IDAES Helmholtz Equation of State Functions

John Eslick

March 31, 2023

Contents

Chapter 1

Introduction

IDAES [1] is a process modeling framework based on Pyomo [2] that enables advanced process optimization. Optimization solvers often require accurate first and second derivative information, which may not be fully available from standard property packages. Thermodynamic cycles using a pure component working fluid are common in power generation and heating and cooling; therefore, high accuracy thermodynamically consistent properties of pure fluids are needed. The IDAES general Helmholtz equation of state (HEoS) functions can be used to calculate high accuracy fluid properties of pure components or mixtures as pseudo-pure components. Azeotropic, nearly azeotropic or single phase fixed composition mixtures can be models as pure components. Future work may extend the functions to general mixtures, but general non-azeotropic mixtures are not currently supported.

Complex property models pose a challenge when solving equation oriented models. Some equation solving is involved in calculating phase equilibrium and changing state variables. Implementing the property calculations as external functions provides problem decomposition where specialized techniques can be used to ensure that the property models solve reliably and correctly, while reducing the complexity of the process model in which they are used.

1.1 Helmholtz Equation of State

The HEoS uses a Helmholtz free energy correlation to predict thermodynamic properties of pure components. The dimensionless Helmholtz free energy can split into an ideal part and real part and written as Equation 1.1. Thermodynamic properties can be calculated from the two parts of the Helmholtz free energy and their first and second derivatives with respect to δ and τ.

$$\frac{f(\delta,\tau)}{RT} = \phi^{\circ}(\delta,\tau) + \phi^{r}(\delta,\tau) \tag{1.1}$$

Where:

- f = Specific Helmholtz free energy [kJ/kg]
- ϕ° = Ideal dimensionless Helmholtz free energy
- ϕ^{r} = Residual dimensionless Helmholtz free energy
- $\delta = \rho/\rho^{*}$
- $\tau = T^{*}/T$

- R = Specific gas constant, ideal gas constant/molecular weight [kJ/kg/K]
- T = Temperature [K]
- T^* = Reducing temperature, usually critical temperature [K]
- ρ = Mass density [kg/m^3]
- ρ^* = Reducing mass density, usually critical density [kg/m^3]

1.2 State Variables

State variables define the state of a system. Usually two thermodynamic quantities define the state for example enthalpy and pressure; however, for temperature and pressure the vapor fraction is also required due to temperature being constant at a given pressure in the two-phase region. The HEoS provides properties as a direct function of density and temperature, but they are not usually the most convenient set of state variables, so the HEoS functions provide properties as a function of several other sets of state variables. Supported state variable sets are listed in Table 1.1.

Table 1.1: Supported state variable sets

symbols	description
(δ, τ)	reduced density, 1/reduced temperature
(h, p)	enthalpy [kJ/kg], pressure [kPa]
(s, p)	entropy [kJ/kg/K], pressure [kPa]
(u, p)	internal energy [kJ/kg], pressure [kPa]
(T, p, x)	temperature [K], pressure [kPa], vapor fraction

1.3 Thermodynamic Properties

The basic set of properties that can be calculated are listed in Table 1.2. For each property there are corresponding functions for other sets of state variables shown in Table 1.1 In addition to the properties listed the components of the dimensions Helmholtz free energy and up to there fourth derivatives are provided. The allows calculation of most thermodynamic properties and their first and second derivatives, as required by optimization solvers.

Table 1.2: Supported thermodynamic properties

property	description
$p(\delta, \tau)$	pressure [kPa]
$u(\delta, \tau)$	internal energy [kJ/kg]
$s(\delta, \tau)$	entropy [kJ/kg/K]
$h(\delta, \tau)$	enthalpy [kJ/kg]
$g(\delta, \tau)$	Gibbs free energy [kJ/kg]
$f(\delta, \tau)$	Helmholtz free energy [kJ/kg]
$c_v(\delta, \tau)$	isochoric heat capacity [kJ/kg/K]
$c_p(\delta, \tau)$	isobaric heat capacity [kJ/kg/K]
$w(\delta, \tau)$	speed of sound [m/s]
$\kappa_T(\delta, \tau)$	isothermal compressibility [1/MPa]

1.4 Transport Properties

Transport properties can also be calculated, but their models are separate from the equation of state, so they may not be available for all components. Transport properties are listed in Table 1.3.

Table 1.3: Supported transport properties

property	description
$\mu(\delta, \tau)$	viscosity [μPa $\cdot$ s]
$\lambda(\delta, \tau)$	thermal conductivity [mW/m/K]
$\sigma(\delta, \tau)$	surface tension [mN/M]

1.5 Saturation Properties

The saturated liquid and vapor properties are calculated by solving the Equations 1.2 and 1.3 for a given temperature for the liquid and vapor density. Once the liquid and vapor densities are found, all other properties of saturated liquid and vapor can be found. The method proposed by Akasaka [3] is used to calculate the saturated phase densities at a given temperature.

$$g(\delta_{\text{liq}}, \tau) = g(\delta_{\text{vap}}, \tau) \tag{1.2}$$

$$p(\delta_{\text{liq}}, \tau) = p(\delta_{\text{vap}}, \tau) \tag{1.3}$$

With the saturation pressure as a function of temperature, the reverse problem can be solved to calculate saturation temperature from pressure, and this allows all saturation properties to be calculated as either a function of temperature or pressure. Saturation density may not be unique for other state variables, so saturated properties are only available as functions of temperature or pressure.

1.6 Deviation from Literature

The Helmholtz free energy correlation for water [4] and CO_2 [5] omit the non-analytic terms. The non-analytic terms improve accuracy near the critical point, but they can cause numerical difficulties since there is a singularity at the critical point. It is possible to add these terms, but the effect of omitting them is minimal.

Chapter 2

Building the Library

This section provides information about compiling the shared library for the HEoS functions. A fully automated build system has not yet be implemented, since the IDAES team uses specialized scripts for binary releases. The HEoS shared library can still be done independently.

2.1 Prerequisites

A C++ compiler is required. There are many option and the suggestions given are not hard requirements, just what the IDAES team uses and find convenient.

- Linux: GCC can be installed using the systems package manager.
- Windows: MinGW-w64 (https://www.mingw-w64.org) and MSYS2 (https://www.msys2.org)
- macOS: Xcode command-line tools (https://developer.apple.com/xcode/)

Boost is used for equation solvers, hashing tuples, and reading json. Below are suggestions for installing

- Linux: Boost can be obtained via package manager
- Windows: Boost can be installed via the MSYS2 package manger, if you use it to run MinGW
- macOS: Boost can be installed via Hombrew (https://brew.sh)

AMPL Solver Library (ASL) is required. The ASL is used to read arbitrary expressions into the C++ functions, and to make functions callable from AMPL solvers. The ASL is available on GitHub (https://github.com/ampl/asl).

2.2 Build

In the `general_helmholtz` directory, edit the make file. There are a few commented out variables `BOOST`, `ASL`, and `ASL_LIB`. Depending on the specific setup, these may or may not be needed. If the build does not find ASL or BOOST the problem can most likely be fixed by setting these.

Run `make` in the `general_helmholtz` directory and the `general_helmholtz.so` file should be built. If there is a problem try setting the variables in the previous paragraph.

2.3 Test

To test the build, set the `IDAES_HELMHOLTZ_DATA_PATH` and `IDAES_HELMHOLTZ_TEST_DATA_PATH` variables to the `param_data` and `param_data` directories. Be sure to end the path in a file separator.

Run `make test`.

