# ExternalMedia

4.1.1

Generated by Doxygen 1.12.0

# Chapter 1

# ExternalMedia

The ExternalMedia library provides a framework for interfacing external codes computing fluid properties to Modelica.Media-compatible component models.

The latest releases of the library can be downloaded here:

- The precompiled Modelica library can be found in the zip-file
- The manual can be downloaded as PDF
- The full source code is also available as compressed file

## 1.1 Library overview

The ExternalMedia library provides a framework for interfacing external codes computing fluid properties to Modelica.Media-compatible component models. The latest 4.1.x and 4.0.x releases are compatible with Modelica Standard Library (MSL) 4.1.x and 4.0.x, while 3.3.x versions are provided for legacy models that still use MSL 3.2.3.

The current version of the library supports pure and pseudo-pure fluids models, possibly two-phase, compliant with the Modelica.Media.Interfaces.PartialTwoPhaseMedium interface. Please have a look at the dedicated introduction section for an in-depth description of the architecture.

The latest releases of the library include built-in access to the open-source CoolProp software and a precompiled interface to the FluidProp commercial software. CoolProp medium models work out of the box without the need of any further installation. FluidProp medium models require to install the FluidProp software with proper licensing to access the media of your interest and to compute the property derivatives, which are required by ExternalMedia. The library works with FluidProp version 3.0 and later. It might work with previous versions of that software, but compatibility is no longer guaranteed. Please refer to the chapter on FluidProp and the dedicated chapter on CoolProp for further details.

The latest releases were tested with Dymola and OpenModelica on Windows and Linux. Support for more tools and operating systems might be added in the future, please let us know if you want to contribute.

You can modify the library to add an interface to your own solver. If your solver is open-source, please contact the developers, so we can add it to the official ExternalMedia library.

## 1.2 Installation instructions for the ExternalMedia library

For OpenModelica, you can install and manage ExternalMedia using the built-in `Package Manager`. Please make sure you are using OpenModelica version 1.25.1 or later to ensure the correct libraries are loaded, in case you want to install more than one version of ExternalMedia on your computer.

For use with Dymola, you can download the zip file with the library and unzip it in your file system. The released library already contains all the pre-compiled binaries for all operating systems, so it should work out of the box.

Install the latest version 3.3.x of External Media if your models still uses Modelica Standard Library 3.2.3, otherwise install the latest version 4.x.x.

If you want to experiment with the code and recompile the libraries, check the compilation instructions.

### 1.2.1 Modelica integration

The Modelica Language Specification mentions annotations for External Libraries and Include Files in `section 12.9.4`. Following the concepts put forward there, the ExternalMedia package provides several pre-compiled shared libraries supporting a selection of operating systems, C-compilers and Modelica tools.

Please open the `package.mo` file inside the `ExternalMedia` folder to load the library. If your Modelica tool is able to find a matching precompiled binary for your configuration, you should now be able to run the examples.

### 1.2.2 Compiling ExternalMedia from sources

ExternalMedia extensively relies of external functions using code from pre-compiled dynamic libraries. The released versions include binaries for Windows and Linux, supporting CoolProp and FluidProp. If you want to experiment with other external codes or operating systems, you can build the ExternalMedia binary libraries yourself. All you need to compile ExternalMedia, besides your C/C++ compiler, is the `CMake software` by Kitware. If you would like to include the CoolProp library, you also need a working Python installation.

Please consult the compilation guide for further instructions and details on how to compile ExternalMedia for different Modelica tools and operating systems.

## 1.3 License

This Modelica package is free software and the use is completely at your own risk; it can be redistributed and/or modified under the terms of the `BSD 3-clause license`.

## 1.4 Development and contribution

ExternalMedia has been around since 2006 and many different people have contributed to it. The history page provides a lot of useful insights and explains how the software became what it is today.

Current main developers:

- `Francesco Casella` started the development in 2006 and coordinates the current development effort.
- `Jorrit Wronski` and Ian Bell took care of the integration of CoolProp in the library and of CMake-based compilation.
- `Federico Terraneo` helped getting the library to work with different Modelica tools and operating systems.

Please report problems using the library `GitHub issue tracker`.

# Chapter 2

# ExternalMedia Change Log

## 2.1 v4.1.1 - 2025/11/03

- Uses Modelica Standard Library 4.1.0
- CoolProp updated to v7.2.0
- Added binaries for macOS ARM (M1/M2)

## 2.2 v4.1.0 - 2025/08/01

- Uses Modelica Standard Library 4.1.0
- CoolProp updated to v6.5.0
- Fixed issue with ModelicaError that causes OMEdit to crash when opening ExternalTwoPhaseMedium

## 2.3 v4.0.0 - 2023/04/19

- Compatible with Modelica Standard Library 4.0.0.
- Works out of the box in Dymola and OpenModelica under Windows and Linux.
- Uses dynamically linked libraries, no dependencies on specific compilers.
- No need to compile the binaries yourself, no special installation procedure required.
- CoolProp updated to v6.4.4 with some extra patches

## 2.4 v3.3.2 - 2023/04/19

- Works out of the box in Dymola and OpenModelica under Windows and Linux.
- Uses dynamically linked libraries, no dependencies on specific compilers.
- No need to compile the binaries yourself, no special installation procedure required.
- CoolProp updated to v6.4.4 with some extra patches

## 2.5 v3.3.1 - 2022/02/17

- Updated CoolProp to v6.4.1.
- Fixed problems with CoolProp interpolation tables.
- Added more precompiled binaries.
- Use git to retrieve the OpenModelica development environment.

## 2.6 v3.3.0 - 2021/05/05

- The first release after a long period of inactivity.
- Added precompiled binaries in subfolders.
- Updated the documentation and restructured the help files.

# Chapter 3

# Compilation guide

## 3.1 Quick-start guide

The heavy-lifting regarding the project configuration is done using the CMake file `CMakeLists.txt`, which makes the `CMake software` a prerequisite for compiling ExternalMedia.

Once you have installed CMake and can access it from a command prompt, you can go to the root folder of the GIT repository and run:

```
cmake -B build -S Projects -DCMAKE_BUILD_TYPE=Release
cmake -B build -S Projects -DCMAKE_BUILD_TYPE=Release
```

NOTE: On Windows to select a 32 or 64 bit build you can append the option `-A Win32` or `-A x64` to the above command.

Please note that there is no typing mistake in the lines above. The current version of ExternalMedia requires you to run the configure step twice. Now you should have a working project configuration and the actual compilation can be triggered using:

```
cmake --build build --config Release --target install
```

By default, the libraries are installed in a subfolder with a name that is determined from the current operating system and the compiler, possible combinations are:

- `Modelica/ExternalMedia ${APP_VERSION}/Resources/Library/win32/vs2015`
- `Modelica/ExternalMedia ${APP_VERSION}/Resources/Library/win64/vs2019`
- `Modelica/ExternalMedia ${APP_VERSION}/Resources/Library/linux64/gcc81`

If you would like to skip the compiler part and make the current configuration the default for the platform, you can use this command below:

```
cmake --build build --config Release --target install-as-default
```

You can override these settings manually using the command line switches for `MODELICA_PLATFORM` and `MODELICA_COMPILER`. The command `cmake -B build -S Projects -DMODELICA_PLATFORM:STRING=mingw64 -DMODELICA_COMPILER:STRING=` would for example configure the installation folder to `Modelica/ExternalMedia ${APP_VERSION}/Resources/Library/mingw64`, which is the preferred search path for OpenModelica that supports side-by-side installations with other compilers and configuration that support other Modelica tools.

## 3.2 Selecting the fluid property libraries

You can disable and enable the FluidProp and the CoolProp integration with command line switches.

The recommended configuration step for Windows systems is

```
cmake -B build -S Projects -DCMAKE_BUILD_TYPE=Release -DFLUIDPROP:BOOL=ON -DCOOLPROP:BOOL=ON
cmake -B build -S Projects -DCMAKE_BUILD_TYPE=Release -DFLUIDPROP:BOOL=ON -DCOOLPROP:BOOL=ON
```

... and for all other systems, you probably want to use

```
cmake -B build -S Projects -DCMAKE_BUILD_TYPE=Release -DFLUIDPROP:BOOL=OFF -DCOOLPROP:BOOL=ON
cmake -B build -S Projects -DCMAKE_BUILD_TYPE=Release -DFLUIDPROP:BOOL=OFF -DCOOLPROP:BOOL=ON
```

## 3.3 Building OpenModelica libraries

Get the OMDEV environment from the git repository:

```
git clone https://openmodelica.org/git/OMDev.git C:/OMDev
```

To install OMDEV in the `C:\OMDev` path, you should start `C:\OMDev\tools\msys\msys.bat`. This gives you a command window that looks like the emulation of a unix prompt. Afterwards, you can run the following commands

```
$ mount d:/Path_to_your_ExternalMediaLibrary_working_copy /ExternalMediaLibrary
$ cd /ExternalMediaLibrary/
$ cmake -B build -S Projects -G "MSYS Makefiles" -DCMAKE_BUILD_TYPE=Release
$ cmake --build build --target install
```

This will build the dynamic library and copy it and the `externalmedia.h` header files in the Resource directories of the Modelica packages, so it can be used right away by just loading the Modelica package in OMC.

# Chapter 4

# CoolProp in ExternalMedia

ExternalMedia provides direct access to the open-source `CoolProp` software, which is integrated in the compiled binaries of the library, so it doesn't need any separate installation.

Please refer to the User Guide of the ExternalMedia library for instructions on how to access specific fluid models of CoolProp. Some ready-to-use examples are provided in the Examples package.

# Chapter 5

# Using the pre-packaged releases with FluidProp

Download and install the latest version of `FluidProp`. 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 the different libraries supported by FluidProp, by extending the External↩ Media.Media.FluidPropMedium package. Please note that only single-component fluids are supported. Set libraryName to "FluidProp.RefProp", "FluidProp.StanMix", "FluidProp.TPSI", or "FluidProp.IF97", 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 model IF97 is already available natively in Modelica.Media as Water.StandardWater, which is much faster than the FluidProp version. If you need ideal gas models (single-component or mixtures), use the medium packages contained in Modelica.Media.IdealGases.

# Chapter 6

# ExternalMedia History

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 were: maximizing the efficiency of the code and minimizing the amount of extra code required to use your own external code within the framework. The library was described in `this paper`.

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 was shipped with an interface to the FluidProp software, provided at the time by TU Delft.

The library was then restructured in 2012 by Francesco Casella and Roberto Bonifetto. The main idea was to get rid of the hidden cache and of the unique ID references and use the Modelica state records instead for caching. In this way, all optimizations performed by Modelica tools are guaranteed to give correct results, also in case of implicit equations, which was previously not the case. The library was mainly used with the Dymola tool, although some limited support for OpenModelica was given.

In 2013, the open-source CoolProp package was integrated in the library, thus providing built-in access to a wide range of fluids.

In 2014, Ian Bell initially provided some makefiles to automatically compile different versions of the library. Later on, Jorrit Wronski added support for CMake scripts.

In 2021, Jorrit Wronksi implemented the entire CMake build pipeline within the GitHub CI environment. New annotations introduced in Modelica 3.4 now allow to build and ship the ExternalMedia package with built-in pre-compiled libraries for many different operating systems, C-compilers, and Modelica tools.

In 2023, Federico Terraneo helped switching from statically linked to dynamically linked compiled binaries, while keeping access to the ModelicaError function to properly handle run-time and compile time errors generated by external media models. This eliminates the need to have different binaries for each compiler, now only one DLL/shared library is needed for each operating system. At long last, ExternalMedia can be used out of the box with different Modelica tools and operating systems, without the need of arcane installation procedures.

# Chapter 7

# An introduction to ExternalMedia

There are two ways to use this library. The easiest way is to use the released download archives. These files come with batteries included since the fluid property library `CoolProp` is part of the code already and it includes a pre-compiled interface to the `FluidProp tool`. 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.

Please refer to the chapter on FluidProp and the dedicated chapter on CoolProp for details.

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 refer to the compilation guide for details regarding the creation of binary files from the source code.

## 7.1 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 ExternalTwoPhase↩ Medium 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.

## 7.2 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.

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 shared library. 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. See the compilation guidefor details.

# Chapter 8

# Hierarchical Index

## 8.1 Class Hierarchy

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

# Chapter 9

# Class Index

## 9.1 Class List

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

# Chapter 10

# File Index

## 10.1 File List

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

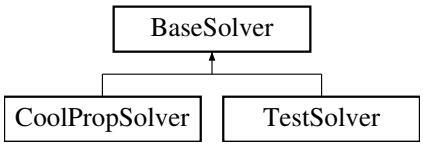
# Chapter 11

# Class Documentation

## 11.1 BaseSolver Class Reference

`#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 void setState_hs (double &h, double &s, int &phase, ExternalThermodynamicState *const properties)

  *Set state from h, s, and phase.*
- virtual double partialDeriv_state (const string &of, const string &wrt, const string &cst, ExternalThermodynamicState *const properties)

  *Compute partial derivative from a populated state record.*
- 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
- string libraryName
- string substanceName

### Protected Attributes

- FluidConstants _fluidConstants

### 11.1.1 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

### 11.1.2 Constructor & Destructor Documentation

#### 11.1.2.1 BaseSolver()

```
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 |

#### 11.1.2.2 ~BaseSolver()

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

Destructor.

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

### 11.1.3 Member Function Documentation

#### 11.1.3.1 a()

```
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 |

Reimplemented in CoolPropSolver.

#### 11.1.3.2 beta()

```
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 |

Reimplemented in CoolPropSolver.

#### 11.1.3.3 computeDerivatives()

```
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 |

#### 11.1.3.4 cp()

```
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 |

Reimplemented in CoolPropSolver.

#### 11.1.3.5 cv()

```
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 |

Reimplemented in CoolPropSolver.

#### 11.1.3.6 d()

```
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 |

Reimplemented in CoolPropSolver.

#### 11.1.3.7 d_der()

```
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 |

Reimplemented in CoolPropSolver.

#### 11.1.3.8 ddhp()

```
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 |

Reimplemented in CoolPropSolver.

#### 11.1.3.9 ddldp()

```
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 |

Reimplemented in CoolPropSolver.

#### 11.1.3.10 ddph()

```
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 |

Reimplemented in CoolPropSolver.

#### 11.1.3.11 ddvdp()

```
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 |

Reimplemented in CoolPropSolver.

#### 11.1.3.12 dhldp()

```
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 |

Reimplemented in CoolPropSolver.

#### 11.1.3.13 dhvdp()

```
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 |

Reimplemented in CoolPropSolver.

#### 11.1.3.14 dl()

```
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 |

Reimplemented in CoolPropSolver.

#### 11.1.3.15 dTp()

```
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 |

Reimplemented in CoolPropSolver.

#### 11.1.3.16 dv()

```
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 |

Reimplemented in CoolPropSolver.

#### 11.1.3.17 eta()

```
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 |

Reimplemented in CoolPropSolver.

#### 11.1.3.18 h()

```
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 |

Reimplemented in CoolPropSolver.

#### 11.1.3.19 hl()

```
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 |

Reimplemented in CoolPropSolver.

#### 11.1.3.20 hv()

```
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 |

Reimplemented in CoolPropSolver.

#### 11.1.3.21 isentropicEnthalpy()

```
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 CoolPropSolver.

#### 11.1.3.22 kappa()

```
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 |

Reimplemented in CoolPropSolver.

#### 11.1.3.23 lambda()

```
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 |

Reimplemented in CoolPropSolver.

#### 11.1.3.24 p()

```
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 |

Reimplemented in CoolPropSolver.

#### 11.1.3.25 partialDeriv_state()

```
double BaseSolver::partialDeriv_state (
            const string & of,
            const string & wrt,
            const string & cst,
            ExternalThermodynamicState *const properties)  [virtual]
```

Compute partial derivative from a populated state record.

This function computes the derivative of the specified input. Note that it requires a populated state record as input.

**Parameters**

| | |
|---|---|
| *of* | Property to differentiate |
| *wrt* | Property to differentiate in |
| *cst* | Property to remain constant |
| *state* | Pointer to input values in state record |
| *mediumName* | Medium name |
| *libraryName* | Library name |
| *substanceName* | Substance name |

Reimplemented in CoolPropSolver.

#### 11.1.3.26 phase()

```
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 |

Reimplemented in CoolPropSolver.

#### 11.1.3.27 Pr()

```
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 |

Reimplemented in CoolPropSolver.

#### 11.1.3.28 psat()

```
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 |

Reimplemented in CoolPropSolver.

#### 11.1.3.29 s()

```
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 |

Reimplemented in CoolPropSolver.

#### 11.1.3.30 setBubbleState()

```
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 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**

| | |
|---|---|
| *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 |

Reimplemented in CoolPropSolver.

#### 11.1.3.31 setDewState()

```
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) |
| *dewProperties* | ExternalThermodynamicState record where to write the dew point properties |

Reimplemented in CoolPropSolver.

#### 11.1.3.32 setFluidConstants()

```
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 CoolPropSolver, and TestSolver.

#### 11.1.3.33 setSat_p()

```
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 CoolPropSolver, and TestSolver.

#### 11.1.3.34 setSat_T()

```
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 CoolPropSolver, and TestSolver.

