ExternalMedia

Generated by Doxygen 1.7.6.1

Mon Feb 20 2012 23:47:57

Contents

Chapter 1

External Media HowTo

1.1 Introduction

The ExternalMedia project was started in 2006 by Francesco Casella and Christoph Richter, with the aim of providing a framework for interfacing external codes computing fluid properties to Modelica.Media-compatible component models. The two main requirements are: maximizing the efficiency of the code and minimizing the amount of extra code required to use your own external code within the framework.

The first implementation featured a hidden cache in the C++ layer and used integer unique IDs to reference that cache. This architecture worked well if the models did not contain implicit algebraic equations involving medium properties, but had serious issues when such equations were involved, which is often the case when solving steady-state initialization problems.

The library has been restructured in 2012 by Francesco Casella and Roberto Bonifetto. The main idea has been to get rid of the hidden cache and of the unique ID references and use the Modelica state records for caching. In this way, all optimizations performed by Modelica tools are guaranteed to give correct results, which was previously not the case. The current version implements the Modelica.Media.Interfaces.PartialTwoPhaseMedium interface, so it can handle pure fluids, either one-phase or two-phase, and is compatible with Modelica and Modelica Standard Library 3.2. Please note that the paths of the medium packages have been changed from the previous versions, so you might need some small changes if you want to upgrade your models from previous versions of the ExternalMedia library.

There are two ways to use this library. The easiest way is to use the releases available on the Modelica website, which include a pre-compiled interface to the FluidProp tool (http://www.fluidprop.com). FluidProp features many built-in fluid models, and can optionally be used to access the whole NIST RefProp database, thus giving easy access to a wide range of fluid models with state-of-the-art accuracy. If you want to use your own fluid property computation code instead, then you need to check out the source code and add the interface to it, as described in this manual.

Please contact the main developer, Francesco Casella (casella@elet.polimi.it) if you have questions or suggestions for improvement.

1.2 Using the pre-packaged releases with FluidProp

Download and install the latest version of FluidProp from http://www.-fluidprop.com. If you want to use the RefProp fluid models, you need to get the full version of FluidProp, which has an extra license fee.

Download and unzip the library corresponding to the version of Microsoft Visual Studio that you use to compile your Modelica models, in order to avoid linker errors. Make sure that you load the ExternalMedia library in your Modelica tool workspace, e.g. by opening the main package.mo file.

You can now define medium models for all the libraries supported by FluidProp, by extending the ExternalMedia.Media.FluidPropMedium package. Set libraryName to "-FluidProp.RefProp", "FluidProp.StanMix", "FluidProp.TPSI", "FluidProp.IF97", or "FluidProp.GasMix" depending on the specific library you need to use. Set substanceNames to a single-element string array containing the name of the specific medium, as specified by the FluidProp documentation. Set mediumName to a string that describes the medium (this only used for documentation purposes but has no effect in selecting the medium model). See ExternalMedia.Examples for examples.

Please note that the medium models IF97 and GasMix are already available natively in Modelica.Media as Water.StandardWater and IdealGases.MixtureGases, respectively - it is recommended to use the Modelica.Media models instead, since they are much faster to compute.

1.3 Architecture of the package

This section gives an overview of the package structure, in order to help you understand how to interface your own code to Modelica using it.

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 Modelica functions within ExternalTwoPhaseMedium communicate to a C/C++ interface layer (called externalmedialib.cpp) via external C functions calls, which in turn make use of C++ objects. This layer takes care of initializing the external fluid computation codes, called solvers from now on. Every solver is wrapped by a C++ class, inheriting from the BaseSolver C++ class. The C/C++ layer maintains a set of active solvers,

one for each different combination of the libraryName and mediumName strings, by means of the SolverMap C++ class. The key to each solver in the map is given by those strings. It is then possible to use multiple instances of many solvers in the same Modelica model at the same time.

All the external C functions pass the libraryName, mediumName and substanceNames strings to the corresponding functions of the interface layer. These in turn use the SolverMap object to look for an active solver in the solver map, corresponding to those strings. If one is found, the corresponding function of the solver is called, otherwise a new solver object is instantiated and added to the map, before calling the corresponding function of the solver.

The default implementation of an external medium model is implemented by the ExternalTwoPhaseMedium Modelica package. The setState_xx() and setSat_x() function calls are rerouted to the corresponding functions of the solver object. These compute all the required properties and return them in the ExternalThermodynamicState and ExternalSaturationProperties C structs, which map onto the corresponding ThermodynamicState and SaturationProperties records defined in ExternalTwoPhaseMedium. All the functions returning properties as a function of the state records are implemented in Modelica and simply return the corresponding element in the state record, which acts as a cache. This is an efficient implementation for many complex fluid models, where most of the CPU time is spent solving the basic equation of state, while the computation of all derived properties adds a minor overhead, so it makes sense to compute them once and for all when the setState_XX() or setSat_xx() functions are called.