The command should run without returning an error code, even though it may generate a small number of error messages. The test data may not contain enough significant figures to calculate pressure from density accurately, and in the neighborhood of the critical point, rapid changes in properties may make matching the test data difficult.

Chapter 3

Property Functions

This section describes the most used property functions. Currently functions return a single property value. Since expressions for the Helmholtz free energy are complex having a large number of parameters and calculation of saturation properties and state variable changes involve iterative solutions to equations, the HEoS functions use a system to cache results. If you call the same function with the same inputs the stored result is returned. This goes for most of the complex intermediate calculations as well. This reduces the computational overhead when calculating multiple properties for the same state variable values.

3.1 Parameter Files

Adding new components or adjusting component parameters is discussed later in this document. For now, an overview of the required parameter files and how they are located is provided. All parameter files must be stored in the same directory. Files for a component are read in based on the component string. The directory to search is set by either the `IDAES_HELMHOLTZ_DATA_PATH` environment variable, which should be set to the directory containing parameter files and should end in a file path separator (i.e. \ or /), or passed to the function call to read in data.

Table 3.1: Parameter Files

file	description
{comp}_parameters.json	Basic parameters and expression mappings
{comp}_expressions_eos.json	Equation of state expressions
{comp}_expressions_tcx.json	Optional, thermal conductivity model
{comp}_expressions_visc.json	Optional, viscosity model
{comp}_expressions_st.json	Optional, surface tension model

To read in data for a component, use the `read_params` function prototyped in the `read_params.h` file.

```
uint read_params(std::string comp, std::string data_path="")
```

The fist argument (`comp`) is a string identifying a component. This is the first part of the parameter files. The second argument is optional, but can provide the search path for parameter files instead of using the environment variable. The function returns an integer, which is an index pointing to parameter data. In all other functions when identifying a component the integer index should be used rather than the component string. Component strings are not case sensitive, and are all converted to lower case, since no all file systems are case sensitive.

3.2 Reference State

Some properties (i.e. enthalpy, entropy, and internal energy) don't have absolute values and are reported relative to a reference state. The Helmholtz free energy correlation parameters directly from the literature are used for the equation of state, and may not use the desired reference state. A reference state is defined from a given reference enthalpy and entropy at a particular state. The reference state can be adjusted by modifying parameters in the ideal component of the dimensionless Helmholtz free energy. Ideal dimensionless Helmholtz free energy correlation can have different forms but they all contain the terms shown in Equation 3.1

$$\phi^{\circ}(\delta,\tau) = \log_e \delta + n_1^{\circ} + \tau n_2^{\circ} + ... \tag{3.1}$$

The reference state can be set by adding an offset to the n_1° and n_2° constants, as in Equation 3.2.

$$\phi^{\circ}(\delta,\tau) = \log_e \delta + n_1^{\circ} + n_{1,\text{off}}^{\circ} + \tau n_2^{\circ} + \tau n_{2,\text{off}}^{\circ} + ... \tag{3.2}$$

To calculate the offsets for a reference state, first calculate the temperature, density, enthalpy, and entropy for the reference state with no offset. The $n_{1,\text{off}}^{\circ}$ value can be calculated by Equation 3.3 and $n_{2,\text{off}}^{\circ}$ can be calculated by Equation 3.4. The non-offset properties, density and temperature can be calculated with property functions with the offset set to (0, 0).

$$n_{1,\text{off}}^{\circ} = \frac{s_1(\delta^{\text{ref}},\tau^{\text{ref}}) - s^{\text{ref}}}{R} \tag{3.3}$$

$$n_{2,\text{off}}^{\circ} = \frac{h^{\text{ref}} - h_1(\delta^{\text{ref}},\tau^{\text{ref}})}{RT^{\text{ref}}} \tag{3.4}$$

Where:

- δ^{ref}: reduced density at the reference point
- τ^{ref}: 1/reduced temperature at the reference point
- $s_1(\delta^{\text{ref}},\tau^{\text{ref}})$: no-offset entropy at the reference point
- $h_1(\delta^{\text{ref}},\tau^{\text{ref}})$: no-offset enthalpy at the reference point
- s^{ref}: specified entropy at the reference point
- h^{ref}: specified enthalpy at the reference point
- T^{ref}: temperature at the reference state [K]
- R: specific gas constant [kJ/kg/K]

There are two ways to set the reference state offset for a component, (1) edit the {comp}_parameters.json file to change the reference_state_offset field or (2) call the `void set_reference_state_offset(uint comp_idx, double n1, double n2)` function prototyped in read_param.h, where n1 and n2.

3.3 Return Structure

The C++ property functions return a structure that contains a property value along with first and second derivatives. Most functions take two state variables, but saturation properties only take one. The structures are defined in the `config.h` file. The most commonly needed structures are `f12_struct` and `f22_struct` given below.

```
struct f12_struct { // return structure for unary functions
  double f;    // Function value
  double f_1;  // First derivative with respect to the argument
  double f_11; // Second derivative with respect to the argument
};

struct f22_struct { // return structure for binary functions
  double f;    // Function value
  double f_1;  // First derivative with respect to the first argument
  double f_11; // Second derivative with respect to the first argument
  double f_2;  // First derivative with respect to the second argument
  double f_12; // Second derivative with respect to the first and second argument
  double f_22; // Second derivative with respect to the second argument
};
```

3.4 Functions of δ and τ

The fundamental property functions are functions of δ and τ. These function prototypes are in the `props.h` file and listed in Table 3.2. These property functions are valid for a single phase density phase not a mixed phase. The transport properties (thermal conductivity, viscosity, and surface tension) are optional and not necessarily provided for every component. They are not derived from the equation of state and require supplemental correlations.

Table 3.2: Property Functions of δ and τ

symbol	function	units
$p(\delta,\tau)$	`f22_struct memo2_pressure(uint comp, double delta, double tau)`	kPa
$u(\delta,\tau)$	`f22_struct memo2_internal_energy(uint comp, double delta, double tau)`	kJ/kg
$s(\delta,\tau)$	`f22_struct memo2_entropy(uint comp, double delta, double tau)`	kJ/kg/K
$h(\delta,\tau)$	`f22_struct memo2_enthalpy(uint comp, double delta, double tau)`	kJ/kg
$g(\delta,\tau)$	`f22_struct memo2_gibbs(uint comp, double delta, double tau)`	kJ/kg
$f(\delta,\tau)$	`f22_struct memo2_helmholtz(uint comp, double delta, double tau)`	kJ/kg
$c_v(\delta,\tau)$	`f22_struct memo2_isochoric_heat_capacity(uint comp, double delta, double tau)`	kJ/kg/K
$c_p(\delta,\tau)$	`f22_struct memo2_isobaric_heat_capacity(uint comp, double delta, double tau)`	kJ/kg/K
$w(\delta,\tau)$	`f22_struct memo2_speed_of_sound(uint comp, double delta, double tau)`	m/s
$v(\delta,\tau)$	`f22_struct memo2_specific_volume(uint comp, double delta, double tau)`	m^3/kg
$\kappa_T(\delta,\tau)$	`f22_struct memo2_isothermal_compressibility(uint comp, double delta, double tau)`	1/MPa
$\lambda(\delta,\tau)$	`f22_struct memo2_thermal_conductivity(uint comp, double delta, double tau)`	mW/M/K
$\mu(\delta,\tau)$	`f22_struct memo2_viscosity(uint comp, double delta, double tau)`	μPa·s
$\sigma(\delta,\tau)$	`f22_struct memo2_surface_tension(uint comp, double delta, double tau)`	mN/m

If a thermodynamic property function is not available, the Helmholtz free energy and derivative functions can be used to calculate it. These functions are listed in Table 3.3.

