ExternalMedia

3.3.2

Generated by Doxygen 1.9.5

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 4.0.x releases are compatible with Modelica Standard Library (MSL) 4.0.0 and later, 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. 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 version 3.3.x of External Media if your models still uses Modelica Standard Library 3.2.3, otherwise install version 4.0.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

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 v3.3.2 - 2022/06/XX

- Improved OpenModelica compatibility
- Restructured the Modelica sources
- ...

2.2 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

2.3 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

Please add some content here ...

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.

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

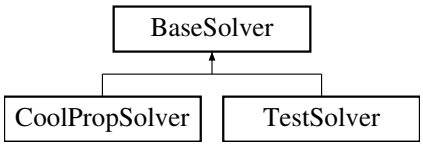
Class Documentation

11.1 BaseSolver Class Reference

Base solver class.

`#include <basesolver.h>`

Inheritance diagram for BaseSolver:



Public Member Functions

- BaseSolver (const string &mediumName, const string &libraryName, const string &substanceName)
  Constructor.
- virtual ~BaseSolver ()
  Destructor.
- double **molarMass** () const
  Return molar mass (Default implementation provided)
- double **criticalTemperature** () const
  Return temperature at critical point (Default implementation provided)
- double **criticalPressure** () const
  Return pressure at critical point (Default implementation provided)
- double **criticalMolarVolume** () const
  Return molar volume at critical point (Default implementation provided)
- double **criticalDensity** () const
  Return density at critical point (Default implementation provided)
- double **criticalEnthalpy** () const
  Return specific enthalpy at critical point (Default implementation provided)
- double **criticalEntropy** () const
  Return specific entropy at critical point (Default implementation provided)

- virtual void setFluidConstants ()

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

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

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

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

  Set state from p, s, and phase.
- virtual 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**

  Medium name.
- string **libraryName**

  Library name.
- string **substanceName**

  Substance name.

Protected Attributes

- FluidConstants **_fluidConstants**

  Fluid constants.

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.

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

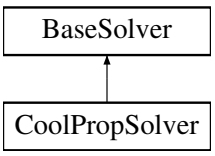
- Sources/basesolver.h
- Sources/basesolver.cpp

11.2 CoolPropSolver Class Reference

CoolProp solver class.

```
#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.

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**

Additional Inherited Members

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

ExternalSaturationProperties property struct.

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

Public Attributes

- double **Tsat**

  Saturation temperature.
- double **dTp**

  Derivative of Ts wrt pressure.
- double **ddldp**

  Derivative of dls wrt pressure.
- double **ddvdp**

  Derivative of dvs wrt pressure.
- double **dhldp**

  Derivative of hls wrt pressure.
- double **dhvdp**

  Derivative of hvs wrt pressure.
- double **dl**

  Density at bubble line (for pressure ps)
- double **dv**

  Density at dew line (for pressure ps)
- double **hl**

  Specific enthalpy at bubble line (for pressure ps)
- double **hv**

  Specific enthalpy at dew line (for pressure ps)
- double **psat**

  Saturation pressure.
- double **sigma**

  Surface tension.
- double **sl**

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

  Specific entropy at dew line (for pressure ps)

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.

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

- Sources/externalmedialib.h

11.4 ExternalThermodynamicState Struct Reference

ExternalThermodynamicState property struct.

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

Public Attributes

- double **T**

  Temperature.
- double **a**

  Velocity of sound.
- double **beta**

  Isobaric expansion coefficient.
- double **cp**

  Specific heat capacity cp.
- double **cv**

  Specific heat capacity cv.
- double **d**

  Density.
- double **ddhp**

  Derivative of density wrt enthalpy at constant pressure.
- double **ddph**

  Derivative of density wrt pressure at constant enthalpy.
- double **eta**

  Dynamic viscosity.
- double **h**

  Specific enthalpy.
- double **kappa**

  Compressibility.
- double **lambda**

  Thermal conductivity.
- double **p**

  Pressure.
- int **phase**

  Phase flag: 2 for two-phase, 1 for one-phase.
- double **s**

  Specific entropy.

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.

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

- Sources/externalmedialib.h

11.5 FluidConstants Struct Reference

Fluid constants struct.

`#include <fluidconstants.h>`

Public Member Functions

- FluidConstants ()
  Constructor.

Public Attributes

- double **MM**
  Molar mass.
- double **pc**
  Pressure at critical point.
- double **Tc**
  Temperature at critical point.
- double **dc**
  Density at critical point.
- double **hc**
  Specific enthalpy at critical point.
- double **sc**
  Specific entropy at critical point.

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.

11.5.2 Constructor & Destructor Documentation

11.5.2.1 FluidConstants()

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

Constructor.

The constructor only initializes the variables.

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

- Sources/fluidconstants.h

11.6 SolverMap Class Reference

Solver map.

`#include <solvermap.h>`

Static Public Member Functions

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

  Get a specific solver.
- static string solverKey (const string &libraryName, const string &substanceName)

  Generate a unique solver key.

Static Protected Attributes

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

  Map for all solver instances identified by the SolverKey.

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.

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

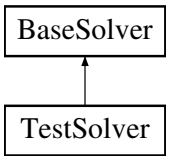
- Sources/solvermap.h
- Sources/solvermap.cpp

11.7 TestSolver Class Reference

Test solver class.

`#include <testsolver.h>`

Inheritance diagram for TestSolver:



Public Member Functions

- **TestSolver** (const string &mediumName, const string &libraryName, const string &substanceName)
- virtual void setFluidConstants ()
  Set fluid constants.
- virtual void setSat_p (double &p, ExternalSaturationProperties *const properties)
  Set saturation properties from p.
- virtual void setSat_T (double &T, ExternalSaturationProperties *const properties)
  Set saturation properties from T.
- virtual void setState_ph (double &p, double &h, int &phase, ExternalThermodynamicState *const properties)
  Set state from p, h, and phase.
- virtual void setState_pT (double &p, double &T, ExternalThermodynamicState *const properties)
  Set state from p and T.
- virtual void setState_dT (double &d, double &T, int &phase, ExternalThermodynamicState *const properties)
  Set state from d, T, and phase.
- virtual void setState_ps (double &p, double &s, int &phase, ExternalThermodynamicState *const properties)
  Set state from p, s, and phase.

Additional Inherited Members

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";

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

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

```
1 #ifndef BASESOLVER_H_
2 #define BASESOLVER_H_
3
4 #include "include.h"
5 #include "fluidconstants.h"
6 #include "externalmedialib.h"
7 #include <stdio.h>
8 #include <stdlib.h>
9 #include <string.h>
10
11 struct FluidConstants;
12
14
25 class BaseSolver{
26 public:
27     BaseSolver(const string &mediumName, const string &libraryName, const string &substanceName);
28     virtual ~BaseSolver();
29
30     double molarMass() const;
31     double criticalTemperature() const;
32     double criticalPressure() const;
33     double criticalMolarVolume() const;
34     double criticalDensity() const;
35     double criticalEnthalpy() const;
36     double criticalEntropy() const;
37
38     virtual void setFluidConstants();
39
40     virtual void setState_ph(double &p, double &h, int &phase, ExternalThermodynamicState *const
       properties);
41     virtual void setState_pT(double &p, double &T, ExternalThermodynamicState *const properties);
42     virtual void setState_dT(double &d, double &T, int &phase, ExternalThermodynamicState *const
       properties);
43     virtual void setState_ps(double &p, double &s, int &phase, ExternalThermodynamicState *const
       properties);
44     virtual void setState_hs(double &h, double &s, int &phase, ExternalThermodynamicState *const
       properties);
45
46     virtual double partialDeriv_state(const string &of, const string &wrt, const string &cst,
       ExternalThermodynamicState *const properties);
47
48     virtual double Pr(ExternalThermodynamicState *const properties);
49     virtual double T(ExternalThermodynamicState *const properties);
50     virtual double a(ExternalThermodynamicState *const properties);
51     virtual double beta(ExternalThermodynamicState *const properties);
52     virtual double cp(ExternalThermodynamicState *const properties);
53     virtual double cv(ExternalThermodynamicState *const properties);
54     virtual double d(ExternalThermodynamicState *const properties);
55     virtual double ddhp(ExternalThermodynamicState *const properties);
56     virtual double ddph(ExternalThermodynamicState *const properties);
57     virtual double eta(ExternalThermodynamicState *const properties);
58     virtual double h(ExternalThermodynamicState *const properties);
59     virtual double kappa(ExternalThermodynamicState *const properties);
60     virtual double lambda(ExternalThermodynamicState *const properties);
61     virtual double p(ExternalThermodynamicState *const properties);
62     virtual int phase(ExternalThermodynamicState *const properties);
63     virtual double s(ExternalThermodynamicState *const properties);
64     virtual double d_der(ExternalThermodynamicState *const properties);
```

