SecondaryCoolantProps

Release 1.1

Matt Mitchell, Edwin Lee

Oct 24, 2022

CONTENTS:

CHAPTER

ONE

FLUID BASE CLASSES

This is a list of the relevant base classes for fluids. If you want to extend, you should inherit one of these.

1.1 Fluid Abstract Base Class

A fluid base class is available for developers to create new derived fluid mixtures:

class scp.base_fluid.**BaseFluid**(*t_min: float*, *t_max: float*, *x: Optional[float] = None*, *x_min: Optional[float] = None*, *x_max: Optional[float] = None*)

Bases: ABC

A fluid base class that provides convenience methods that can be accessed in derived classes.

alpha(*temp: float*) → float

Convenience function for returning the thermal diffusivity by the common shorthand 'alpha'

@param temp: Fluid temperature, in degrees Celsius @return: Returns the thermal diffusivity in [m2/s]

abstract conductivity(*temp: float*) → float

Abstract method; derived classes should override to return the thermal conductivity of that fluid.

@param temp: Fluid temperature, in degrees Celsius @return: Returns the thermal conductivity in [W/m-K]

static conductivity_units() → str

cp(*temp: float*) → float

Convenience function for returning the specific heat by the common shorthand 'cp'

@param temp: Fluid temperature, in degrees Celsius @return: Returns the specific heat in [J/kg-K]

abstract density(*temp: float*) → float

Abstract method; derived classes should override to return the density of that fluid.

@param temp: Fluid temperature, in degrees Celsius @return: Returns the density in [kg/m3]

static density_units() → str

abstract property fluid_name: str

An abstract property that needs to return the fluid name in derived fluid classes

Derived function must be decorated with @property

@return: string name of the fluid

abstract freeze_point(*x: float*) → float

Abstract method; derived classes shoule override the freezing point of that fluid

@param x: Fluid concentration fraction, ranging from 0 to 1 @return Returns the freezing point of the fluid, in Celsius

static freeze_point_units() → str

k(*temp: float*) → float

Convenience function for returning the thermal conductivity by the common shorthand 'k'

@param temp: Fluid temperature, in degrees Celsius @return: Returns the thermal conductivity, in [W/m-K]

mu(*temp: float*) → float

Convenience function for returning the dynamic viscosity by the common letter 'mu'

@param temp: Fluid temperature, in degrees Celsius @return: Returns the dynamic viscosity – which one is mu in [Pa-s]

pr(*temp: float = 0.0*) → float

Convenience function for returning the Prandtl number by the common shorthand 'pr'

@param temp: Fluid temperature, in degrees Celsius @return: Returns the dimensionless Prandtl number

prandtl(*temp: float*) → float

Returns the Prandtl number for this fluid

@param temp: Fluid temperature, in degrees Celsius @return: Returns the dimensionless Prandtl number

static prandtl_units() → str

rho(*temp: float*) → float

Convenience function for returning the density by the common shorthand 'rho'

@param temp: Fluid temperature, in degrees Celsius @return: Returns the density, in [kg/m3]

abstract specific_heat(*temp: float*) → float

Abstract method; derived classes should override to return the specific heat of that fluid.

@param temp: Fluid temperature, in degrees Celsius @return: Returns the specific heat in [J/kg-K]

static specific_heat_units() → str

thermal_diffusivity(*temp: float*) → float

Returns the thermal diffusivity for this fluid

@param temp: Fluid temperature, in degrees Celsius @return: Returns the thermal diffusivity in [m2/s]

static thermal_diffusivity_units() → str

abstract viscosity(*temp: float*) → float

Abstract method; derived classes should override to return the dynamic viscosity of that fluid.