#### 11.1.3.35 setState_dT()

```
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 CoolPropSolver, and TestSolver.

#### 11.1.3.36 setState_hs()

```
void BaseSolver::setState_hs (
            double & h,
            double & s,
            int & phase,
            ExternalThermodynamicState *const properties)  [virtual]
```

Set state from h, s, and phase.

This function sets the thermodynamic state record for the given specific enthalpy 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**

| | |
|---|---|
| *h* | Specific enthalpy |
| *s* | Specific entropy |
| *phase* | Phase (2 for two-phase, 1 for one-phase, 0 if not known) |
| *properties* | ExternalThermodynamicState property struct |

Reimplemented in CoolPropSolver.

#### 11.1.3.37 setState_ph()

```
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 CoolPropSolver, and TestSolver.

#### 11.1.3.38 setState_ps()

```
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 CoolPropSolver, and TestSolver.

#### 11.1.3.39 setState_pT()

```
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 CoolPropSolver, and TestSolver.

#### 11.1.3.40 sigma()

```
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 |

Reimplemented in CoolPropSolver.

#### 11.1.3.41 sl()

```
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 |

Reimplemented in CoolPropSolver.

#### 11.1.3.42 sv()

```
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 |

Reimplemented in CoolPropSolver.

#### 11.1.3.43 T()

```
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 |

Reimplemented in CoolPropSolver.

#### 11.1.3.44 Tsat()

```
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 |

Reimplemented in CoolPropSolver.

### 11.1.4 Member Data Documentation

#### 11.1.4.1 _fluidConstants

```
FluidConstants BaseSolver::_fluidConstants  [protected]
```

Fluid constants

#### 11.1.4.2 libraryName

```
string BaseSolver::libraryName
```

Library name

#### 11.1.4.3 mediumName

```
string BaseSolver::mediumName
```

Medium name

#### 11.1.4.4 substanceName

```
string BaseSolver::substanceName
```

Substance name

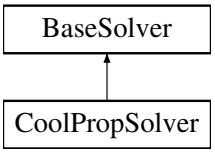
The documentation for this class was generated from the following files:

- Sources/basesolver.h
- Sources/basesolver.cpp

## 11.2 CoolPropSolver Class Reference

```
#include <coolpropsolver.h>
```

Inheritance diagram for CoolPropSolver:



### Public Member Functions

- **CoolPropSolver** (const std::string &mediumName, const std::string &libraryName, const std::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 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 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 void setState_hs (double &h, double &s, int &phase, ExternalThermodynamicState *const properties)

  *Set state from h, s, and phase.*
- virtual double partialDeriv_state (const string &of, const string &wrt, const string &cst, ExternalThermodynamicState *const properties)

  *Compute partial derivative from a populated state record.*
- 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 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 double psat (ExternalSaturationProperties *const properties)

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

  *Compute saturation temperature.*

## Public Member Functions inherited from BaseSolver

- 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 bool computeDerivatives (ExternalThermodynamicState ∗const properties)

  *Compute derivatives.*

#### Protected Member Functions

- virtual void postStateChange (ExternalThermodynamicState ∗const properties)
- long **makeDerivString** (const string &of, const string &wrt, const string &cst)
- double **interp_linear** (double Q, double valueL, double valueV)

  *Interpolation routines.*
- double **interp_recip** (double Q, double valueL, double valueV)

#### Protected Attributes

- shared_ptr< CoolProp::AbstractState > **state**
- bool **enable_TTSE**
- bool **enable_BICUBIC**
- bool **calc_transport**
- bool **extend_twophase**
- bool **isCompressible**
- int **debug_level**
- double **twophase_derivsmoothing_xend**
- double **rho_smoothing_xend**
- double **_p_eps**
- double **_delta_h**
- ExternalSaturationProperties **_satPropsClose2Crit**

### Protected Attributes inherited from BaseSolver

- FluidConstants _fluidConstants

#### Additional Inherited Members

### Public Attributes inherited from BaseSolver

- string mediumName
- string libraryName
- string substanceName

### 11.2.1 Detailed Description

CoolProp solver class

This class defines a solver that calls out to the open-source CoolProp property database and is partly inspired by the fluidpropsolver that was part of the first ExternalMedia release.

libraryName = "CoolProp";

Ian Bell ( ian.h.bell@gmail.com) University of Liege, Liege, Belgium

Jorrit Wronski ( jowr@mek.dtu.dk) Technical University of Denmark, Kgs. Lyngby, Denmark

2012-2014

### 11.2.2 Member Function Documentation

#### 11.2.2.1 a()

```
double CoolPropSolver::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 |

Reimplemented from BaseSolver.

#### 11.2.2.2 beta()

```
double CoolPropSolver::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 |

Reimplemented from BaseSolver.

#### 11.2.2.3 cp()

```
double CoolPropSolver::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 |

Reimplemented from BaseSolver.

#### 11.2.2.4 cv()

```
double CoolPropSolver::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 |

Reimplemented from BaseSolver.

#### 11.2.2.5 d()

```
double CoolPropSolver::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 |

Reimplemented from BaseSolver.

#### 11.2.2.6 d_der()

```
double CoolPropSolver::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 |

Reimplemented from BaseSolver.

#### 11.2.2.7 ddhp()

```
double CoolPropSolver::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 |

Reimplemented from BaseSolver.

#### 11.2.2.8 ddldp()

```
double CoolPropSolver::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 |

Reimplemented from BaseSolver.

#### 11.2.2.9 ddph()

```
double CoolPropSolver::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 |

Reimplemented from BaseSolver.

#### 11.2.2.10 ddvdp()

```
double CoolPropSolver::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 |

Reimplemented from BaseSolver.

#### 11.2.2.11 dhldp()

```
double CoolPropSolver::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 |

Reimplemented from BaseSolver.

#### 11.2.2.12 dhvdp()

```
double CoolPropSolver::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 |

Reimplemented from BaseSolver.

#### 11.2.2.13 dl()

```
double CoolPropSolver::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 |

Reimplemented from BaseSolver.

#### 11.2.2.14 dTp()

```
double CoolPropSolver::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 |

Reimplemented from BaseSolver.

#### 11.2.2.15 dv()

```
double CoolPropSolver::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 |

Reimplemented from BaseSolver.

#### 11.2.2.16 eta()

```
double CoolPropSolver::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 |

Reimplemented from BaseSolver.

#### 11.2.2.17 h()

```
double CoolPropSolver::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 |

Reimplemented from BaseSolver.

#### 11.2.2.18 hl()

```
double CoolPropSolver::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 |

Reimplemented from BaseSolver.

#### 11.2.2.19 hv()

```
double CoolPropSolver::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 |

Reimplemented from BaseSolver.

#### 11.2.2.20 isentropicEnthalpy()

```
double CoolPropSolver::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 from BaseSolver.

#### 11.2.2.21 kappa()

```
double CoolPropSolver::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 |

Reimplemented from BaseSolver.

#### 11.2.2.22 lambda()

```
double CoolPropSolver::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 |

Reimplemented from BaseSolver.

#### 11.2.2.23 p()

```
double CoolPropSolver::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 |

Reimplemented from BaseSolver.

#### 11.2.2.24 partialDeriv_state()

```
double CoolPropSolver::partialDeriv_state (
            const string & of,
            const string & wrt,
            const string & cst,
            ExternalThermodynamicState *const properties)  [virtual]
```

Compute partial derivative from a populated state record.

This function computes the derivative of the specified input. Note that it requires a populated state record as input.

**Parameters**

| | |
|---|---|
| *of* | Property to differentiate |
| *wrt* | Property to differentiate in |
| *cst* | Property to remain constant |
| *state* | Pointer to input values in state record |
| *mediumName* | Medium name |
| *libraryName* | Library name |
| *substanceName* | Substance name |

Reimplemented from BaseSolver.

#### 11.2.2.25 phase()

```
int CoolPropSolver::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 |

Reimplemented from BaseSolver.

#### 11.2.2.26 postStateChange()

```
void CoolPropSolver::postStateChange (
            ExternalThermodynamicState *const properties)  [protected], [virtual]
```

Some common code to avoid pitfalls from incompressibles

#### 11.2.2.27 Pr()

```
double CoolPropSolver::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 |

Reimplemented from BaseSolver.

#### 11.2.2.28 psat()

```
double CoolPropSolver::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 |

Reimplemented from BaseSolver.

#### 11.2.2.29 s()

```
double CoolPropSolver::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 |

Reimplemented from BaseSolver.

#### 11.2.2.30 setBubbleState()

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

Set bubble state.

Reimplemented from BaseSolver.

#### 11.2.2.31 setDewState()

```
void CoolPropSolver::setDewState (
            ExternalSaturationProperties *const properties,
            int phase,
            ExternalThermodynamicState *const bubbleProperties)  [virtual]
```

Set dew state.

Reimplemented from BaseSolver.

#### 11.2.2.32 setFluidConstants()

```
void CoolPropSolver::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.

#### 11.2.2.33 setSat_p()

```
void CoolPropSolver::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.

#### 11.2.2.34 setSat_T()

```
void CoolPropSolver::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.

#### 11.2.2.35 setState_dT()

```
void CoolPropSolver::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.

#### 11.2.2.36 setState_hs()

```
void CoolPropSolver::setState_hs (
            double & h,
            double & s,
            int & phase,
            ExternalThermodynamicState *const properties)  [virtual]
```

Set state from h, s, and phase.

This function sets the thermodynamic state record for the given specific enthalpy 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**

| | |
|---|---|
| *h* | Specific enthalpy |
| *s* | Specific entropy |
| *phase* | Phase (2 for two-phase, 1 for one-phase, 0 if not known) |
| *properties* | ExternalThermodynamicState property struct |

Reimplemented from BaseSolver.

#### 11.2.2.37 setState_ph()

```
void CoolPropSolver::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.

#### 11.2.2.38 setState_ps()

```
void CoolPropSolver::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.

#### 11.2.2.39 setState_pT()

```
void CoolPropSolver::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.

#### 11.2.2.40 sigma()

```
double CoolPropSolver::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 |

Reimplemented from BaseSolver.

#### 11.2.2.41 sl()

```
double CoolPropSolver::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 |

Reimplemented from BaseSolver.

#### 11.2.2.42 sv()

```
double CoolPropSolver::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 |

Reimplemented from BaseSolver.

#### 11.2.2.43 T()

```
double CoolPropSolver::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 |

Reimplemented from BaseSolver.

#### 11.2.2.44 Tsat()

```
double CoolPropSolver::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 |

Reimplemented from BaseSolver.

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

- Sources/coolpropsolver.h
- Sources/coolpropsolver.cpp

## 11.3 ExternalSaturationProperties Struct Reference

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

**Public Attributes**

- double Tsat
- double dTp
- double ddldp
- double ddvdp
- double dhldp
- double dhvdp
- double dl
- double dv
- double hl
- double hv
- double psat
- double sigma
- double sl
- double sv

### 11.3.1 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. Note: the exported interface functions use typeless void *sat instead of ExternalSaturationProperties *sat as Modelica does not treat external struct types so far.

### 11.3.2 Member Data Documentation

#### 11.3.2.1 ddldp

```
double ExternalSaturationProperties::ddldp
```

Derivative of dls wrt pressure

#### 11.3.2.2 ddvdp

```
double ExternalSaturationProperties::ddvdp
```

Derivative of dvs wrt pressure

#### 11.3.2.3 dhldp

```
double ExternalSaturationProperties::dhldp
```

Derivative of hls wrt pressure

#### 11.3.2.4 dhvdp

```
double ExternalSaturationProperties::dhvdp
```

Derivative of hvs wrt pressure

#### 11.3.2.5 dl

```
double ExternalSaturationProperties::dl
```

Density at bubble line (for pressure ps)

#### 11.3.2.6 dTp

```
double ExternalSaturationProperties::dTp
```

Derivative of Ts wrt pressure

#### 11.3.2.7 dv

```
double ExternalSaturationProperties::dv
```

Density at dew line (for pressure ps)

#### 11.3.2.8 hl

```
double ExternalSaturationProperties::hl
```

Specific enthalpy at bubble line (for pressure ps)

#### 11.3.2.9 hv

```
double ExternalSaturationProperties::hv
```

Specific enthalpy at dew line (for pressure ps)

#### 11.3.2.10 psat

```
double ExternalSaturationProperties::psat
```

Saturation pressure

#### 11.3.2.11 sigma

```
double ExternalSaturationProperties::sigma
```

Surface tension

#### 11.3.2.12 sl

```
double ExternalSaturationProperties::sl
```

Specific entropy at bubble line (for pressure ps)

#### 11.3.2.13 sv

```
double ExternalSaturationProperties::sv
```

Specific entropy at dew line (for pressure ps)

#### 11.3.2.14 Tsat

```
double ExternalSaturationProperties::Tsat
```

Saturation temperature

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

- Sources/externalmedialib.h

## 11.4 ExternalThermodynamicState Struct Reference

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

### Public Attributes

- double T
- double a
- double beta
- double cp
- double cv
- double d
- double ddhp
- double ddph
- double eta
- double h
- double kappa
- double lambda
- double p
- int phase
- double s

### 11.4.1 Detailed Description

ExternalThermodynamicState property struct

The ExternalThermodynamicState propery struct defines all the properties that are computed by external Modelica medium models extending from PartialExternalTwoPhaseMedium. Note: the exported interface functions use type-less void *state instead of ExternalThermodynamicState *state as Modelica does not treat external struct types so far.

### 11.4.2 Member Data Documentation

#### 11.4.2.1 a

```
double ExternalThermodynamicState::a
```

Velocity of sound

#### 11.4.2.2 beta

```
double ExternalThermodynamicState::beta
```

Isobaric expansion coefficient

#### 11.4.2.3 cp

```
double ExternalThermodynamicState::cp
```

Specific heat capacity cp

#### 11.4.2.4 cv

```
double ExternalThermodynamicState::cv
```

Specific heat capacity cv

#### 11.4.2.5 d

```
double ExternalThermodynamicState::d
```

Density

#### 11.4.2.6 ddhp

```
double ExternalThermodynamicState::ddhp
```

Derivative of density wrt enthalpy at constant pressure

#### 11.4.2.7 ddph

```
double ExternalThermodynamicState::ddph
```

Derivative of density wrt pressure at constant enthalpy

#### 11.4.2.8 eta

```
double ExternalThermodynamicState::eta
```

Dynamic viscosity

#### 11.4.2.9 h

```
double ExternalThermodynamicState::h
```

Specific enthalpy

#### 11.4.2.10 kappa

```
double ExternalThermodynamicState::kappa
```

Compressibility

#### 11.4.2.11 lambda

```
double ExternalThermodynamicState::lambda
```

Thermal conductivity

#### 11.4.2.12 p

```
double ExternalThermodynamicState::p
```

Pressure

#### 11.4.2.13 phase

```
int ExternalThermodynamicState::phase
```

Phase flag: 2 for two-phase, 1 for one-phase

#### 11.4.2.14 s

```
double ExternalThermodynamicState::s
```

Specific entropy

#### 11.4.2.15 T

```
double ExternalThermodynamicState::T
```

Temperature

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

- Sources/externalmedialib.h

## 11.5 FluidConstants Struct Reference

```
#include <fluidconstants.h>
```

### Public Member Functions

- FluidConstants ()

### Public Attributes

- double MM
- double pc
- double Tc
- double dc
- double hc
- double sc

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

### 11.5.2 Constructor & Destructor Documentation

#### 11.5.2.1 FluidConstants()

```
FluidConstants::FluidConstants ()  [inline]
```

Constructor.

The constructor only initializes the variables.

### 11.5.3 Member Data Documentation

#### 11.5.3.1 dc

```
double FluidConstants::dc
```

Density at critical point

#### 11.5.3.2 hc

```
double FluidConstants::hc
```

Specific enthalpy at critical point

#### 11.5.3.3 MM

```
double FluidConstants::MM
```

Molar mass

#### 11.5.3.4 pc

```
double FluidConstants::pc
```

Pressure at critical point

#### 11.5.3.5 sc

```
double FluidConstants::sc
```

Specific entropy at critical point

#### 11.5.3.6 Tc

```
double FluidConstants::Tc
```

Temperature at critical point

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

- Sources/fluidconstants.h

## 11.6 SolverMap Class Reference

```
#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

## 11.6.1 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

## 11.6.2 Member Function Documentation

### 11.6.2.1 getSolver()

```
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 |

### 11.6.2.2 solverKey()

```
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.

### 11.6.3 Member Data Documentation

#### 11.6.3.1 _solvers

```
map< string, BaseSolver * > SolverMap::_solvers  [static], [protected]
```

Map for all solver instances identified by the SolverKey

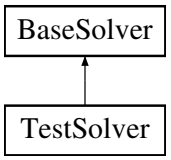
The documentation for this class was generated from the following files:

- Sources/solvermap.h
- Sources/solvermap.cpp

## 11.7 TestSolver Class Reference

```
#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.*

## Public Member Functions inherited from BaseSolver

- 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 setState_hs (double &h, double &s, int &phase, ExternalThermodynamicState *const properties)

  *Set state from h, s, and phase.*
- virtual double partialDeriv_state (const string &of, const string &wrt, const string &cst, ExternalThermodynamicState *const properties)

  *Compute partial derivative from a populated state record.*
- 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 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.*

**Additional Inherited Members**

## Public Attributes inherited from BaseSolver

- string mediumName
- string libraryName
- string substanceName

## Protected Attributes inherited from BaseSolver

- FluidConstants _fluidConstants

### 11.7.1 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";



### 11.7.2 Member Function Documentation

#### 11.7.2.1 setFluidConstants()

```
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.

#### 11.7.2.2 setSat_p()

```
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.