In case some of the thermodynamic properties require a significant amount of CPU time on their own, it is possible to override this default implementation. On one hand, it is necessary to extend the ExternalTwoPhaseMedium Modelica package and redeclare those functions, so that they call the corresponding external C functions defined in externalmedium.cpp, instead of returning the value cached in the state record. On the other hand, it is also necessary to provide an implementation of the corresponding functions in the C++ solver object, by overriding the virtual functions of the BaseSolver object. In this case, the setState_xx() and setSat_X() functions need not compute all the values of the cache state records; uncomputed properties might be set to zero. This is not a problem, since Modelica.Media compatible models should never access the elements of the state records directly, but only through the appropriate functions, so these values should never be actually used by component models using the medium package.

1.4 Developing your own external medium package

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.

The most straightforward implementation is the one in which all fluid properties are computed at once by the setState_XX() and setSat_X() functions and all the other functions

return the values cached in the state records.

Get the source code from the SVN repository of the Modelica Association: https://svn.modelica.org/projects/ExternalMediaLibrary/trunk.

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 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 and setSat functions need to compute and fill in all the fields of the corresponding C structs for the library to work correctly. On the other hand, you don't necessarily need to implement all of the four setState functions: if you know in advance that your models will only use certain combinations of variables as inputs (e.g. p, h), then you might omit implementing the setState and setSat 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 is up to you to decode that syntax within the addSolver() function, and within the constructor of your solver object. Look at how the FluidProp solver is implemented for an example.

Finally, add the .cpp and .h files of the solver object to the C/C++ project, set the include.h file according to your needs and recompile it to a static library (or to a DLL). The compiled libraries and the externalmedialib.h files must then be copied into the Include subdirectory of the Modelica package so that the Modelica tool can link them when compiling the models.

As already mentioned in the previous section, you might provide customized implementations where some of the properties are not computed by the setState and setSat functions and stored in the cache records, but rather computed on demand, based on a smaller set of thermodynamic properties computed by the setState and setSat functions and stored in the state C struct.

Please note that compiling ExternalMedia from source code might require the professional version of Microsoft Visual Studio, which includes the COM libraries used by the FluidProp interface. However, if you remove all the FluidProp files and references from the project, then you should be able to compile the source code with the Express edition, or possibly also with gcc.

Chapter 2

Class Index

2.1 Class Hierarchy

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

Chapter 3

Class Index

3.1 Class List

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

Chapter 4

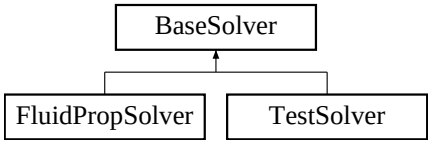
Class Documentation

4.1 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.
- 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 criticalMolarVolume () const

  Return molar volume at critical point (Default implementation provided)
- double criticalDensity () const

Return density 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)

- virtual void setFluidConstants ()

  Set fluid constants.

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

  Set state from p, h, and phase.

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

  Set state from p and T.

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

  Set state from d, T, and phase.

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

  Set state from p, s, and phase.

- virtual double Pr (ExternalThermodynamicState *const properties)

  Compute Prandtl number.

- virtual double T (ExternalThermodynamicState *const properties)

  Compute temperature.

- virtual double a (ExternalThermodynamicState *const properties)

  Compute velocity of sound.

- virtual double beta (ExternalThermodynamicState *const properties)

  Compute isobaric expansion coefficient.

- virtual double cp (ExternalThermodynamicState *const properties)

  Compute specific heat capacity cp.

- virtual double cv (ExternalThermodynamicState *const properties)

  Compute specific heat capacity cv.

- virtual double d (ExternalThermodynamicState *const properties)

  Compute density.

- virtual double ddhp (ExternalThermodynamicState *const properties)

  Compute derivative of density wrt enthalpy at constant pressure.

- virtual double ddph (ExternalThermodynamicState *const properties)

  Compute derivative of density wrt pressure at constant enthalpy.

- virtual double eta (ExternalThermodynamicState *const properties)

  Compute dynamic viscosity.

- virtual double h (ExternalThermodynamicState *const properties)

  Compute specific enthalpy.

- virtual double kappa (ExternalThermodynamicState *const properties)

  Compute compressibility.

- virtual double lambda (ExternalThermodynamicState *const properties)

Compute thermal conductivity.

- virtual double p (ExternalThermodynamicState *const properties)

  Compute pressure.
- virtual int phase (ExternalThermodynamicState *const properties)

  Compute phase flag.
- virtual double s (ExternalThermodynamicState *const properties)

  Compute specific entropy.
- virtual double d_der (ExternalThermodynamicState *const properties)

  Compute total derivative of density ph.
- virtual double isentropicEnthalpy (double &p, ExternalThermodynamicState *const properties)

  Compute isentropic enthalpy.
- virtual void setSat_p (double &p, ExternalSaturationProperties *const properties)

  Set saturation properties from p.
- virtual void setSat_T (double &T, ExternalSaturationProperties *const properties)

  Set saturation properties from T.
- virtual void setBubbleState (ExternalSaturationProperties *const properties, int phase, ExternalThermodynamicState *const bubbleProperties)

  Set bubble state.
- virtual void setDewState (ExternalSaturationProperties *const properties, int phase, ExternalThermodynamicState *const bubbleProperties)

  Set dew state.
- virtual double dTp (ExternalSaturationProperties *const properties)

  Compute derivative of Ts wrt pressure.
- virtual double ddldp (ExternalSaturationProperties *const properties)