```
65      virtual double isentropicEnthalpy(double &p, ExternalThermodynamicState *const properties);
66
67      virtual void setSat_p(double &p, ExternalSaturationProperties *const properties);
68      virtual void setSat_T(double &T, ExternalSaturationProperties *const properties);
69
70      virtual void setBubbleState(ExternalSaturationProperties *const properties, int phase,
71                                  ExternalThermodynamicState *const bubbleProperties);
72      virtual void setDewState(ExternalSaturationProperties *const properties, int phase,
73                               ExternalThermodynamicState *const bubbleProperties);
74
75      virtual double dTp(ExternalSaturationProperties *const properties);
76      virtual double ddldp(ExternalSaturationProperties *const properties);
77      virtual double ddvdp(ExternalSaturationProperties *const properties);
78      virtual double dhldp(ExternalSaturationProperties *const properties);
79      virtual double dhvdp(ExternalSaturationProperties *const properties);
80      virtual double dl(ExternalSaturationProperties *const properties);
81      virtual double dv(ExternalSaturationProperties *const properties);
82      virtual double hl(ExternalSaturationProperties *const properties);
83      virtual double hv(ExternalSaturationProperties *const properties);
84      virtual double sigma(ExternalSaturationProperties *const properties);
85      virtual double sl(ExternalSaturationProperties *const properties);
86      virtual double sv(ExternalSaturationProperties *const properties);
87
88      virtual bool computeDerivatives(ExternalThermodynamicState *const properties);
89
90      virtual double psat(ExternalSaturationProperties *const properties);
91      virtual double Tsat(ExternalSaturationProperties *const properties);
92
94      string mediumName;
96      string libraryName;
98      string substanceName;
99
100 protected:
102      FluidConstants _fluidConstants;
103 };
104
105 #endif // BASESOLVER_H_
```

12.2 coolpropsolver.h

```
1  #ifndef COOLPROPSOLVER_H_
2  #define COOLPROPSOLVER_H_
3
4  #include "include.h"
5  #if (EXTERNALMEDIA_COOLPROP == 1)
6
7  #include "basesolver.h"
8  #include "AbstractState.h"
9  #include "crossplatform_shared_ptr.h"
10
12
30 class CoolPropSolver : public BaseSolver{
31
32 protected:
33      //class CoolProp::AbstractState *state;
34      shared_ptr<CoolProp::AbstractState> state;
35      bool enable_TTSE, enable_BICUBIC, calc_transport, extend_twophase, isCompressible;
36      int debug_level;
37      double twophase_derivsmoothing_xend;
38      double rho_smoothing_xend;
39      double _p_eps   ; // relative tolerance margin for subcritical pressure conditions
40      double _delta_h ; // delta_h for one-phase/two-phase discrimination
41      ExternalSaturationProperties _satPropsClose2Crit; // saturation properties close to  critical
        conditions
42
43      virtual void postStateChange(ExternalThermodynamicState *const properties);
44      long makeDerivString(const string &of, const string &wrt, const string &cst);
45      double interp_linear(double Q, double valueL, double valueV);
46      double interp_recip(double Q, double valueL, double valueV);
47
48 public:
49      CoolPropSolver(const std::string &mediumName, const std::string &libraryName, const std::string
        &substanceName);
50      ~CoolPropSolver();
51      virtual void setFluidConstants();
52
53      virtual void setSat_p(double &p, ExternalSaturationProperties *const properties);
54      virtual void setSat_T(double &T, ExternalSaturationProperties *const properties);
55
56      virtual void setBubbleState(ExternalSaturationProperties *const properties, int phase,
        ExternalThermodynamicState *const bubbleProperties);
57      virtual void setDewState   (ExternalSaturationProperties *const properties, int phase,
        ExternalThermodynamicState *const bubbleProperties);
```

```
58
59      virtual void setState_ph(double &p, double &h, int &phase, ExternalThermodynamicState *const
        properties);
60      virtual void setState_pT(double &p, double &T, ExternalThermodynamicState *const properties);
61      virtual void setState_dT(double &d, double &T, int &phase, ExternalThermodynamicState *const
        properties);
62      virtual void setState_ps(double &p, double &s, int &phase, ExternalThermodynamicState *const
        properties);
63      virtual void setState_hs(double &h, double &s, int &phase, ExternalThermodynamicState *const
        properties);
64
65      virtual double partialDeriv_state(const string &of, const string &wrt, const string &cst,
        ExternalThermodynamicState *const properties);
66
67      virtual double Pr(ExternalThermodynamicState *const properties);
68      virtual double T(ExternalThermodynamicState *const properties);
69      virtual double a(ExternalThermodynamicState *const properties);
70      virtual double beta(ExternalThermodynamicState *const properties);
71      virtual double cp(ExternalThermodynamicState *const properties);
72      virtual double cv(ExternalThermodynamicState *const properties);
73      virtual double d(ExternalThermodynamicState *const properties);
74      virtual double ddhp(ExternalThermodynamicState *const properties);
75      virtual double ddph(ExternalThermodynamicState *const properties);
76      virtual double eta(ExternalThermodynamicState *const properties);
77      virtual double h(ExternalThermodynamicState *const properties);
78      virtual double kappa(ExternalThermodynamicState *const properties);
79      virtual double lambda(ExternalThermodynamicState *const properties);
80      virtual double p(ExternalThermodynamicState *const properties);
81      virtual int phase(ExternalThermodynamicState *const properties);
82      virtual double s(ExternalThermodynamicState *const properties);
83      virtual double d_der(ExternalThermodynamicState *const properties);
84      virtual double isentropicEnthalpy(double &p, ExternalThermodynamicState *const properties);
85
86      virtual double dTp(ExternalSaturationProperties *const properties);
87      virtual double ddldp(ExternalSaturationProperties *const properties);
88      virtual double ddvdp(ExternalSaturationProperties *const properties);
89      virtual double dhldp(ExternalSaturationProperties *const properties);
90      virtual double dhvdp(ExternalSaturationProperties *const properties);
91      virtual double dl(ExternalSaturationProperties *const properties);
92      virtual double dv(ExternalSaturationProperties *const properties);
93      virtual double hl(ExternalSaturationProperties *const properties);
94      virtual double hv(ExternalSaturationProperties *const properties);
95      virtual double sigma(ExternalSaturationProperties *const properties);
96      virtual double sl(ExternalSaturationProperties *const properties);
97      virtual double sv(ExternalSaturationProperties *const properties);
98
99      virtual double psat(ExternalSaturationProperties *const properties);
100      virtual double Tsat(ExternalSaturationProperties *const properties);
101
102 };
103
104 #endif
105
106 #endif // COOLPROPSOLVER_H_
```

12.3 Sources/errorhandling.h File Reference

Error handling for external library.

Functions

- void errorMessage (char ∗errorMsg)

  Function to display error message.
- void warningMessage (char ∗warningMsg)

  Function to display warning message.

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.

```
1
13 #ifndef ERRORHANDLING_H_
14 #define ERRORHANDLING_H_
15
16 #ifdef WIN32
17 extern void (*ModelicaErrorPtr)(const char *);
18 extern void (*ModelicaWarningPtr)(const char *);
19 #endif
20
22
27 void errorMessage(char *errorMsg);
29
33 void warningMessage(char *warningMsg);
34
35 #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

  ExternalThermodynamicState property struct.
- struct ExternalSaturationProperties

  ExternalSaturationProperties property struct.

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

  ExternalThermodynamicState property struct.
- typedef struct ExternalSaturationProperties ExternalSaturationProperties

  ExternalSaturationProperties property struct.