Table 3.3: Dimensionless Helmholtz Free Energy and Derivative Functions of δ and τ

file	description
$\phi^\circ(\delta,\tau)$	`f22_struct memo2_phi_ideal(uint comp, double delta, double tau)`
$\phi^\circ_\delta(\delta,\tau)$	`f22_struct memo2_phi_ideal_d(uint comp, double delta, double tau)`
$\phi^\circ_\tau(\delta,\tau)$	`f22_struct memo2_phi_ideal_t(uint comp, double delta, double tau)`
$\phi^\circ_{\delta\delta}(\delta,\tau)$	`f22_struct memo2_phi_ideal_dd(uint comp, double delta, double tau))`
$\phi^\circ_{\delta\tau}(\delta,\tau)$	`f22_struct memo2_phi_ideal_dt(uint comp, double delta, double tau)`
$\phi^\circ_{\tau\tau}(\delta,\tau)$	`f22_struct memo2_phi_ideal_tt(uint comp, double delta, double tau)`
$\phi^r(\delta,\tau)$	`f22_struct memo2_phi_resi(uint comp, double delta, double tau)`
$\phi^r_\delta(\delta,\tau)$	`f22_struct memo2_phi_resi_d(uint comp, double delta, double tau)`
$\phi^r_\tau(\delta,\tau)$	`f22_struct memo2_phi_resi_t(uint comp, double delta, double tau)`
$\phi^r_{\delta\delta}(\delta,\tau)$	`f22_struct memo2_phi_resi_dd(uint comp, double delta, double tau))`
$\phi^r_{\delta\tau}(\delta,\tau)$	`f22_struct memo2_phi_resi_dt(uint comp, double delta, double tau)`
$\phi^r_{\tau\tau}(\delta,\tau)$	`f22_struct memo2_phi_resi_tt(uint comp, double delta, double tau)`

3.5 Property Functions of h, p

Property functions of enthalpy and pressure are available and prototyped in `props_hp.h`. These functions are available in single and mixed phase form. The enthalpy argument is the mixed phase enthalpy given in Equation 3.5, where h is enthalpy and x is vapor fraction.

$$h = xh_{\text{vap}} + (1-x)h_{\text{liq}} \tag{3.5}$$

Mixed phase property functions of enthalpy and pressure are provided in Table 3.4. Pressure has units of kPa and enthalpy has units of kJ/kg. The mixed phase properties w, κ_T, λ, and μ are not physically meaningful for mixtures of two phases, but are provided for convenience, since they will be valid for whatever phase exists in the single phase region. While there is only a surface tension in the two-phase region, the function will provide a smooth and continuous result even in the single phase region. As much as possible an effort is made to extend functions smoothly even in regions where they are not valid to minimize difficulty with equation solving. This means post solve checks should be implemented to ensure the final results are valid. In the supercritical region there is only one supercritical phase, so liquid and vapor properties are the same. The vapor fraction in the supercritical region for the purpose of the vapor fraction function is defined to be 0.

Single phase property functions are given in Table 3.5, where {`phase`} is either `vap` or `liq`. In the two-phase region these functions return the single phase properties. For liquid in the vapor region, the functions try to solve for liquid density in the vapor region, this extends the functions smoothly to the critical temperature. Above the critical temperature, the liquid property functions return vapor/supercritical properties. The vapor properties in the liquid region attempt to solve for vapor density extending the properties to the critical pressure. Above the critical pressure, the vapor functions return liquid/supercritical properties.

Table 3.4: Mixed Phase Property Functions of h and p

function	units
`f22_struct memo2_temperature_hp(uint comp, double h, double p)`	K
`f22_struct memo2_vapor_fraction_hp(uint comp, double h, double p)`	–
`f22_struct memo2_internal_energy_hp(uint comp, double h, double p)`	kJ/kg
`f22_struct memo2_entropy_hp(uint comp, double h, double p)`	kJ/kg/K
`f22_struct memo2_gibbs_hp(uint comp, double h, double p)`	kJ/kg
`f22_struct memo2_helmholtz_hp(uint comp, double h, double p)`	kJ/kg
`f22_struct memo2_isochoric_heat_capacity_hp(uint comp, double h, double p)`	kJ/kg/K
`f22_struct memo2_isobaric_heat_capacity_hp(uint comp, double h, double p)`	kJ/kg/K
`f22_struct memo2_speed_of_sound_hp(uint comp, double h, double p)`	m/s
`f22_struct memo2_specific_volume_hp(uint comp, double h, double p)`	m^3/kg
`f22_struct memo2_isothermal_compressibility_hp(uint comp, double h, double p)`	1/MPa
`f22_struct memo2_thermal_conductivity_hp(uint comp, double h, double p)`	mW/M/K
`f22_struct memo2_viscosity_hp(uint comp, double h, double p)`	μPa·s
`f22_struct memo2_surface_tension_hp(uint comp, double h, double p)`	mN/m

Table 3.5: Single Phase Property Functions of h and p

function	units
`f22_struct memo2_enthalpy_{phase}_hp(uint comp, double h, double p)`	–
`f22_struct memo2_internal_energy_{phase}_hp(uint comp, double h, double p)`	kJ/kg
`f22_struct memo2_entropy_{phase}_hp(uint comp, double h, double p)`	kJ/kg/K
`f22_struct memo2_gibbs_{phase}_hp(uint comp, double h, double p)`	kJ/kg
`f22_struct memo2_helmholtz_{phase}_hp(uint comp, double h, double p)`	kJ/kg
`f22_struct memo2_isochoric_heat_capacity_{phase}_hp(uint comp, double h, double p)`	kJ/kg/K
`f22_struct memo2_isobaric_heat_capacity_{phase}_hp(uint comp, double h, double p)`	kJ/kg/K
`f22_struct memo2_speed_of_sound_{phase}_hp(uint comp, double h, double p)`	m/s
`f22_struct memo2_specific_volume_{phase}_hp(uint comp, double h, double p)`	m^3/kg
`f22_struct memo2_isothermal_compressibility_{phase}_hp(uint comp, double h, double p)`	1/MPa
`f22_struct memo2_thermal_conductivity_{phase}_hp(uint comp, double h, double p)`	mW/M/K
`f22_struct memo2_viscosity_{phase}_hp(uint comp, double h, double p)`	μPa·s
`f22_struct memo2_surface_tension_{phase}_hp(uint comp, double h, double p)`	mN/m

3.6 Property Functions of s, p

Property functions of entropy and pressure are available and prototyped in `props_sp.h`. These functions are available in single and mixed phase form. The entropy argument is the mixed phase entropy given in Equation 3.6, where s is entropy and x is vapor fraction.

$$s = x s_{\text{vap}} + (1 - x) s_{\text{liq}} \tag{3.6}$$

Mixed phase property functions of entropy and pressure are provided in Table 3.6. Pressure has units of kPa and entropy has units of kJ/kg/K. The mixed phase properties w, κ_T, λ, and μ are not physically meaningful for mixtures of two phases, but are provided for convenience, since they will be valid for whatever phase exists in the single phase region. While there is only a surface tension in the two-phase region, the function will provide a smooth and continuous result even in the single phase region. As much as possible an effort is made to extend functions smoothly even in regions where they are not valid to minimize difficulty with equation solving. This means post solve checks should be implemented to ensure the final results are valid. In the supercritical region there is only one supercritical phase, so liquid and vapor properties are the same. The vapor fraction in the supercritical region for the purpose of the vapor fraction function is defined to be 0.