  Compute derivative of dls wrt pressure.
- virtual double ddvdp (ExternalSaturationProperties *const properties)

  Compute derivative of dvs wrt pressure.
- virtual double dhldp (ExternalSaturationProperties *const properties)

  Compute derivative of hls wrt pressure.
- virtual double dhvdp (ExternalSaturationProperties *const properties)

  Compute derivative of hvs wrt pressure.
- virtual double dl (ExternalSaturationProperties *const properties)

  Compute density at bubble line.
- virtual double dv (ExternalSaturationProperties *const properties)

  Compute density at dew line.
- virtual double hl (ExternalSaturationProperties *const properties)

  Compute enthalpy at bubble line.
- virtual double hv (ExternalSaturationProperties *const properties)

  Compute enthalpy at dew line.
- virtual double sigma (ExternalSaturationProperties *const properties)

  Compute surface tension.

- virtual double sl (ExternalSaturationProperties *const properties)

  Compute entropy at bubble line.
- virtual double sv (ExternalSaturationProperties *const properties)

  Compute entropy at dew line.
- virtual bool computeDerivatives (ExternalThermodynamicState *const properties)

  Compute derivatives.
- virtual double psat (ExternalSaturationProperties *const properties)

  Compute saturation pressure.
- virtual double Tsat (ExternalSaturationProperties *const properties)

  Compute saturation temperature.

Public Attributes

- string mediumName

  Medium name.
- string libraryName

  Library name.
- string substanceName

  Substance name.

Protected Attributes

- FluidConstants _fluidConstants

  Fluid constants.

4.1.1 Detailed Description

Base solver class.

This is the base class for all external solver objects (e.g. TestSolver, FluidPropSolver). A solver object encapsulates the interface to external fluid property computation routines

Francesco Casella, Christoph Richter, Roberto Bonifetto 2006-2012 Copyright - Politecnico di Milano, TU Braunschweig, Politecnico di Torino

4.1.2 Constructor & Destructor Documentation

4.1.2.1 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

4.1.2.2 BaseSolver::~BaseSolver ( ) [virtual]

Destructor.

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

4.1.3 Member Function Documentation

4.1.3.1 double BaseSolver::a (ExternalThermodynamicState *const *properties*) [virtual]

Compute velocity of sound.

This function returns the velocity of sound from the state specified by the properties input

Must be re-implemented in the specific solver

Parameters

properties	ExternalThermodynamicState property struct corresponding to current state

4.1.3.2 double BaseSolver::beta (ExternalThermodynamicState *const *properties*) [virtual]

Compute isobaric expansion coefficient.

This function returns the isobaric expansion coefficient from the state specified by the properties input

Must be re-implemented in the specific solver

Parameters

properties	ExternalThermodynamicState property struct corresponding to current state

4.1.3.3 bool BaseSolver::computeDerivatives (ExternalThermodynamicState *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	ExternalThermodynamicState property record

4.1.3.4 double BaseSolver::cp (ExternalThermodynamicState *const *properties*) [virtual]

Compute specific heat capacity cp.

This function returns the specific heat capacity cp from the state specified by the properties input

Must be re-implemented in the specific solver

Parameters

properties	ExternalThermodynamicState property struct corresponding to current state

4.1.3.5 double BaseSolver::cv (ExternalThermodynamicState *const *properties*) [virtual]

Compute specific heat capacity cv.

This function returns the specific heat capacity cv from the state specified by the properties input

Must be re-implemented in the specific solver

Parameters

properties	ExternalThermodynamicState property struct corresponding to current state

4.1.3.6 double BaseSolver::d (ExternalThermodynamicState *const *properties*) [virtual]

Compute density.

This function returns the density from the state specified by the properties input

Must be re-implemented in the specific solver

Parameters

properties	ExternalThermodynamicState property struct corresponding to current state

4.1.3.7 double BaseSolver::d_der (ExternalThermodynamicState *const *properties*) [virtual]

Compute total derivative of density ph.

This function returns the total derivative of density ph from the state specified by the properties input

Must be re-implemented in the specific solver

Parameters

properties	ExternalThermodynamicState property struct corresponding to current state

4.1.3.8 double BaseSolver::ddhp (ExternalThermodynamicState *const *properties*) [virtual]

Compute derivative of density wrt enthalpy at constant pressure.

This function returns the derivative of density wrt enthalpy at constant pressure from the state specified by the properties input

Must be re-implemented in the specific solver

Parameters

properties	ExternalThermodynamicState property struct corresponding to current state

4.1.3.9 double BaseSolver::ddldp (ExternalSaturationProperties *const *properties*) [virtual]

Compute derivative of dls wrt pressure.

This function returns the derivative of dls wrt pressure from the state specified by the

properties input

Must be re-implemented in the specific solver

Parameters

properties	ExternalSaturationProperties property struct corresponding to current state

4.1.3.10 double BaseSolver::ddph (ExternalThermodynamicState *const *properties) [virtual]

Compute derivative of density wrt pressure at constant enthalpy.

This function returns the derivative of density wrt pressure at constant enthalpy from the state specified by the properties input

Must be re-implemented in the specific solver

Parameters

properties	ExternalThermodynamicState property struct corresponding to current state

4.1.3.11 double BaseSolver::ddvdp (ExternalSaturationProperties *const *properties) [virtual]

Compute derivative of dvs wrt pressure.