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

Parameters

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.

```
1
20 #ifndef EXTERNALMEDIALIB_H_
21 #define EXTERNALMEDIALIB_H_
22
23 // Constants for input choices (see ExternalMedia.Common.InputChoices)
24 #define CHOICE_dT 1
25 #define CHOICE_hs 2
26 #define CHOICE_ph 3
27 #define CHOICE_ps 4
28 #define CHOICE_pT 5
29
33 #if !defined(EXTERNALMEDIA_EXPORT)
34 #  if !defined(EXTERNALMEDIA_LIBRARY_EXPORTS)
35 #    define EXTERNALMEDIA_EXPORT
36 #  else
37 #    if (EXTERNALMEDIA_LIBRARY_EXPORTS == 1)
38 #      if defined(_WIN32) || defined(__WIN32__) || defined(_WIN64) || defined(__WIN64__)
39 #        if !defined(__EXTERNALMEDIA_ISWINDOWS__)
40 #          define __EXTERNALMEDIA_ISWINDOWS__
41 #        endif
42 #      elif __APPLE__
43 #        if !defined(__EXTERNALMEDIA_ISAPPLE__)
44 #          define __EXTERNALMEDIA_ISAPPLE__
45 #        endif
46 #      elif __linux
47 #        if !defined(__EXTERNALMEDIA_ISLINUX__)
48 #          define __EXTERNALMEDIA_ISLINUX__
49 #        endif
50 #      endif
51 #
52 #      if defined(__EXTERNALMEDIA_ISLINUX__)
53 #        define EXTERNALMEDIA_EXPORT
54 #      elif defined(__EXTERNALMEDIA_ISAPPLE__)
55 #        define EXTERNALMEDIA_EXPORT
56 #      else
57 #        define EXTERNALMEDIA_EXPORT __declspec(dllexport)
58 #      endif
59 #    else
60 #      define EXTERNALMEDIA_EXPORT
61 #    endif
62 #  endif
63 #endif
64
65 // Define struct
67
76 typedef struct ExternalThermodynamicState {
77
79     double T;
81     double a;
83     double beta;
85     double cp;
87     double cv;
89     double d;
91     double ddhp;
93     double ddph;
95     double eta;
97     double h;
99     double kappa;
101     double lambda;
```

```
103     double p;
105     int phase;
107     double s;
108
110
113     #ifdef __cplusplus
114     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) {};
115     #endif
116
117 } ExternalThermodynamicState;
118
120
129 typedef struct ExternalSaturationProperties {
131     double Tsat;
133     double dTp;
135     double ddldp;
137     double ddvdp;
139     double dhldp;
141     double dhvdp;
143     double dl;
145     double dv;
147     double hl;
149     double hv;
151     double psat;
153     double sigma;
155     double sl;
157     double sv;
158
160
163     #ifdef __cplusplus
164     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) {};
165     #endif
166
167 } ExternalSaturationProperties;
168
169
170 #ifdef __cplusplus
171 extern "C" {
172 #endif // __cplusplus
173
174     EXTERNALMEDIA_EXPORT double TwoPhaseMedium_getMolarMass_C_impl(const char *mediumName, const char
      *libraryName, const char *substanceName);
175     EXTERNALMEDIA_EXPORT double TwoPhaseMedium_getCriticalTemperature_C_impl(const char *mediumName,
      const char *libraryName, const char *substanceName);
176     EXTERNALMEDIA_EXPORT double TwoPhaseMedium_getCriticalPressure_C_impl(const char *mediumName, const
      char *libraryName, const char *substanceName);
177     EXTERNALMEDIA_EXPORT double TwoPhaseMedium_getCriticalMolarVolume_C_impl(const char *mediumName,
      const char *libraryName, const char *substanceName);
178
179     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);
180     EXTERNALMEDIA_EXPORT void TwoPhaseMedium_setState_pT_C_impl(double p, double T,            void
      *state, const char *mediumName, const char *libraryName, const char *substanceName);
181     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);
182     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);
183     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);
184
185     // 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
186     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 *));
187     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 *));
188     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 *));
189     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 *));
190     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 *));
191
192     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);
193
194     EXTERNALMEDIA_EXPORT double TwoPhaseMedium_prandtlNumber_C_impl(void *state, const char *mediumName,
      const char *libraryName, const char *substanceName);
195     EXTERNALMEDIA_EXPORT double TwoPhaseMedium_temperature_C_impl(void *state, const char *mediumName,
```

```
    const char *libraryName, const char *substanceName);
196  EXTERNALMEDIA_EXPORT double TwoPhaseMedium_velocityOfSound_C_impl(void *state, const char
    *mediumName, const char *libraryName, const char *substanceName);
197  EXTERNALMEDIA_EXPORT double TwoPhaseMedium_isobaricExpansionCoefficient_C_impl(void *state, const
    char *mediumName, const char *libraryName, const char *substanceName);
198  EXTERNALMEDIA_EXPORT double TwoPhaseMedium_specificHeatCapacityCp_C_impl(void *state,   const char
    *mediumName, const char *libraryName, const char *substanceName);
199  EXTERNALMEDIA_EXPORT double TwoPhaseMedium_specificHeatCapacityCv_C_impl(void *state,   const char
    *mediumName, const char *libraryName, const char *substanceName);
200  EXTERNALMEDIA_EXPORT double TwoPhaseMedium_density_C_impl(void *state, const char *mediumName, const
    char *libraryName, const char *substanceName);
201  EXTERNALMEDIA_EXPORT double TwoPhaseMedium_density_derh_p_C_impl(void *state,   const char
    *mediumName, const char *libraryName, const char *substanceName);
202  EXTERNALMEDIA_EXPORT double TwoPhaseMedium_density_derp_h_C_impl(void *state,   const char
    *mediumName, const char *libraryName, const char *substanceName);
203  EXTERNALMEDIA_EXPORT double TwoPhaseMedium_dynamicViscosity_C_impl(void *state, const char
    *mediumName, const char *libraryName, const char *substanceName);
204  EXTERNALMEDIA_EXPORT double TwoPhaseMedium_specificEnthalpy_C_impl(void *state, const char
    *mediumName, const char *libraryName, const char *substanceName);
205  EXTERNALMEDIA_EXPORT double TwoPhaseMedium_isothermalCompressibility_C_impl(void *state, const char
    *mediumName, const char *libraryName, const char *substanceName);
206  EXTERNALMEDIA_EXPORT double TwoPhaseMedium_thermalConductivity_C_impl(void *state, const char
    *mediumName, const char *libraryName, const char *substanceName);
207  EXTERNALMEDIA_EXPORT double TwoPhaseMedium_pressure_C_impl(void *state, const char *mediumName,
    const char *libraryName, const char *substanceName);
208  EXTERNALMEDIA_EXPORT double TwoPhaseMedium_specificEntropy_C_impl(void *state, const char
    *mediumName, const char *libraryName, const char *substanceName);
209  EXTERNALMEDIA_EXPORT double TwoPhaseMedium_density_ph_der_C_impl(void *state,   const char
    *mediumName, const char *libraryName, const char *substanceName);
210  EXTERNALMEDIA_EXPORT double TwoPhaseMedium_isentropicEnthalpy_C_impl(double p_downstream, void
    *refState,   const char *mediumName, const char *libraryName, const char *substanceName);
211
212  EXTERNALMEDIA_EXPORT void TwoPhaseMedium_setSat_p_C_impl(double p, void *sat, const char
    *mediumName, const char *libraryName, const char *substanceName);
213  EXTERNALMEDIA_EXPORT void TwoPhaseMedium_setSat_T_C_impl(double T, void *sat, const char
    *mediumName, const char *libraryName, const char *substanceName);
214  EXTERNALMEDIA_EXPORT void TwoPhaseMedium_setBubbleState_C_impl(void *sat, int phase, void *state,
    const char *mediumName, const char *libraryName, const char *substanceName);
215  EXTERNALMEDIA_EXPORT void TwoPhaseMedium_setDewState_C_impl(void *sat, int phase, void *state, const
    char *mediumName, const char *libraryName, const char *substanceName);
216
217  EXTERNALMEDIA_EXPORT double TwoPhaseMedium_saturationTemperature_C_impl(double p, const char
    *mediumName, const char *libraryName, const char *substanceName);
218  EXTERNALMEDIA_EXPORT double TwoPhaseMedium_saturationTemperature_derp_C_impl(double p, const char
    *mediumName, const char *libraryName, const char *substanceName);
219  EXTERNALMEDIA_EXPORT double TwoPhaseMedium_saturationTemperature_derp_sat_C_impl(void *sat, const
    char *mediumName, const char *libraryName, const char *substanceName);
220
221  EXTERNALMEDIA_EXPORT double TwoPhaseMedium_dBubbleDensity_dPressure_C_impl(void *sat, const char
    *mediumName, const char *libraryName, const char *substanceName);
222  EXTERNALMEDIA_EXPORT double TwoPhaseMedium_dDewDensity_dPressure_C_impl(void *sat, const char
    *mediumName, const char *libraryName, const char *substanceName);
223  EXTERNALMEDIA_EXPORT double TwoPhaseMedium_dBubbleEnthalpy_dPressure_C_impl(void *sat, const char
    *mediumName, const char *libraryName, const char *substanceName);
224  EXTERNALMEDIA_EXPORT double TwoPhaseMedium_dDewEnthalpy_dPressure_C_impl(void *sat, const char
    *mediumName, const char *libraryName, const char *substanceName);
225  EXTERNALMEDIA_EXPORT double TwoPhaseMedium_bubbleDensity_C_impl(void *sat, const char *mediumName,
    const char *libraryName, const char *substanceName);
226  EXTERNALMEDIA_EXPORT double TwoPhaseMedium_dewDensity_C_impl(void *sat, const char *mediumName,
    const char *libraryName, const char *substanceName);
227  EXTERNALMEDIA_EXPORT double TwoPhaseMedium_bubbleEnthalpy_C_impl(void *sat, const char *mediumName,
    const char *libraryName, const char *substanceName);
228  EXTERNALMEDIA_EXPORT double TwoPhaseMedium_dewEnthalpy_C_impl(void *sat, const char *mediumName,
    const char *libraryName, const char *substanceName);
229  EXTERNALMEDIA_EXPORT double TwoPhaseMedium_saturationPressure_C_impl(double T, const char
    *mediumName, const char *libraryName, const char *substanceName);
230  EXTERNALMEDIA_EXPORT double TwoPhaseMedium_surfaceTension_C_impl(void *sat, const char *mediumName,
    const char *libraryName, const char *substanceName);
231  EXTERNALMEDIA_EXPORT double TwoPhaseMedium_bubbleEntropy_C_impl(void *sat, const char *mediumName,
    const char *libraryName, const char *substanceName);
232  EXTERNALMEDIA_EXPORT double TwoPhaseMedium_dewEntropy_C_impl(void *sat, const char *mediumName,
    const char *libraryName, const char *substanceName);
233
234 #ifdef __cplusplus
235 }
236 #endif // __cplusplus
237
238 #endif /*EXTERNALMEDIALIB_H_*/
```