#### 11.7.2.3 setSat_T()

```
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.

#### 11.7.2.4 setState_dT()

```
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.

#### 11.7.2.5 setState_ph()

```
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.

#### 11.7.2.6 setState_ps()

```
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.

#### 11.7.2.7 setState_pT()

```
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:

- Sources/testsolver.h
- Sources/testsolver.cpp

## 11.8 TFluidProp Class Reference

### Public Member Functions

- bool **IsValid** ()
- 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 *Error↩ Msg)
- void **GetCompSet** (string ModelName, int *nComps, string *CompSet, 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 Min↩ Val, 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)
- void **GetVersion** (string ModelName, int *version)
- double * **FugaCoef** (string InputSpec, double Input1, double Input2, string *ErrorMsg)
- double **SurfTens** (string InputSpec, double Input1, double Input2, string *ErrorMsg)
- double **GibbsEnergy** (string InputSpec, double Input1, double Input2, string *ErrorMsg)
- void **CapeOpenDeriv** (string InputSpec, double Input1, double Input2, double *v, double *h, double *s, double *G, double *Inphi, string *ErrorMsg)
- double * **SpecVolume_Deriv** (string InputSpec, double Input1, double Input2, string *ErrorMsg)
- double * **Enthalpy_Deriv** (string InputSpec, double Input1, double Input2, string *ErrorMsg)
- double * **Entropy_Deriv** (string InputSpec, double Input1, double Input2, string *ErrorMsg)
- double * **GibbsEnergy_Deriv** (string InputSpec, double Input1, double Input2, string *ErrorMsg)
- double * **FugaCoef_Deriv** (string InputSpec, double Input1, double Input2, string *ErrorMsg)

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

- Sources/FluidProp_IF.h
- Sources/FluidProp_IF.cpp

# Chapter 12

# File Documentation

## 12.1 basesolver.h

```
00001 #ifndef BASESOLVER_H_
00002 #define BASESOLVER_H_
00003
00004 #include "include.h"
00005 #include "fluidconstants.h"
00006 #include "externalmedialib.h"
00007 #include <stdio.h>
00008 #include <stdlib.h>
00009 #include <string.h>
00010
00011 struct FluidConstants;
00012
00025 class BaseSolver{
00026 public:
00027     BaseSolver(const string &mediumName, const string &libraryName, const string &substanceName);
00028     virtual ~BaseSolver();
00029
00030     double molarMass() const;
00031     double criticalTemperature() const;
00032     double criticalPressure() const;
00033     double criticalMolarVolume() const;
00034     double criticalDensity() const;
00035     double criticalEnthalpy() const;
00036     double criticalEntropy() const;
00037
00038     virtual void setFluidConstants();
00039
00040     virtual void setState_ph(double &p, double &h, int &phase, ExternalThermodynamicState *const
      properties);
00041     virtual void setState_pT(double &p, double &T, ExternalThermodynamicState *const properties);
00042     virtual void setState_dT(double &d, double &T, int &phase, ExternalThermodynamicState *const
      properties);
00043     virtual void setState_ps(double &p, double &s, int &phase, ExternalThermodynamicState *const
      properties);
00044     virtual void setState_hs(double &h, double &s, int &phase, ExternalThermodynamicState *const
      properties);
00045
00046     virtual double partialDeriv_state(const string &of, const string &wrt, const string &cst,
      ExternalThermodynamicState *const properties);
00047
00048     virtual double Pr(ExternalThermodynamicState *const properties);
00049     virtual double T(ExternalThermodynamicState *const properties);
00050     virtual double a(ExternalThermodynamicState *const properties);
00051     virtual double beta(ExternalThermodynamicState *const properties);
00052     virtual double cp(ExternalThermodynamicState *const properties);
00053     virtual double cv(ExternalThermodynamicState *const properties);
00054     virtual double d(ExternalThermodynamicState *const properties);
00055     virtual double ddhp(ExternalThermodynamicState *const properties);
00056     virtual double ddph(ExternalThermodynamicState *const properties);
00057     virtual double eta(ExternalThermodynamicState *const properties);
00058     virtual double h(ExternalThermodynamicState *const properties);
00059     virtual double kappa(ExternalThermodynamicState *const properties);
00060     virtual double lambda(ExternalThermodynamicState *const properties);
00061     virtual double p(ExternalThermodynamicState *const properties);
00062     virtual int phase(ExternalThermodynamicState *const properties);
00063     virtual double s(ExternalThermodynamicState *const properties);
00064     virtual double d_der(ExternalThermodynamicState *const properties);
00065     virtual double isentropicEnthalpy(double &p, ExternalThermodynamicState *const properties);
```

```
00066
00067     virtual void setSat_p(double &p, ExternalSaturationProperties *const properties);
00068     virtual void setSat_T(double &T, ExternalSaturationProperties *const properties);
00069
00070     virtual void setBubbleState(ExternalSaturationProperties *const properties, int phase,
00071                                 ExternalThermodynamicState *const bubbleProperties);
00072     virtual void setDewState(ExternalSaturationProperties *const properties, int phase,
00073                              ExternalThermodynamicState *const bubbleProperties);
00074
00075     virtual double dTp(ExternalSaturationProperties *const properties);
00076     virtual double ddldp(ExternalSaturationProperties *const properties);
00077     virtual double ddvdp(ExternalSaturationProperties *const properties);
00078     virtual double dhldp(ExternalSaturationProperties *const properties);
00079     virtual double dhvdp(ExternalSaturationProperties *const properties);
00080     virtual double dl(ExternalSaturationProperties *const properties);
00081     virtual double dv(ExternalSaturationProperties *const properties);
00082     virtual double hl(ExternalSaturationProperties *const properties);
00083     virtual double hv(ExternalSaturationProperties *const properties);
00084     virtual double sigma(ExternalSaturationProperties *const properties);
00085     virtual double sl(ExternalSaturationProperties *const properties);
00086     virtual double sv(ExternalSaturationProperties *const properties);
00087
00088     virtual bool computeDerivatives(ExternalThermodynamicState *const properties);
00089
00090     virtual double psat(ExternalSaturationProperties *const properties);
00091     virtual double Tsat(ExternalSaturationProperties *const properties);
00092
00094     string mediumName;
00096     string libraryName;
00098     string substanceName;
00099
00100 protected:
00102     FluidConstants _fluidConstants;
00103 };
00104
00105 #endif /* BASESOLVER_H_ */
```

## 12.2 coolpropsolver.h

```
00001 #ifndef COOLPROPSOLVER_H_
00002 #define COOLPROPSOLVER_H_
00003
00004 #include "include.h"
00005 #if (EXTERNALMEDIA_COOLPROP == 1)
00006
00007 #include "basesolver.h"
00008 #include "AbstractState.h"
00009 #include "crossplatform_shared_ptr.h"
00010
00030 class CoolPropSolver : public BaseSolver{
00031
00032 protected:
00033     /* class CoolProp::AbstractState *state; */
00034     shared_ptr<CoolProp::AbstractState> state;
00035     bool enable_TTSE, enable_BICUBIC, calc_transport, extend_twophase, isCompressible;
00036     int debug_level;
00037     double twophase_derivsmoothing_xend;
00038     double rho_smoothing_xend;
00039     double _p_eps   ; /* relative tolerance margin for subcritical pressure conditions */
00040     double _delta_h ; /* delta_h for one-phase/two-phase discrimination */
00041     ExternalSaturationProperties _satPropsClose2Crit; /* saturation properties close to  critical
      conditions */
00042
00043     virtual void postStateChange(ExternalThermodynamicState *const properties);
00044     long makeDerivString(const string &of, const string &wrt, const string &cst);
00045     double interp_linear(double Q, double valueL, double valueV);
00046     double interp_recip(double Q, double valueL, double valueV);
00047
00048 public:
00049     CoolPropSolver(const std::string &mediumName, const std::string &libraryName, const std::string
      &substanceName);
00050     ~CoolPropSolver();
00051     virtual void setFluidConstants();
00052
00053     virtual void setSat_p(double &p, ExternalSaturationProperties *const properties);
00054     virtual void setSat_T(double &T, ExternalSaturationProperties *const properties);
00055
00056     virtual void setBubbleState(ExternalSaturationProperties *const properties, int phase,
      ExternalThermodynamicState *const bubbleProperties);
00057     virtual void setDewState   (ExternalSaturationProperties *const properties, int phase,
      ExternalThermodynamicState *const bubbleProperties);
00058
00059     virtual void setState_ph(double &p, double &h, int &phase, ExternalThermodynamicState *const
      properties);
```

```
00060     virtual void setState_pT(double &p, double &T, ExternalThermodynamicState *const properties);
00061     virtual void setState_dT(double &d, double &T, int &phase, ExternalThermodynamicState *const
      properties);
00062     virtual void setState_ps(double &p, double &s, int &phase, ExternalThermodynamicState *const
      properties);
00063     virtual void setState_hs(double &h, double &s, int &phase, ExternalThermodynamicState *const
      properties);
00064
00065     virtual double partialDeriv_state(const string &of, const string &wrt, const string &cst,
      ExternalThermodynamicState *const properties);
00066
00067     virtual double Pr(ExternalThermodynamicState *const properties);
00068     virtual double T(ExternalThermodynamicState *const properties);
00069     virtual double a(ExternalThermodynamicState *const properties);
00070     virtual double beta(ExternalThermodynamicState *const properties);
00071     virtual double cp(ExternalThermodynamicState *const properties);
00072     virtual double cv(ExternalThermodynamicState *const properties);
00073     virtual double d(ExternalThermodynamicState *const properties);
00074     virtual double ddhp(ExternalThermodynamicState *const properties);
00075     virtual double ddph(ExternalThermodynamicState *const properties);
00076     virtual double eta(ExternalThermodynamicState *const properties);
00077     virtual double h(ExternalThermodynamicState *const properties);
00078     virtual double kappa(ExternalThermodynamicState *const properties);
00079     virtual double lambda(ExternalThermodynamicState *const properties);
00080     virtual double p(ExternalThermodynamicState *const properties);
00081     virtual int phase(ExternalThermodynamicState *const properties);
00082     virtual double s(ExternalThermodynamicState *const properties);
00083     virtual double d_der(ExternalThermodynamicState *const properties);
00084     virtual double isentropicEnthalpy(double &p, ExternalThermodynamicState *const properties);
00085
00086     virtual double dTp(ExternalSaturationProperties *const properties);
00087     virtual double ddldp(ExternalSaturationProperties *const properties);
00088     virtual double ddvdp(ExternalSaturationProperties *const properties);
00089     virtual double dhldp(ExternalSaturationProperties *const properties);
00090     virtual double dhvdp(ExternalSaturationProperties *const properties);
00091     virtual double dl(ExternalSaturationProperties *const properties);
00092     virtual double dv(ExternalSaturationProperties *const properties);
00093     virtual double hl(ExternalSaturationProperties *const properties);
00094     virtual double hv(ExternalSaturationProperties *const properties);
00095     virtual double sigma(ExternalSaturationProperties *const properties);
00096     virtual double sl(ExternalSaturationProperties *const properties);
00097     virtual double sv(ExternalSaturationProperties *const properties);
00098
00099     virtual double psat(ExternalSaturationProperties *const properties);
00100     virtual double Tsat(ExternalSaturationProperties *const properties);
00101
00102 };
00103
00104 #endif
00105
00106 #endif /* COOLPROPSOLVER_H_ */
```

## 12.3 Sources/errorhandling.h File Reference

Error handling for external library.

### Functions

- void errorMessage (char *errorMsg)
- void warningMessage (char *warningMsg)

### 12.3.1 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 Copyright Politecnico di Milano and TU Braunschweig

### 12.3.2 Function Documentation

#### 12.3.2.1 errorMessage()

```
void errorMessage (
            char * errorMsg)
```

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 |

#### 12.3.2.2 warningMessage()

```
void warningMessage (
            char * warningMsg)
```

Function to display warning message

Calling this function will display the specified warning message.

**Parameters**

| | |
|---|---|
| *warningMessage* | Warning message to be displayed |

## 12.4 errorhandling.h

Go to the documentation of this file.