This function returns the derivative of dvs wrt pressure from the state specified by the properties input

Must be re-implemented in the specific solver

Parameters

properties	ExternalSaturationProperties property struct corresponding to current state

4.1.3.12 double BaseSolver::dhldp (ExternalSaturationProperties *const *properties) [virtual]

Compute derivative of hls wrt pressure.

This function returns the derivative of hls wrt pressure from the state specified by the properties input

Must be re-implemented in the specific solver

Parameters

properties	ExternalSaturationProperties property struct corresponding to current state

4.1.3.13 double BaseSolver::dhvdp (ExternalSaturationProperties *const *properties) [virtual]

Compute derivative of hvs wrt pressure.

This function returns the derivative of hvs wrt pressure from the state specified by the properties input

Must be re-implemented in the specific solver

Parameters

properties	ExternalSaturationProperties property struct corresponding to current state

4.1.3.14 double BaseSolver::dl (ExternalSaturationProperties *const *properties*) [virtual]

Compute density at bubble line.

This function returns the density at bubble line from the state specified by the properties input

Must be re-implemented in the specific solver

Parameters

properties	ExternalSaturationProperties property struct corresponding to current state

4.1.3.15 double BaseSolver::dTp (ExternalSaturationProperties *const *properties*) [virtual]

Compute derivative of Ts wrt pressure.

This function returns the derivative of Ts wrt pressure from the state specified by the properties input

Must be re-implemented in the specific solver

Parameters

properties	ExternalSaturationProperties property struct corresponding to current state

4.1.3.16 double BaseSolver::dv (ExternalSaturationProperties *const *properties*) [virtual]

Compute density at dew line.

This function returns the density at dew line from the state specified by the properties input

Must be re-implemented in the specific solver

Parameters

properties	ExternalSaturationProperties property struct corresponding to current state

4.1.3.17 double BaseSolver::eta (ExternalThermodynamicState *const *properties*) [virtual]

Compute dynamic viscosity.

This function returns the dynamic viscosity from the state specified by the properties input

Must be re-implemented in the specific solver

Parameters

properties	ExternalThermodynamicState property struct corresponding to current state

4.1.3.18 double BaseSolver::h (ExternalThermodynamicState *const *properties*) [virtual]

Compute specific enthalpy.

This function returns the specific enthalpy from the state specified by the properties input

Must be re-implemented in the specific solver

Parameters

properties	ExternalThermodynamicState property struct corresponding to current state

4.1.3.19 double BaseSolver::hl (ExternalSaturationProperties *const *properties*) [virtual]

Compute enthalpy at bubble line.

This function returns the enthalpy at bubble line from the state specified by the properties input

Must be re-implemented in the specific solver

Parameters

properties	ExternalSaturationProperties property struct corresponding to current state

4.1.3.20 double BaseSolver::hv (ExternalSaturationProperties *const *properties*) [virtual]

Compute enthalpy at dew line.

This function returns the enthalpy at dew line from the state specified by the properties input

Must be re-implemented in the specific solver

Parameters

properties	ExternalSaturationProperties property struct corresponding to current state

4.1.3.21 double BaseSolver::isentropicEnthalpy (double & *p*, ExternalThermodynamicState *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	ExternalThermodynamicState property struct corresponding to current state

Reimplemented in FluidPropSolver.

4.1.3.22 double BaseSolver::kappa (ExternalThermodynamicState *const *properties*) [virtual]

Compute compressibility.

This function returns the compressibility from the state specified by the properties input

Must be re-implemented in the specific solver

Parameters

properties	ExternalThermodynamicState property struct corresponding to current state

4.1.3.23 double BaseSolver::lambda (ExternalThermodynamicState *const *properties) [virtual]

Compute thermal conductivity.

This function returns the thermal conductivity from the state specified by the properties input

Must be re-implemented in the specific solver

Parameters

properties	ExternalThermodynamicState property struct corresponding to current state

4.1.3.24 double BaseSolver::p (ExternalThermodynamicState *const *properties*) [virtual]

Compute pressure.

This function returns the pressure from the state specified by the properties input

Must be re-implemented in the specific solver

Parameters

properties	ExternalThermodynamicState property struct corresponding to current state

4.1.3.25 int BaseSolver::phase (ExternalThermodynamicState *const *properties*) [virtual]

Compute phase flag.

This function returns the phase flag from the state specified by the properties input

Must be re-implemented in the specific solver

Parameters

properties	ExternalThermodynamicState property struct corresponding to current state

4.1.3.26 double BaseSolver::Pr (ExternalThermodynamicState *const *properties*) [virtual]

Compute Prandtl number.

This function returns the Prandtl number from the state specified by the properties input

Must be re-implemented in the specific solver

Parameters

properties	ExternalThermodynamicState property struct corresponding to current state

4.1.3.27 double BaseSolver::psat (ExternalSaturationProperties *const *properties*) [virtual]

Compute saturation pressure.

This function returns the saturation pressure from the state specified by the properties input

Must be re-implemented in the specific solver

Parameters

properties	ExternalSaturationProperties property struct corresponding to current state

4.1.3.28 double BaseSolver::s (ExternalThermodynamicState *const *properties*) [virtual]

Compute specific entropy.