12.7 fluidconstants.h

```
1 #ifndef FLUIDCONSTANTS_H_
2 #define FLUIDCONSTANTS_H_
```

```
3
4 #include "include.h"
5
7
17 struct FluidConstants{
19     double MM;
21     double pc;
23     double Tc;
25     double dc;
26     // The following two functions are currently only available internally
27     // but do not have the required interface functions to be accessible from
28     // Modelica.
30     double hc;
32     double sc;
33
35
38     FluidConstants() : MM(-1), pc(-1), Tc(-1), dc(-1), hc(-1), sc(-1) {};
39 };
40
41 #endif // FLUIDCONSTANTS_H_
```

12.8 FluidProp_COM.h

```
1 //=============================================================================//
2 //                                                                             //
3 //                        FluidProp C++ COM interface                          //
4 //                        ---------------------------                          //
5 //                                                                             //
6 //  The interface defined in this file, IFluidProp_COM is the direct C++       //
7 //  interface to the FluidProp COM server.  It is not recommended to use       //
8 //  this interface directly, please use the TFluidProp wrapper class.          //
9 //  This file should not be altered.                                           //
10 //                                                                            //
11 //  July, 2004, for FluidProp 1                                               //
12 //  January, 2006, for FluidProp 2                                            //
13 //  April, 2007, for FluidProp 2.3                                            //
14 //  November, 2012, for FluidProp 2.5                                         //
15 //                                                                            //
16 //=============================================================================//
17
18 #ifndef FluidProp_COM_h
19 #define FluidProp_COM_h
20
21 #include "include.h"
22 #include <comutil.h>
23
24
25 // The IFluidProp interface
26 interface IFluidProp_COM : public IDispatch
27 {
28   public:
29      virtual void __stdcall CreateObject  ( BSTR ModelName, BSTR* ErrorMsg) = 0;
30      virtual void __stdcall ReleaseObjects( ) = 0;
31
32      virtual void __stdcall SetFluid      ( BSTR ModelName, long nComp, SAFEARRAY** sa_Comp,
33                                             SAFEARRAY** sa_Conc, BSTR* ErrorMsg) = 0;
34      virtual void __stdcall SetFluid_M    ( BSTR ModelName, long nComp, SAFEARRAY* sa_Comp,
35                                             SAFEARRAY* sa_Conc, BSTR* ErrorMsg) = 0;
36
37      virtual void __stdcall GetFluid      ( BSTR ModelName, long* nComp, SAFEARRAY** sa_Comp,
38                                             SAFEARRAY** sa_Conc) = 0;
39      virtual void __stdcall GetFluid_M    ( BSTR ModelName, long* nComp, SAFEARRAY** sa_Comp,
40                                             SAFEARRAY** sa_Conc) = 0;
41
42      virtual void __stdcall GetFluidNames ( BSTR LongShort, BSTR ModelName, long* nComp,
43                                             SAFEARRAY** sa_CompSet, BSTR* ErrorMsg) = 0;
44      virtual void __stdcall GetFluidNames_M( BSTR LongShort, BSTR ModelName, long* nComp,
45                                             SAFEARRAY** sa_CompSet, BSTR* ErrorMsg) = 0;
46
47      virtual void __stdcall GetCompSet    ( BSTR ModelName, long* nComp, SAFEARRAY** sa_CompSet,
48                                             BSTR* ErrorMsg) = 0;
49      virtual void __stdcall GetCompSet_M  ( BSTR ModelName, long* nComp, SAFEARRAY** sa_CompSet,
50                                             BSTR* ErrorMsg) = 0;
51
52      virtual void __stdcall Pressure      ( BSTR InputSpec, double Input1, double Input2,
53                                             double* Output, BSTR* ErrorMsg) = 0;
54      virtual void __stdcall Temperature   ( BSTR InputSpec, double Input1, double Input2,
55                                             double* Output, BSTR* ErrorMsg) = 0;
56      virtual void __stdcall SpecVolume    ( BSTR InputSpec, double Input1, double Input2,
57                                             double* Output, BSTR* ErrorMsg) = 0;
58      virtual void __stdcall Density       ( BSTR InputSpec, double Input1, double Input2,
59                                             double* Output, BSTR* ErrorMsg) = 0;
60      virtual void __stdcall Enthalpy      ( BSTR InputSpec, double Input1, double Input2,
```

```
61                                          double* Output, BSTR* ErrorMsg) = 0;
62      virtual void __stdcall Entropy       ( BSTR InputSpec, double Input1, double Input2,
63                                            double* Output, BSTR* ErrorMsg) = 0;
64      virtual void __stdcall IntEnergy     ( BSTR InputSpec, double Input1, double Input2,
65                                            double* Output, BSTR* ErrorMsg) = 0;
66      virtual void __stdcall VaporQual     ( BSTR InputSpec, double Input1, double Input2,
67                                            double* Output, BSTR* ErrorMsg) = 0;
68
69      virtual void __stdcall LiquidCmp     ( BSTR InputSpec, double Input1, double Input2,
70                                            SAFEARRAY** Output, BSTR* ErrorMsg) = 0;
71      virtual void __stdcall LiquidCmp_M   ( BSTR InputSpec, double Input1, double Input2,
72                                            SAFEARRAY** Output, BSTR* ErrorMsg) = 0;
73
74      virtual void __stdcall VaporCmp      ( BSTR InputSpec, double Input1, double Input2,
75                                            SAFEARRAY** Output, BSTR* ErrorMsg) = 0;
76      virtual void __stdcall VaporCmp_M    ( BSTR InputSpec, double Input1, double Input2,
77                                            SAFEARRAY** Output, BSTR* ErrorMsg) = 0;
78
79      virtual void __stdcall HeatCapV      ( BSTR InputSpec, double Input1, double Input2,
80                                            double* Output, BSTR* ErrorMsg) = 0;
81      virtual void __stdcall HeatCapP      ( BSTR InputSpec, double Input1, double Input2,
82                                            double* Output, BSTR* ErrorMsg) = 0;
83      virtual void __stdcall SoundSpeed    ( BSTR InputSpec, double Input1, double Input2,
84                                            double* Output, BSTR* ErrorMsg) = 0;
85      virtual void __stdcall Alpha         ( BSTR InputSpec, double Input1, double Input2,
86                                            double* Output, BSTR* ErrorMsg) = 0;
87      virtual void __stdcall Beta          ( BSTR InputSpec, double Input1, double Input2,
88                                            double* Output, BSTR* ErrorMsg) = 0;
89      virtual void __stdcall Chi           ( BSTR InputSpec, double Input1, double Input2,
90                                            double* Output, BSTR* ErrorMsg) = 0;
91      virtual void __stdcall Fi            ( BSTR InputSpec, double Input1, double Input2,
92                                            double* Output, BSTR* ErrorMsg) = 0;
93      virtual void __stdcall Ksi           ( BSTR InputSpec, double Input1, double Input2,
94                                            double* Output, BSTR* ErrorMsg) = 0;
95      virtual void __stdcall Psi           ( BSTR InputSpec, double Input1, double Input2,
96                                            double* Output, BSTR* ErrorMsg) = 0;
97      virtual void __stdcall Zeta          ( BSTR InputSpec, double Input1, double Input2,
98                                            double* Output, BSTR* ErrorMsg) = 0;
99      virtual void __stdcall Theta         ( BSTR InputSpec, double Input1, double Input2,
100                                           double* Output, BSTR* ErrorMsg) = 0;
101      virtual void __stdcall Kappa        ( BSTR InputSpec, double Input1, double Input2,
102                                           double* Output, BSTR* ErrorMsg) = 0;
103      virtual void __stdcall Gamma        ( BSTR InputSpec, double Input1, double Input2,
104                                           double* Output, BSTR* ErrorMsg) = 0;
105
106      virtual void __stdcall Viscosity    ( BSTR InputSpec, double Input1, double Input2,
107                                           double* Output, BSTR* ErrorMsg) = 0;
108      virtual void __stdcall ThermCond    ( BSTR InputSpec, double Input1, double Input2,
109                                           double* Output, BSTR* ErrorMsg) = 0;