@param temp: Fluid temperature, in degrees Celsius @return: Returns the dynamic viscosity in [Pa-s]

static viscosity_units() → str

1.2 Melinder Fluid Abstract Base Class

A Melinder fluid base class is available for developers to create new derived fluid mixtures:

class scp.base_melinder.**BaseMelinder**(*t_min: float*, *t_max: float*, *x: float*, *x_min: float*, *x_max: float*)

Bases: BaseFluid

A base class for Melinder fluids that provides convenience methods that can be accessed in derrived classes.

Melinder, Å. 2010. Properties of Secondary Working Fluids for Indirect Systems. 2nd ed. International Institute of Refrigeration.

abstract coefficient_conductivity() → Tuple

Abstract method; derived classes should override to return the coefficient matrix for conductivity.

abstract coefficient_density() → Tuple

Abstract method; derived classes should override to return the coefficient matrix for density.

abstract coefficient_freezing() → Tuple

Abstract method; derived classes should override to return the coefficient matrix for freezing point.

abstract coefficient_specific_heat() → Tuple

Abstract method; derived classes should override to return the coefficient matrix for specific heat.

abstract coefficient_viscosity() → Tuple

Abstract method; derived classes should override to return the coefficient matrix for viscosity.

conductivity(*temp: float*) → float

Calculates the thermal conductivity of the mixture

@param temp: Fluid temperature, in degrees Celsius @return: Thermal conductivity, in W/m-K

density(*temp: float*) → float

Calculates the density of the mixture

@param temp: Fluid temperature, in degrees Celsius @return: Density, in kg/m3

freeze_point(*x: float*) → float

Calculate the freezing point temperature of the mixture

@param x: Concentration fraction @return Freezing point temperature, in Celsius

specific_heat(*temp: float*) → float

Calculates the specific heat of the mixture

@param temp: Fluid temperature, in degrees Celsius @return: Specific heat, in J/kg-K

viscosity(*temp: float*) → float

Calculate the dynamic viscosity of the mixture

@param temp: Fluid temperature, in degrees Celsius @return: Dynamic viscosity, in N/m2-s, or Pa-s

CHAPTER

TWO

COMMAND LINE USAGE

This library comes with a command line interface. Once this library is pip installed, a new binary executable will be available with the name `scprop`. The command has a help argument with output similar to this (execute manually to verify latest syntax):

```
$ scprop --help
Usage: scprop [OPTIONS]

Options:
  -f, --fluid [water|ethyl_alcohol|ethylene_glycol|methyl_alcohol|propylene_glycol]
                                  Which fluid to use?  [required]
  -x, --concentration FLOAT RANGE
                                  Mixture concentration fraction. Default 0.0.
                                  [0.0<=x<=1.0]
  -p, --property [viscosity|specific_heat|density|conductivity|prandtl|thermal_
↪diffusivity|freeze_point]
                                  Which fluid property to evaluate.
                                  [required]
  -t, --temperature FLOAT         Fluid temperature, in degrees Celsius.
                                  [required]
  -q, --quick                     Just report the value, good for scripts
                                  [default: False]
  --help                          Show this message and exit.
```

Some example usages:

```
$ scprop --fluid water --property density --temperature 25
Fluid:    water
Property: density
Value:    997.0448954179155
Units:    [kg/m3]

$ scprop --fluid water --property density --temperature 25 --quick
997.0448954179155

$ scprop --fluid ethylene_glycol --concentration 0.3 --property specific_heat --
↪temperature 40
Fluid:    ethylene_glycol
Property: specific_heat
Value:    3775.3537088494118
Units:    [J/kg-K]
```

Note that the fluid argument must be from the list of supported fluids, and the property must be from the list of supported properties. Both of these are available from the help message. For glycol mixtures, the glycol concentration is a decimal value. The temperature input value is in Celsius.

By default the output is a nice summary, but there is a quick option that just reports the value. The quick value is a nice integration into some workflows that would require the CLI.

For interactive bash sessions, there is a tab-completion capability available. Simply add the following line to the `.bashrc` file, or execute it manually in the current session to apply it temporarily:

```
eval "$(_SCPROP_COMPLETE=bash_source scprop)"
```

Enabling this will turn on tab completion for the different argument names, as well as the supported fluid and property names.