```
00001
00013 #ifndef ERRORHANDLING_H_
00014 #define ERRORHANDLING_H_
00015
00016 #ifdef WIN32
00017 extern void (*ModelicaErrorPtr)(const char *);
00018 extern void (*ModelicaWarningPtr)(const char *);
00019 #endif
00020
00027 void errorMessage(char *errorMsg);
00033 void warningMessage(char *warningMsg);
00034
00035 #endif /* ERRORHANDLING_H_ */
```

## 12.5 Sources/externalmedialib.h File Reference

Header file to be included in the Modelica tool, with external function interfaces.

### Classes

- struct ExternalThermodynamicState
- struct ExternalSaturationProperties

### Macros

- #define **CHOICE_dT** 1
- #define **CHOICE_hs** 2
- #define **CHOICE_ph** 3
- #define **CHOICE_ps** 4
- #define **CHOICE_pT** 5
- #define EXTERNALMEDIA_EXPORT

### Typedefs

- typedef struct ExternalThermodynamicState ExternalThermodynamicState
- typedef struct ExternalSaturationProperties ExternalSaturationProperties

### Functions

- EXTERNALMEDIA_EXPORT double TwoPhaseMedium_getMolarMass_C_impl (const char ∗mediumName, const char ∗libraryName, const char ∗substanceName)

  *Get molar mass.*
- EXTERNALMEDIA_EXPORT double TwoPhaseMedium_getCriticalTemperature_C_impl (const char ∗mediumName, const char ∗libraryName, const char ∗substanceName)

  *Get critical temperature.*
- EXTERNALMEDIA_EXPORT double TwoPhaseMedium_getCriticalPressure_C_impl (const char ∗mediumName, const char ∗libraryName, const char ∗substanceName)

  *Get critical pressure.*
- EXTERNALMEDIA_EXPORT double TwoPhaseMedium_getCriticalMolarVolume_C_impl (const char ∗mediumName, const char ∗libraryName, const char ∗substanceName)

  *Get critical molar volume.*
- EXTERNALMEDIA_EXPORT void TwoPhaseMedium_setState_ph_C_impl (double p, double h, int phase, void ∗state, const char ∗mediumName, const char ∗libraryName, const char ∗substanceName)

  *Compute properties from p, h, and phase.*
- EXTERNALMEDIA_EXPORT void TwoPhaseMedium_setState_pT_C_impl (double p, double T, void ∗state, const char ∗mediumName, const char ∗libraryName, const char ∗substanceName)

  *Compute properties from p and T.*
- EXTERNALMEDIA_EXPORT void TwoPhaseMedium_setState_dT_C_impl (double d, double T, int phase, void ∗state, const char ∗mediumName, const char ∗libraryName, const char ∗substanceName)

  *Compute properties from d, T, and phase.*
- EXTERNALMEDIA_EXPORT void TwoPhaseMedium_setState_ps_C_impl (double p, double s, int phase, void ∗state, const char ∗mediumName, const char ∗libraryName, const char ∗substanceName)

  *Compute properties from p, s, and phase.*
- EXTERNALMEDIA_EXPORT void TwoPhaseMedium_setState_hs_C_impl (double h, double s, int phase, void ∗state, const char ∗mediumName, const char ∗libraryName, const char ∗substanceName)

  *Compute properties from h, s, and phase.*
- EXTERNALMEDIA_EXPORT void **TwoPhaseMedium_setState_ph_C_impl_err** (double p, double h, int phase, void ∗state, const char ∗mediumName, const char ∗libraryName, const char ∗substanceName, void(∗ModelicaErrorPtr)(const char ∗), void(∗ModelicaWarningPtr)(const char ∗))

- EXTERNALMEDIA_EXPORT void **TwoPhaseMedium_setState_pT_C_impl_err** (double p, double T, void *state, const char *mediumName, const char *libraryName, const char *substanceName, void(*ModelicaErrorPtr)(const char *), void(*ModelicaWarningPtr)(const char *))
- EXTERNALMEDIA_EXPORT void **TwoPhaseMedium_setState_dT_C_impl_err** (double d, double T, int phase, void *state, const char *mediumName, const char *libraryName, const char *substanceName, void(*ModelicaErrorPtr)(const char *), void(*ModelicaWarningPtr)(const char *))
- EXTERNALMEDIA_EXPORT void **TwoPhaseMedium_setState_ps_C_impl_err** (double p, double s, int phase, void *state, const char *mediumName, const char *libraryName, const char *substanceName, void(*ModelicaErrorPtr)(const char *), void(*ModelicaWarningPtr)(const char *))
- EXTERNALMEDIA_EXPORT void **TwoPhaseMedium_setState_hs_C_impl_err** (double h, double s, int phase, void *state, const char *mediumName, const char *libraryName, const char *substanceName, void(*ModelicaErrorPtr)(const char *), void(*ModelicaWarningPtr)(const char *))
- EXTERNALMEDIA_EXPORT double TwoPhaseMedium_partialDeriv_state_C_impl (const char *of, const char *wrt, const char *cst, void *state, const char *mediumName, const char *libraryName, const char *substanceName)

  *Compute partial derivative from a populated state record.*
- EXTERNALMEDIA_EXPORT double TwoPhaseMedium_prandtlNumber_C_impl (void *state, const char *mediumName, const char *libraryName, const char *substanceName)

  *Return Prandtl number of specified medium.*
- EXTERNALMEDIA_EXPORT double TwoPhaseMedium_temperature_C_impl (void *state, const char *mediumName, const char *libraryName, const char *substanceName)

  *Return temperature of specified medium.*
- EXTERNALMEDIA_EXPORT double TwoPhaseMedium_velocityOfSound_C_impl (void *state, const char *mediumName, const char *libraryName, const char *substanceName)

  *Return velocity of sound of specified medium.*
- EXTERNALMEDIA_EXPORT double TwoPhaseMedium_isobaricExpansionCoefficient_C_impl (void *state, const char *mediumName, const char *libraryName, const char *substanceName)

  *Return isobaric expansion coefficient of specified medium.*
- EXTERNALMEDIA_EXPORT double TwoPhaseMedium_specificHeatCapacityCp_C_impl (void *state, const char *mediumName, const char *libraryName, const char *substanceName)

  *Return specific heat capacity cp of specified medium.*
- EXTERNALMEDIA_EXPORT double TwoPhaseMedium_specificHeatCapacityCv_C_impl (void *state, const char *mediumName, const char *libraryName, const char *substanceName)

  *Return specific heat capacity cv of specified medium.*
- EXTERNALMEDIA_EXPORT double TwoPhaseMedium_density_C_impl (void *state, const char *mediumName, const char *libraryName, const char *substanceName)

  *Return density of specified medium.*
- EXTERNALMEDIA_EXPORT double TwoPhaseMedium_density_derh_p_C_impl (void *state, const char *mediumName, const char *libraryName, const char *substanceName)

  *Return derivative of density wrt specific enthalpy at constant pressure of specified medium.*
- EXTERNALMEDIA_EXPORT double TwoPhaseMedium_density_derp_h_C_impl (void *state, const char *mediumName, const char *libraryName, const char *substanceName)

  *Return derivative of density wrt pressure at constant specific enthalpy of specified medium.*
- EXTERNALMEDIA_EXPORT double TwoPhaseMedium_dynamicViscosity_C_impl (void *state, const char *mediumName, const char *libraryName, const char *substanceName)

  *Return dynamic viscosity of specified medium.*
- EXTERNALMEDIA_EXPORT double TwoPhaseMedium_specificEnthalpy_C_impl (void *state, const char *mediumName, const char *libraryName, const char *substanceName)

  *Return specific enthalpy of specified medium.*
- EXTERNALMEDIA_EXPORT double TwoPhaseMedium_isothermalCompressibility_C_impl (void *state, const char *mediumName, const char *libraryName, const char *substanceName)

  *Return isothermal compressibility of specified medium.*
- EXTERNALMEDIA_EXPORT double TwoPhaseMedium_thermalConductivity_C_impl (void *state, const char *mediumName, const char *libraryName, const char *substanceName)

*Return thermal conductivity of specified medium.*

- EXTERNALMEDIA_EXPORT double TwoPhaseMedium_pressure_C_impl (void *state, const char *mediumName, const char *libraryName, const char *substanceName)

  *Return pressure of specified medium.*
- EXTERNALMEDIA_EXPORT double TwoPhaseMedium_specificEntropy_C_impl (void *state, const char *mediumName, const char *libraryName, const char *substanceName)

  *Return specific entropy of specified medium.*
- EXTERNALMEDIA_EXPORT double TwoPhaseMedium_density_ph_der_C_impl (void *state, const char *mediumName, const char *libraryName, const char *substanceName)

  *Return derivative of density wrt pressure and specific enthalpy of specified medium.*
- EXTERNALMEDIA_EXPORT double **TwoPhaseMedium_isentropicEnthalpy_C_impl** (double p_↩ downstream, void *refState, const char *mediumName, const char *libraryName, const char *substance↩ Name)
- EXTERNALMEDIA_EXPORT void TwoPhaseMedium_setSat_p_C_impl (double p, void *sat, const char *mediumName, const char *libraryName, const char *substanceName)

  *Compute saturation properties from p.*
- EXTERNALMEDIA_EXPORT void TwoPhaseMedium_setSat_T_C_impl (double T, void *sat, const char *mediumName, const char *libraryName, const char *substanceName)

  *Compute saturation properties from T.*
- EXTERNALMEDIA_EXPORT void TwoPhaseMedium_setBubbleState_C_impl (void *sat, int phase, void *state, const char *mediumName, const char *libraryName, const char *substanceName)

  *Compute bubble state.*
- EXTERNALMEDIA_EXPORT void TwoPhaseMedium_setDewState_C_impl (void *sat, int phase, void *state, const char *mediumName, const char *libraryName, const char *substanceName)

  *Compute dew state.*
- EXTERNALMEDIA_EXPORT double **TwoPhaseMedium_saturationTemperature_C_impl** (double p, const char *mediumName, const char *libraryName, const char *substanceName)

  *Compute saturation temperature for specified medium and pressure.*
- EXTERNALMEDIA_EXPORT double **TwoPhaseMedium_saturationTemperature_derp_C_impl** (double p, const char *mediumName, const char *libraryName, const char *substanceName)

  *Compute derivative of saturation temperature for specified medium and pressure.*
- EXTERNALMEDIA_EXPORT double TwoPhaseMedium_saturationTemperature_derp_sat_C_impl (void *sat, const char *mediumName, const char *libraryName, const char *substanceName)

  *Return derivative of saturation temperature of specified medium from saturation properties.*
- EXTERNALMEDIA_EXPORT double TwoPhaseMedium_dBubbleDensity_dPressure_C_impl (void *sat, const char *mediumName, const char *libraryName, const char *substanceName)

  *Return derivative of bubble density wrt pressure of specified medium from saturation properties.*
- EXTERNALMEDIA_EXPORT double TwoPhaseMedium_dDewDensity_dPressure_C_impl (void *sat, const char *mediumName, const char *libraryName, const char *substanceName)

  *Return derivative of dew density wrt pressure of specified medium from saturation properties.*
- EXTERNALMEDIA_EXPORT double TwoPhaseMedium_dBubbleEnthalpy_dPressure_C_impl (void *sat, const char *mediumName, const char *libraryName, const char *substanceName)

  *Return derivative of bubble specific enthalpy wrt pressure of specified medium from saturation properties.*
- EXTERNALMEDIA_EXPORT double TwoPhaseMedium_dDewEnthalpy_dPressure_C_impl (void *sat, const char *mediumName, const char *libraryName, const char *substanceName)

  *Return derivative of dew specific enthalpy wrt pressure of specified medium from saturation properties.*
- EXTERNALMEDIA_EXPORT double TwoPhaseMedium_bubbleDensity_C_impl (void *sat, const char *mediumName, const char *libraryName, const char *substanceName)

  *Return bubble density of specified medium from saturation properties.*
- EXTERNALMEDIA_EXPORT double TwoPhaseMedium_dewDensity_C_impl (void *sat, const char *mediumName, const char *libraryName, const char *substanceName)

  *Return dew density of specified medium from saturation properties.*

- EXTERNALMEDIA_EXPORT double TwoPhaseMedium_bubbleEnthalpy_C_impl (void *sat, const char *mediumName, const char *libraryName, const char *substanceName)

  *Return bubble specific enthalpy of specified medium from saturation properties.*
- EXTERNALMEDIA_EXPORT double TwoPhaseMedium_dewEnthalpy_C_impl (void *sat, const char *mediumName, const char *libraryName, const char *substanceName)

  *Return dew specific enthalpy of specified medium from saturation properties.*
- EXTERNALMEDIA_EXPORT double TwoPhaseMedium_saturationPressure_C_impl (double T, const char *mediumName, const char *libraryName, const char *substanceName)

  *Compute saturation pressure for specified medium and temperature.*
- EXTERNALMEDIA_EXPORT double TwoPhaseMedium_surfaceTension_C_impl (void *sat, const char *mediumName, const char *libraryName, const char *substanceName)

  *Return surface tension of specified medium.*
- EXTERNALMEDIA_EXPORT double TwoPhaseMedium_bubbleEntropy_C_impl (void *sat, const char *mediumName, const char *libraryName, const char *substanceName)

  *Return bubble specific entropy of specified medium from saturation properties.*
- EXTERNALMEDIA_EXPORT double TwoPhaseMedium_dewEntropy_C_impl (void *sat, const char *mediumName, const char *libraryName, const char *substanceName)

  *Return dew specific entropy of specified medium from saturation properties.*

### 12.5.1 Detailed Description

Header file to be included in the Modelica tool, with external function interfaces.

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

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

Minor additions in 2014 to make ExternalMedia compatible with GCC on Linux operating systems Jorrit Wronski (Technical University of Denmark)

Adapted to work with dynamically linked libraries by Francesco Casella and Federico Terraneo 2022 (Politecnico di Milano)

### 12.5.2 Macro Definition Documentation

#### 12.5.2.1 EXTERNALMEDIA_EXPORT

```
#define EXTERNALMEDIA_EXPORT
```

Portable definitions of the EXPORT macro

### 12.5.3 Typedef Documentation

#### 12.5.3.1 ExternalSaturationProperties

```
typedef struct ExternalSaturationProperties ExternalSaturationProperties
```

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. Note: the exported interface functions use typeless void *sat instead of ExternalSaturationProperties *sat as Modelica does not treat external struct types so far.

#### 12.5.3.2 ExternalThermodynamicState

```
typedef struct ExternalThermodynamicState ExternalThermodynamicState
```

ExternalThermodynamicState property struct

The ExternalThermodynamicState propery struct defines all the properties that are computed by external Modelica medium models extending from PartialExternalTwoPhaseMedium. Note: the exported interface functions use typeless void *state instead of ExternalThermodynamicState *state as Modelica does not treat external struct types so far.

### 12.5.4 Function Documentation

#### 12.5.4.1 TwoPhaseMedium_bubbleDensity_C_impl()

```
EXTERNALMEDIA_EXPORT double TwoPhaseMedium_bubbleDensity_C_impl (
            void * sat,
            const char * mediumName,
            const char * libraryName,
            const char * substanceName)
```

Return bubble density of specified medium from saturation properties.

Note: This function is not used by the default implementation of ExternalTwoPhaseMedium class. It might be used by external medium models customized solvers redeclaring the default functions

#### 12.5.4.2 TwoPhaseMedium_bubbleEnthalpy_C_impl()

```
EXTERNALMEDIA_EXPORT double TwoPhaseMedium_bubbleEnthalpy_C_impl (
            void * sat,
            const char * mediumName,
            const char * libraryName,
            const char * substanceName)
```

Return bubble specific enthalpy of specified medium from saturation properties.

Note: This function is not used by the default implementation of ExternalTwoPhaseMedium class. It might be used by external medium models customized solvers redeclaring the default functions

#### 12.5.4.3 TwoPhaseMedium_bubbleEntropy_C_impl()

```
EXTERNALMEDIA_EXPORT double TwoPhaseMedium_bubbleEntropy_C_impl (
            void * sat,
            const char * mediumName,
            const char * libraryName,
            const char * substanceName)
```

Return bubble specific entropy of specified medium from saturation properties.

Note: This function is not used by the default implementation of ExternalTwoPhaseMedium class. It might be used by external medium models customized solvers redeclaring the default functions

#### 12.5.4.4 TwoPhaseMedium_dBubbleDensity_dPressure_C_impl()

```
EXTERNALMEDIA_EXPORT double TwoPhaseMedium_dBubbleDensity_dPressure_C_impl (
            void * sat,
            const char * mediumName,
            const char * libraryName,
            const char * substanceName)
```

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

Note: This function is not used by the default implementation of ExternalTwoPhaseMedium class. It might be used by external medium models customized solvers redeclaring the default functions

#### 12.5.4.5 TwoPhaseMedium_dBubbleEnthalpy_dPressure_C_impl()

```
EXTERNALMEDIA_EXPORT double TwoPhaseMedium_dBubbleEnthalpy_dPressure_C_impl (
            void * sat,
            const char * mediumName,
            const char * libraryName,
            const char * substanceName)
```

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

Note: This function is not used by the default implementation of ExternalTwoPhaseMedium class. It might be used by external medium models customized solvers redeclaring the default functions

#### 12.5.4.6 TwoPhaseMedium_dDewDensity_dPressure_C_impl()

```
EXTERNALMEDIA_EXPORT double TwoPhaseMedium_dDewDensity_dPressure_C_impl (
            void * sat,
            const char * mediumName,
            const char * libraryName,
            const char * substanceName)
```

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

Note: This function is not used by the default implementation of ExternalTwoPhaseMedium class. It might be used by external medium models customized solvers redeclaring the default functions

#### 12.5.4.7 TwoPhaseMedium_dDewEnthalpy_dPressure_C_impl()

```
EXTERNALMEDIA_EXPORT double TwoPhaseMedium_dDewEnthalpy_dPressure_C_impl (
            void * sat,
            const char * mediumName,
            const char * libraryName,
            const char * substanceName)
```

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

Note: This function is not used by the default implementation of ExternalTwoPhaseMedium class. It might be used by external medium models customized solvers redeclaring the default functions

#### 12.5.4.8 TwoPhaseMedium_density_C_impl()

```
EXTERNALMEDIA_EXPORT double TwoPhaseMedium_density_C_impl (
            void * state,
            const char * mediumName,
            const char * libraryName,
            const char * substanceName)
```

Return density of specified medium.

Note: This function is not used by the default implementation of ExternalTwoPhaseMedium class. It might be used by external medium models customized solvers redeclaring the default functions

#### 12.5.4.9 TwoPhaseMedium_density_derh_p_C_impl()

```
EXTERNALMEDIA_EXPORT double TwoPhaseMedium_density_derh_p_C_impl (
            void * state,
            const char * mediumName,
            const char * libraryName,
            const char * substanceName)
```

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

Note: This function is not used by the default implementation of ExternalTwoPhaseMedium class. It might be used by external medium models customized solvers redeclaring the default functions

#### 12.5.4.10 TwoPhaseMedium_density_derp_h_C_impl()

```
EXTERNALMEDIA_EXPORT double TwoPhaseMedium_density_derp_h_C_impl (
            void * state,
            const char * mediumName,
            const char * libraryName,
            const char * substanceName)
```

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

Note: This function is not used by the default implementation of ExternalTwoPhaseMedium class. It might be used by external medium models customized solvers redeclaring the default functions

#### 12.5.4.11 TwoPhaseMedium_density_ph_der_C_impl()

```
EXTERNALMEDIA_EXPORT double TwoPhaseMedium_density_ph_der_C_impl (
            void * state,
            const char * mediumName,
            const char * libraryName,
            const char * substanceName)
```

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

Note: This function is not used by the default implementation of ExternalTwoPhaseMedium class. It might be used by external medium models customized solvers redeclaring the default functions

#### 12.5.4.12 TwoPhaseMedium_dewDensity_C_impl()

```
EXTERNALMEDIA_EXPORT double TwoPhaseMedium_dewDensity_C_impl (
            void * sat,
            const char * mediumName,
            const char * libraryName,
            const char * substanceName)
```

Return dew density of specified medium from saturation properties.

Note: This function is not used by the default implementation of ExternalTwoPhaseMedium class. It might be used by external medium models customized solvers redeclaring the default functions

#### 12.5.4.13 TwoPhaseMedium_dewEnthalpy_C_impl()

```
EXTERNALMEDIA_EXPORT double TwoPhaseMedium_dewEnthalpy_C_impl (
            void * sat,
            const char * mediumName,
            const char * libraryName,
            const char * substanceName)
```

Return dew specific enthalpy of specified medium from saturation properties.

Note: This function is not used by the default implementation of ExternalTwoPhaseMedium class. It might be used by external medium models customized solvers redeclaring the default functions

#### 12.5.4.14 TwoPhaseMedium_dewEntropy_C_impl()

```
EXTERNALMEDIA_EXPORT double TwoPhaseMedium_dewEntropy_C_impl (
            void * sat,
            const char * mediumName,
            const char * libraryName,
            const char * substanceName)
```

Return dew specific entropy of specified medium from saturation properties.

Note: This function is not used by the default implementation of ExternalTwoPhaseMedium class. It might be used by external medium models customized solvers redeclaring the default functions

#### 12.5.4.15 TwoPhaseMedium_dynamicViscosity_C_impl()

```
EXTERNALMEDIA_EXPORT double TwoPhaseMedium_dynamicViscosity_C_impl (
            void * state,
            const char * mediumName,
            const char * libraryName,
            const char * substanceName)
```

Return dynamic viscosity of specified medium.

Note: This function is not used by the default implementation of ExternalTwoPhaseMedium class. It might be used by external medium models customized solvers redeclaring the default functions

#### 12.5.4.16 TwoPhaseMedium_getCriticalMolarVolume_C_impl()

```
EXTERNALMEDIA_EXPORT double TwoPhaseMedium_getCriticalMolarVolume_C_impl (
            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 |

#### 12.5.4.17 TwoPhaseMedium_getCriticalPressure_C_impl()

```
EXTERNALMEDIA_EXPORT double TwoPhaseMedium_getCriticalPressure_C_impl (
            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 |

#### 12.5.4.18 TwoPhaseMedium_getCriticalTemperature_C_impl()

```
EXTERNALMEDIA_EXPORT double TwoPhaseMedium_getCriticalTemperature_C_impl (
            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 |