110
111      virtual void __stdcall AllProps     ( BSTR InputSpec, double Input1, double Input2,
112                                           double* P, double* T, double* v, double* d,
113                                           double* h, double* s, double* u, double* q,
114                                           SAFEARRAY** x, SAFEARRAY** y, double* cv, double* cp,
115                                           double* c, double* alpha, double* beta, double* chi,
116                                           double* fi, double* ksi, double* psi, double* zeta,
117                                           double* theta, double* kappa, double* gamma,
118                                           double* eta, double* lambda, BSTR* ErrorMsg) = 0;
119      virtual void __stdcall AllProps_M   ( BSTR InputSpec, double Input1, double Input2,
120                                           double* P, double* T, double* v, double* d,
121                                           double* h, double* s, double* u, double* q,
122                                           SAFEARRAY** x, SAFEARRAY** y, double* cv, double* cp,
123                                           double* c, double* alpha, double* beta, double* chi,
124                                           double* fi, double* ksi, double* psi, double* zeta,
125                                           double* theta, double* kappa, double* gamma,
126                                           double* eta, double* lambda, BSTR* ErrorMsg) = 0;
127
128      virtual void __stdcall AllPropsSat  ( BSTR InputSpec, double Input1, double Input2,
129                                           double* P, double* T, double* v, double* d,
130                                           double* h, double* s, double* u, double* q,
131                                           SAFEARRAY** x, SAFEARRAY** y, double* cv, double* cp,
132                                           double* c, double* alpha, double* beta, double* chi,
133                                           double* fi, double* ksi, double* psi, double* zeta,
134                                           double* theta, double* kappa, double* gamma,
135                                           double* eta, double* lambda, double* d_liq,
136                                           double* d_vap, double* h_liq, double* h_vap,
137                                           double* T_sat, double* dd_liq_dP, double* dd_vap_dP,
138                                           double* dh_liq_dP, double* dh_vap_dP,
139                                           double* dT_sat_dP, BSTR* ErrorMsg) = 0;
140      virtual void __stdcall AllPropsSat_M ( BSTR InputSpec, double Input1, double Input2,
141                                           double* P, double* T, double* v, double* d,
142                                           double* h, double* s, double* u, double* q,
143                                           SAFEARRAY** x, SAFEARRAY** y, double* cv, double* cp,
144                                           double* c, double* alpha, double* beta, double* chi,
145                                           double* fi, double* ksi, double* psi, double* zeta,
146                                           double* theta, double* kappa, double* gamma,
147                                           double* eta, double* lambda, double* d_liq,
```

```
148                                          double* d_vap, double* h_liq, double* h_vap,
149                                          double* T_sat, double* dd_liq_dP, double* dd_vap_dP,
150                                          double* dh_liq_dP, double* dh_vap_dP,
151                                          double* dT_sat_dP, BSTR* ErrorMsg) = 0;
152
153      virtual void __stdcall Solve         ( BSTR FuncSpec, double FuncVal, BSTR InputSpec,
154                                             long Target, double FixedVal, double MinVal,
155                                             double MaxVal, double* Output, BSTR* ErrorMsg) = 0;
156
157      virtual void __stdcall Mmol          ( double* Output, BSTR* ErrorMsg) = 0;
158      virtual void __stdcall Tcrit         ( double* Output, BSTR* ErrorMsg) = 0;
159      virtual void __stdcall Pcrit         ( double* Output, BSTR* ErrorMsg) = 0;
160      virtual void __stdcall Tmin          ( double* Output, BSTR* ErrorMsg) = 0;
161      virtual void __stdcall Tmax          ( double* Output, BSTR* ErrorMsg) = 0;
162      virtual void __stdcall AllInfo       ( double* M_mol, double* T_crit, double* P_crit,
163                                             double* T_min, double* T_max , BSTR* ErrorMsg) = 0;
164
165      virtual void __stdcall SetUnits      ( BSTR UnitSet, BSTR MassOrMole, BSTR Properties,
166                                             BSTR Units, BSTR* ErrorMsg) = 0;
167      virtual void __stdcall SetRefState   ( double T_ref, double P_ref, BSTR* ErrorMsg) = 0;
168
169      virtual void __stdcall freeStanMix_Psat_k1   ( ) = 0;    // C++ interface not yet implemented
170      virtual void __stdcall zFlow_vu          ( ) = 0;    // C++ interface not yet implemented
171      virtual void __stdcall GetVersion        ( BSTR ModelName, SAFEARRAY** sa_version) = 0;
172
173      virtual void __stdcall AllTransProps     ( ) = 0;    // C++ interface not yet implemented
174      virtual void __stdcall SaturationLine    ( ) = 0;    // C++ interface not yet implemented
175      virtual void __stdcall IsoLine           ( ) = 0;    // C++ interface not yet implemented
176      virtual void __stdcall freeStanMix_xy_A_alfa ( ) = 0;    // C++ interface not yet implemented
177      virtual void __stdcall PCP_SAFT_xy_kij   ( ) = 0;    // C++ interface not yet implemented
178      virtual void __stdcall PCP_SAFT_hsxy_mp  ( ) = 0;    // C++ interface not yet implemented
179      virtual void __stdcall PCP_SAFT_hsxy_mp_M( ) = 0;    // C++ interface not yet implemented
180
181      virtual void __stdcall FugaCoef          ( BSTR InputSpec, double Input1, double Input2,
182                                                 SAFEARRAY** Output, BSTR* ErrorMsg) = 0;
183      virtual void __stdcall FugaCoef_M        ( BSTR InputSpec, double Input1, double Input2,
184                                                 SAFEARRAY** Output, BSTR* ErrorMsg) = 0;
185
186      virtual void __stdcall SurfTens          ( BSTR InputSpec, double Input1, double Input2,
187                                                 double* Output, BSTR* ErrorMsg) = 0;
188
189      virtual void __stdcall GibbsEnergy       ( BSTR InputSpec, double Input1, double Input2,
190                                                 double* Output, BSTR* ErrorMsg) = 0;
191
192      virtual void __stdcall CapeOpenDeriv     ( BSTR InputSpec, double Input1, double Input2,
193                                                 SAFEARRAY** v, SAFEARRAY** h, SAFEARRAY** s,
194                                                 SAFEARRAY** G, SAFEARRAY** lnphi, BSTR* ErrorMsg) = 0;
195
196      virtual void __stdcall SpecVolume_Deriv  ( BSTR InputSpec, double Input1, double Input2,
197                                                 SAFEARRAY** Output, BSTR* ErrorMsg) = 0;
198      virtual void __stdcall Enthalpy_Deriv    ( BSTR InputSpec, double Input1, double Input2,
199                                                 SAFEARRAY** Output, BSTR* ErrorMsg) = 0;
200      virtual void __stdcall Entropy_Deriv     ( BSTR InputSpec, double Input1, double Input2,
201                                                 SAFEARRAY** Output, BSTR* ErrorMsg) = 0;
202      virtual void __stdcall GibbsEnergy_Deriv ( BSTR InputSpec, double Input1, double Input2,
203                                                 SAFEARRAY** Output, BSTR* ErrorMsg) = 0;
204      virtual void __stdcall FugaCoef_Deriv    ( BSTR InputSpec, double Input1, double Input2,
205                                                 SAFEARRAY** Output, BSTR* ErrorMsg) = 0;
206
207      virtual void __stdcall PCP_SAFT_P_kij    ( ) = 0;    // C++ interface not yet implemented
208      virtual void __stdcall PCP_SAFT_T_kij    ( ) = 0;    // C++ interface not yet implemented
209      virtual void __stdcall PCP_SAFT_Prho_mseT( ) = 0;    // C++ interface not yet implemented
210      virtual void __stdcall CalcProp          ( ) = 0;    // C++ interface not yet implemented
211 };
212
213 #endif // FluidProp_COM_h
```

