

The Modelica.Media Library

Francesco Casella

Dipartimento di Elettronica e Informazione
Politecnico di Milano



Introduction

- The Modelica.Media library is a package of the Modelica Standard Library to compute fluid properties
- Goals of the library
 - Provide a standard framework for replaceable fluid models
 - Provide ready-made models for the most commonly used fluids
 - Allow the computations of all the relevant fluid properties, including partial derivatives of thermodynamic functions
 - Allow efficient numerical simulations+
- The ExternalMedia provides a general framework to implement efficient Modelica.Media-compatible fluid models using external routines

Structure of the library

- Each medium model of the library is contained in a **package**
- Each package contains
 - all the functions and models defined by the base class (interface definition)
 - re-definition of unit types with appropriate default attributes
 - additional functions and constants needed to compute the fluid properties
- Base classes are defined in Modelica.Media.Interfaces for different categories of fluids
 - PartialMedium: the “mother of all medium models”
 - PartialPureSubstance: pure substance. Defines functions without the composition input
 - PartialMixtureMedium: defines functions to convert mole to mass fractions
 - PartialTwoPhaseMedium: defines functions to compute saturation properties
- Fluid properties can be computed via:
 - setState_xx() functions / ThermodynamicState records (functional interface)
 - BaseProperties models (equation-based interface)

Using ThermodynamicState

- A ThermodynamicState record is obtained from two (three for mixtures) state variable inputs
 - $p, T, (X)$
 - $p, h, (X)$
 - $d, T, (X)$
 - $p, s, (X)$
- All the other properties can be computed as a function of the ThermodynamicState record
- The ThermodynamicState record contains some properties from which it is “easy” to compute all other ones actual content of the record is medium-dependent
- Models with replaceable media should not access the fields of ThermodynamicState directly

Using ThermodynamicState (pure fluid)

```
model ComponentWithPureFluid
  replaceable package Medium =
    Modelica.Media.Interfaces.PartialPureSubstance;
  Medium.ThermodynamicState state1, state2;
  Medium.SpecificEnthalpy h1;
  Medium.Density d1, d2;
  Medium.ThermalConductivity lambda1;
equation
  state1 = Medium.setState_pT(1.0e5, 293.15);
  state2 = Medium.setState_dT(500, 650);
  h1 = Medium.specificEnthalpy(state1);
  d1 = Medium.density(state1);
  d2 = Medium.density(state2);
  lambda1 = Medium.ThermalConductivity(state1);
end ComponentWithPureFluid;
```

```
model ComponentWithWater
  extends ComponentWithPureFluid(
    redeclare package Medium =
      Modelica.Media.Water.StandardWater);
end ComponentWithWater;
```

```
model Plant
  ComponentWithPureFluid C
  ..
end Plant;
```

```
Plant P(C(redeclare package Medium =
  Modelica.Media.Water.StandardWater));
```



Using ThermodynamicState (mixture fluid)

```
model ComponentWithMixtureFluid
  replaceable package Medium =
    Modelica.Media.Interfaces.PartialMixtureMedium;
  parameter Medium.MassFraction X[Medium.nX];
  Medium.ThermodynamicState state;
  Medium.SpecificEnthalpy h;
  Medium.Density d;
  Medium.SpecificHeatCapacityCp cp;
equation
  state = Medium.setState_pTX(1.0e5, 293.15, X);
  h = Medium.specificEnthalpy(state);
  d = Medium.density(state);
  cp = Medium.ThermalConductivity(state);
end ComponentWithPureFluid;



model ComponentWithAir
  extends ComponentWithMixtureFluid(
    redeclare package Medium = Modelica.Media.Air.DryAirNasa,
    X = Modelica.Media.Air.DryAirNasa.reference_X);
end ComponentWithAir;
```

Two-phase media

- Two-phase medium models also allow to compute the saturation properties

One point on the thermodynamic plane  ThermodynamicState  All thermodynamic properties

One point on the saturation curve  SaturationProperties  All saturation properties

SaturationProperties  ThermodynamicState  All thermodynamic properties

Using ThermodynamicState (two-phase fluid)

```
model ComponentWithTwoPhaseFluid
  replaceable package Medium =
    Modelica.Media.Interfaces.PartialTwoPhaseMedium;
  Medium.SaturationProperties sat;
  Medium.Temperature Ts;
  Medium.SpecificEnthalpy hl, hv
  Medium.Density dl, dv;
  Medium.SurfaceTension sigma;
  Medium.ThermodynamicState state;
  Medium.DerDensityByPressure dd_dp;
```

equation

```
  sat = Medium.setSat_p(10e5);
  Ts = Medium.saturationTemperature_sat(sat);
  hl = Medium.bubbleEnthalpy(sat);
  hv = Medium.dewEnthalpy(sat);
  dl = Medium.bubbleDensity(sat);
  dv = Medium.dewDensity(sat);
  sigma = Medium.surfaceTension(sat);
  state = Medium.setDewState(sat, 2);
  dd_dp = Medium.density_derp_h(state);
end ComponentWithPureFluid;
```

```
model ComponentWithWater
  extends ComponentWithPureFluid(
    redeclare package Medium = Modelica.Media.Water.StandardWater);
end ComponentWithWater;
```