Table 3.6: Mixed Phase Property Functions of s and p

function	units
`f22_struct memo2_temperature_sp(uint comp, double s, double p)`	K
`f22_struct memo2_vapor_fraction_sp(uint comp, double s, double p)`	–
`f22_struct memo2_internal_energy_sp(uint comp, double s, double p)`	kJ/kg
`f22_struct memo2_enthalpy_sp(uint comp, double s, double p)`	kJ/kg
`f22_struct memo2_gibbs_sp(uint comp, double s, double p)`	kJ/kg
`f22_struct memo2_helmholtz_sp(uint comp, double s, double p)`	kJ/kg
`f22_struct memo2_isochoric_heat_capacity_sp(uint comp, double s, double p)`	kJ/kg/K
`f22_struct memo2_isobaric_heat_capacity_sp(uint comp, double s, double p)`	kJ/kg/K
`f22_struct memo2_speed_of_sound_sp(uint comp, double s, double p)`	m/s
`f22_struct memo2_specific_volume_sp(uint comp, double s, double p)`	m^3/kg
`f22_struct memo2_isothermal_compressibility_sp(uint comp, double s, double p)`	1/MPa
`f22_struct memo2_thermal_conductivity_sp(uint comp, double s, double p)`	mW/M/K
`f22_struct memo2_viscosity_sp(uint comp, double s, double p)`	μPa·s
`f22_struct memo2_surface_tension_sp(uint comp, double s, double p)`	mN/m

Single phase property functions are given in Table 3.7, where {`phase`} is either `vap` or `liq`. In the two-phase region these functions return the single phase properties. For liquid in the vapor region, the functions try to solve for liquid density in the vapor region, this extends the functions smoothly to the critical temperature. Above the critical temperature, the liquid property functions return vapor/supercritical properties. The vapor properties in the liquid region attempt to solve for vapor density extending the properties to the critical pressure. Above the critical pressure, the vapor functions return liquid/supercritical properties.

3.7 Property Functions of u, p

Property functions of internal energy and pressure are available and prototyped in `props_up.h`. These functions are available in single and mixed phase form. The internal energy argument is the mixed phase internal energy given in Equation 3.7, where u is internal energy and x is vapor fraction.

$$u = xu_{\text{vap}} + (1 - x)u_{\text{liq}} \tag{3.7}$$

Mixed phase property functions of internal energy and pressure are provided in Table 3.8. Pressure has units of kPa and internal energy has units of kJ/kg. The mixed phase properties w, κ_T, λ, and μ are not physically meaningful for mixtures of two phases, but are provided for convenience, since they will be valid for whatever phase exists in the single phase region. While there is only a surface tension in the two-phase region, the function will provide a smooth and continuous result even

Table 3.7: Single Phase Property Functions of s and p

function	units
`f22_struct memo2_enthalpy_{phase}_sp(uint comp, double s, double p)`	–
`f22_struct memo2_internal_energy_{phase}_sp(uint comp, double s, double p)`	kJ/kg
`f22_struct memo2_entropy_{phase}_sp(uint comp, double s, double p)`	kJ/kg/K
`f22_struct memo2_gibbs_{phase}_sp(uint comp, double s, double p)`	kJ/kg
`f22_struct memo2_helmholtz_{phase}_sp(uint comp, double s, double p)`	kJ/kg
`f22_struct memo2_isochoric_heat_capacity_{phase}_sp(uint comp, double s, double p)`	kJ/kg/K
`f22_struct memo2_isobaric_heat_capacity_{phase}_sp(uint comp, double s, double p)`	kJ/kg/K
`f22_struct memo2_speed_of_sound_{phase}_sp(uint comp, double s, double p)`	m/s
`f22_struct memo2_specific_volume_{phase}_sp(uint comp, double s, double p)`	m^3/kg
`f22_struct memo2_isothermal_compressibility_{phase}_sp(uint comp, double s, double p)`	1/MPa
`f22_struct memo2_thermal_conductivity_{phase}_sp(uint comp, double s, double p)`	mW/M/K
`f22_struct memo2_viscosity_{phase}_sp(uint comp, double s, double p)`	μPa·s
`f22_struct memo2_surface_tension_{phase}_sp(uint comp, double s, double p)`	mN/m

in the single phase region. As much as possible an effort is made to extend functions smoothly even in regions where they are not valid to minimize difficulty with equation solving. This means post solve checks should be implemented to ensure the final results are valid. In the supercritical region there is only one supercritical phase, so liquid and vapor properties are the same. The vapor fraction in the supercritical region for the purpose of the vapor fraction function is defined to be 0.

Table 3.8: Mixed Phase Property Functions of u and p

function	units
`f22_struct memo2_temperature_up(uint comp, double u, double p)`	K
`f22_struct memo2_vapor_fraction_up(uint comp, double u, double p)`	–
`f22_struct memo2_entropy_up(uint comp, double u, double p)`	kJ/kg/K
`f22_struct memo2_enthalpy_up(uint comp, double u, double p)`	kJ/kg
`f22_struct memo2_gibbs_up(uint comp, double u, double p)`	kJ/kg
`f22_struct memo2_helmholtz_up(uint comp, double u, double p)`	kJ/kg
`f22_struct memo2_isochoric_heat_capacity_up(uint comp, double u, double p)`	kJ/kg/K
`f22_struct memo2_isobaric_heat_capacity_up(uint comp, double u, double p)`	kJ/kg/K
`f22_struct memo2_speed_of_sound_up(uint comp, double u, double p)`	m/s
`f22_struct memo2_specific_volume_up(uint comp, double u, double p)`	m^3/kg
`f22_struct memo2_isothermal_compressibility_up(uint comp, double u, double p)`	1/MPa
`f22_struct memo2_thermal_conductivity_up(uint comp, double u, double p)`	mW/M/K
`f22_struct memo2_viscosity_up(uint comp, double u, double p)`	μPa·s
`f22_struct memo2_surface_tension_up(uint comp, double u, double p)`	mN/m

Single phase property functions are given in Table 3.9, where {`phase`} is either `vap` or `liq`. In the two-phase region these functions return the single phase properties. For liquid in the vapor region, the functions try to solve for liquid density in the vapor region, this extends the functions smoothly to the critical temperature. Above the critical temperature, the liquid property functions return vapor/supercritical properties. The vapor properties in the liquid region attempt to solve

for vapor density extending the properties to the critical pressure. Above the critical pressure, the vapor functions return liquid/supercritical properties.

Table 3.9: Single Phase Property Functions of u and p

function	units
`f22_struct memo2_enthalpy_{phase}_sp(uint comp, double s, double p)`	–
`f22_struct memo2_internal_energy_{phase}_sp(uint comp, double s, double p)`	kJ/kg
`f22_struct memo2_entropy_{phase}_sp(uint comp, double s, double p)`	kJ/kg/K
`f22_struct memo2_gibbs_{phase}_sp(uint comp, double s, double p)`	kJ/kg
`f22_struct memo2_helmholtz_{phase}_sp(uint comp, double s, double p)`	kJ/kg
`f22_struct memo2_isochoric_heat_capacity_{phase}_sp(uint comp, double s, double p)`	kJ/kg/K
`f22_struct memo2_isobaric_heat_capacity_{phase}_sp(uint comp, double s, double p)`	kJ/kg/K
`f22_struct memo2_speed_of_sound_{phase}_sp(uint comp, double s, double p)`	m/s
`f22_struct memo2_specific_volume_{phase}_sp(uint comp, double s, double p)`	m^3/kg
`f22_struct memo2_isothermal_compressibility_{phase}_sp(uint comp, double s, double p)`	1/MPa
`f22_struct memo2_thermal_conductivity_{phase}_sp(uint comp, double s, double p)`	mW/M/K
`f22_struct memo2_viscosity_{phase}_sp(uint comp, double s, double p)`	μPa·s
`f22_struct memo2_surface_tension_{phase}_sp(uint comp, double s, double p)`	mN/m

3.8 Property Functions of T, p

Properties as a function of temperature with units of K and pressure with units of kPa are only available for single phases, since vapor fraction cannot be determined from temperature and pressure. These function are prototyped in `props_tp.h`. These functions are extended past the saturation curve by solving for liquid density in the vapor region or vapor density in the liquid region as a smooth continuation past the saturation curve. For liquid, past the critical temperature vapor properties are returned for the liquid functions. For vapor past the critical pressure, liquid properties are returned. Available functions are given in Table 3.10 where {`phase`} is either `liq` or `vap`. In the supercritical region, both functions return the same result corresponding to the supercritical fluid.