12.9 FluidProp_IF.h

```
1 //=============================================================================//
2 //                                                                             //
3 //                          FluidProp C++ interface                            //
4 //                          -----------------------                            //
5 //                                                                             //
6 //  The class implemented in this file, TFluidProp, is as a wrapper class for //
7 //  the IFluidProp_COM interface.  TFluidProp hides COM specific details       //
8 //  like safe arrays (SAFEARRAY) and binary strings (BSTR) in IFluidProp_COM.  //
9 //  In the TFluidProp class only standard C++ data types are used.  This is    //
10 //  the recommended way working with the FluidProp COM server in C++.          //
11 //                                                                            //
12 //  July, 2004, for FluidProp 1                                               //
13 //  January, 2006, for FluidProp 2                                            //
14 //  April, 2007, for FluidProp 2.3                                            //
```

```
15 //  November, 2012, for FluidProp 2.5                                      //
16 //                                                                         //
17 //=========================================================================//
18
19 #ifndef FluidProp_IF_h
20 #define FluidProp_IF_h
21
22 #include "include.h"
23
24 #pragma comment(lib, "comsuppw.lib")
25
26 #include <string>
27 using std::string;
28
29 #include "FluidProp_COM.h"
30
31
32 // The TFluidProp class
33 class TFluidProp
34 {
35    public:
36
37       TFluidProp();
38       ~TFluidProp();
39
40       bool IsValid();
41
42       void CreateObject  ( string ModelName, string* ErrorMsg);
43       void ReleaseObjects( );
44
45       void SetFluid      ( string ModelName, int nComp, string* Comp, double* Conc,
46                            string* ErrorMsg);
47       void GetFluid      ( string* ModelName, int* nComp, string* Comp, double* Conc,
48                            bool CompInfo = true);
49       void GetFluidNames ( string LongShort, string ModelName, int* nFluids, string* FluidNames,
50                            string* ErrorMsg);
51       void GetCompSet    ( string ModelName, int* nComps, string* CompSet, string* ErrorMsg);
52
53       double Pressure    ( string InputSpec, double Input1, double Input2, string* ErrorMsg);
54       double Temperature ( string InputSpec, double Input1, double Input2, string* ErrorMsg);
55       double SpecVolume  ( string InputSpec, double Input1, double Input2, string* ErrorMsg);
56       double Density     ( string InputSpec, double Input1, double Input2, string* ErrorMsg);
57       double Enthalpy    ( string InputSpec, double Input1, double Input2, string* ErrorMsg);
58       double Entropy     ( string InputSpec, double Input1, double Input2, string* ErrorMsg);
59       double IntEnergy   ( string InputSpec, double Input1, double Input2, string* ErrorMsg);
60       double VaporQual   ( string InputSpec, double Input1, double Input2, string* ErrorMsg);
61       double* LiquidCmp  ( string InputSpec, double Input1, double Input2, string* ErrorMsg);
62       double* VaporCmp   ( string InputSpec, double Input1, double Input2, string* ErrorMsg);
63       double HeatCapV    ( string InputSpec, double Input1, double Input2, string* ErrorMsg);
64       double HeatCapP    ( string InputSpec, double Input1, double Input2, string* ErrorMsg);
65       double SoundSpeed  ( string InputSpec, double Input1, double Input2, string* ErrorMsg);
66       double Alpha       ( string InputSpec, double Input1, double Input2, string* ErrorMsg);
67       double Beta        ( string InputSpec, double Input1, double Input2, string* ErrorMsg);
68       double Chi         ( string InputSpec, double Input1, double Input2, string* ErrorMsg);
69       double Fi          ( string InputSpec, double Input1, double Input2, string* ErrorMsg);
70       double Ksi         ( string InputSpec, double Input1, double Input2, string* ErrorMsg);
71       double Psi         ( string InputSpec, double Input1, double Input2, string* ErrorMsg);
72       double Zeta        ( string InputSpec, double Input1, double Input2, string* ErrorMsg);
73       double Theta       ( string InputSpec, double Input1, double Input2, string* ErrorMsg);
74       double Kappa       ( string InputSpec, double Input1, double Input2, string* ErrorMsg);
75       double Gamma       ( string InputSpec, double Input1, double Input2, string* ErrorMsg);
76       double Viscosity   ( string InputSpec, double Input1, double Input2, string* ErrorMsg);
77       double ThermCond   ( string InputSpec, double Input1, double Input2, string* ErrorMsg);
78
79       void AllProps      ( string InputSpec, double Input1, double Input2, double& P, double& T,
80                            double& v, double& d, double& h, double& s, double& u, double& q,
81                            double* x, double* y, double& cv, double& cp, double& c, double& alpha,
82                            double& beta, double& chi, double& fi, double& ksi, double& psi,
83                            double& zeta, double& theta, double& kappa, double& gamma, double& eta,
84                            double& lambda, string* ErrorMsg);
85
86       // Compute all the properties at once, including saturation properties
87       void AllPropsSat   ( string InputSpec, double Input1, double Input2, double& P, double& T,
88                            double& v, double& d, double& h, double& s, double& u, double& q,
89                            double* x, double* y, double& cv, double& cp, double& c, double& alpha,
90                            double& beta, double& chi, double& fi, double& ksi, double& psi,
91                            double& zeta, double& theta, double& kappa, double& gamma, double& eta,
92                            double& lambda, double& d_liq, double& d_vap, double& h_liq, double& h_vap,
93                            double& T_sat, double& dd_liq_dP, double& dd_vap_dP, double& dh_liq_dP,
94                            double& dh_vap_dP, double& dT_sat_dP, string* ErrorMsg);
95
96       double Solve       ( string FuncSpec, double FuncVal, string InputSpec, long Target,
97                            double FixedVal, double MinVal, double MaxVal, string* ErrorMsg);
98
99       double Mmol        ( string* ErrorMsg);
100       double Tcrit       ( string* ErrorMsg);
101       double Pcrit       ( string* ErrorMsg);
```

```
102       double Tmin        ( string* ErrorMsg);
103       double Tmax        ( string* ErrorMsg);
104       void AllInfo       ( double& Mmol, double& Tcrit, double& Pcrit, double& Tmin, double& Tmax,
105                            string* ErrorMsg);
106
107       void SetUnits      ( string UnitSet, string MassOrMole, string Properties, string Units,
108                            string* ErrorMsg);
109       void SetRefState   ( double T_ref, double P_ref, string* ErrorMsg);
110       void GetVersion    ( string ModelName, int* version);
111
112       double* FugaCoef   ( string InputSpec, double Input1, double Input2, string* ErrorMsg);
113       double SurfTens    ( string InputSpec, double Input1, double Input2, string* ErrorMsg);
114       double GibbsEnergy ( string InputSpec, double Input1, double Input2, string* ErrorMsg);
115       void CapeOpenDeriv ( string InputSpec, double Input1, double Input2, double* v, double* h,
116                            double* s, double* G, double* lnphi, string* ErrorMsg);
117
118       double* SpecVolume_Deriv ( string InputSpec, double Input1, double Input2, string* ErrorMsg);
119       double* Enthalpy_Deriv   ( string InputSpec, double Input1, double Input2, string* ErrorMsg);
120       double* Entropy_Deriv    ( string InputSpec, double Input1, double Input2, string* ErrorMsg);
121       double* GibbsEnergy_Deriv( string InputSpec, double Input1, double Input2, string* ErrorMsg);
122       double* FugaCoef_Deriv   ( string InputSpec, double Input1, double Input2, string* ErrorMsg);
123
124    private:
125
126       IClassFactory*  ClassFactory ;      // Pointer to class factory
127       IFluidProp_COM* FluidProp_COM;      // Pointer to FluidProp interface
128
129 };
130
131 #endif // FluidProp_IF_h
```