#### 12.5.4.19 TwoPhaseMedium_getMolarMass_C_impl()

```
EXTERNALMEDIA_EXPORT double TwoPhaseMedium_getMolarMass_C_impl (
            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 |

#### 12.5.4.20 TwoPhaseMedium_isobaricExpansionCoefficient_C_impl()

```
EXTERNALMEDIA_EXPORT double TwoPhaseMedium_isobaricExpansionCoefficient_C_impl (
            void * state,
            const char * mediumName,
            const char * libraryName,
            const char * substanceName)
```

Return isobaric expansion coefficient of specified medium.

Note: This function is not used by the default implementation of ExternalTwoPhaseMedium class. It might be used by external medium models customized solvers redeclaring the default functions

#### 12.5.4.21 TwoPhaseMedium_isothermalCompressibility_C_impl()

```
EXTERNALMEDIA_EXPORT double TwoPhaseMedium_isothermalCompressibility_C_impl (
            void * state,
            const char * mediumName,
            const char * libraryName,
            const char * substanceName)
```

Return isothermal compressibility of specified medium.

Note: This function is not used by the default implementation of ExternalTwoPhaseMedium class. It might be used by external medium models customized solvers redeclaring the default functions

#### 12.5.4.22 TwoPhaseMedium_partialDeriv_state_C_impl()

```
EXTERNALMEDIA_EXPORT double TwoPhaseMedium_partialDeriv_state_C_impl (
            const char * of,
            const char * wrt,
            const char * cst,
            void * state,
            const char * mediumName,
            const char * libraryName,
            const char * substanceName)
```

Compute partial derivative from a populated state record.

This function computes the derivative of the specified input.

**Parameters**

| | |
|---|---|
| *of* | Property to differentiate |
| *wrt* | Property to differentiate in |
| *cst* | Property to remain constant |
| *state* | Pointer to input values in state record |
| *mediumName* | Medium name |
| *libraryName* | Library name |
| *substanceName* | Substance name |

#### 12.5.4.23 TwoPhaseMedium_prandtlNumber_C_impl()

```
EXTERNALMEDIA_EXPORT double TwoPhaseMedium_prandtlNumber_C_impl (
            void * state,
            const char * mediumName,
            const char * libraryName,
            const char * substanceName)
```

Return Prandtl number of specified medium.

Note: This function is not used by the default implementation of ExternalTwoPhaseMedium class. It might be used by external medium models customized solvers redeclaring the default functions

#### 12.5.4.24 TwoPhaseMedium_pressure_C_impl()

```
EXTERNALMEDIA_EXPORT double TwoPhaseMedium_pressure_C_impl (
            void * state,
            const char * mediumName,
            const char * libraryName,
            const char * substanceName)
```

Return pressure of specified medium.

Note: This function is not used by the default implementation of ExternalTwoPhaseMedium class. It might be used by external medium models customized solvers redeclaring the default functions

#### 12.5.4.25 TwoPhaseMedium_saturationPressure_C_impl()

```
EXTERNALMEDIA_EXPORT double TwoPhaseMedium_saturationPressure_C_impl (
            double T,
            const char * mediumName,
            const char * libraryName,
            const char * substanceName)
```

Compute saturation pressure for specified medium and temperature.

Note: This function is not used by the default implementation of ExternalTwoPhaseMedium class. It might be used by external medium models customized solvers redeclaring the default functions

#### 12.5.4.26 TwoPhaseMedium_saturationTemperature_derp_sat_C_impl()

```
EXTERNALMEDIA_EXPORT double TwoPhaseMedium_saturationTemperature_derp_sat_C_impl (
            void * sat,
            const char * mediumName,
            const char * libraryName,
            const char * substanceName)
```

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

Note: This function is not used by the default implementation of ExternalTwoPhaseMedium class. It might be used by external medium models customized solvers redeclaring the default functions

#### 12.5.4.27 TwoPhaseMedium_setBubbleState_C_impl()

```
EXTERNALMEDIA_EXPORT void TwoPhaseMedium_setBubbleState_C_impl (
            void * sat,
            int phase,
            void * state,
            const char * mediumName,
            const char * libraryName,
            const char * substanceName)
```

Compute bubble state.

This function computes the bubble state for the specified medium.

**Parameters**

| | |
|---|---|
| *sat* | Pointer to values of ExternalSaturationProperties struct |
| *phase* | Phase (2 for two-phase, 1 for one-phase, 0 if not known) |
| *state* | Pointer to return values for ExternalThermodynamicState struct |
| *mediumName* | Medium name |
| *libraryName* | Library name |
| *substanceName* | Substance name |

#### 12.5.4.28 TwoPhaseMedium_setDewState_C_impl()

```
EXTERNALMEDIA_EXPORT void TwoPhaseMedium_setDewState_C_impl (
            void * sat,
            int phase,
            void * state,
            const char * mediumName,
            const char * libraryName,
            const char * substanceName)
```

Compute dew state.

This function computes the dew state for the specified medium.

**Parameters**

| | |
|---|---|
| *sat* | Pointer to values of ExternalSaturationProperties struct |
| *phase* | Phase (2 for two-phase, 1 for one-phase, 0 if not known) |
| *state* | Pointer to return values for ExternalThermodynamicState struct |
| *mediumName* | Medium name |
| *libraryName* | Library name |
| *substanceName* | Substance name |

#### 12.5.4.29 TwoPhaseMedium_setSat_p_C_impl()

```
EXTERNALMEDIA_EXPORT void TwoPhaseMedium_setSat_p_C_impl (
            double p,
            void * sat,
            const char * mediumName,
            const char * libraryName,
            const char * substanceName)
```

Compute saturation properties from p.

This function computes the saturation properties for the specified inputs.

**Parameters**

| | |
|---|---|
| *p* | Pressure |
| *sat* | Pointer to return values for ExternalSaturationProperties struct |
| *mediumName* | Medium name |
| *libraryName* | Library name |
| *substanceName* | Substance name |

#### 12.5.4.30 TwoPhaseMedium_setSat_T_C_impl()

```
EXTERNALMEDIA_EXPORT void TwoPhaseMedium_setSat_T_C_impl (
            double T,
            void * sat,
            const char * mediumName,
            const char * libraryName,
            const char * substanceName)
```

Compute saturation properties from T.

This function computes the saturation properties for the specified inputs.

**Parameters**

| | |
|---|---|
| *T* | Temperature |
| *sat* | Pointer to return values for ExternalSaturationProperties struct |
| *mediumName* | Medium name |
| *libraryName* | Library name |
| *substanceName* | Substance name |

#### 12.5.4.31 TwoPhaseMedium_setState_dT_C_impl()

```
EXTERNALMEDIA_EXPORT void TwoPhaseMedium_setState_dT_C_impl (
            double d,
            double T,
            int phase,
            void * state,
            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.

**Parameters**

| | |
|---|---|
| *d* | Density |
| *T* | Temperature |
| *phase* | Phase (2 for two-phase, 1 for one-phase, 0 if not known) |
| *state* | Pointer to return values for ExternalThermodynamicState struct |
| *mediumName* | Medium name |
| *libraryName* | Library name |
| *substanceName* | Substance name |

#### 12.5.4.32 TwoPhaseMedium_setState_hs_C_impl()

```
EXTERNALMEDIA_EXPORT void TwoPhaseMedium_setState_hs_C_impl (
            double h,
            double s,
            int phase,
            void * state,
            const char * mediumName,
            const char * libraryName,
            const char * substanceName)
```

Compute properties from h, s, and phase.

This function computes the properties for the specified inputs.

**Parameters**

| | |
|---|---|
| *h* | Specific enthalpy |
| *s* | Specific entropy |
| *phase* | Phase (2 for two-phase, 1 for one-phase, 0 if not known) |
| *state* | Pointer to return values for ExternalThermodynamicState struct |
| *mediumName* | Medium name |
| *libraryName* | Library name |
| *substanceName* | Substance name |

#### 12.5.4.33 TwoPhaseMedium_setState_ph_C_impl()

```
EXTERNALMEDIA_EXPORT void TwoPhaseMedium_setState_ph_C_impl (
            double p,
            double h,
            int phase,
            void * state,
            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.

**Parameters**

| | |
|---|---|
| *p* | Pressure |
| *h* | Specific enthalpy |
| *phase* | Phase (2 for two-phase, 1 for one-phase, 0 if not known) |
| *state* | Pointer to return values for ExternalThermodynamicState struct |
| *mediumName* | Medium name |
| *libraryName* | Library name |
| *substanceName* | Substance name |

#### 12.5.4.34 TwoPhaseMedium_setState_ps_C_impl()

```
EXTERNALMEDIA_EXPORT void TwoPhaseMedium_setState_ps_C_impl (
            double p,
            double s,
            int phase,
            void * state,
            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.

**Parameters**

| | |
|---|---|
| *p* | Pressure |
| *s* | Specific entropy |
| *phase* | Phase (2 for two-phase, 1 for one-phase, 0 if not known) |
| *state* | Pointer to return values for ExternalThermodynamicState struct |
| *mediumName* | Medium name |
| *libraryName* | Library name |
| *substanceName* | Substance name |

#### 12.5.4.35 TwoPhaseMedium_setState_pT_C_impl()

```
EXTERNALMEDIA_EXPORT void TwoPhaseMedium_setState_pT_C_impl (
            double p,
            double T,
            void * state,
            const char * mediumName,
            const char * libraryName,
            const char * substanceName)
```

Compute properties from p and T.

This function computes the properties for the specified inputs.

Attention: The phase input is ignored for this function!

**Parameters**

| | |
|---|---|
| *p* | Pressure |
| *T* | Temperature |
| *state* | Pointer to return values for ExternalThermodynamicState struct |
| *mediumName* | Medium name |
| *libraryName* | Library name |
| *substanceName* | Substance name |

#### 12.5.4.36 TwoPhaseMedium_specificEnthalpy_C_impl()

```
EXTERNALMEDIA_EXPORT double TwoPhaseMedium_specificEnthalpy_C_impl (
            void * state,
            const char * mediumName,
            const char * libraryName,
            const char * substanceName)
```

Return specific enthalpy of specified medium.

Note: This function is not used by the default implementation of ExternalTwoPhaseMedium class. It might be used by external medium models customized solvers redeclaring the default functions

#### 12.5.4.37 TwoPhaseMedium_specificEntropy_C_impl()

```
EXTERNALMEDIA_EXPORT double TwoPhaseMedium_specificEntropy_C_impl (
            void * state,
            const char * mediumName,
            const char * libraryName,
            const char * substanceName)
```

Return specific entropy of specified medium.

Note: This function is not used by the default implementation of ExternalTwoPhaseMedium class. It might be used by external medium models customized solvers redeclaring the default functions

#### 12.5.4.38 TwoPhaseMedium_specificHeatCapacityCp_C_impl()

```
EXTERNALMEDIA_EXPORT double TwoPhaseMedium_specificHeatCapacityCp_C_impl (
            void * state,
            const char * mediumName,
            const char * libraryName,
            const char * substanceName)
```

Return specific heat capacity cp of specified medium.

Note: This function is not used by the default implementation of ExternalTwoPhaseMedium class. It might be used by external medium models customized solvers redeclaring the default functions

#### 12.5.4.39 TwoPhaseMedium_specificHeatCapacityCv_C_impl()

```
EXTERNALMEDIA_EXPORT double TwoPhaseMedium_specificHeatCapacityCv_C_impl (
            void * state,
            const char * mediumName,
            const char * libraryName,
            const char * substanceName)
```

Return specific heat capacity cv of specified medium.

Note: This function is not used by the default implementation of ExternalTwoPhaseMedium class. It might be used by external medium models customized solvers redeclaring the default functions

#### 12.5.4.40 TwoPhaseMedium_surfaceTension_C_impl()

```
EXTERNALMEDIA_EXPORT double TwoPhaseMedium_surfaceTension_C_impl (
            void * sat,
            const char * mediumName,
            const char * libraryName,
            const char * substanceName)
```

Return surface tension of specified medium.

Note: This function is not used by the default implementation of ExternalTwoPhaseMedium class. It might be used by external medium models customized solvers redeclaring the default functions

#### 12.5.4.41 TwoPhaseMedium_temperature_C_impl()

```
EXTERNALMEDIA_EXPORT double TwoPhaseMedium_temperature_C_impl (
            void * state,
            const char * mediumName,
            const char * libraryName,
            const char * substanceName)
```

Return temperature of specified medium.

Note: This function is not used by the default implementation of ExternalTwoPhaseMedium class. It might be used by external medium models customized solvers redeclaring the default functions

#### 12.5.4.42 TwoPhaseMedium_thermalConductivity_C_impl()

```
EXTERNALMEDIA_EXPORT double TwoPhaseMedium_thermalConductivity_C_impl (
            void * state,
            const char * mediumName,
            const char * libraryName,
            const char * substanceName)
```

Return thermal conductivity of specified medium.

Note: This function is not used by the default implementation of ExternalTwoPhaseMedium class. It might be used by external medium models customized solvers redeclaring the default functions

#### 12.5.4.43 TwoPhaseMedium_velocityOfSound_C_impl()

```
EXTERNALMEDIA_EXPORT double TwoPhaseMedium_velocityOfSound_C_impl (
            void * state,
            const char * mediumName,
            const char * libraryName,
            const char * substanceName)
```

Return velocity of sound of specified medium.

Note: This function is not used by the default implementation of ExternalTwoPhaseMedium class. It might be used by external medium models customized solvers redeclaring the default functions

