Documentation

Overview:

Simulator is the Openmodelica library created at FOSEE, IITB. The main objective of this is to create general purpose steady state chemical engineering simulator using OpenModelica. You can download this library from- https://github.com/FOSSEE/OMChemSim

This documentation discusses structure of this library and also gives details about all models.

Table of Contents

Structure:
Simulator is mainly categorized in four sections:

1. Files
2. Streams
3. Unit_Operations
4. Test

1. Files:

This section contains all files other than unit operations and Streams. These files are further subcategorized according to use.

2. Streams:

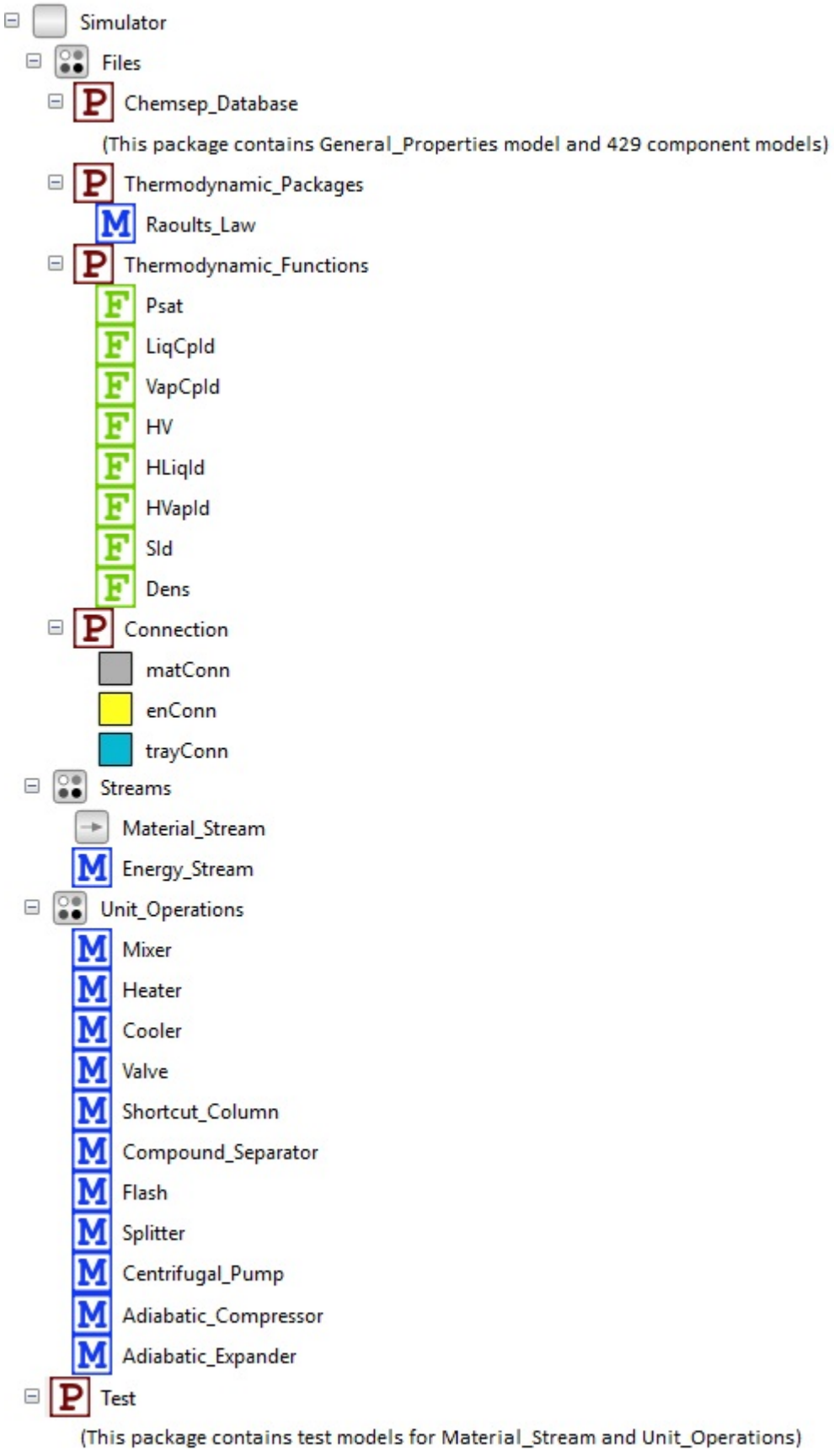
This section contains Material_Stream and Energy_Stream.

3. Unit_Operations:

This section contains all Unit Operations.

4. Test:

This section contains the test models of Material_Stream and Unit_Operations. Models in Files, Streams and Unit_Operations cannot be used individually. For creating simulation we need to follow some steps and create new models. Examples of such models are created in Test.