This function returns the specific entropy from the state specified by the properties input

Must be re-implemented in the specific solver

Parameters

properties	ExternalThermodynamicState property struct corresponding to current state

4.1.3.29 void BaseSolver::setBubbleState (ExternalSaturationProperties *const *properties*, int *phase*, ExternalThermodynamicState *const *bubbleProperties*) [virtual]

Set bubble state.

This function sets the bubble state record bubbleProperties corresponding to the satu-

ration 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

properties	ExternalSaturationProperties record with saturation properties data
phase	Phase (1: one-phase, 2: two-phase)
bubbleProperties	ExternalThermodynamicState record where to write the bubble point properties

4.1.3.30 void BaseSolver::setDewState (ExternalSaturationProperties *const *properties, int *phase, ExternalThermodynamicState *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

properties	ExternalSaturationProperties record with saturation properties data
phase	Phase (1: one-phase, 2: two-phase)
bubbleProperties	ExternalThermodynamicState record where to write the dew point properties

4.1.3.31 void BaseSolver::setFluidConstants ( ) [virtual]

Set fluid constants.

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

Must be re-implemented in the specific solver

Reimplemented in FluidPropSolver, and TestSolver.

4.1.3.32 void BaseSolver::setSat_p (double & *p, ExternalSaturationProperties *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 ExternalSaturationProperties propery struct.

Must be re-implemented in the specific solver

Parameters

p	Pressure
properties	ExternalSaturationProperties property struct

Reimplemented in FluidPropSolver, and TestSolver.

4.1.3.33 void BaseSolver::setSat_T (double & *T,* ExternalSaturationProperties *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 ExternalSaturationProperties propery struct.

Must be re-implemented in the specific solver

Parameters

T	Temperature
properties	ExternalSaturationProperties property struct

Reimplemented in FluidPropSolver, and TestSolver.

4.1.3.34 void BaseSolver::setState_dT (double & *d,* double & *T,* int & *phase,* ExternalThermodynamicState *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 ExternalThermodynamicState 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	ExternalThermodynamicState property struct

Reimplemented in FluidPropSolver, and TestSolver.

4.1.3.35 void BaseSolver::setState_ph (double & *p*, double & *h*, int & *phase*, ExternalThermodynamicState *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 ExternalThermodynamicState 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	ExternalThermodynamicState property struct

Reimplemented in FluidPropSolver, and TestSolver.

4.1.3.36 void BaseSolver::setState_ps (double & *p*, double & *s*, int & *phase*, ExternalThermodynamicState *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 ExternalThermodynamicState 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	ExternalThermodynamicState property struct

Reimplemented in FluidPropSolver, and TestSolver.

4.1.3.37 void BaseSolver::setState_pT (double & *p*, double & *T*, ExternalThermodynamicState *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 ExternalThermodynamicState property struct.

Must be re-implemented in the specific solver

Parameters

p	Pressure
T	Temperature
properties	ExternalThermodynamicState property struct

Reimplemented in FluidPropSolver, and TestSolver.

4.1.3.38 double BaseSolver::sigma (ExternalSaturationProperties *const *properties) [virtual]

Compute surface tension.

This function returns the surface tension from the state specified by the properties input

Must be re-implemented in the specific solver

Parameters

properties	ExternalSaturationProperties property struct corresponding to current state

4.1.3.39 double BaseSolver::sl (ExternalSaturationProperties *const *properties*) [virtual]

Compute entropy at bubble line.

This function returns the entropy at bubble line from the state specified by the properties input

Must be re-implemented in the specific solver

Parameters

properties	ExternalSaturationProperties property struct corresponding to current state

4.1.3.40 double BaseSolver::sv (ExternalSaturationProperties *const *properties*) [virtual]

Compute entropy at dew line.

This function returns the entropy at dew line from the state specified by the properties input

Must be re-implemented in the specific solver

Parameters

properties	ExternalSaturationProperties property struct corresponding to current state

4.1.3.41 double BaseSolver::T (ExternalThermodynamicState *const *properties*) [virtual]

Compute temperature.

This function returns the temperature from the state specified by the properties input

Must be re-implemented in the specific solver

Parameters

properties	ExternalThermodynamicState property struct corresponding to current state

4.1.3.42 double BaseSolver::Tsat (ExternalSaturationProperties *const *properties*) [virtual]

Compute saturation temperature.

This function returns the saturation temperature from the state specified by the properties input

Must be re-implemented in the specific solver

Parameters

properties	ExternalSaturationProperties property struct corresponding to current state

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

- D:/Lavoro/ModelicaSVN/ExternalMediaLibrary/Projects/Sources/basesolver.h
- D:/Lavoro/ModelicaSVN/ExternalMediaLibrary/Projects/Sources/basesolver.cpp

4.2 ExternalSaturationProperties Struct Reference

ExternalSaturationProperties property struct.

`#include <externalmedialib.h>`

Public Attributes

- double Tsat

  Saturation temperature.
- double dTp

  Derivative of Ts wrt pressure.
- double ddldp

  Derivative of dls wrt pressure.