# 12.6 externalmedialib.h

Go to the documentation of this file.

```
00001
00020 #ifndef EXTERNALMEDIALIB_H_
00021 #define EXTERNALMEDIALIB_H_
00022
00023 /* Constants for input choices (see ExternalMedia.Common.InputChoices) */
00024 #define CHOICE_dT 1
00025 #define CHOICE_hs 2
00026 #define CHOICE_ph 3
00027 #define CHOICE_ps 4
00028 #define CHOICE_pT 5
00029
00033 #if !defined(EXTERNALMEDIA_EXPORT)
00034 #  if !defined(EXTERNALMEDIA_LIBRARY_EXPORTS)
00035 #    define EXTERNALMEDIA_EXPORT
00036 #  else
00037 #    if (EXTERNALMEDIA_LIBRARY_EXPORTS == 1)
00038 #      if defined(_WIN32) || defined(__WIN32__) || defined(_WIN64) || defined(__WIN64__)
00039 #        if !defined(__EXTERNALMEDIA_ISWINDOWS__)
00040 #          define __EXTERNALMEDIA_ISWINDOWS__
00041 #        endif
00042 #      elif __APPLE__
00043 #        if !defined(__EXTERNALMEDIA_ISAPPLE__)
00044 #          define __EXTERNALMEDIA_ISAPPLE__
00045 #        endif
00046 #      elif __linux
00047 #        if !defined(__EXTERNALMEDIA_ISLINUX__)
00048 #          define __EXTERNALMEDIA_ISLINUX__
00049 #        endif
00050 #      endif
00051 #
00052 #      if defined(__EXTERNALMEDIA_ISLINUX__)
00053 #        define EXTERNALMEDIA_EXPORT
00054 #      elif defined(__EXTERNALMEDIA_ISAPPLE__)
00055 #        define EXTERNALMEDIA_EXPORT
00056 #      else
00057 #        define EXTERNALMEDIA_EXPORT __declspec(dllexport)
00058 #      endif
00059 #    else
00060 #      define EXTERNALMEDIA_EXPORT
00061 #    endif
00062 #  endif
00063 #endif
00064
00065 /* Define struct */
00076 typedef struct ExternalThermodynamicState {
00077
00079     double T;
00081     double a;
00083     double beta;
00085     double cp;
00087     double cv;
00089     double d;
00091     double ddhp;
00093     double ddph;
00095     double eta;
00097     double h;
00099     double kappa;
00101     double lambda;
00103     double p;
00105     int phase;
00107     double s;
00108
00113     #ifdef __cplusplus
00114     ExternalThermodynamicState() : T(-1), a(-1), beta(-1), cp(-1), cv(-1), d(-1), ddhp(-1), ddph(-1),
      eta(-1), h(-1), kappa(-1), lambda(-1), p(-1), phase(-1), s(-1) {};
00115     #endif
00116
00117 } ExternalThermodynamicState;
00118
00129 typedef struct ExternalSaturationProperties {
00131     double Tsat;
00133     double dTp;
00135     double ddldp;
00137     double ddvdp;
00139     double dhldp;
00141     double dhvdp;
00143     double dl;
00145     double dv;
00147     double hl;
00149     double hv;
00151     double psat;
00153     double sigma;
```

```
00155     double sl;
00157     double sv;
00158
00163     #ifdef __cplusplus
00164     ExternalSaturationProperties() : Tsat(-1), dTp(-1), ddldp(-1), ddvdp(-1), dhldp(-1), dhvdp(-1),
      dl(-1), dv(-1), hl(-1), hv(-1), psat(-1), sigma(-1), sl(-1), sv(-1) {};
00165     #endif
00166
00167 } ExternalSaturationProperties;
00168
00169
00170 #ifdef __cplusplus
00171 extern "C" {
00172 #endif /* __cplusplus */
00173
00174     EXTERNALMEDIA_EXPORT double TwoPhaseMedium_getMolarMass_C_impl(const char *mediumName, const char
      *libraryName, const char *substanceName);
00175     EXTERNALMEDIA_EXPORT double TwoPhaseMedium_getCriticalTemperature_C_impl(const char *mediumName,
      const char *libraryName, const char *substanceName);
00176     EXTERNALMEDIA_EXPORT double TwoPhaseMedium_getCriticalPressure_C_impl(const char *mediumName,
      const char *libraryName, const char *substanceName);
00177     EXTERNALMEDIA_EXPORT double TwoPhaseMedium_getCriticalMolarVolume_C_impl(const char *mediumName,
      const char *libraryName, const char *substanceName);
00178
00179     EXTERNALMEDIA_EXPORT void TwoPhaseMedium_setState_ph_C_impl(double p, double h, int phase, void
      *state, const char *mediumName, const char *libraryName, const char *substanceName);
00180     EXTERNALMEDIA_EXPORT void TwoPhaseMedium_setState_pT_C_impl(double p, double T,            void
      *state, const char *mediumName, const char *libraryName, const char *substanceName);
00181     EXTERNALMEDIA_EXPORT void TwoPhaseMedium_setState_dT_C_impl(double d, double T, int phase, void
      *state, const char *mediumName, const char *libraryName, const char *substanceName);
00182     EXTERNALMEDIA_EXPORT void TwoPhaseMedium_setState_ps_C_impl(double p, double s, int phase, void
      *state, const char *mediumName, const char *libraryName, const char *substanceName);
00183     EXTERNALMEDIA_EXPORT void TwoPhaseMedium_setState_hs_C_impl(double h, double s, int phase, void
      *state, const char *mediumName, const char *libraryName, const char *substanceName);
00184
00185     /* These functions implement a workaround to handle ModelicaError and ModelicaWarning on Windows
      until a proper solution based on exporting symbols becomes available in Modelica tools */
00186     EXTERNALMEDIA_EXPORT void TwoPhaseMedium_setState_ph_C_impl_err(double p, double h, int phase,
      void *state, const char *mediumName, const char *libraryName, const char *substanceName, void
      (*ModelicaErrorPtr)(const char *), void (*ModelicaWarningPtr)(const char *));
00187     EXTERNALMEDIA_EXPORT void TwoPhaseMedium_setState_pT_C_impl_err(double p, double T,
      void *state, const char *mediumName, const char *libraryName, const char *substanceName, void
      (*ModelicaErrorPtr)(const char *), void (*ModelicaWarningPtr)(const char *));
00188     EXTERNALMEDIA_EXPORT void TwoPhaseMedium_setState_dT_C_impl_err(double d, double T, int phase,
      void *state, const char *mediumName, const char *libraryName, const char *substanceName, void
      (*ModelicaErrorPtr)(const char *), void (*ModelicaWarningPtr)(const char *));
00189     EXTERNALMEDIA_EXPORT void TwoPhaseMedium_setState_ps_C_impl_err(double p, double s, int phase,
      void *state, const char *mediumName, const char *libraryName, const char *substanceName, void
      (*ModelicaErrorPtr)(const char *), void (*ModelicaWarningPtr)(const char *));
00190     EXTERNALMEDIA_EXPORT void TwoPhaseMedium_setState_hs_C_impl_err(double h, double s, int phase,
      void *state, const char *mediumName, const char *libraryName, const char *substanceName, void
      (*ModelicaErrorPtr)(const char *), void (*ModelicaWarningPtr)(const char *));
00191
00192     EXTERNALMEDIA_EXPORT double TwoPhaseMedium_partialDeriv_state_C_impl(const char *of, const char
      *wrt, const char *cst, void *state, const char *mediumName, const char *libraryName, const char
      *substanceName);
00193
00194     EXTERNALMEDIA_EXPORT double TwoPhaseMedium_prandtlNumber_C_impl(void *state, const char
      *mediumName, const char *libraryName, const char *substanceName);
00195     EXTERNALMEDIA_EXPORT double TwoPhaseMedium_temperature_C_impl(void *state, const char *mediumName,
      const char *libraryName, const char *substanceName);
00196     EXTERNALMEDIA_EXPORT double TwoPhaseMedium_velocityOfSound_C_impl(void *state, const char
      *mediumName, const char *libraryName, const char *substanceName);
00197     EXTERNALMEDIA_EXPORT double TwoPhaseMedium_isobaricExpansionCoefficient_C_impl(void *state, const
      char *mediumName, const char *libraryName, const char *substanceName);
00198     EXTERNALMEDIA_EXPORT double TwoPhaseMedium_specificHeatCapacityCp_C_impl(void *state,   const char
      *mediumName, const char *libraryName, const char *substanceName);
00199     EXTERNALMEDIA_EXPORT double TwoPhaseMedium_specificHeatCapacityCv_C_impl(void *state,   const char
      *mediumName, const char *libraryName, const char *substanceName);
00200     EXTERNALMEDIA_EXPORT double TwoPhaseMedium_density_C_impl(void *state, const char *mediumName,
      const char *libraryName, const char *substanceName);
00201     EXTERNALMEDIA_EXPORT double TwoPhaseMedium_density_derh_p_C_impl(void *state,   const char
      *mediumName, const char *libraryName, const char *substanceName);
00202     EXTERNALMEDIA_EXPORT double TwoPhaseMedium_density_derp_h_C_impl(void *state,   const char
      *mediumName, const char *libraryName, const char *substanceName);
00203     EXTERNALMEDIA_EXPORT double TwoPhaseMedium_dynamicViscosity_C_impl(void *state, const char
      *mediumName, const char *libraryName, const char *substanceName);
00204     EXTERNALMEDIA_EXPORT double TwoPhaseMedium_specificEnthalpy_C_impl(void *state, const char
      *mediumName, const char *libraryName, const char *substanceName);
00205     EXTERNALMEDIA_EXPORT double TwoPhaseMedium_isothermalCompressibility_C_impl(void *state, const
      char *mediumName, const char *libraryName, const char *substanceName);
00206     EXTERNALMEDIA_EXPORT double TwoPhaseMedium_thermalConductivity_C_impl(void *state, const char
      *mediumName, const char *libraryName, const char *substanceName);
00207     EXTERNALMEDIA_EXPORT double TwoPhaseMedium_pressure_C_impl(void *state, const char *mediumName,
      const char *libraryName, const char *substanceName);
00208     EXTERNALMEDIA_EXPORT double TwoPhaseMedium_specificEntropy_C_impl(void *state, const char
      *mediumName, const char *libraryName, const char *substanceName);
```

```
00209     EXTERNALMEDIA_EXPORT double TwoPhaseMedium_density_ph_der_C_impl(void *state,   const char
      *mediumName, const char *libraryName, const char *substanceName);
00210     EXTERNALMEDIA_EXPORT double TwoPhaseMedium_isentropicEnthalpy_C_impl(double p_downstream, void
      *refState,   const char *mediumName, const char *libraryName, const char *substanceName);
00211
00212     EXTERNALMEDIA_EXPORT void TwoPhaseMedium_setSat_p_C_impl(double p, void *sat, const char
      *mediumName, const char *libraryName, const char *substanceName);
00213     EXTERNALMEDIA_EXPORT void TwoPhaseMedium_setSat_T_C_impl(double T, void *sat, const char
      *mediumName, const char *libraryName, const char *substanceName);
00214     EXTERNALMEDIA_EXPORT void TwoPhaseMedium_setBubbleState_C_impl(void *sat, int phase, void *state,
      const char *mediumName, const char *libraryName, const char *substanceName);
00215     EXTERNALMEDIA_EXPORT void TwoPhaseMedium_setDewState_C_impl(void *sat, int phase, void *state,
      const char *mediumName, const char *libraryName, const char *substanceName);
00216
00217     EXTERNALMEDIA_EXPORT double TwoPhaseMedium_saturationTemperature_C_impl(double p, const char
      *mediumName, const char *libraryName, const char *substanceName);
00218     EXTERNALMEDIA_EXPORT double TwoPhaseMedium_saturationTemperature_derp_C_impl(double p, const char
      *mediumName, const char *libraryName, const char *substanceName);
00219     EXTERNALMEDIA_EXPORT double TwoPhaseMedium_saturationTemperature_derp_sat_C_impl(void *sat, const
      char *mediumName, const char *libraryName, const char *substanceName);
00220
00221     EXTERNALMEDIA_EXPORT double TwoPhaseMedium_dBubbleDensity_dPressure_C_impl(void *sat, const char
      *mediumName, const char *libraryName, const char *substanceName);
00222     EXTERNALMEDIA_EXPORT double TwoPhaseMedium_dDewDensity_dPressure_C_impl(void *sat, const char
      *mediumName, const char *libraryName, const char *substanceName);
00223     EXTERNALMEDIA_EXPORT double TwoPhaseMedium_dBubbleEnthalpy_dPressure_C_impl(void *sat, const char
      *mediumName, const char *libraryName, const char *substanceName);
00224     EXTERNALMEDIA_EXPORT double TwoPhaseMedium_dDewEnthalpy_dPressure_C_impl(void *sat, const char
      *mediumName, const char *libraryName, const char *substanceName);
00225     EXTERNALMEDIA_EXPORT double TwoPhaseMedium_bubbleDensity_C_impl(void *sat, const char *mediumName,
      const char *libraryName, const char *substanceName);
00226     EXTERNALMEDIA_EXPORT double TwoPhaseMedium_dewDensity_C_impl(void *sat, const char *mediumName,
      const char *libraryName, const char *substanceName);
00227     EXTERNALMEDIA_EXPORT double TwoPhaseMedium_bubbleEnthalpy_C_impl(void *sat, const char
      *mediumName, const char *libraryName, const char *substanceName);
00228     EXTERNALMEDIA_EXPORT double TwoPhaseMedium_dewEnthalpy_C_impl(void *sat, const char *mediumName,
      const char *libraryName, const char *substanceName);
00229     EXTERNALMEDIA_EXPORT double TwoPhaseMedium_saturationPressure_C_impl(double T, const char
      *mediumName, const char *libraryName, const char *substanceName);
00230     EXTERNALMEDIA_EXPORT double TwoPhaseMedium_surfaceTension_C_impl(void *sat, const char
      *mediumName, const char *libraryName, const char *substanceName);
00231     EXTERNALMEDIA_EXPORT double TwoPhaseMedium_bubbleEntropy_C_impl(void *sat, const char *mediumName,
      const char *libraryName, const char *substanceName);
00232     EXTERNALMEDIA_EXPORT double TwoPhaseMedium_dewEntropy_C_impl(void *sat, const char *mediumName,
      const char *libraryName, const char *substanceName);
00233
00234 #ifdef __cplusplus
00235 }
00236 #endif /* __cplusplus */
00237
00238 #endif /*EXTERNALMEDIALIB_H_*/
```

## 12.7 fluidconstants.h

```
00001 #ifndef FLUIDCONSTANTS_H_
00002 #define FLUIDCONSTANTS_H_
00003
00004 #include "include.h"
00005
00017 struct FluidConstants{
00019     double MM;
00021     double pc;
00023     double Tc;
00025     double dc;
00026     /* The following two functions are currently only available internally
00027     but do not have the required interface functions to be accessible from
00028     Modelica. */
00030     double hc;
00032     double sc;
00033
00038     FluidConstants() : MM(-1), pc(-1), Tc(-1), dc(-1), hc(-1), sc(-1) {};
00039 };
00040
00041 #endif /* FLUIDCONSTANTS_H_ */
```

