps, 15
    setState_pT, 16
BaseTwoPhaseMedium, 18
    BaseTwoPhaseMedium, 22
BaseTwoPhaseMedium
    ~BaseTwoPhaseMedium, 23
    BaseTwoPhaseMedium, 22
    getBubbleUniqueID, 25
    getDewUniqueID, 25
    reinitMedium, 23
    setBubbleState, 25
    setDewState, 25
    setSat_p, 23
    setSat_p_state, 24
    setSat_T, 24
    setSolver, 23
    setState_dT, 24
    setState_ph, 24
    setState_ps, 24
    setState_pT, 25
BUILD_DLL
    include.h, 65
BUILD_LIB
    include.h, 66

changeMedium
    MediumMap, 30
computeDerivatives
    BaseSolver, 17
createMedium_
    externaltwophasemedium.cpp, 49
    externaltwophasemedium.h, 58

deleteMedium
    MediumMap, 30
DYMOLA
    include.h, 66

errorhandling.h, 43
    errorMessage, 43
    warningMessage, 43
errorMessage
    errorhandling.h, 43
externaltwophasemedium.cpp, 45
    createMedium_, 49
    getCriticalMolarVolume_, 49
    getCriticalPressure_, 49
    getCriticalTemperature_, 49
    getMolarMass_, 50
    setBubbleState_, 50
    setDewState_, 50
    setSat_p_, 50
    setSat_p_state_, 51
    setSat_T_, 51
    setState_dT_, 52
    setState_ph_, 52
    setState_ps_, 53
    setState_pT_, 53
externaltwophasemedium.h, 55
    createMedium_, 58
    getCriticalMolarVolume_, 59

getCriticalPressure_, 59
getCriticalTemperature_, 59
getMolarMass_, 59
setBubbleState_, 60
setDewState_, 60
setSat_p_, 60
setSat_p_state_, 61
setSat_T_, 61
setState_dT_, 62
setState_ph_, 62
setState_ps_, 63
setState_pT_, 63

FluidConstants, 27
FLUIDPROP
include.h, 66

getBubbleUniqueID
BaseTwoPhaseMedium, 25
getCriticalMolarVolume_
externaltwophasemedium.cpp, 49
externaltwophasemedium.h, 59
getCriticalPressure_
externaltwophasemedium.cpp, 49
externaltwophasemedium.h, 59
getCriticalTemperature_
externaltwophasemedium.cpp, 49
externaltwophasemedium.h, 59
getDewUniqueID
BaseTwoPhaseMedium, 25
getMolarMass_
externaltwophasemedium.cpp, 50
externaltwophasemedium.h, 59
getSolver
SolverMap, 32

include.h, 65
BUILD_DLL, 65
BUILD_LIB, 66
DYMOLA, 66
FLUIDPROP, 66
MAX_TRANSIENT_MEDIUM, 66
NAN, 66
OPEN_MODELICA, 66
initializeFields
TwoPhaseMediumProperties, 42
isentropicEnthalpy
BaseSolver, 17

MAX_TRANSIENT_MEDIUM
include.h, 66
medium
MediumMap, 30
MediumMap, 28
MediumMap
addMedium, 29
addSolverMedium, 29
addTransientMedium, 29
changeMedium, 30
deleteMedium, 30
medium, 30
solverMedium, 30

NAN
include.h, 66

OPEN_MODELICA
include.h, 66

phase
TwoPhaseMediumProperties, 42

reinitMedium
BaseTwoPhaseMedium, 23

setBubbleState
BaseSolver, 16
BaseTwoPhaseMedium, 25
setBubbleState_
externaltwophasemedium.cpp, 50
externaltwophasemedium.h, 60
setDewState
BaseSolver, 16
BaseTwoPhaseMedium, 25
setDewState_
externaltwophasemedium.cpp, 50
externaltwophasemedium.h, 60
setFluidConstants
BaseSolver, 14
TestSolver, 35
setSat_p
BaseSolver, 14
BaseTwoPhaseMedium, 23
TestSolver, 35
setSat_p_
externaltwophasemedium.cpp, 50
externaltwophasemedium.h, 60
setSat_p_state
BaseSolver, 14
BaseTwoPhaseMedium, 24
TestSolver, 36
setSat_p_state_
externaltwophasemedium.cpp, 51
externaltwophasemedium.h, 61
setSat_T
BaseSolver, 14
BaseTwoPhaseMedium, 24
TestSolver, 35
setSat_T_

externaltwophasemedium.cpp, 51
externaltwophasemedium.h, 61
setSolver
BaseTwoPhaseMedium, 23
setState_dT
BaseSolver, 15
BaseTwoPhaseMedium, 24
TestSolver, 36
setState_dT_
externaltwophasemedium.cpp, 52
externaltwophasemedium.h, 62
setState_ph
BaseSolver, 15
BaseTwoPhaseMedium, 24
TestSolver, 36
setState_ph_
externaltwophasemedium.cpp, 52
externaltwophasemedium.h, 62
setState_ps
BaseSolver, 15
BaseTwoPhaseMedium, 24
TestSolver, 37
setState_ps_
externaltwophasemedium.cpp, 53
externaltwophasemedium.h, 63
setState_pT
BaseSolver, 16
BaseTwoPhaseMedium, 25
TestSolver, 37
setState_pT_
externaltwophasemedium.cpp, 53
externaltwophasemedium.h, 63
solverKey
SolverMap, 32
SolverMap, 32
SolverMap
getSolver, 32
solverKey, 32
solverMedium
MediumMap, 30

TestSolver, 34
TestSolver
setFluidConstants, 35
setSat_p, 35
setSat_p_state, 36
setSat_T, 35
setState_dT, 36
setState_ph, 36
setState_ps, 37
setState_pT, 37
TwoPhaseMedium, 38
TwoPhaseMedium, 38
TwoPhaseMedium
~TwoPhaseMedium, 38
TwoPhaseMedium, 38
TwoPhaseMediumProperties, 40
TwoPhaseMediumProperties, 42
TwoPhaseMediumProperties
initializeFields, 42
phase, 42
TwoPhaseMediumProperties, 42

warningMessage
errorhandling.h, 43