- double ddvdp

  Derivative of dvs wrt pressure.
- double dhldp

  Derivative of hls wrt pressure.
- double dhvdp

  Derivative of hvs wrt pressure.
- 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 psat

  Saturation pressure.
- double sigma

  Surface tension.
- double sl

  Specific entropy at bubble line (for pressure ps)
- double sv

  Specific entropy at dew line (for pressure ps)

4.2.1 Detailed Description

ExternalSaturationProperties property struct.

The ExternalSaturationProperties propery struct defines all the saturation properties for the dew and the bubble line that are computed by external Modelica medium models extending from PartialExternalTwoPhaseMedium.

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

- D:/Lavoro/ModelicaSVN/ExternalMediaLibrary/Projects/Sources/externalmedialib.-h

4.3 ExternalThermodynamicState Struct Reference

ExternalThermodynamicState property struct.

```
#include <externalmedialib.h>
```

Public Attributes

- double Pr
  Prandtl number.
- double T
  Temperature.
- double a
  Velocity of sound.
- double beta
  Isobaric expansion coefficient.
- double cp
  Specific heat capacity cp.
- double cv
  Specific heat capacity cv.
- double d
  Density.
- double ddhp
  Derivative of density wrt enthalpy at constant pressure.
- double ddph
  Derivative of density wrt pressure at constant enthalpy.
- double eta
  Dynamic viscosity.
- double h
  Specific enthalpy.
- double kappa
  Compressibility.
- double lambda
  Thermal conductivity.
- double p
  Pressure.
- int phase
  Phase flag: 2 for two-phase, 1 for one-phase.
- double s
  Specific entropy.

4.3.1 Detailed Description

ExternalThermodynamicState property struct.

The ExternalThermodynamicState propery struct defines all the properties that are computed by external Modelica medium models extending from PartialExternalTwo-PhaseMedium.

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

- D:/Lavoro/ModelicaSVN/ExternalMediaLibrary/Projects/Sources/externalmedialib.-h

4.4 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.

4.4.1 Detailed Description

Fluid constants struct.

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

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

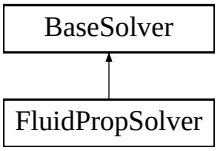
- D:/Lavoro/ModelicaSVN/ExternalMediaLibrary/Projects/Sources/fluidconstants.h

4.5 FluidPropSolver Class Reference

FluidProp solver interface class.

`#include <fluidpropsolver.h>`

Inheritance diagram for FluidPropSolver:



Public Member Functions

- **FluidPropSolver** (const string &mediumName, const string &libraryName, const string &substanceName)
- virtual void setFluidConstants ()

  Set fluid constants.
- virtual void setSat_p (double &p, ExternalSaturationProperties *const properties)

  Set saturation properties from p.
- virtual void setSat_T (double &T, ExternalSaturationProperties *const properties)

  Set saturation properties from T.
- virtual void setState_ph (double &p, double &h, int &phase, ExternalThermodynamicState *const properties)

  Computes the properties of the state vector from p and h.
- virtual void setState_pT (double &p, double &T, ExternalThermodynamicState *const properties)

  Computes the properties of the state vector from p and T.
- virtual void setState_dT (double &d, double &T, int &phase, ExternalThermodynamicState *const properties)
- virtual void setState_ps (double &p, double &s, int &phase, ExternalThermodynamicState *const properties)

  Computes the properties of the state vector from p and s.
- virtual double isentropicEnthalpy (double &p, ExternalThermodynamicState *const properties)

  Compute isentropic enthalpy.

Protected Member Functions

- bool isError (string ErrorMsg)

  Check if FluidProp returned an error.

Protected Attributes

- TFluidProp **FluidProp**

4.5.1 Detailed Description

FluidProp solver interface class.

This class defines a solver object encapsulating a FluidProp object

The class will work if FluidProp is correctly installed, and if the following files, defining the CFluidProp object, are included in the C project:

- FluidProp_IF.h

- FluidProp_IF.cpp
- FluidProp_COM.h These files are developed and maintained by TU Delft and distributed as a part of the FluidProp suite http://www.fluidprop.com

Compilation requires support of the COM libraries: http://en.wikipedia.org/wiki/Component_Object_Model

To instantiate a specific FluidProp fluid, it is necessary to set the libraryName and substanceNames package constants as in the following example:

libraryName = "FluidProp.RefProp"; substanceNames = {"H2O"};

Instead of RefProp, it is possible to indicate TPSI, StanMix, etc. Instead of H2O, it is possible to indicate any supported substance

See also the solvermap.cpp code

4.5.2 Member Function Documentation

4.5.2.1 double FluidPropSolver::isentropicEnthalpy (double & *p*, ExternalThermodynamicState *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

Parameters

p	New pressure
properties	ExternalThermodynamicState property record corresponding to current state

Reimplemented from BaseSolver.

4.5.2.2 void FluidPropSolver::setFluidConstants ( ) [virtual]

Set fluid constants.

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

Must be re-implemented in the specific solver

Reimplemented from BaseSolver.

4.5.2.3 void FluidPropSolver::setSat_p (double & *p*, ExternalSaturationProperties *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 ExternalSaturationProperties propery struct.

Must be re-implemented in the specific solver

Parameters

p	Pressure
properties	ExternalSaturationProperties property struct

Reimplemented from BaseSolver.