## 12.8 FluidProp_COM.h

```
00001 /*  ============================================================================ */
00002 /*                                                                              */
```

```
00003 /*                        FluidProp C++ COM interface                        */
00004 /*                        ---------------------------                        */
00005 /*                                                                           */
00006 /*   The interface defined in this file, IFluidProp_COM is the direct C++    */
00007 /*   interface to the FluidProp COM server.  It is not recommended to use    */
00008 /*   this interface directly, please use the TFluidProp wrapper class.       */
00009 /*   This file should not be altered.                                        */
00010 /*                                                                           */
00011 /*   July, 2004, for FluidProp 1                                             */
00012 /*   January, 2006, for FluidProp 2                                          */
00013 /*   April, 2007, for FluidProp 2.3                                          */
00014 /*   November, 2012, for FluidProp 2.5                                       */
00015 /*                                                                           */
00016 /*  ======================================================================== */
00017
00018 #ifndef FluidProp_COM_h
00019 #define FluidProp_COM_h
00020
00021 #include "include.h"
00022 #include <comutil.h>
00023
00024
00025 /* The IFluidProp interface */
00026 interface IFluidProp_COM : public IDispatch
00027 {
00028   public:
00029     virtual void __stdcall CreateObject  ( BSTR ModelName, BSTR* ErrorMsg) = 0;
00030     virtual void __stdcall ReleaseObjects( ) = 0;
00031
00032     virtual void __stdcall SetFluid      ( BSTR ModelName, long nComp, SAFEARRAY** sa_Comp,
00033                                            SAFEARRAY** sa_Conc, BSTR* ErrorMsg) = 0;
00034     virtual void __stdcall SetFluid_M    ( BSTR ModelName, long nComp, SAFEARRAY* sa_Comp,
00035                                            SAFEARRAY* sa_Conc, BSTR* ErrorMsg) = 0;
00036
00037     virtual void __stdcall GetFluid      ( BSTR ModelName, long* nComp, SAFEARRAY** sa_Comp,
00038                                            SAFEARRAY** sa_Conc) = 0;
00039     virtual void __stdcall GetFluid_M    ( BSTR ModelName, long* nComp, SAFEARRAY** sa_Comp,
00040                                            SAFEARRAY** sa_Conc) = 0;
00041
00042     virtual void __stdcall GetFluidNames ( BSTR LongShort, BSTR ModelName, long* nComp,
00043                                            SAFEARRAY** sa_CompSet, BSTR* ErrorMsg) = 0;
00044     virtual void __stdcall GetFluidNames_M( BSTR LongShort, BSTR ModelName, long* nComp,
00045                                            SAFEARRAY** sa_CompSet, BSTR* ErrorMsg) = 0;
00046
00047     virtual void __stdcall GetCompSet    ( BSTR ModelName, long* nComp, SAFEARRAY** sa_CompSet,
00048                                            BSTR* ErrorMsg) = 0;
00049     virtual void __stdcall GetCompSet_M  ( BSTR ModelName, long* nComp, SAFEARRAY** sa_CompSet,
00050                                            BSTR* ErrorMsg) = 0;
00051
00052     virtual void __stdcall Pressure      ( BSTR InputSpec, double Input1, double Input2,
00053                                            double* Output, BSTR* ErrorMsg) = 0;
00054     virtual void __stdcall Temperature   ( BSTR InputSpec, double Input1, double Input2,
00055                                            double* Output, BSTR* ErrorMsg) = 0;
00056     virtual void __stdcall SpecVolume    ( BSTR InputSpec, double Input1, double Input2,
00057                                            double* Output, BSTR* ErrorMsg) = 0;
00058     virtual void __stdcall Density       ( BSTR InputSpec, double Input1, double Input2,
00059                                            double* Output, BSTR* ErrorMsg) = 0;
00060     virtual void __stdcall Enthalpy      ( BSTR InputSpec, double Input1, double Input2,
00061                                            double* Output, BSTR* ErrorMsg) = 0;
00062     virtual void __stdcall Entropy       ( BSTR InputSpec, double Input1, double Input2,
00063                                            double* Output, BSTR* ErrorMsg) = 0;
00064     virtual void __stdcall IntEnergy     ( BSTR InputSpec, double Input1, double Input2,
00065                                            double* Output, BSTR* ErrorMsg) = 0;
00066     virtual void __stdcall VaporQual     ( BSTR InputSpec, double Input1, double Input2,
00067                                            double* Output, BSTR* ErrorMsg) = 0;
00068
00069     virtual void __stdcall LiquidCmp     ( BSTR InputSpec, double Input1, double Input2,
00070                                            SAFEARRAY** Output, BSTR* ErrorMsg) = 0;
00071     virtual void __stdcall LiquidCmp_M   ( BSTR InputSpec, double Input1, double Input2,
00072                                            SAFEARRAY** Output, BSTR* ErrorMsg) = 0;
00073
00074     virtual void __stdcall VaporCmp      ( BSTR InputSpec, double Input1, double Input2,
00075                                            SAFEARRAY** Output, BSTR* ErrorMsg) = 0;
00076     virtual void __stdcall VaporCmp_M    ( BSTR InputSpec, double Input1, double Input2,
00077                                            SAFEARRAY** Output, BSTR* ErrorMsg) = 0;
00078
00079     virtual void __stdcall HeatCapV      ( BSTR InputSpec, double Input1, double Input2,
00080                                            double* Output, BSTR* ErrorMsg) = 0;
00081     virtual void __stdcall HeatCapP      ( BSTR InputSpec, double Input1, double Input2,
00082                                            double* Output, BSTR* ErrorMsg) = 0;
00083     virtual void __stdcall SoundSpeed    ( BSTR InputSpec, double Input1, double Input2,
00084                                            double* Output, BSTR* ErrorMsg) = 0;
00085     virtual void __stdcall Alpha         ( BSTR InputSpec, double Input1, double Input2,
00086                                            double* Output, BSTR* ErrorMsg) = 0;
00087     virtual void __stdcall Beta          ( BSTR InputSpec, double Input1, double Input2,
00088                                            double* Output, BSTR* ErrorMsg) = 0;
00089     virtual void __stdcall Chi           ( BSTR InputSpec, double Input1, double Input2,
```

```
00090                                           double* Output, BSTR* ErrorMsg) = 0;
00091      virtual void __stdcall Fi            ( BSTR InputSpec, double Input1, double Input2,
00092                                           double* Output, BSTR* ErrorMsg) = 0;
00093      virtual void __stdcall Ksi           ( BSTR InputSpec, double Input1, double Input2,
00094                                           double* Output, BSTR* ErrorMsg) = 0;
00095      virtual void __stdcall Psi           ( BSTR InputSpec, double Input1, double Input2,
00096                                           double* Output, BSTR* ErrorMsg) = 0;
00097      virtual void __stdcall Zeta          ( BSTR InputSpec, double Input1, double Input2,
00098                                           double* Output, BSTR* ErrorMsg) = 0;
00099      virtual void __stdcall Theta         ( BSTR InputSpec, double Input1, double Input2,
00100                                           double* Output, BSTR* ErrorMsg) = 0;
00101      virtual void __stdcall Kappa         ( BSTR InputSpec, double Input1, double Input2,
00102                                           double* Output, BSTR* ErrorMsg) = 0;
00103      virtual void __stdcall Gamma         ( BSTR InputSpec, double Input1, double Input2,
00104                                           double* Output, BSTR* ErrorMsg) = 0;
00105
00106      virtual void __stdcall Viscosity     ( BSTR InputSpec, double Input1, double Input2,
00107                                           double* Output, BSTR* ErrorMsg) = 0;
00108      virtual void __stdcall ThermCond     ( BSTR InputSpec, double Input1, double Input2,
00109                                           double* Output, BSTR* ErrorMsg) = 0;
00110
00111      virtual void __stdcall AllProps      ( BSTR InputSpec, double Input1, double Input2,
00112                                           double* P, double* T, double* v, double* d,
00113                                           double* h, double* s, double* u, double* q,
00114                                           SAFEARRAY** x, SAFEARRAY** y, double* cv, double* cp,
00115                                           double* c, double* alpha, double* beta, double* chi,
00116                                           double* fi, double* ksi, double* psi, double* zeta,
00117                                           double* theta, double* kappa, double* gamma,
00118                                           double* eta, double* lambda, BSTR* ErrorMsg) = 0;
00119      virtual void __stdcall AllProps_M    ( BSTR InputSpec, double Input1, double Input2,
00120                                           double* P, double* T, double* v, double* d,
00121                                           double* h, double* s, double* u, double* q,
00122                                           SAFEARRAY** x, SAFEARRAY** y, double* cv, double* cp,
00123                                           double* c, double* alpha, double* beta, double* chi,
00124                                           double* fi, double* ksi, double* psi, double* zeta,
00125                                           double* theta, double* kappa, double* gamma,
00126                                           double* eta, double* lambda, BSTR* ErrorMsg) = 0;
00127
00128      virtual void __stdcall AllPropsSat   ( BSTR InputSpec, double Input1, double Input2,
00129                                           double* P, double* T, double* v, double* d,
00130                                           double* h, double* s, double* u, double* q,
00131                                           SAFEARRAY** x, SAFEARRAY** y, double* cv, double* cp,
00132                                           double* c, double* alpha, double* beta, double* chi,
00133                                           double* fi, double* ksi, double* psi, double* zeta,
00134                                           double* theta, double* kappa, double* gamma,
00135                                           double* eta, double* lambda, double* d_liq,
00136                                           double* d_vap, double* h_liq, double* h_vap,
00137                                           double* T_sat, double* dd_liq_dP, double* dd_vap_dP,
00138                                           double* dh_liq_dP, double* dh_vap_dP,
00139                                           double* dT_sat_dP, BSTR* ErrorMsg) = 0;
00140      virtual void __stdcall AllPropsSat_M ( BSTR InputSpec, double Input1, double Input2,
00141                                           double* P, double* T, double* v, double* d,
00142                                           double* h, double* s, double* u, double* q,
00143                                           SAFEARRAY** x, SAFEARRAY** y, double* cv, double* cp,
00144                                           double* c, double* alpha, double* beta, double* chi,
00145                                           double* fi, double* ksi, double* psi, double* zeta,
00146                                           double* theta, double* kappa, double* gamma,
00147                                           double* eta, double* lambda, double* d_liq,
00148                                           double* d_vap, double* h_liq, double* h_vap,
00149                                           double* T_sat, double* dd_liq_dP, double* dd_vap_dP,
00150                                           double* dh_liq_dP, double* dh_vap_dP,
00151                                           double* dT_sat_dP, BSTR* ErrorMsg) = 0;
00152
00153      virtual void __stdcall Solve         ( BSTR FuncSpec, double FuncVal, BSTR InputSpec,
00154                                           long Target, double FixedVal, double MinVal,
00155                                           double MaxVal, double* Output, BSTR* ErrorMsg) = 0;
00156
00157      virtual void __stdcall Mmol          ( double* Output, BSTR* ErrorMsg) = 0;
00158      virtual void __stdcall Tcrit         ( double* Output, BSTR* ErrorMsg) = 0;
00159      virtual void __stdcall Pcrit         ( double* Output, BSTR* ErrorMsg) = 0;
00160      virtual void __stdcall Tmin          ( double* Output, BSTR* ErrorMsg) = 0;
00161      virtual void __stdcall Tmax          ( double* Output, BSTR* ErrorMsg) = 0;
00162      virtual void __stdcall AllInfo       ( double* M_mol, double* T_crit, double* P_crit,
00163                                           double* T_min, double* T_max , BSTR* ErrorMsg) = 0;
00164
00165      virtual void __stdcall SetUnits      ( BSTR UnitSet, BSTR MassOrMole, BSTR Properties,
00166                                           BSTR Units, BSTR* ErrorMsg) = 0;
00167      virtual void __stdcall SetRefState   ( double T_ref, double P_ref, BSTR* ErrorMsg) = 0;
00168
00169      virtual void __stdcall freeStanMix_Psat_k1   ( ) = 0;    /* C++ interface not yet implemented */
00170      virtual void __stdcall zFlow_vu          ( ) = 0;    /* C++ interface not yet implemented */
00171      virtual void __stdcall GetVersion        ( BSTR ModelName, SAFEARRAY** sa_version) = 0;
00172
00173      virtual void __stdcall AllTransProps     ( ) = 0;    /* C++ interface not yet implemented */
00174      virtual void __stdcall SaturationLine    ( ) = 0;    /* C++ interface not yet implemented */
00175      virtual void __stdcall IsoLine           ( ) = 0;    /* C++ interface not yet implemented */
00176      virtual void __stdcall freeStanMix_xy_A_alfa ( ) = 0;    /* C++ interface not yet implemented */
```

```
00177      virtual void __stdcall PCP_SAFT_xy_kij   ( ) = 0;    /* C++ interface not yet implemented */
00178      virtual void __stdcall PCP_SAFT_hsxy_mp  ( ) = 0;    /* C++ interface not yet implemented */
00179      virtual void __stdcall PCP_SAFT_hsxy_mp_M( ) = 0;    /* C++ interface not yet implemented */
00180
00181      virtual void __stdcall FugaCoef          ( BSTR InputSpec, double Input1, double Input2,
00182                                                 SAFEARRAY** Output, BSTR* ErrorMsg) = 0;
00183      virtual void __stdcall FugaCoef_M        ( BSTR InputSpec, double Input1, double Input2,
00184                                                 SAFEARRAY** Output, BSTR* ErrorMsg) = 0;
00185
00186      virtual void __stdcall SurfTens          ( BSTR InputSpec, double Input1, double Input2,
00187                                                 double* Output, BSTR* ErrorMsg) = 0;
00188
00189      virtual void __stdcall GibbsEnergy       ( BSTR InputSpec, double Input1, double Input2,
00190                                                 double* Output, BSTR* ErrorMsg) = 0;
00191
00192      virtual void __stdcall CapeOpenDeriv     ( BSTR InputSpec, double Input1, double Input2,
00193                                                 SAFEARRAY** v, SAFEARRAY** h, SAFEARRAY** s,
00194                                                 SAFEARRAY** G, SAFEARRAY** lnphi, BSTR* ErrorMsg) = 0;
00195
00196      virtual void __stdcall SpecVolume_Deriv  ( BSTR InputSpec, double Input1, double Input2,
00197                                                 SAFEARRAY** Output, BSTR* ErrorMsg) = 0;
00198      virtual void __stdcall Enthalpy_Deriv    ( BSTR InputSpec, double Input1, double Input2,
00199                                                 SAFEARRAY** Output, BSTR* ErrorMsg) = 0;
00200      virtual void __stdcall Entropy_Deriv     ( BSTR InputSpec, double Input1, double Input2,
00201                                                 SAFEARRAY** Output, BSTR* ErrorMsg) = 0;
00202      virtual void __stdcall GibbsEnergy_Deriv ( BSTR InputSpec, double Input1, double Input2,
00203                                                 SAFEARRAY** Output, BSTR* ErrorMsg) = 0;
00204      virtual void __stdcall FugaCoef_Deriv    ( BSTR InputSpec, double Input1, double Input2,
00205                                                 SAFEARRAY** Output, BSTR* ErrorMsg) = 0;
00206
00207      virtual void __stdcall PCP_SAFT_P_kij    ( ) = 0;    /* C++ interface not yet implemented */
00208      virtual void __stdcall PCP_SAFT_T_kij    ( ) = 0;    /* C++ interface not yet implemented */
00209      virtual void __stdcall PCP_SAFT_Prho_mseT( ) = 0;    /* C++ interface not yet implemented */
00210      virtual void __stdcall CalcProp          ( ) = 0;    /* C++ interface not yet implemented */
00211 };
00212
00213 #endif /* FluidProp_COM_h */
```

## 12.9 FluidProp_IF.h

```
00001 /* ========================================================================== */
00002 /*                                                                            */
00003 /*                          FluidProp C++ interface                           */
00004 /*                          -----------------------                           */
00005 /*                                                                            */
00006 /*  The class implemented in this file, TFluidProp, is as a wrapper class for */
00007 /*  the IFluidProp_COM interface.  TFluidProp hides COM specific details      */
00008 /*  like safe arrays (SAFEARRAY) and binary strings (BSTR) in IFluidProp_COM. */
00009 /*  In the TFluidProp class only standard C++ data types are used.  This is   */
00010 /*  the recommended way working with the FluidProp COM server in C++.         */
00011 /*                                                                            */
00012 /*  July, 2004, for FluidProp 1                                               */
00013 /*  January, 2006, for FluidProp 2                                            */
00014 /*  April, 2007, for FluidProp 2.3                                            */
00015 /*  November, 2012, for FluidProp 2.5                                         */
00016 /*                                                                            */
00017 /* ========================================================================== */
00018
00019 #ifndef FluidProp_IF_h
00020 #define FluidProp_IF_h
00021
00022 #include "include.h"
00023
00024 #pragma comment(lib, "comsuppw.lib")
00025
00026 #include <string>
00027 using std::string;
00028
00029 #include "FluidProp_COM.h"
00030
00031
00032 /* The TFluidProp class */
00033 class TFluidProp
00034 {
00035    public:
00036
00037       TFluidProp();
00038       ~TFluidProp();
00039
00040       bool IsValid();
00041
00042       void CreateObject  ( string ModelName, string* ErrorMsg);
00043       void ReleaseObjects( );
```

```
00044
00045       void SetFluid      ( string ModelName, int nComp, string* Comp, double* Conc,
00046                            string* ErrorMsg);
00047       void GetFluid      ( string* ModelName, int* nComp, string* Comp, double* Conc,
00048                            bool CompInfo = true);
00049       void GetFluidNames ( string LongShort, string ModelName, int* nFluids, string* FluidNames,
00050                            string* ErrorMsg);
00051       void GetCompSet    ( string ModelName, int* nComps, string* CompSet, string* ErrorMsg);
00052
00053       double Pressure    ( string InputSpec, double Input1, double Input2, string* ErrorMsg);
00054       double Temperature ( string InputSpec, double Input1, double Input2, string* ErrorMsg);
00055       double SpecVolume  ( string InputSpec, double Input1, double Input2, string* ErrorMsg);
00056       double Density     ( string InputSpec, double Input1, double Input2, string* ErrorMsg);
00057       double Enthalpy    ( string InputSpec, double Input1, double Input2, string* ErrorMsg);
00058       double Entropy     ( string InputSpec, double Input1, double Input2, string* ErrorMsg);
00059       double IntEnergy   ( string InputSpec, double Input1, double Input2, string* ErrorMsg);
00060       double VaporQual   ( string InputSpec, double Input1, double Input2, string* ErrorMsg);
00061       double* LiquidCmp  ( string InputSpec, double Input1, double Input2, string* ErrorMsg);
00062       double* VaporCmp   ( string InputSpec, double Input1, double Input2, string* ErrorMsg);
00063       double HeatCapV    ( string InputSpec, double Input1, double Input2, string* ErrorMsg);
00064       double HeatCapP    ( string InputSpec, double Input1, double Input2, string* ErrorMsg);
00065       double SoundSpeed  ( string InputSpec, double Input1, double Input2, string* ErrorMsg);
00066       double Alpha       ( string InputSpec, double Input1, double Input2, string* ErrorMsg);
00067       double Beta        ( string InputSpec, double Input1, double Input2, string* ErrorMsg);
00068       double Chi         ( string InputSpec, double Input1, double Input2, string* ErrorMsg);
00069       double Fi          ( string InputSpec, double Input1, double Input2, string* ErrorMsg);
00070       double Ksi         ( string InputSpec, double Input1, double Input2, string* ErrorMsg);
00071       double Psi         ( string InputSpec, double Input1, double Input2, string* ErrorMsg);
00072       double Zeta        ( string InputSpec, double Input1, double Input2, string* ErrorMsg);
00073       double Theta       ( string InputSpec, double Input1, double Input2, string* ErrorMsg);
00074       double Kappa       ( string InputSpec, double Input1, double Input2, string* ErrorMsg);
00075       double Gamma       ( string InputSpec, double Input1, double Input2, string* ErrorMsg);
00076       double Viscosity   ( string InputSpec, double Input1, double Input2, string* ErrorMsg);
00077       double ThermCond   ( string InputSpec, double Input1, double Input2, string* ErrorMsg);
00078
00079       void AllProps      ( string InputSpec, double Input1, double Input2, double& P, double& T,
00080                            double& v, double& d, double& h, double& s, double& u, double& q,
00081                            double* x, double* y, double& cv, double& cp, double& c, double& alpha,
00082                            double& beta, double& chi, double& fi, double& ksi, double& psi,
00083                            double& zeta, double& theta, double& kappa, double& gamma, double& eta,
00084                            double& lambda, string* ErrorMsg);
00085
00086      /*  Compute all the properties at once, including saturation properties */
00087       void AllPropsSat   ( string InputSpec, double Input1, double Input2, double& P, double& T,
00088                            double& v, double& d, double& h, double& s, double& u, double& q,
00089                            double* x, double* y, double& cv, double& cp, double& c, double& alpha,
00090                            double& beta, double& chi, double& fi, double& ksi, double& psi,
00091                            double& zeta, double& theta, double& kappa, double& gamma, double& eta,
00092                            double& lambda, double& d_liq, double& d_vap, double& h_liq, double& h_vap,
00093                            double& T_sat, double& dd_liq_dP, double& dd_vap_dP, double& dh_liq_dP,
00094                            double& dh_vap_dP, double& dT_sat_dP, string* ErrorMsg);
00095
00096       double Solve       ( string FuncSpec, double FuncVal, string InputSpec, long Target,
00097                            double FixedVal, double MinVal, double MaxVal, string* ErrorMsg);
00098
00099       double Mmol        ( string* ErrorMsg);
00100       double Tcrit       ( string* ErrorMsg);
00101       double Pcrit       ( string* ErrorMsg);
00102       double Tmin        ( string* ErrorMsg);
00103       double Tmax        ( string* ErrorMsg);
00104       void AllInfo       ( double& Mmol, double& Tcrit, double& Pcrit, double& Tmin, double& Tmax,
00105                            string* ErrorMsg);
00106
00107       void SetUnits      ( string UnitSet, string MassOrMole, string Properties, string Units,
00108                            string* ErrorMsg);
00109       void SetRefState   ( double T_ref, double P_ref, string* ErrorMsg);
00110       void GetVersion    ( string ModelName, int* version);
00111
00112       double* FugaCoef   ( string InputSpec, double Input1, double Input2, string* ErrorMsg);
00113       double SurfTens    ( string InputSpec, double Input1, double Input2, string* ErrorMsg);
00114       double GibbsEnergy ( string InputSpec, double Input1, double Input2, string* ErrorMsg);
00115       void CapeOpenDeriv ( string InputSpec, double Input1, double Input2, double* v, double* h,
00116                            double* s, double* G, double* lnphi, string* ErrorMsg);
00117
00118       double* SpecVolume_Deriv ( string InputSpec, double Input1, double Input2, string* ErrorMsg);
00119       double* Enthalpy_Deriv   ( string InputSpec, double Input1, double Input2, string* ErrorMsg);
00120       double* Entropy_Deriv    ( string InputSpec, double Input1, double Input2, string* ErrorMsg);
00121       double* GibbsEnergy_Deriv( string InputSpec, double Input1, double Input2, string* ErrorMsg);
00122       double* FugaCoef_Deriv   ( string InputSpec, double Input1, double Input2, string* ErrorMsg);
00123
00124    private:
00125
00126       IClassFactory*  ClassFactory ;      /* Pointer to class factory */
00127       IFluidProp_COM* FluidProp_COM;      /* Pointer to FluidProp interface */
00128
00129 };
00130
```

```
00131 #endif /* FluidProp_IF_h */
```