Table 3.10: Single Phase Property Functions of T and p

function	units
`f22_struct memo2_enthalpy_{phase}_tp(uint comp, double T, double p)`	–
`f22_struct memo2_internal_energy_{phase}_tp(uint comp, double T, double p)`	kJ/kg
`f22_struct memo2_entropy_{phase}_tp(uint comp, double T, double p)`	kJ/kg/K
`f22_struct memo2_gibbs_{phase}_tp(uint comp, double T, double p)`	kJ/kg
`f22_struct memo2_helmholtz_{phase}_tp(uint comp, double T, double p)`	kJ/kg
`f22_struct memo2_isochoric_heat_capacity_{phase}_tp(uint comp, double T, double p)`	kJ/kg/K
`f22_struct memo2_isobaric_heat_capacity_{phase}_tp(uint comp, double T, double p)`	kJ/kg/K
`f22_struct memo2_speed_of_sound_{phase}_tp(uint comp, double T, double p)`	m/s
`f22_struct memo2_specific_volume_{phase}_tp(uint comp, double T, double p)`	m^3/kg
`f22_struct memo2_isothermal_compressibility_{phase}_tp(uint comp, double T, double p)`	1/MPa
`f22_struct memo2_thermal_conductivity_{phase}_tp(uint comp, double T, double p)`	mW/M/K
`f22_struct memo2_viscosity_{phase}_tp(uint comp, double T, double p)`	μPa·s
`f22_struct memo2_surface_tension_{phase}_tp(uint comp, double T, double p)`	mN/m

3.9 Saturated Property Functions

Saturated properties are available as unary functions of temperature or pressure. The saturated property function are prototyped in `sat.h`. The basic saturation curve functions are a function of τ these functions are listed in Table 3.11, where the delta functions return density of liquid or vapor.

Table 3.11: Saturated Property Functions of τ

function	units
`f12_struct sat_p(uint comp, double tau)`	kPa
`f12_struct sat_delta_v(uint comp, double tau)`	–
`f12_struct sat_delta_l(uint comp, double tau)`	–

Saturation property function of temperature in K are given in Table 3.12.

Table 3.12: Saturated Property Functions of T

function	units
`f12_struct sat_p_t(uint comp, double T)`	–
`f12_struct sat_h_liq_t(uint comp, double T)`	kJ/kg
`f12_struct sat_h_vap_t(uint comp, double T)`	kJ/kg
`f12_struct sat_s_liq_t(uint comp, double T)`	kJ/kg/K
`f12_struct sat_s_vap_t(uint comp, double T)`	kJ/kg/K
`f12_struct sat_u_liq_t(uint comp, double T)`	kJ/kg
`f12_struct sat_u_vap_t(uint comp, double T)`	kJ/kg
`f12_struct sat_v_liq_t(uint comp, double T)`	m^3/kg
`f12_struct sat_v_vap_t(uint comp, double T)`	m^3/kg

Saturation property function of pressure in kPa are given in Table 3.13.

Table 3.13: Saturated Property Functions of P

function	units
`f12_struct sat_tau(uint comp, double p)`	–
`f12_struct sat_t(uint comp, double p)`	K
`f12_struct sat_h_liq_p(uint comp, double p)`	kJ/kg
`f12_struct sat_h_vap_p(uint comp, double p)`	kJ/kg
`f12_struct sat_s_liq_p(uint comp, double p)`	kJ/kg/K
`f12_struct sat_s_vap_p(uint comp, double p)`	kJ/kg/K
`f12_struct sat_u_liq_p(uint comp, double p)`	kJ/kg
`f12_struct sat_u_vap_p(uint comp, double p)`	kJ/kg
`f12_struct sat_v_liq_p(uint comp, double p)`	m^3/kg
`f12_struct sat_v_vap_p(uint comp, double p)`	m^3/kg

Chapter 4

AMPL Wrapper Functions

The property functions are available as AMPL user-defined functions, which can be used in AMPL solvers and with Pyomo. This documentation is a work-in-progress.

Chapter 5

Parameter Files

Components can be added or modified by creating or changing parameter files. The expressions for the equation of state and transport properties are not assumed to have a standard fixed form, so AMPL NL files are used to define them. One NL file is required for the equation of state and optional NL files can be provided for transport properties. Other basic parameters are provided as a json file. Pyomo [2] and IDAES are used to generate parameter files, so you should install IDAES from https://github.com/IDAES/idaes-pse.

There is a python module `helmholtz_paramters.py` in the `param_data` directory which can be used to generate parameter files. Common forms of the Helmholtz free energy correlation are also built into the `helmholtz_paramters.py` so often the expression for the equation of state do not need to be written out. Basic parameters fed into the parameter file generation class can be provided as a python dictionary or json file. In this section we'll focus on the json file option.

To generate parameter files, typically a python script is used. The script below shows the simplest script which reads parameters form `r32.json` to generate the parameter files needed by the shared library to calculate properties for R32 as listed in Table 3.1.

```
from helmholtz_parameters import WriteParameters

def main():
    we = WriteParameters(parameters="r32.json")
    we.write()

if __name__ == "__main__":
    main()
```

For examples, see the {comp}.json and {comp}.py files in the `param_data` directory.

5.1 Basic Parameter JSON File

The structure of the most basic parameter file for CO2 is shown below. The `eos`, `aux`, and `transport` sections are omitted from the example for now and will be discussed later. Those sections are optional and can be used to add parameters for the equation of state or transport models. The example show the minimum required parameters.

```
{
    "comp": "co2",
    "basic": {
        "R": 0.1889241,
```

```
        "MW": 44.0098,
        "T_star": 304.1282,
        "rho_star": 467.6,
        "Tc": 304.1282,
        "rhoc": 467.6,
        "Pc": 7377.3,
        "Tt": 216.592,
        "Pt": 517.95,
        "rhot_l": 1178.46,
        "rhot_v": 13.761,
        "P_min": 1e-9,
        "P_max": 5e5,
        "rho_max": 1600.0,
        "T_min": 216,
        "T_max": 1400
    },
    "eos":{},
    "aux":{},
    "transport":{},
}
```

The `comp` field is the name which will be used to read in data for a component. Table 5.1 shows the required parameters in the `basic` section of the json file. As noted in the table some parameters are just used to define approximate bounds for equation solving so exact values are not needed.