12.10 fluidpropsolver.h

```
1
35 #ifndef FLUIDPROPSOLVER_H_
36 #define FLUIDPROPSOLVER_H_
37
38 #include "include.h"
39 #if (EXTERNALMEDIA_FLUIDPROP == 1)
40
41 #include "basesolver.h"
42
43 #include "FluidProp_IF.h"
44
45 class FluidPropSolver : public BaseSolver{
46 public:
47     FluidPropSolver(const string &mediumName, const string &libraryName, const string &substanceName);
48     ~FluidPropSolver();
49     virtual void setFluidConstants();
50
51     virtual void setSat_p(double &p, ExternalSaturationProperties *const properties);
52     virtual void setSat_T(double &T, ExternalSaturationProperties *const properties);
53
54     virtual void setState_ph(double &p, double &h, int &phase, ExternalThermodynamicState *const
       properties);
55     virtual void setState_pT(double &p, double &T, ExternalThermodynamicState *const properties);
56     virtual void setState_dT(double &d, double &T, int &phase, ExternalThermodynamicState *const
       properties);
57     virtual void setState_ps(double &p, double &s, int &phase, ExternalThermodynamicState *const
       properties);
58     virtual void setBubbleState(ExternalSaturationProperties *const properties, int phase,
59                                 ExternalThermodynamicState *const bubbleProperties);
60     virtual void setDewState(ExternalSaturationProperties *const properties, int phase,
61                              ExternalThermodynamicState *const dewProperties);
62     virtual double isentropicEnthalpy(double &p, ExternalThermodynamicState *const properties);
63
64 protected:
65     TFluidProp FluidProp;  // Instance of FluidProp wrapper object
66     bool isError(string ErrorMsg);
67     bool licenseError(string ErrorMsg);
68 };
69
70 #endif
71
72 #endif /*FLUIDPROPSOLVER_H_*/
```

12.11 importer.h

```
1 /* ****************************************************************
```

```
2  * Working around Windows' dynamic linker
3  *
4  * Federico Terraneo, Mahder Gebremedhin May 2022
5  **********************************************************************/
6
7 #pragma once
8
9 #ifdef _WIN32
10
11 #define WIN32_LEAN_AND_MEAN
12 #include <windows.h>
13 #include <libloaderapi.h>
14 #include <errhandlingapi.h>
15 #include <psapi.h>
16
17 template<typename T>
18 T importSymbol(const char *funcName)
19 {
20     //TODO: we should do caching
21
22     // First check if the executable itself exports the symbol we want
23     HMODULE exe = GetModuleHandleA(NULL);
24     if(exe == NULL)
25     {
26         fprintf(stderr, "Can't get handle to executable (error %d)\n", GetLastError());
27         exit(1);
28     }
29     T pfn = reinterpret_cast<T>(GetProcAddress(exe, funcName));
30     if(pfn) return pfn;
31
32     // If we don't find it in the executable, then we search it in all loaded DLLs
33     HANDLE process = GetCurrentProcess();
34     if(process == NULL)
35     {
36         fprintf(stderr, "Can't get a handle to current process (error %d)\n", GetLastError());
37         exit(1);
38     }
39
40     HMODULE loaded_modules[1024];
41     DWORD cbNeeded;
42     auto result = EnumProcessModules(process, loaded_modules, sizeof(loaded_modules), &cbNeeded);
43     CloseHandle(process);
44     if(!result)
45     {
46         fprintf(stderr, "Can't enumerate loaded modules (error %d)\n", GetLastError());
47         exit(1);
48     }
49
50     int num_modules = cbNeeded / sizeof(HMODULE); // Actual number of loaded modules
51     for(int i = 0; i < num_modules; i++)
52     {
53         T pfn = reinterpret_cast<T>(GetProcAddress(loaded_modules[i], funcName));
54         if(pfn) return pfn;
55     }
56
57     fprintf(stderr, "Can't get handle to %s in all loaded modules.\n", funcName);
58     exit(1);
59 }
60
61 #define IMPORT(x,y) auto y = importSymbol<x>(#y)
62
63 #else //_WIN32
64
65 #include "ModelicaUtilities.h"
66
67 // Nothing to do on Linux, its linker just works
68 #define IMPORT(x,y)
69
70 #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

  FluidProp solver.
- #define EXTERNALMEDIA_COOLPROP 1

  CoolProp solver.
- #define NAN 0xffffffff

  Not a number.
- #define **ISNAN**(x) (x == NAN)

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 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.

```
1
15 #ifndef INCLUDE_H_
16 #define INCLUDE_H_
17
18 /*********************************************************************
19  *               Start of user option selection
20  *********************************************************************/
21
22 // Selection of used external fluid property computation packages.
24
28 #ifndef EXTERNALMEDIA_FLUIDPROP
29 #define EXTERNALMEDIA_FLUIDPROP 0
30 #endif
31
32 // Selection of used external fluid property computation packages.
34
38 #ifndef EXTERNALMEDIA_COOLPROP
39 #define EXTERNALMEDIA_COOLPROP 1
40 #endif
41
43
47 #include <math.h>
48 #ifndef NAN
49 #define NAN 0xffffffff
50 #endif
51 #ifndef ISNAN
52 #define ISNAN(x) (x == NAN)
53 #endif
54
55 /*********************************************************************
56  *                End of user option selection
57  *            Do not change anything below this line
58  *********************************************************************/
59
60 // General purpose includes
61 #include <map>
62 using std::map;
63
64 #include <string>
65 using std::string;
66
67 // Include error handling
68 #include "errorhandling.h"
69
70 #endif /*INCLUDE_H_*/
```

12.14 ModelicaUtilities.h

```
1 /* ModelicaUtilities.h - External utility functions header
2
3    Copyright (C) 2010-2020, Modelica Association and contributors
4    All rights reserved.
5
6    Redistribution and use in source and binary forms, with or without
7    modification, are permitted provided that the following conditions are met:
8
9    1. Redistributions of source code must retain the above copyright notice,
10       this list of conditions and the following disclaimer.
11
12    2. Redistributions in binary form must reproduce the above copyright
13       notice, this list of conditions and the following disclaimer in the
14       documentation and/or other materials provided with the distribution.
15
16    3. Neither the name of the copyright holder nor the names of its
17       contributors may be used to endorse or promote products derived from
18       this software without specific prior written permission.
19
20    THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND
21    ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
22    WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
23    DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE
24    FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
25    DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
26    SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
27    CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
28    OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
29    OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
30 */
```