4.5.2.4 void FluidPropSolver::setSat_T (double & *T*, ExternalSaturationProperties *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 ExternalSaturationProperties propery struct.

Must be re-implemented in the specific solver

Parameters

T	Temperature
properties	ExternalSaturationProperties property struct

Reimplemented from BaseSolver.

4.5.2.5 void FluidPropSolver::setState_dT (double & *d*, double & *T*, int & *phase*, ExternalThermodynamicState *const *properties*) [virtual]

Note: the phase input is currently not supported according to the standard, the phase input is returned in the state record

Reimplemented from BaseSolver.

4.5.2.6 void FluidPropSolver::setState_ph (double & *p*, double & *h*, int & *phase*, ExternalThermodynamicState *const *properties*) [virtual]

Computes the properties of the state vector from p and h.

Note: the phase input is currently not supported according to the standard, the phase input is returned in the state record

Reimplemented from BaseSolver.

4.5.2.7 void FluidPropSolver::setState_ps (double & *p*, double & *s*, int & *phase*, ExternalThermodynamicState *const *properties*) [virtual]

Computes the properties of the state vector from p and s.

Note: the phase input is currently not supported according to the standard, the phase input is returned in the state record

Reimplemented from BaseSolver.

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

- D:/Lavoro/ModelicaSVN/ExternalMediaLibrary/Projects/Sources/fluidpropsolver.h
- D:/Lavoro/ModelicaSVN/ExternalMediaLibrary/Projects/Sources/fluidpropsolver.cpp

4.6 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.

4.6.1 Detailed Description

Solver map.

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

Francesco Casella, Christoph Richter, Roberto Bonifetto 2006-2012 Copyright - Politecnico di Milano, TU Braunschweig, Politecnico di Torino

4.6.2 Member Function Documentation

4.6.2.1 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

4.6.2.2 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:

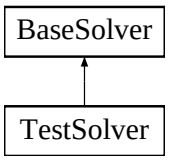
- D:/Lavoro/ModelicaSVN/ExternalMediaLibrary/Projects/Sources/solvermap.h
- D:/Lavoro/ModelicaSVN/ExternalMediaLibrary/Projects/Sources/solvermap.cpp

4.7 TestSolver Class Reference

Test solver class.

`#include <testsolver.h>`

Inheritance diagram for TestSolver:



Public Member Functions

- **TestSolver** (const string &mediumName, const string &libraryName, const string &substanceName)
- virtual void setFluidConstants ()

  Set fluid constants.
- virtual void setSat_p (double &p, ExternalSaturationProperties *const properties)

  Set saturation properties from p.
- virtual void setSat_T (double &T, ExternalSaturationProperties *const properties)

  Set saturation properties from T.
- virtual void setState_ph (double &p, double &h, int &phase, ExternalThermodynamicState *const properties)

  Set state from p, h, and phase.
- virtual void setState_pT (double &p, double &T, ExternalThermodynamicState *const properties)

  Set state from p and T.
- virtual void setState_dT (double &d, double &T, int &phase, ExternalThermodynamicState *const properties)

  Set state from d, T, and phase.
- virtual void setState_ps (double &p, double &s, int &phase, ExternalThermodynamicState *const properties)

  Set state from p, s, and phase.

4.7.1 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, Roberto Bonifetto 2006-2012 Copyright - Politecnico di Milano, TU Braunschweig, Politecnico di Torino

4.7.2 Member Function Documentation

4.7.2.1 void TestSolver::setFluidConstants ( ) [virtual]

Set fluid constants.

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

Must be re-implemented in the specific solver

Reimplemented from BaseSolver.

4.7.2.2 void TestSolver::setSat_p (double & *p*, ExternalSaturationProperties *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 ExternalSaturationProperties propery struct.

Must be re-implemented in the specific solver

Parameters

p	Pressure
properties	ExternalSaturationProperties property struct

Reimplemented from BaseSolver.

4.7.2.3 void TestSolver::setSat_T (double & *T*, ExternalSaturationProperties *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 ExternalSaturationProperties propery struct.

Must be re-implemented in the specific solver

Parameters

T	Temperature
properties	ExternalSaturationProperties property struct

Reimplemented from BaseSolver.

4.7.2.4 void TestSolver::setState_dT (double & *d*, double & *T*, int & *phase*, ExternalThermodynamicState *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 ExternalThermodynamicState 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	ExternalThermodynamicState property struct

Reimplemented from BaseSolver.

4.7.2.5 void TestSolver::setState_ph (double & *p*, double & *h*, int & *phase*, ExternalThermodynamicState *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 ExternalThermodynamicState 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	ExternalThermodynamicState property struct

Reimplemented from BaseSolver.

4.7.2.6 void TestSolver::setState_ps (double & *p*, double & *s*, int & *phase*, ExternalThermodynamicState *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 ExternalThermodynamicState 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	ExternalThermodynamicState property struct

Reimplemented from BaseSolver.

4.7.2.7 void TestSolver::setState_pT (double & *p*, double & *T*, ExternalThermodynamicState *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 ExternalThermodynamicState property struct.

Must be re-implemented in the specific solver

Parameters

p	Pressure
T	Temperature
properties	ExternalThermodynamicState property struct

Reimplemented from BaseSolver.