Table 5.1: Basic Parameters

quantity	description	units
R	specific gas constant (ideal gas constant/molecular weight)	kJ/kg/K
MW	molecular weight	g/mol
T_star	reducing temperature	K
rho_star	reducing density	kg/m^3
Tc	critical temperature	K
rhoc	critical density	kg/m^3
Pc	critical pressure (recalculated to match Tc and rhoc in generating parameter files)	kPa
Tt	triple point temperature (can be approximate)	K
Pt	triple point pressure (can be approximate)	kPa
rhot_l	triple point liquid density (can be approximate)	kg/m^3
rhot_v	triple point vapor density (can be approximate)	kg/m^3
P_min	minimum pressure	kPa
P_max	maximum pressure	kPa
rho_max	maximum density	kg/m^3
T_min	minimum temperature	K
T_max	maximum temperature	K

5.2 Expressions

Once the basic parameters are entered expressions for the equation of state are required and transport property expressions can optionally be defined. These expressions are written using Pyomo, but the most common expression for the equation of state and surface tension are prepackaged in the `WriteParameters` class. Prepackaged expressions are not yet available for viscosity and thermal conductivity, so they must be written out as Pyomo expressions if needed.

The `WriteParameters` contains four Pyomo models used to generate expressions. Each model has the `delta` and `tau` variables, along with all of the basic parameters as parameters. These models are attributes of a `WriteExpression` object are are shown in Table 5.2. Currently all expressions must contain only the delta and tau variables. For transport properties, the expressions may call ample functions for properties, so it (for example) heat capacity is used in the thermal conductivity correlation, the head capacity ASL user-defined function an be called.

Table 5.2: WriteParameter Object Pyomo Models For Expression

model	description
`model`	model for dimensionless Helmholtz free energy and aux sat density curves
`model_visc`	model viscosity
`model_tcx`	model thermal conductivity
`model_st`	model surface tension

`WriteParameters` have an `add` method that takes a dictionary of pyomo expressions where the expression name is the key. The required expressions are detailed in the next sections. Some expressions are prepackaged and can be generated directly from parameters.

5.2.1 Equation of State

The Helmholtz equation of state uses the Helmholtz free energy as a function of density and temperature (Equation 1.1) split into an ideal and residual part and it derivatives to calculate vapor and liquid thermodynamic properties of a pure substance. To assist with the saturated properties calculation auxiliary curves for saturated liquid and vapor density are also required. Table 5.3 provides a summary of required equation of state expressions.

Likely every form of these expressions are already available prepackaged, so it is unlikely that you will need to type out the expressions explicitly. The next sections summarize the available expressions and required parameters. Expression parameters for the equation of state are defined in the `eos` section of the JSON file. The

Ideal Helmholtz Free Energy

Several canned expressions for the ideal part of the Helmholtz free energy. For the purpose, of the parameter module and not having, better names each expression is just assigned a type number. To select the form to use, add the `phi_ideal_type` to the `eos` section of the JSON file. See below for a truncated example.

```
...
"eos":{
        "phi_ideal_type": 2,
        "phi_residual_type": 3,
...
```

Table 5.3: EoS Expressions

quantity	expression name	description
$\phi^\circ(\delta,\tau)$	`phii`	ideal part of the Helmholtz free energy
$\phi^\circ_\delta(\delta,\tau)$	`phii_d`	$\frac{\partial}{\partial\delta}\phi^\circ(\delta,\tau)$
$\phi^\circ_{\delta\delta}(\delta,\tau)$	`phii_dd`	$\frac{\partial^2}{\partial\delta^2}\phi^\circ(\delta,\tau)$
$\phi^\circ_\tau(\delta,\tau)$	`phii_t`	$\frac{\partial}{\partial\tau}\phi^\circ(\delta,\tau)$
$\phi^\circ_{\tau\tau}(\delta,\tau)$	`phii_tt`	$\frac{\partial^2}{\partial\tau^2}\phi^\circ(\delta,\tau)$
$\phi^\circ_{\delta\tau}(\delta,\tau)$	`phii_dt`	$\frac{\partial^2}{\partial\delta\partial\tau}\phi^\circ(\delta,\tau)$
$\phi^r(\delta,\tau)$	`phir`	residual part of the Helmholtz free energy
$\phi^r_\delta\delta,\tau)$	`phir_d`	$\frac{\partial}{\partial\delta}\phi^r(\delta,\tau)$
$\phi^r_{\delta\delta}(\delta,\tau)$	`phir_dd`	$\frac{\partial^2}{\partial\delta^2}\phi^r(\delta,\tau)$
$\phi^r_\tau(\delta,\tau)$	`phir_t`	$\frac{\partial}{\partial\tau}\phi^r(\delta,\tau)$
$\phi^r_{\tau\tau}(\delta,\tau)$	`phir_tt`	$\frac{\partial^2}{\partial\tau^2}\phi^r(\delta,\tau)$
$\phi^r_{\delta\tau}(\delta,\tau)$	`phir_dt`	$\frac{\partial^2}{\partial\delta\partial\tau}\phi^r(\delta,\tau)$
$\delta^{\mathrm{liq}}_{\mathrm{sat}}(\delta,\tau)$	`delta_l_sat_approx`	approximate saturated liquid δ as a function of τ
$\delta^{\mathrm{vap}}_{\mathrm{sat}}(\tau)$	`delta_v_sat_approx`	approximate saturated vapor δ as a function of τ

The parameters for the expressions are also included in the `eos` section of the parameter file, but the required parameters may differ depending on the expression chosen.

Type 01

The type 1 form of the ideal portion of the Helmholtz free energy is shown in Equation 5.1. This form of the expression requires a dictionary of the n°_i terms as `n0` and the γ° as `g0`. The last term index in the sum (h) is provided as `last_term_ideal`.

$$\log_e\delta + n^\circ_1 + n^\circ_2\tau + n^\circ_3\log_e\tau + \sum_{i=4}^{h} n^\circ_i\log_e\left[1-\exp(-\gamma^\circ_i\tau)\right] \tag{5.1}$$

A truncated example of the parameters for this is shown below.

```
"eos":{
        "n0": {
            "1": -8.258096,
            "2": 6.353098,
            "3": 3.004486,
            "4": 1.160761,
            "5": 2.645151,
            "6": 5.794987,
            "7": 1.129475
        },
        "g0": {
            "4": 2.2718538,
            "5": 11.9144210,
```

```
            "6": 5.1415638,
            "7": 32.7682170
        },
        "last_term_ideal": 7,
        "phi_ideal_type": 1,
...
```

Type 02

The type 2 form of the ideal part of Helmholtz free energy is given in 5.2. The parameters for this are basically the same as type 1 but the `last_term_ideal` is a list with two elements [h_1, h_2].

$$\log_e \delta + n_1^\circ + n_2^\circ \tau + n_3^\circ \log_e \tau + \sum_{i=4}^{h_1} n_i^\circ \tau^{\gamma_i^\circ} + \sum_{i=h_1+1}^{h_2} n_i^\circ \log_e \left[1 - \exp(-\gamma_i^\circ \tau)\right] \tag{5.2}$$

Type 03

The type 2 form of the ideal part of Helmholtz free energy is given in 5.3. The parameters are the same as for type 1.

$$\log_e \delta + n_1^\circ + n_2^\circ \tau + n_3^\circ \log_e \tau + \sum_{i=4}^{h} n_i^\circ \tau^{\gamma_i^\circ} \tag{5.3}$$

5.2.2 Residual Helmholtz Free Energy Forms

Type 01

Equation 5.4 shows the type 1 form. This form requires the parameters n_i as dictionary n, d_i as dictionary `d`, t_i as dictionary `t`, c_i as dictionary `c`, and `last_term_residual` as a list.