Simulator
Files
Chemsep_Database
(This package contains General_Properties model and 429 component models)
Thermodynamic_Packages
Raoults_Law
Thermodynamic_Functions
Psat
LiqCpId
VapCpId
HV
HLiqId
HVapId
Sld
Dens
Connection
matConn
enConn
trayConn
Streams
Material_Stream
Energy_Stream
Unit_Operations
Mixer
Heater
Cooler
Valve
Shortcut_Column
Compound_Separator
Flash
Splitter
Centrifugal_Pump
Adiabatic_Compressor
Adiabatic_Expander
Test
(This package contains test models for Material_Stream and Unit_Operations)

1. Files

1.1 Chemsep_Database:

Chemsep_Database is the collection of pure component properties and coefficients of empirical equations for calculating some properties. This package is created from chemsep1.xml file from dwsim3 using python script. Chemsep_Database contain one General_Properties model and 429 component models.

We can import Chemsep_Database as

```
import database_name = Simulator.Files.Chemsep_Database;
```

We can instantiate components as

```
parameter database_name.component_name_1 comp1;
parameter database_name.component_name_2 comp2;
```

Here `database_name`, `comp1` and `comp2` are arbitrary names given to instantiated database and components. `component_name_1` and `component_name_2` are model names of particular components in Chemsep_Database.

In Chemsep_Database, all component models are subclasses of General_Properties model. Hence we can instance General_Properties for creating components array.

```
parameter database_name.General_Properties comp_array[2] = {comp1, comp2};
```

`comp_array` is an array of 2 elements where each element of an array is one component model.

Note that all the components and component array are declared as `parameter`, Otherwise property variables of these models would also be added as variables in simulation.

1.2 Thermodynamic_Packages:

This section contains all thermodynamic packages available. User needs to select proper package for simulation and extend it with material stream to create composite material stream. Some unit operations also needs thermodynamic package extension.

For using these packages we need to write following line for extending required thermodynamic package.

```
extends Simulator.Files.Thermodynamic_Packages.package_name;
```

`package_name` is the model name in Thermodynamic_Packages.

1.2.1 Raoults_Law:

For using Raoults_Law we need to write

```
extends Simulator.Files.Thermodynamic_Packages.Raoults_Law;
```

1.3 Thermodynamic_Functions:

This section contains different thermodynamic functions. These function calculates pure components thermodynmic properties using Chemsep_Database and empirical equations.

1.3.1 Psat:

This function calculates temperature dependent saturated pressure for given component.

```
vap_press = Simulator.Files.Thermodynamic_Functions.Psat(comp1.VP, T);
```

`vap_press` is arbitrary variable for the vapor pressure of comp1 at temperature T. `comp1.VP` and `T` are input for `Psat` function. `comp1.VP` is VP array of comp1 from Chemsep_Database and T is temperature in K.

1.3.2 LiqCpId:

This function calculates liquid phase specific heat of pure component at given temperature.

```
L_Cp = Simulator.Files.Thermodynamic_Functions.LiqCpId(comp1.LiqCp, T);
```

`L_Cp` is the liquid phase specific heat of comp1 at temperature T. `comp1.LiqCp` and `T` are input for `LiqCpId` function. `comp1.LiqCp` is LiqCp array of comp1 from Chemsep_Database and T is temperature in K.

1.3.3 VapCpId:

This function calculates vapor phase specific heat of pure component at given temperature

```
V_Cp = Simulator.Files.Thermodynamic_Functions.VapCpId(comp1.VapCp, T);
```

`V_Cp` is the vapor phase specific heat of comp1 at temperature T. `comp1.VapCp` and `T` are input for `VapCpId` function. `comp1.VapCp` is VapCp array of comp1 from Chemsep_Database and T is temperature in K.

1.3.4 HV:

This function calculates heat of vaporization of pure component at given temperature.

```
H_Vap = Simulator.Files.Thermodynamic_Functions.HV(comp1.HOV, comp1.Tc, T);
```

`H_Vap` is the heat of vaporization of comp1 at temperature T. `comp1.HOV`, `comp1.Tc` and `T` are input for `HV` function. `comp1.HOV` is HOV array and `comp1.Tc` is critical point of comp1 from Chemsep_Database and T is temperature in K.

1.3.5 HLiqId:

This function calculates ideal liquid phase enthalpy for pure component at given temperature.

```
liq_enth = Simulator.Files.Thermodynamic_Functions.HliqId(comp1.SH, comp1.VapCp, comp1.HOV, comp1.Tc, T);
```

`liq_enth` is liquid phase enthalpy of comp1 at temperature T. `comp1.SH, comp1.VapCp, comp1.HOV, comp1.Tc` are values taken from Chemsep_Database and T is temperature in K.

1.3.6 HVapId:

This function calculates ideal vapor phase enthalpy for pure component at given temperature.

```
vap_enth = Simulator.Files.Thermodynamic_Functions.HvapId(comp1.SH, comp1.VapCp, comp1.HOV, comp1.Tc, T);
```

`vap_enth` is vapor phase enthalpy of comp1 at temperature T. `comp1.SH, comp1.VapCp, comp1.HOV, comp1.Tc` are values taken from Chemsep_Database and T is temperature in K.