Using BaseProperties (pure fluids)

- The BaseProperties model contains
 - the N most frequently used thermodynamic properties (p, T, d, u, h, X, MM)
 - N-2 equation equations-of-state relating them
- The model can be used by adding 2 more equations to completely specify the thermodynamic conditions

```
model ComponentWithPureFluid
  replaceable package Medium =
    Modelica.Media.Interfaces.PartialPureSubstance;
  Medium.BaseProperties medium;
equation
  medium.p = 1.0e5;
  medium.T = 293.15;
  // medium.d, medium.h, medium.u, medium.MM are available
end ComponentWithPureFluid;
```

Using BaseProperties (mixture fluids)

- The number of additional equations is in general $2 + \text{Medium.nXi}$
- Medium.nX : number of components of the medium model
- Medium.nXi : number of *independent* components in the BaseProperties model
 - pure fluid ($nS = 1$) $nXi = 0$;
 - mixture with $\text{reducedX} = \text{false}$ $nXi = nX$;
 - mixture with $\text{reducedX} = \text{true}$ $nXi = nX - 1$;
 - mixture with $\text{fixedX} = \text{true}$ $nXi = 0$;
- The equations to determine X from Xi are included in the base class

```
Xi = X[1:nXi];  
if nX > 1 then  
  if fixedX then  
    X = reference_X;  
  elseif reducedX then  
    X[nX] = 1 - sum(Xi);  
  end if;  
end if;
```

Using BaseProperties (mixture fluids)

```
model ComponentWithMixtureFluid
  replaceable package Medium =
    Modelica.Media.Interfaces.PartialMixtureMedium;
  Medium.BaseProperties medium;
  parameter Medium.MassFraction X[Medium.nX];
equation
  medium.p = 1.0e5;
  medium.T = 293.15;
  medium.nXi = X[1:Medium.nXi]
  // medium.d, medium.h, medium.u, medium.MM are available
end ComponentWithMixtureFluid;
```

```
model ComponentWithAir
  extends ComponentWithMixtureFluid(
    redeclare package Medium = Modelica.Media.Air.DryAirNasa,
    X = Modelica.Media.Air.DryAirNasa.reference_X);
end ComponentWithAir;
```

Using BaseProperties (extra properties)

- The BaseProperties model contains an instance of ThermodynamicState: this can be used to compute any extra property on-demand
- If the medium is 2-phase, BaseProperties also contains a SaturationProperties record, set at the medium *pressure*

```
model ComponentWithTwoPhaseFluid
  replaceable package Medium =
    Modelica.Media.Interfaces.PartialTwoPhaseMedium;
  Medium.BaseProperties medium;
  Medium.SpecificHeatCapacity cv;
  Medium.Density dl, dv;
equation
  medium.p = 1.0e5;
  medium.T = 293.15;
  // medium.d, medium.h, medium.u, medium.MM are available

  // other properties
  cv = Medium.specificHeatCapacityCv(medium.state);

  // saturation properties at the medium pressure
  dl = Medium.bubbleDensity(medium.sat);
  dv = Medium.dewDensity(medium.sat);
end ComponentWithPureFluid;
```

Using BaseProperties (dynamic models)

- By setting preferredMediumStates = true, the medium model automatically select the preferred state variables (using the stateSelect attribute)

```
model DynamicBalanceComponent
  replaceable package Medium =
    Modelica.Media.Interfaces.PartialPureSubstance;
  Medium.BaseProperties medium(preferredMediumStates = true);
  ...
equation
  M = V*medium.d;
  U = M*medium.u;
  der(M) = port.m_flow; // mass balance
  der(U) = port.H_flow; // energy balance
  ...
end DynamicBalanceComponent;
```

- The tool will automatically differentiate the differential equations in order to change the state variables to the preferred set

```
der(M) = V*der(medium.d) = V*pder(medium.d,p)*der(medium.p) +
                                     +V*pder(medium.d,T)*der(medium.T);
der(U) = M*der(medium.u) + medium.u * der(M) = ...
```

Using BaseProperties (dynamic mixtures)