$$\phi^r(\delta, \tau) = \sum_{1}^{h_1} n_i \delta^{d_i} \tau^{t_i} + \sum_{h_1+1}^{h_2} n_i \delta^{d_i} \tau^{t_i} \exp(-\delta^{c_i}) \tag{5.4}$$

An example of the EOS is shown below using the type 1 ideal and type 1 residual expressions. Most off the coefficients have been removed for brevity.

```
    "eos": {
        "reference": [
            "Tillner-Roth, R., A. Yokozeki. An International Standard Equation of State for",
            "    Difluoromethane (R-32) for Temperatures from the Triple Point at 136.34 K to",
            "    435 K and Pressures up to 70 MPa, Journal of Physical and Chemical Reference",
            "    Data 26, 1273 (1997); https://doi.org/10.1063/1.556002"
        ],
        "c": {
            "9": 4,
   ...
```

```
            "19": 1
        },
        "d": {
            "1": 1.00,
            ...
            "19": 3.00
        },
        "t": {
            "1": 0.250,
            ...
            "19": 0.500
        },
        "n": {
            "1": 0.1046634e1,
...
            "19": 0.8263017
        },
        "n0": {
            "1": -8.258096,
            ...
            "7": 1.129475
        },
        "g0": {
            "4": 2.2718538,
            ...
            "7": 32.7682170
        },
        "last_term_ideal": 7,
        "last_term_residual": [8, 19],
        "phi_ideal_type": 1,
        "phi_residual_type": 1
    },
```


Type 02

Equation 5.5 shows the type 2 form [5]. This expression takes the same paramters as type 1 plus `g` for γ_i, `e` for ε_i, `a` for α_i and `b` for β_i.

$$\phi^r(\delta,\tau) = \sum_{i=1}^{h_1} n_i \delta^{d_i} \tau^{t_1} + \sum_{i=h_1+1}^{h_2} n_i \delta^{d_i} \tau^{t_1} \exp(-\delta^{c_i}) + \sum_{i=h_2+1}^{h_3} n_i \delta^{d_i} \tau^{t_1} \exp\left[-\alpha_i(\delta-\varepsilon_i)^2 - \beta_i(\tau-\gamma_i)^2\right] \tag{5.5}$$

Type 03

Equation 5.6 shows the type 3 form. This expression uses the same parameters as type 1 plus `b` for b_i.

$$\phi^r(\delta,\tau) = \sum_{i=1}^{h_1} n_i \delta^{d_i} \tau^{t_1} + \sum_{i=h_1+1}^{h_2} n_i \delta^{d_i} \tau^{t_1} \exp(-\delta^{c_i}) \sum_{i=h_2+1}^{h_3} n_i \delta^{d_i} \tau^{t_1} \exp(-\delta^{c_i}) \exp(-\tau^{b_i}) \tag{5.6}$$

Type 04

Equation 5.7 shows the type 4 form. This expression uses the same parameters as type 1.

$$\phi^r(\delta,\tau) = \sum_{i=1}^{h_0} n_i \delta^{d_i} \tau^{t_1} + \sum_{j=1}^{m} \left[\exp(-\delta^j) \sum_{i=h_{j-1}+1}^{h_j} n_i \delta^{d_i} \tau^{t_1} \right] \tag{5.7}$$

5.2.3 Auxiliary Saturated Density

The auxiliary saturated density equations are approximate, and provide a good initial guess when solving the phase equilibrium equations. These expressions go in the `aux` section of the parameter JSON file. The there are two types of expressions used for the approximate saturated density curves.

Type 01

The first type of saturated density expression is given by Equation 5.8.

$$\delta = c + \sum_{i=1}^{h} n_i \left(1 - \frac{T}{T_c}\right)^{t_i} \tag{5.8}$$

Type 02

The first type of saturated density expression is given by Equation 5.9.

$$\delta = c \exp\left[\sum_{i=0}^{h} \left(1 - \frac{T}{T_c}\right)^{t_i} \right] \tag{5.9}$$

Aux Parameter Example

An example "aux" section for R32 is shown below. This shows both types of auxiliary curves.

```
"aux": {
    "reference": [
        "Tillner-Roth, R., A. Yokozeki. An International Standard Equation of State for",
        "    Difluoromethane (R-32) for Temperatures from the Triple Point at 136.34 K to",
        "    435 K and Pressures up to 70 MPa, Journal of Physical and Chemical Reference",
        "    Data 26, 1273 (1997); https://doi.org/10.1063/1.556002"
    ],
    "delta_l_sat_approx": {
        "c": 1,
        "n": {
            "1": 1.024882075471698,
            "2": 3.0578537735849056,
            "3": -1.8337028301886793,
```

```
                "4": 0.865188679245283
            },
            "t": {
                "1": 0.25,
                "2": 0.6666666666666666,
                "3": 1.0,
                "4": 1.6666666666666667
            },
            "type": 1
        },
        "delta_v_sat_approx": {
            "c": 1,
            "n": {
                "1": -1.969,
                "2": -2.0222,
                "3": -6.7409,
                "4": -27.479
            },
            "t": {
                "1": 0.3333333333333333,
                "2": 0.6666666666666666,
                "3": 1.3333333333333333,
                "4": 3.6666666666666665
            },
            "type": 2
        }
    },
```


5.2.4 Transport Properties

The IDAES Helmholtz equation of state functions also support calculation of the transport properties thermal conductivity, viscosity, and surface tension. The transport properties must be expressions of τ and δ. Transport property expressions can call property functions, so if for example you need to use heat capacity in the thermal conductivity calculation, you can call the heat capacity function.

The expected units of measure for the transport properties are given in Table 5.4.

Table 5.4: Units of Measure for Transport Properties

property	expression name	units
thermal conductivity	`thermal_conductivity`	mW/m/K
viscosity	`viscosity`	μPa $\cdot$ s
surface tension	`surface_tension`	mN/m

The built in surface tension surface tension expression should cover most substances, but for viscosity and thermal conductivity general prepackaged expressions are not yet provided. Viscosity and thermal conductivity are usually provided as functions that take a Pyomo model are return an expression.