CHAPTER

THREE

PREFERRED PROGRAMMATIC USAGE

For programmatic usage, the preferred approach is for users to directly consume the fluid classes directly in your applications. Applications of using the CLI approach with `scprop` should be limited to that use case.

An example usage:

```
from scp.water import Water

class MyClass:
    def __init__(self):
        self.my_fluid = Water()

    def do_something(self):
        # do some calculations
        # ...

        # get fluid properties
        temp = 20  # Celsius
        visc = self.my_fluid.viscosity(temp)
        dens = self.my_fluid.density(temp)

        # continue
        # ...
```

Other fluids can also be imported and applied in a similar fashion.

Other examples:

```
from scp.ethyl_alcohol import EthylAlcohol
from scp.ethylene_glycol imoprt EthyleneGlycol
from scp.methyl_alcohol import MethylAlcohol
from scp.propylene_glycol import PropyleneGycol
```

CHAPTER

FOUR

SUPPORTED FLUIDS

Here are the currently supported fluids.

4.1 Ethyl Alcohol (Ethylene)

Provides fluid properties for aqueous mixtures of Ethylene for temperatures <= 40 C, with concentrations from 0-0.6.

Example:

```
from scp.ethyl_alcohol import MethylAlcohol

if __name__ == "__main__":
    x_fraction = 0.2  # concentration fraction
    ea = EthylAlcohol(x_fraction)

    temp = 10  # Celsius
    print(ea.viscosity(temp))
```

class scp.ethyl_alcohol.**EthylAlcohol**(*x: float*)

Bases: BaseMelinder

A derived fluid class for ethylene glycol and water mixtures

coefficient_conductivity() → Tuple

Abstract method; derived classes should override to return the coefficient matrix for conductivity.

coefficient_density() → Tuple

Abstract method; derived classes should override to return the coefficient matrix for density.

coefficient_freezing() → Tuple

Abstract method; derived classes should override to return the coefficient matrix for freezing point.

coefficient_specific_heat() → Tuple

Abstract method; derived classes should override to return the coefficient matrix for specific heat.

coefficient_viscosity() → Tuple

Abstract method; derived classes should override to return the coefficient matrix for viscosity.

property fluid_name: str

Returns a descriptive title for this fluid @return: "EthylAlcohol"

4.2 Ethylene Glycol

Provides fluid properties for aqueous mixtures of Ethylene Glycol for temperatures <= 100 C, with concentrations from 0-0.6.

Example:

```
from scp.ethylene_glycol import EthyleneGlycol

if __name__ == "__main__":
    x_fraction = 0.2  # concentration fraction
    eg = EthyleneGlycol(x_fraction)

    temp = 10  # Celsius
    print(eg.viscosity(temp))
```

class scp.ethylene_glycol.**EthyleneGlycol**(*x: float*)

Bases: BaseMelinder

A derived fluid class for ethylene glycol and water mixtures

coefficient_conductivity() → Tuple

Abstract method; derived classes should override to return the coefficient matrix for conductivity.

coefficient_density() → Tuple

Abstract method; derived classes should override to return the coefficient matrix for density.

coefficient_freezing() → Tuple

Abstract method; derived classes should override to return the coefficient matrix for freezing point.

coefficient_specific_heat() → Tuple

Abstract method; derived classes should override to return the coefficient matrix for specific heat.

coefficient_viscosity() → Tuple

Abstract method; derived classes should override to return the coefficient matrix for viscosity.

property fluid_name: str

Returns a descriptive title for this fluid @return: "EthyleneGlycol"

4.3 Methyl Alcohol (Methylene)

Provides fluid properties for aqueous mixtures of Methylene for temperatures <= 40 C, with concentrations from 0-0.6.