## 12.10 fluidpropsolver.h

```
00001
00035 #ifndef FLUIDPROPSOLVER_H_
00036 #define FLUIDPROPSOLVER_H_
00037
00038 #include "include.h"
00039 #if (EXTERNALMEDIA_FLUIDPROP == 1)
00040
00041 #include "basesolver.h"
00042
00043 #include "FluidProp_IF.h"
00044
00045 class FluidPropSolver : public BaseSolver{
00046 public:
00047     FluidPropSolver(const string &mediumName, const string &libraryName, const string &substanceName);
00048     ~FluidPropSolver();
00049     virtual void setFluidConstants();
00050
00051     virtual void setSat_p(double &p, ExternalSaturationProperties *const properties);
00052     virtual void setSat_T(double &T, ExternalSaturationProperties *const properties);
00053
00054     virtual void setState_ph(double &p, double &h, int &phase, ExternalThermodynamicState *const
      properties);
00055     virtual void setState_pT(double &p, double &T, ExternalThermodynamicState *const properties);
00056     virtual void setState_dT(double &d, double &T, int &phase, ExternalThermodynamicState *const
      properties);
00057     virtual void setState_ps(double &p, double &s, int &phase, ExternalThermodynamicState *const
      properties);
00058     virtual void setBubbleState(ExternalSaturationProperties *const properties, int phase,
00059                                 ExternalThermodynamicState *const bubbleProperties);
00060     virtual void setDewState(ExternalSaturationProperties *const properties, int phase,
00061                              ExternalThermodynamicState *const dewProperties);
00062     virtual double isentropicEnthalpy(double &p, ExternalThermodynamicState *const properties);
00063
00064 protected:
00065     TFluidProp FluidProp;  /* Instance of FluidProp wrapper object */
00066     bool isError(string ErrorMsg);
00067     bool licenseError(string ErrorMsg);
00068 };
00069
00070 #endif
00071
00072 #endif /* FLUIDPROPSOLVER_H_ */
```

## 12.11 importer.h

```
00001 /* ****************************************************************
00002  * Working around Windows' dynamic linker
00003  *
00004  * Federico Terraneo, Mahder Gebremedhin May 2022
00005  ******************************************************************/
00006
00007 #pragma once
00008
00009 #ifdef _WIN32
00010
00011 #define WIN32_LEAN_AND_MEAN
00012 #include <windows.h>
00013 #include <libloaderapi.h>
00014 #include <errhandlingapi.h>
00015 #include <psapi.h>
00016
00017 template<typename T>
00018 T tryImportSymbol(const char *funcName)
00019 {
00020     /* TODO: we should do caching */
00021
00022     /* First check if the executable itself exports the symbol we want */
00023     HMODULE exe = GetModuleHandleA(NULL);
00024     if(exe == NULL)
00025     {
00026         fprintf(stderr, "Can't get handle to executable (error %d)\n", GetLastError());
00027         exit(1);
00028     }
00029     T pfn = reinterpret_cast<T>(GetProcAddress(exe, funcName));
00030     if(pfn) return pfn;
```

```
00031
00032     /* If we don't find it in the executable, then we search it in all loaded DLLs */
00033     HANDLE process = GetCurrentProcess();
00034     if(process == NULL)
00035     {
00036         fprintf(stderr, "Can't get a handle to current process (error %d)\n", GetLastError());
00037         exit(1);
00038     }
00039
00040     HMODULE loaded_modules[1024];
00041     DWORD cbNeeded;
00042     auto result = EnumProcessModules(process, loaded_modules, sizeof(loaded_modules), &cbNeeded);
00043     CloseHandle(process);
00044     if(!result)
00045     {
00046         fprintf(stderr, "Can't enumerate loaded modules (error %d)\n", GetLastError());
00047         exit(1);
00048     }
00049
00050     int num_modules = cbNeeded / sizeof(HMODULE); /* Actual number of loaded modules */
00051     for(int i = 0; i < num_modules; i++)
00052     {
00053         T pfn = reinterpret_cast<T>(GetProcAddress(loaded_modules[i], funcName));
00054         if(pfn) return pfn;
00055     }
00056
00057     /* not found */
00058     return NULL;
00059 }
00060
00061 template<typename T>
00062 T importSymbol(const char *funcName)
00063 {
00064     T result = tryImportSymbol<T>(funcName);
00065     if(result) return result;
00066
00067     fprintf(stderr, "Can't get handle to %s in all loaded modules.\n", funcName);
00068     exit(1);
00069 }
00070
00071 #define IMPORT(x,y) auto y = importSymbol<x>(#y)
00072
00073 #else /* _WIN32 */
00074
00075 #include "ModelicaUtilities.h"
00076
00077 /*  Nothing to do on Linux, its linker just works */
00078 #define IMPORT(x,y)
00079
00080 #endif /* _WIN32 */
```

## 12.12 Sources/include.h File Reference

Main include file.

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

### Macros

- #define EXTERNALMEDIA_FLUIDPROP 0
- #define EXTERNALMEDIA_COOLPROP 1
- #define NAN 0xffffffff
- #define ISNAN(x)

### 12.12.1 Detailed Description

Main include file.

This is a main include file 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



### 12.12.2 Macro Definition Documentation

#### 12.12.2.1 EXTERNALMEDIA_COOLPROP

```
#define EXTERNALMEDIA_COOLPROP 1
```

CoolProp solver

Set this preprocessor variable to 1 to include the interface to the CoolProp solver developed and maintained by Jorrit Wronski et al.

#### 12.12.2.2 EXTERNALMEDIA_FLUIDPROP

```
#define EXTERNALMEDIA_FLUIDPROP 0
```

FluidProp solver

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

#### 12.12.2.3 ISNAN

```
#define ISNAN(
              x)
```

**Value:**
```
(x == NAN)
```

#### 12.12.2.4 NAN

```
#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.

## 12.13 include.h

Go to the documentation of this file.

```
00001
00015 #ifndef INCLUDE_H_
00016 #define INCLUDE_H_
00017
00018 /*************************************************************************
00019  *               Start of user option selection
00020  ************************************************************************/
00021
00022 /* Selection of used external fluid property computation packages. */
00028 #ifndef EXTERNALMEDIA_FLUIDPROP
00029 #define EXTERNALMEDIA_FLUIDPROP 0
00030 #endif
00031
00032 /* Selection of used external fluid property computation packages. */
00038 #ifndef EXTERNALMEDIA_COOLPROP
00039 #define EXTERNALMEDIA_COOLPROP 1
00040 #endif
00041
00047 #include <math.h>
00048 #ifndef NAN
00049 #define NAN 0xffffffff
00050 #endif
00051 #ifndef ISNAN
00052 #define ISNAN(x) (x == NAN)
00053 #endif
00054
00055 /*************************************************************************
00056  *                End of user option selection
00057  *            Do not change anything below this line
00058  ************************************************************************/
00059
00060 /* General purpose includes */
00061 #include <map>
00062 using std::map;
00063
00064 #include <string>
00065 using std::string;
00066
00067 /* Include error handling */
00068 #include "errorhandling.h"
00069
00070 #endif /* INCLUDE_H_ */
```

## 12.14 ModelicaUtilities.h

```
00001 /* ModelicaUtilities.h - External utility functions header
00002
00003    Copyright (C) 2010-2020, Modelica Association and contributors
00004    All rights reserved.
00005
00006    Redistribution and use in source and binary forms, with or without
00007    modification, are permitted provided that the following conditions are met:
00008
00009    1. Redistributions of source code must retain the above copyright notice,
00010       this list of conditions and the following disclaimer.
00011
00012    2. Redistributions in binary form must reproduce the above copyright
00013       notice, this list of conditions and the following disclaimer in the
00014       documentation and/or other materials provided with the distribution.
00015
00016    3. Neither the name of the copyright holder nor the names of its
00017       contributors may be used to endorse or promote products derived from
00018       this software without specific prior written permission.
00019
00020    THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND
00021    ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
00022    WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
00023    DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE
00024    FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
00025    DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
00026    SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
00027    CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
00028    OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
00029    OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
00030 */
00031
00032 /* Utility functions which can be called by external Modelica functions.
00033
```

```
00034    These functions are defined in section 12.8.6 of the
00035    Modelica Specification 3.0 and section 12.9.6 of the
00036    Modelica Specification 3.1 and later.
00037
00038    A generic C-implementation of these functions cannot be given,
00039    because it is tool dependent how strings are output in a
00040    window of the respective simulation tool. Therefore, only
00041    this header file is shipped with the Modelica Standard Library.
00042 */
00043
00044 #ifndef MODELICA_UTILITIES_H
00045 #define MODELICA_UTILITIES_H
00046
00047 #include <stddef.h>
00048 #include <stdarg.h>
00049
00050 #if defined(__cplusplus)
00051 extern "C" {
00052 #endif
00053
00054 /*
00055   Some of the functions never return to the caller. In order to compile
00056   external Modelica C-code in most compilers, noreturn attributes need to
00057   be present to avoid warnings or errors.
00058
00059   The following macros handle noreturn attributes according to the latest
00060   C11/C++11 standard with fallback to GNU, Clang or MSVC extensions if using
00061   an older compiler.
00062 */
00063 #undef MODELICA_NORETURN
00064 #undef MODELICA_NORETURNATTR
00065 #if __STDC_VERSION__ >= 201112L
00066 #define MODELICA_NORETURN _Noreturn
00067 #define MODELICA_NORETURNATTR
00068 #elif defined(__cplusplus) && __cplusplus >= 201103L
00069 #if (defined(__GNUC__) && __GNUC__ >= 5) || \
00070     (defined(__GNUC__) && defined(__GNUC_MINOR__) && __GNUC__ == 4 && __GNUC_MINOR__ >= 8)
00071 #define MODELICA_NORETURN [[noreturn]]
00072 #define MODELICA_NORETURNATTR
00073 #elif (defined(__GNUC__) && __GNUC__ >= 3) || \
00074       (defined(__GNUC__) && defined(__GNUC_MINOR__) && __GNUC__ == 2 && __GNUC_MINOR__ >= 8)
00075 #define MODELICA_NORETURN
00076 #define MODELICA_NORETURNATTR __attribute__((noreturn))
00077 #elif defined(__GNUC__)
00078 #define MODELICA_NORETURN
00079 #define MODELICA_NORETURNATTR
00080 #else
00081 #define MODELICA_NORETURN [[noreturn]]
00082 #define MODELICA_NORETURNATTR
00083 #endif
00084 #elif defined(__clang__)
00085 /* Encapsulated for Clang since GCC fails to process __has_attribute */
00086 #if __has_attribute(noreturn)
00087 #define MODELICA_NORETURN
00088 #define MODELICA_NORETURNATTR __attribute__((noreturn))
00089 #else
00090 #define MODELICA_NORETURN
00091 #define MODELICA_NORETURNATTR
00092 #endif
00093 #elif (defined(__GNUC__) && __GNUC__ >= 3) || \
00094       (defined(__GNUC__) && defined(__GNUC_MINOR__) && __GNUC__ == 2 && __GNUC_MINOR__ >= 8) || \
00095       (defined(__SUNPRO_C) && __SUNPRO_C >= 0x5110)
00096 #define MODELICA_NORETURN
00097 #define MODELICA_NORETURNATTR __attribute__((noreturn))
00098 #elif (defined(_MSC_VER) && _MSC_VER >= 1200) || \
00099       defined(__BORLANDC__)
00100 #define MODELICA_NORETURN __declspec(noreturn)
00101 #define MODELICA_NORETURNATTR
00102 #else
00103 #define MODELICA_NORETURN
00104 #define MODELICA_NORETURNATTR
00105 #endif
00106
00107 /*
00108   The following macros handle format attributes for type-checks against a
00109   format string.
00110 */
00111
00112 #if defined(__clang__)
00113 /* Encapsulated for Clang since GCC fails to process __has_attribute */
00114 #if __has_attribute(format)
00115 #define MODELICA_FORMATATTR_PRINTF __attribute__((format(printf, 1, 2)))
00116 #define MODELICA_FORMATATTR_VPRINTF __attribute__((format(printf, 1, 0)))
00117 #else
00118 #define MODELICA_FORMATATTR_PRINTF
00119 #define MODELICA_FORMATATTR_VPRINTF
00120 #endif
```

```
00121 #elif defined(__GNUC__) && __GNUC__ >= 3
00122 #define MODELICA_FORMATATTR_PRINTF __attribute__((format(printf, 1, 2)))
00123 #define MODELICA_FORMATATTR_VPRINTF __attribute__((format(printf, 1, 0)))
00124 #else
00125 #define MODELICA_FORMATATTR_PRINTF
00126 #define MODELICA_FORMATATTR_VPRINTF
00127 #endif
00128
00129 void ModelicaMessage(const char *string);
00130 /*
00131 Output the message string (no format control).
00132 */
00133
00134
00135 void ModelicaFormatMessage(const char *string, ...) MODELICA_FORMATATTR_PRINTF;
00136 /*
00137 Output the message under the same format control as the C-function printf.
00138 */
00139
00140
00141 void ModelicaVFormatMessage(const char *string, va_list args) MODELICA_FORMATATTR_VPRINTF;
00142 /*
00143 Output the message under the same format control as the C-function vprintf.
00144 */
00145
00146
00147 MODELICA_NORETURN void ModelicaError(const char *string) MODELICA_NORETURNATTR;
00148 /*
00149 Output the error message string (no format control). This function
00150 never returns to the calling function, but handles the error
00151 similarly to an assert in the Modelica code.
00152 */
00153
00154 void ModelicaWarning(const char *string);
00155 /*
00156 Output the warning message string (no format control).
00157 */
00158
00159 void ModelicaFormatWarning(const char *string, ...) MODELICA_FORMATATTR_PRINTF;
00160 /*
00161 Output the warning message under the same format control as the C-function printf.
00162 */
00163
00164 void ModelicaVFormatWarning(const char *string, va_list args) MODELICA_FORMATATTR_VPRINTF;
00165 /*
00166 Output the warning message under the same format control as the C-function vprintf.
00167 */
00168
00169 MODELICA_NORETURN void ModelicaFormatError(const char *string, ...) MODELICA_NORETURNATTR
      MODELICA_FORMATATTR_PRINTF;
00170 /*
00171 Output the error message under the same format control as the C-function
00172 printf. This function never returns to the calling function,
00173 but handles the error similarly to an assert in the Modelica code.
00174 */
00175
00176
00177 MODELICA_NORETURN void ModelicaVFormatError(const char *string, va_list args) MODELICA_NORETURNATTR
      MODELICA_FORMATATTR_VPRINTF;
00178 /*
00179 Output the error message under the same format control as the C-function
00180 vprintf. This function never returns to the calling function,
00181 but handles the error similarly to an assert in the Modelica code.
00182 */
00183
00184
00185 char* ModelicaAllocateString(size_t len);
00186 /*
00187 Allocate memory for a Modelica string which is used as return
00188 argument of an external Modelica function. Note, that the storage
00189 for string arrays (= pointer to string array) is still provided by the
00190 calling program, as for any other array. If an error occurs, this
00191 function does not return, but calls "ModelicaError".
00192 */
00193
00194
00195 char* ModelicaAllocateStringWithErrorReturn(size_t len);
00196 /*
00197 Same as ModelicaAllocateString, except that in case of error, the
00198 function returns 0. This allows the external function to close files
00199 and free other open resources in case of error. After cleaning up
00200 resources use ModelicaError or ModelicaFormatError to signal
00201 the error.
00202 */
00203
00204 #if defined(__cplusplus)
00205 }
```

```
00206 #endif
00207
00208 #endif
```

## 12.15 solvermap.h

```
00001 #ifndef SOLVERMAP_H_
00002 #define SOLVERMAP_H_
00003
00004 #include "include.h"
00005
00006 class BaseSolver;
00007
00018 class SolverMap{
00019 public:
00020     static BaseSolver *getSolver(const string &mediumName, const string &libraryName, const string
      &substanceName);
00021     static string solverKey(const string &libraryName, const string &substanceName);
00022
00023 protected:
00025     static map<string, BaseSolver*> _solvers;
00026 };
00027
00028 #endif /* SOLVERMAP_H_ */
```

## 12.16 testsolver.h

```
00001 #ifndef TESTSOLVER_H_
00002 #define TESTSOLVER_H_
00003
00004 #include "basesolver.h"
00005
00035 class TestSolver : public BaseSolver{
00036 public:
00037     TestSolver(const string &mediumName, const string &libraryName, const string &substanceName);
00038     ~TestSolver();
00039     virtual void setFluidConstants();
00040
00041     virtual void setSat_p(double &p, ExternalSaturationProperties *const properties);
00042     virtual void setSat_T(double &T, ExternalSaturationProperties *const properties);
00043
00044     virtual void setState_ph(double &p, double &h, int &phase, ExternalThermodynamicState *const
      properties);
00045     virtual void setState_pT(double &p, double &T, ExternalThermodynamicState *const properties);
00046     virtual void setState_dT(double &d, double &T, int &phase, ExternalThermodynamicState *const
      properties);
00047     virtual void setState_ps(double &p, double &s, int &phase, ExternalThermodynamicState *const
      properties);
00048 };
00049
00050 #endif /* TESTSOLVER_H_ */
```

# Index