Viscosity and Thermal Conductivity

The example below for r134a, shows the complete Python for generating R134a parameters. This provides complete examples for viscosity and thermal conductivity.

```
##############################################################################################
#                                                                                            #
# R134A EOS Expressions and Parameters:                                                      #
#                                                                                            #
# Tillner-Roth, R.; Baehr, H.D., An International Standard Formulation for the              #
#    Thermodynamic Properties of 1,1,1,2-Tetrafluoroethane (HFC-134a) for                    #
#    Temperatures from 170 K to 455 K and Pressures up to 70 MPa, J. Phys. Chem.             #
#    Ref. Data, 1994, 23, 5, 657-729, https://doi.org/10.1063/1.555958                      #
#                                                                                            #
# Perkins, R.A.; Laesecke, A.; Howley, J.; Ramires, M.L.V.; Gurova, A.N.; Cusco, L.,         #
#    Experimental thermal conductivity values for the IUPAC round-robin sample of            #
#    1,1,1,2-tetrafluoroethane (R134a), NIST Interagency/Internal Report (NISTIR)            #
#    - 6605, 2000, https://doi.org/10.6028/NIST.IR.6605.                                     #
#                                                                                            #
# Huber, M.L.; Laesecke, A.; Perkins, R.A., Model for the Viscosity and Thermal              #
#    Conductivity of Refrigerants, Including a New Correlation for the Viscosity             #
#    of R134a, Ind. Eng. Chem. Res., 2003, 42, 13, 3163-3178,                                #
#    https://doi.org/10.1021/ie0300880.                                                      #
#                                                                                            #
# Thermal conductivity parameter errata correction from CoolProp parameter file.             #
# lambda^d.g. a1: 8.00982 -> 8.00982e-5                                                      #
#                                                                                            #
# Mulero, A., I. Cachadina, Parra, M., "Recommended Correlations for the                     #
#     Surface Tension of Common Fluids," J. Phys. Chem. Ref. Data 41, 043105                 #
#     (2012).                                                                                #
#                                                                                            #
##############################################################################################

import math
import pyomo.environ as pyo
from idaes.core.util.math import smooth_max
from helmholtz_parameters import WriteParameters


def thermal_conductivity_rule(m):
    a = [
        -1.05248e-2,
        8.00982e-5,
    ]
    b = {
        1: 1.836526,
        2: 5.126143,
        3: -1.436883,
        4: 6.261441e-1,
    }

    T = m.T_star / m.tau
    MW = m.MW / 1000.0  # kg/mol
    rho_star = m.rho_star / MW
    rho = m.delta * rho_star  # mol/m^3
    tau = m.tau
    delta = m.delta
    rho_crit = 5049.886  # mol/m^3
    lamb_red = 2.055e-3  # reducing tc W/m/K
    Tref = 561.411  # reference T K
    k = 1.380649e-23
    Pc = 4.05928  # MPa
    big_gam = 0.0496
    R0 = 1.03
    gamma = 1.239
    qd = 1892020000.0
    xi0 = 1.94e-10  # m
    nu = 0.63

    m.cp = pyo.ExternalFunction(library="", function="cp")
    m.cv = pyo.ExternalFunction(library="", function="cv")
    m.mu = pyo.ExternalFunction(library="", function="mu")
```

```
    m.itc = pyo.ExternalFunction(library="", function="itc")

    drho_dp = m.itc("r134a", delta, m.T_star / T) * rho_star * m.delta
    drho_dp_ref = m.itc("r134a", delta, m.T_star / Tref) * rho_star * m.delta

    deltchi = smooth_max(
        Pc * rho / rho_crit**2 * (drho_dp - drho_dp_ref * Tref / T), 0, eps=1e-8
    )
    xi = xi0 * (deltchi / big_gam) ** (nu / gamma)

    cp = m.cp("r134a", delta, tau)
    cv = m.cv("r134a", delta, tau)
    mu = m.mu("r134a", delta, tau) / 1e6
    y = qd * xi
    kappa_inv = cv / cp
    pi = math.pi
    Omega = 2.0 / pi * ((1.0 - kappa_inv) * pyo.atan(y) + kappa_inv * y)
    Omega0 = (
        2.0
        / pi
        * (1.0 - pyo.exp(-1.0 / (1.0 / y + 1.0 / 3.0 * (y * rho_crit / rho) ** 2)))
    )
    lambda_c = (
        1000 * MW * cp * rho * R0 * k * T / 6.0 / math.pi / mu / xi * (Omega - Omega0)
    )
    lambda_dg = a[0] + a[1] * T
    lambda_r = lamb_red * sum(bi * (rho / rho_crit) ** i for i, bi in b.items())
    return 1000 * (lambda_dg + lambda_r + lambda_c)


def viscosity_rule(m):
    a = [
        0.355404,
        -0.464337,
        0.257353e-1,
    ]
    b = [
        -19.572881,
        219.73999,
        -1015.3226,
        2471.01251,
        -3375.1717,
        2491.6597,
        -787.26086,
        14.085455,
        -0.34664158,
    ]
    te = [
        0.0,
        -0.25,
        -0.50,
        -0.75,
        -1.00,
        -1.25,
        -1.50,
        -2.50,
        -5.50,
    ]
    c = {
        1: -20.6900719,
        2: 0.356029549,
        3: 2.11101816,
        4: 13.9601415,
        5: -4.5643502,
        6: -3.51593275,
        7: 214.76332,
        8: -0.890173375e-1,
        9: 0.100035295,
```

```
        10: 3.163695636,
    }
    T = m.T_star / m.tau
    rho = m.delta * m.rho_star / m.MW * 1000
    M = 102.031
    sigma = 0.46893
    eok = 299.363
    NA = 6.0221408e23
    Ts = T / eok
    vs = pyo.exp(sum(ai * pyo.log(Ts) ** i for i, ai in enumerate(a)))
    Bs = sum(bi * Ts**ti for bi, ti in zip(b, te))
    B = NA * sigma**3 * Bs / 1e9**3
    etas = 0.021357 * pyo.sqrt(M * T) / (sigma**2 * vs)
    tau = T / m.Tc
    delta0 = c[10] / (1 + c[8] * tau + c[9] * tau**2)
    delta = rho / 5017.053
    eta = (
        c[1] * delta
        + (c[2] / tau**6 + c[3] / tau**2 + c[4] / tau**0.5 + c[5] * tau**2)
        * delta**2
        + c[6] * delta**3
        + c[7] / (delta0 - delta)
        - c[7] / delta0
    )
    return etas * (1 + B * rho) + eta


def main():
    we = WriteParameters("r134a.json")
    we.add(
        {
            "viscosity": viscosity_rule,
            "thermal_conductivity": thermal_conductivity_rule,
        }
    )
    we.write()

    print("ASHRAE Offset")
    print(we.calculate_reference_offset(2.79075439914, 1.60488955608, 0, 0))

if __name__ == "__main__":
    main()
```

Surface Tension

There is currently only one built in expression for surface tension.

$$\sigma = \sum_{i=0}^{h} s_i \max\left(1 - \frac{T}{T_c}, 0\right)^{n_i} \tag{5.10}$$

An example surface tension section for R32 is given below.

```
    "transport": {
        "thermal_conductivity": {

        },
        "viscosity": {

        },
```

```
        "surface_tension": {
            "reference": [
                "Mulero, A., I. Cachadina, Parra, M., Recommended Correlations for the",
                "    Surface Tension of Common Fluids, J. Phys. Chem. Ref. Data 41, 043105",
                "     (2012)."
            ],
            "Tc": 351.255,
            "s": {
                "0": 71.47
            },
            "n": {
                "0": 1.246
            },
            "type": 1
        }
    }
```

References

[1] A. Lee, J. H. Ghouse, J. C. Eslick, C. D. Laird, J. D. Siirola, M. A. Zamarripa, D. Gunter, J. H. Shinn, A. W. Dowling, D. Bhattacharyya, L. T. Biegler, A. P. Burgard, and D. C. Miller, "The idaes process modeling framework and model library—flexibility for process simulation and optimization," *Journal of Advanced Manufacturing and Processing*, vol. 3, no. 3, p. e10095, 2021.

[2] M. L. Bynum, G. A. Hackebeil, W. E. Hart, C. D. Laird, B. L. Nicholson, J. D. Siirola, J.-P. Watson, and D. L. Woodruff, *Pyomo–optimization modeling in python*, vol. 67. Springer Science & Business Media, third ed., 2021.

[3] R. Akasaka, "A reliable and useful method to determine the saturation state from helmholtz energy equations of state," *Journal of Thermal Science and Technology*, vol. 3, no. 3, pp. 442–451, 2008.

[4] W. Wagner and A. Pruß, "The iapws formulation 1995 for the thermodynamic properties of ordinary water substance for general and scientific use," *Journal of Physical and Chemical Reference Data*, vol. 31, no. 2, pp. 387–535, 2002.

[5] R. Span and W. Wagner, "A new equation of state for carbon dioxide covering the fluid region from the triple-point temperature to 1100 k at pressures up to 800 mpa," *Journal of Physical and Chemical Reference Data*, vol. 25, 11 1996.