Example:

```
from scp.methyl_alcohol import MethylAlcohol

if __name__ == "__main__":
    x_fraction = 0.2  # concentration fraction
    ma = MethylAlcohol(x_fraction)

    temp = 10  # Celsius
    print(ma.viscosity(temp))
```

class scp.methyl_alcohol.**MethylAlcohol**(*x: float*)

Bases: BaseMelinder

A derived fluid class for methyl alcohol and water mixtures

coefficient_conductivity() → Tuple

Abstract method; derived classes should override to return the coefficient matrix for conductivity.

coefficient_density() → Tuple

Abstract method; derived classes should override to return the coefficient matrix for density.

coefficient_freezing() → Tuple

Abstract method; derived classes should override to return the coefficient matrix for freezing point.

coefficient_specific_heat() → Tuple

Abstract method; derived classes should override to return the coefficient matrix for specific heat.

coefficient_viscosity() → Tuple

Abstract method; derived classes should override to return the coefficient matrix for viscosity.

property fluid_name: str

Returns a descriptive title for this fluid @return: "MethylAlcohol"

4.4 Propylene Glycol

Provides fluid properties for aqueous mixtures of Propylene Gycol for temperatures <=100 C, with concentrations from 0-0.6.

Example:

```
from scp.propylene_glycol import PropyleneGycol

if __name__ == "__main__":
    x_fraction = 0.2  # concentration fraction
    pg = PropyleneGycol(x_fraction)

    temp = 10  # Celsius
    print(pg.viscosity(temp))
```

class scp.propylene_glycol.**PropyleneGlycol**(*x: float*)

Bases: BaseMelinder

A derived fluid class for propylene glycol and water mixtures

coefficient_conductivity() → Tuple

Abstract method; derived classes should override to return the coefficient matrix for conductivity.

coefficient_density() → Tuple

Abstract method; derived classes should override to return the coefficient matrix for density.

coefficient_freezing() → Tuple

Abstract method; derived classes should override to return the coefficient matrix for freezing point.

coefficient_specific_heat() → Tuple

Abstract method; derived classes should override to return the coefficient matrix for specific heat.

coefficient_viscosity() → Tuple

Abstract method; derived classes should override to return the coefficient matrix for viscosity.

property fluid_name: str

Returns a descriptive title for this fluid @return: "PropyleneGlycol"

4.5 Water

Provides fluid properties for liquid water between 0-100 C.

Example:

```
from scp.water import Water

if __name__ == "__main__":
    water = Water()
    temp = 10  # Celsius
    print(water.viscosity(temp))
```

class scp.water.**Water**

Bases: BaseFluid

conductivity(*temp: float*) → float

Returns the fluid thermal conductivity for this derived fluid.

Thermal conductivity equation from linear least-squares fit to data in CRC Handbook (op.cit.), page E-11

@param temp: Fluid temperature, in degrees Celsius @return: Thermal conductivity, in [W/m-K]

density(*temp: float*) → float

Returns the fluid density for this derived fluid.

Density eq. for water at 1 atm., from CRC Handbook of Chem. & Phys., 61st Edition (1980-1981), p. F-6.

@param temp: Fluid temperature, in degrees Celsius @return: Density, in [kg/m3]

property fluid_name: str

Returns the fluid name for this derived fluid. @return: "Water"

freeze_point(*_=None*) → float

Returns the freezing point temperature of water

@param _: Unused variable @return Freezing point temperature, in Celsius

specific_heat(*temp: float*) → float

Returns the fluid specific heat.

Specific heat of water at 1 atmosphere, 0 to 100 C. Equation from linear least-squares regression of data from CRC Handbook (op.cit.) page D-174

@param temp: Fluid temperature, in degrees Celsius @return: Specific heat, in [J/kg-K]

viscosity(*temp: float*) → float

Returns the viscosity of water.

@param temp: Fluid temperature, in degrees Celsius @return: The dynamic viscosity of water in [Pa-s]

CHAPTER

FIVE

INDICES AND TABLES

- genindex
- modindex
- search