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

- D:/Lavoro/ModelicaSVN/ExternalMediaLibrary/Projects/Sources/testsolver.h
- D:/Lavoro/ModelicaSVN/ExternalMediaLibrary/Projects/Sources/testsolver.cpp

4.8 TFluidProp Class Reference

Public Member Functions

- void **CreateObject** (string ModelName, string *ErrorMsg)
- void **ReleaseObjects** ()
- void **SetFluid** (string ModelName, int nComp, string *Comp, double *Conc, string *ErrorMsg)
- void **GetFluid** (string *ModelName, int *nComp, string *Comp, double *Conc, bool CompInfo=true)
- void **GetFluidNames** (string LongShort, string ModelName, int *nFluids, string *FluidNames, string *ErrorMsg)
- double **Pressure** (string InputSpec, double Input1, double Input2, string *ErrorMsg)
- double **Temperature** (string InputSpec, double Input1, double Input2, string *ErrorMsg)
- double **SpecVolume** (string InputSpec, double Input1, double Input2, string *ErrorMsg)
- double **Density** (string InputSpec, double Input1, double Input2, string *ErrorMsg)
- double **Enthalpy** (string InputSpec, double Input1, double Input2, string *ErrorMsg)

- double **Entropy** (string InputSpec, double Input1, double Input2, string *ErrorMsg)
- double **IntEnergy** (string InputSpec, double Input1, double Input2, string *ErrorMsg)
- double **VaporQual** (string InputSpec, double Input1, double Input2, string *ErrorMsg)
- double * **LiquidCmp** (string InputSpec, double Input1, double Input2, string *ErrorMsg)
- double * **VaporCmp** (string InputSpec, double Input1, double Input2, string *ErrorMsg)
- double **HeatCapV** (string InputSpec, double Input1, double Input2, string *ErrorMsg)
- double **HeatCapP** (string InputSpec, double Input1, double Input2, string *ErrorMsg)
- double **SoundSpeed** (string InputSpec, double Input1, double Input2, string *ErrorMsg)
- double **Alpha** (string InputSpec, double Input1, double Input2, string *ErrorMsg)
- double **Beta** (string InputSpec, double Input1, double Input2, string *ErrorMsg)
- double **Chi** (string InputSpec, double Input1, double Input2, string *ErrorMsg)
- double **Fi** (string InputSpec, double Input1, double Input2, string *ErrorMsg)
- double **Ksi** (string InputSpec, double Input1, double Input2, string *ErrorMsg)
- double **Psi** (string InputSpec, double Input1, double Input2, string *ErrorMsg)
- double **Zeta** (string InputSpec, double Input1, double Input2, string *ErrorMsg)
- double **Theta** (string InputSpec, double Input1, double Input2, string *ErrorMsg)
- double **Kappa** (string InputSpec, double Input1, double Input2, string *ErrorMsg)
- double **Gamma** (string InputSpec, double Input1, double Input2, string *ErrorMsg)
- double **Viscosity** (string InputSpec, double Input1, double Input2, string *ErrorMsg)
- double **ThermCond** (string InputSpec, double Input1, double Input2, string *ErrorMsg)
- void **AllProps** (string InputSpec, double Input1, double Input2, double &P, double &T, double &v, double &d, double &h, double &s, double &u, double &q, double *x, double *y, double &cv, double &cp, double &c, double &alpha, double &beta, double &chi, double &fi, double &ksi, double &psi, double &zeta, double &theta, double &kappa, double &gamma, double &eta, double &lambda, string *ErrorMsg)
- void **AllPropsSat** (string InputSpec, double Input1, double Input2, double &P, double &T, double &v, double &d, double &h, double &s, double &u, double &q, double *x, double *y, double &cv, double &cp, double &c, double &alpha, double &beta, double &chi, double &fi, double &ksi, double &psi, double &zeta, double &theta, double &kappa, double &gamma, double &eta, double &lambda, double &d_liq, double &d_vap, double &h_liq, double &h_vap, double &T_sat, double &dd_liq_dP, double &dd_vap_dP, double &dh_liq_dP, double &dh_vap_dP, double &dT_sat_dP, string *ErrorMsg)
- double **Solve** (string FuncSpec, double FuncVal, string InputSpec, long Target, double FixedVal, double MinVal, double MaxVal, string *ErrorMsg)
- double **Mmol** (string *ErrorMsg)

- double **Tcrit** (string *ErrorMsg)
- double **Pcrit** (string *ErrorMsg)
- double **Tmin** (string *ErrorMsg)
- double **Tmax** (string *ErrorMsg)
- void **AllInfo** (double &Mmol, double &Tcrit, double &Pcrit, double &Tmin, double &Tmax, string *ErrorMsg)
- void **SetUnits** (string UnitSet, string MassOrMole, string Properties, string Units, string *ErrorMsg)
- void **SetRefState** (double T_ref, double P_ref, string *ErrorMsg)

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

- D:/Lavoro/ModelicaSVN/ExternalMediaLibrary/Projects/Sources/FluidProp_IF.H
- D:/Lavoro/ModelicaSVN/ExternalMediaLibrary/Projects/Sources/FluidProp_IF.-cpp