```
31
32 /* Utility functions which can be called by external Modelica functions.
33
34    These functions are defined in section 12.8.6 of the
35    Modelica Specification 3.0 and section 12.9.6 of the
36    Modelica Specification 3.1 and later.
37
38    A generic C-implementation of these functions cannot be given,
39    because it is tool dependent how strings are output in a
40    window of the respective simulation tool. Therefore, only
41    this header file is shipped with the Modelica Standard Library.
42 */
43
44 #ifndef MODELICA_UTILITIES_H
45 #define MODELICA_UTILITIES_H
46
47 #include <stddef.h>
48 #include <stdarg.h>
49
50 #if defined(__cplusplus)
51 extern "C" {
52 #endif
53
54 /*
55   Some of the functions never return to the caller. In order to compile
56   external Modelica C-code in most compilers, noreturn attributes need to
57   be present to avoid warnings or errors.
58
59   The following macros handle noreturn attributes according to the latest
60   C11/C++11 standard with fallback to GNU, Clang or MSVC extensions if using
61   an older compiler.
62 */
63 #undef MODELICA_NORETURN
64 #undef MODELICA_NORETURNATTR
65 #if __STDC_VERSION__ >= 201112L
66 #define MODELICA_NORETURN _Noreturn
67 #define MODELICA_NORETURNATTR
68 #elif defined(__cplusplus) && __cplusplus >= 201103L
69 #if (defined(__GNUC__) && __GNUC__ >= 5) || \
70     (defined(__GNUC__) && defined(__GNUC_MINOR__) && __GNUC__ == 4 && __GNUC_MINOR__ >= 8)
71 #define MODELICA_NORETURN [[noreturn]]
72 #define MODELICA_NORETURNATTR
73 #elif (defined(__GNUC__) && __GNUC__ >= 3) || \
74       (defined(__GNUC__) && defined(__GNUC_MINOR__) && __GNUC__ == 2 && __GNUC_MINOR__ >= 8)
75 #define MODELICA_NORETURN
76 #define MODELICA_NORETURNATTR __attribute__((noreturn))
77 #elif defined(__GNUC__)
78 #define MODELICA_NORETURN
79 #define MODELICA_NORETURNATTR
80 #else
81 #define MODELICA_NORETURN [[noreturn]]
82 #define MODELICA_NORETURNATTR
83 #endif
84 #elif defined(__clang__)
85 /* Encapsulated for Clang since GCC fails to process __has_attribute */
86 #if __has_attribute(noreturn)
87 #define MODELICA_NORETURN
88 #define MODELICA_NORETURNATTR __attribute__((noreturn))
89 #else
90 #define MODELICA_NORETURN
91 #define MODELICA_NORETURNATTR
92 #endif
93 #elif (defined(__GNUC__) && __GNUC__ >= 3) || \
94       (defined(__GNUC__) && defined(__GNUC_MINOR__) && __GNUC__ == 2 && __GNUC_MINOR__ >= 8) || \
95       (defined(__SUNPRO_C) && __SUNPRO_C >= 0x5110)
96 #define MODELICA_NORETURN
97 #define MODELICA_NORETURNATTR __attribute__((noreturn))
98 #elif (defined(_MSC_VER) && _MSC_VER >= 1200) || \
99       defined(__BORLANDC__)
100 #define MODELICA_NORETURN __declspec(noreturn)
101 #define MODELICA_NORETURNATTR
102 #else
103 #define MODELICA_NORETURN
104 #define MODELICA_NORETURNATTR
105 #endif
106
107 /*
108   The following macros handle format attributes for type-checks against a
109   format string.
110 */
111
112 #if defined(__clang__)
113 /* Encapsulated for Clang since GCC fails to process __has_attribute */
114 #if __has_attribute(format)
115 #define MODELICA_FORMATATTR_PRINTF __attribute__((format(printf, 1, 2)))
116 #define MODELICA_FORMATATTR_VPRINTF __attribute__((format(printf, 1, 0)))
117 #else
```

```
118 #define MODELICA_FORMATATTR_PRINTF
119 #define MODELICA_FORMATATTR_VPRINTF
120 #endif
121 #elif defined(__GNUC__) && __GNUC__ >= 3
122 #define MODELICA_FORMATATTR_PRINTF __attribute__((format(printf, 1, 2)))
123 #define MODELICA_FORMATATTR_VPRINTF __attribute__((format(printf, 1, 0)))
124 #else
125 #define MODELICA_FORMATATTR_PRINTF
126 #define MODELICA_FORMATATTR_VPRINTF
127 #endif
128
129 void ModelicaMessage(const char *string);
130 /*
131 Output the message string (no format control).
132 */
133
134
135 void ModelicaFormatMessage(const char *string, ...) MODELICA_FORMATATTR_PRINTF;
136 /*
137 Output the message under the same format control as the C-function printf.
138 */
139
140
141 void ModelicaVFormatMessage(const char *string, va_list args) MODELICA_FORMATATTR_VPRINTF;
142 /*
143 Output the message under the same format control as the C-function vprintf.
144 */
145
146
147 MODELICA_NORETURN void ModelicaError(const char *string) MODELICA_NORETURNATTR;
148 /*
149 Output the error message string (no format control). This function
150 never returns to the calling function, but handles the error
151 similarly to an assert in the Modelica code.
152 */
153
154 void ModelicaWarning(const char *string);
155 /*
156 Output the warning message string (no format control).
157 */
158
159 void ModelicaFormatWarning(const char *string, ...) MODELICA_FORMATATTR_PRINTF;
160 /*
161 Output the warning message under the same format control as the C-function printf.
162 */
163
164 void ModelicaVFormatWarning(const char *string, va_list args) MODELICA_FORMATATTR_VPRINTF;
165 /*
166 Output the warning message under the same format control as the C-function vprintf.
167 */
168
169 MODELICA_NORETURN void ModelicaFormatError(const char *string, ...) MODELICA_NORETURNATTR
       MODELICA_FORMATATTR_PRINTF;
170 /*
171 Output the error message under the same format control as the C-function
172 printf. This function never returns to the calling function,
173 but handles the error similarly to an assert in the Modelica code.
174 */
175
176
177 MODELICA_NORETURN void ModelicaVFormatError(const char *string, va_list args) MODELICA_NORETURNATTR
       MODELICA_FORMATATTR_VPRINTF;
178 /*
179 Output the error message under the same format control as the C-function
180 vprintf. This function never returns to the calling function,
181 but handles the error similarly to an assert in the Modelica code.
182 */
183
184
185 char* ModelicaAllocateString(size_t len);
186 /*
187 Allocate memory for a Modelica string which is used as return
188 argument of an external Modelica function. Note, that the storage
189 for string arrays (= pointer to string array) is still provided by the
190 calling program, as for any other array. If an error occurs, this
191 function does not return, but calls "ModelicaError".
192 */
193
194
195 char* ModelicaAllocateStringWithErrorReturn(size_t len);
196 /*
197 Same as ModelicaAllocateString, except that in case of error, the
198 function returns 0. This allows the external function to close files
199 and free other open resources in case of error. After cleaning up
200 resources use ModelicaError or ModelicaFormatError to signal
201 the error.
202 */
```

```
203
204 #if defined(__cplusplus)
205 }
206 #endif
207
208 #endif
```

12.15 solvermap.h

```
1 #ifndef SOLVERMAP_H_
2 #define SOLVERMAP_H_
3
4 #include "include.h"
5
6 class BaseSolver;
7
9
18 class SolverMap{
19 public:
20     static BaseSolver *getSolver(const string &mediumName, const string &libraryName, const string
      &substanceName);
21     static string solverKey(const string &libraryName, const string &substanceName);
22
23 protected:
25     static map<string, BaseSolver*> _solvers;
26 };
27
28 #endif // SOLVERMAP_H_
```

12.16 testsolver.h

```
1 #ifndef TESTSOLVER_H_
2 #define TESTSOLVER_H_
3
4 #include "basesolver.h"
5
7
35 class TestSolver : public BaseSolver{
36 public:
37     TestSolver(const string &mediumName, const string &libraryName, const string &substanceName);
38     ~TestSolver();
39     virtual void setFluidConstants();
40
41     virtual void setSat_p(double &p, ExternalSaturationProperties *const properties);
42     virtual void setSat_T(double &T, ExternalSaturationProperties *const properties);
43
44     virtual void setState_ph(double &p, double &h, int &phase, ExternalThermodynamicState *const
      properties);
45     virtual void setState_pT(double &p, double &T, ExternalThermodynamicState *const properties);
46     virtual void setState_dT(double &d, double &T, int &phase, ExternalThermodynamicState *const
      properties);
47     virtual void setState_ps(double &p, double &s, int &phase, ExternalThermodynamicState *const
      properties);
48 };
49
50 #endif // TESTSOLVER_H_
```

Index