- If a mixture medium is used, the nX_i independent mass fractions will also become states

```
model DynamicBalanceMixtureComponent
  replaceable package Medium =
    Modelica.Media.Interfaces.PartialMixtureMedium;
    Medium.BaseProperties medium(preferredMediumStates = true);
    SI.Mass MXi[Medium.nXi];
    ...
equation
  M = V*medium.d;
  MXi = M*medium.Xi;
  U = M*medium.u;
  der(M) = port.m_flow;           // mass balance
  der(MXi) = port.m_flow*port.Xi; // component mass balance
  der(U) = port.H_flow;           // energy balance
  ...
end DynamicBalanceMixtureComponent;
```

Using BaseProperties (dynamic models)

How can the tool compute the derivative of the density?

- Density computed by equations
 - ⇒ automatic differentiation of the equations
- Density computed by a Modelica function
 - ⇒ automatic differentiation of the function algorithm (hard / possibly inefficient)
 - ⇒ function inlining, then symbolic differentiation of equations
 - ⇒ derivative() annotation
- Density computed by an external function
 - ⇒ derivative() annotation

```
function f
  input Real x;
  input Real y;
  output f;
  annotation(derivative=f_der);
algorithm
  ...
end f;
```

```
function f_der
  "Total time derivative of f"
  input Real x;
  input Real y;
  input Real x_der;
  input Real y_der;
  output f_der;
algorithm
  ...
end f_der;
```

Implementation of BaseProperties (I)

```
package MyMedium extends Modelica.Media.Interfaces.PureSubstance;
...
redeclare model extends BaseProperties(
  p(stateSelect = if preferredMediumStates then StateSelect.prefer
    else StateSelect.default),
  T(stateSelect = if preferredMediumStates then StateSelect.prefer
    else StateSelect.default));
  equation
    d = density_pT(p,T);
    h = f_h(p,T);
    u = h - p/d;
    MM = 0.042;
end BaseProperties;

...
end MyMedium;
```


Implementation of BaseProperties (II)

```
package MyMedium extends Modelica.Media.Interfaces.PureSubstance;
...
function density_pT
  input Pressure p;
  input Temperature T;
  output Density d;
  annotation(derivative = density_pT_der);
algorithm
  d := density(setState_pT(p,T));
end density_pT;

function density_pT_der
  input Pressure p;
  input Temperature T;
  input Real p_der;
  input Real T_der;
  output Real d_der;
algorithm
  d_der := p_der*density_derp_h(setState_pT(p,T)) +
           T_der*density_derh_p(setState_pT(p,T));
end density_pT_der;

...
end MyMedium;
```

Issues w/ BaseProperties (derivatives)

The current approach with automatic state selection and differentiation is not completely satisfactory

- It is necessary to write time derivative functions and all the derivative annotations, unless the tool is able to derive the algorithm of the derivative function automatically (not always possible / efficient)
- It is necessary to write functions for the partial derivatives (mandatory, if external functions are possibly going to be used)

Medium models in the Standard Library

- As of version 3.1/3.2, the Modelica Standard library contains many medium model implementations
 - Constant-composition ideal gas with constant c_p
 - Mixtures of ideal gases, with accurate $h(T)$ (NASA data)
 - Humid air
 - Detailed water (IF97)
 - Incompressible fluid, with table-based $d(T)$
 - Linear compressible fluid (constant c_p , compressibility, th. expansion coeff.) (with inlining, pure equations at the end)
- The ExternalMedia has been developed in 2007 to interface Modelica.Media with external property computation codes
 - Interface with any external C/C++/Fortran code/Java
 - Fully compatible with PartialTwoPhaseMedium – mixture media not (yet) supported
 - Available for Modelica 2 (MSL 2.2.x) and Modelica 3 (MSL 3.x)

Future of Modelica.Media

- Some basic design flaws have been pointed out during time:
 - BaseProperties should be a part of the Fluid library (or other thermofluid libraries), not of Media.
 - State selection is not a concern for the library
 - The handling of independent compositions (n_S , n_X , n_{Xi}) is messy
- Proposal for new (non backwards-compatible) Media library
 - No more BaseProperties
 - Two separate libraries
 - Mass-fraction-based for thermal / energy applications
 - Mole-fraction-based for chemical / process application
 - Only a functional interface available
 - Always provide the full composition vector as input (n_S)
 - Partial class framework allows to develop new medium models by only implementing a few class
- The proposal has been around for almost two years now. Some resources (e.g. some European Project) are required to have it implemented and become part of the MSL
- It might be worthwhile to also wait for Modelica 4, which might provide automatic medium propagation via type inference