1.3.7 SId:

This function calculates ideal liquid phase entropy and ideal vapor phase entropy for given T and P.

```
(liq_entr, vap_entr) = Thermodynamic_Functions.SId(comp1.AS, comp1.VapCp, comp1.HOV, comp1.Tb, comp1.Tc, T, P, liq_frac, vap_frac);
```

Output of this function is array of two elements. `liq_entr` is liquid phase entropy and `vap_entr` is vapor phase entropy. `comp1.AS, comp1.VapCp, comp1.HOV, comp1.Tb, comp1.Tc` are values taken from database. T is temperature in K. P is pressure in Pa. `liq_frac` and `vap_frac` are liquid phase mole fraction of comp1 and vapor phase mole fraction of comp1 respectively.

1.3.8 Dens:

This function calculates molar density of pure component.

```
Den = Simulator.Files.Thermodynamic_Functions.Dens(comp1.LiqDen, comp1.Tc, T, P);
```

`Den` is the molar density of `comp1` at `T` and `P`. `comp1.LiqDen, comp1.Tc` are taken from database. T is temperature in K. P is pressure in Pa.

1.4 Connection:

This section contains connectors which are used in Streams and Unit_Operations. Connectors are used for making process of creating simulation easier. Overall equations and variables count increases because of connectors hence we have created connectors with minimum number of variables possible with which simulation can be created.

1.4.1 matConn:

This connector is used in Material_Stream and all Unit_Operations. This connector is created for establishing connection between composite material stream and unit operations. While creating new model this connector can be instantiated by dragging and dropping. After dragging and dropping following code is generated.

```
Simulator.Files.Connection.matConn matConn1 annotation(.....);
```

For completely defineing connector, we need to assign value to parameter connNOC. It can be done as follows,

```
Simulator.Files.Connection.matConn conn_name(connNOC =
no_of_comp) annotation(.....);
```

`conn_name` is arbitrary name assigned to connector. `no_of_comp` is variable declared for number of components value in model.

1.4.2 enConn:

This connector is used in Energy_Stream and for showing energy connections in some Unit_Operations. This connector is created for conecting Unit_Operations with Energy_Stream. This connector can be instantiated by dragging and dropping. After dragging and dropping following code is generated.

```
Simulator.Files.Connection.enConn conn_name annotation(.....);
```

1.4.3 trayConn:

This connector is created for establishing connection beween trays for rigorous column models. This is also instantiated by dragging and dropping and value for connNOC needs to be specified as in matConn.Complete code looks like following,

```
Simulator.Files.Connection.trayConn conn_name(connNOC =
no_of_comp) annotation(.....);
```

1.5 Models:

This section contains useful models for used in Unit_Operations.

1.5.2 Flash:

This model is useful for if VLE calculation is requried in any unit operation. This needs to be extended. NOC and comp variables needs to be defined in model. Code for this looks like follows:

```
parameter Integer NOC = no_of_comp;
parameter database_name.General_Properties comp_array[NOC] =
comp_array;
extends Simulator.Files.Models.Flash;
extends Simulator.Files.Thermodynamic_Packages.package_name;
```

`no_of_comp` is value of number of components or variable to which value of number of components is assigned. `comp_array` is component array.

Mixture mole fraction and molar flow has to be specified. Other two variables are selected according to mode. These variables are written in equation section.

```
equation
    mode_var_1 = value_of_mv1;
    mode_var_2 = value_of_mv2;
    compMolFrac[1, :] = mole_frac_array[no_of_comp];
    totMolFlo[1] = value_of_mol_flo;
```

`mode_var_1` and `mode_var_2` are mode variables. `CompMolFrac[1,:]` is mixture mole fraction array of all components. `TotMolFlo[1]` is the miture molar flow. Right hand side of these equations are values of respective variables.

Material_Stream support five modes:

1. PT: values of Pressure `(P)` and Temperature `(T)` are given
2. PH: value of Pressure `(P)` and Mixture molar enthalpy `(phasMolEnth[1])` are given
3. PVF: values of Pressure `(P)` and Vapor phase mole fraction `(vapPhasMolFrac)` are given.
4. TVF: values of Temperature `(T)` and Vapor phase mole fraction `(vapPhasMolFrac)` are given.
5. PS: values of Pressure `(P)` and Mixture molar entropy `(phasMolEntr[1])` are given.

2. Streams

2.1 Material_Stream:

This is one of the important model for creating simulations. This model needs to be extended with required thermodynamic package to create useable model of material stream. Value for parameters NOC and comp in Material_Stream has to be specified in bracket. Code looks like below

```
    extends Simulator.Streams.Material_Stream(NOC = no_of_comp, comp
= comp_array);
    extends Simulator.Files.Thermodynamic_Packages.package_name;
```

`no_of_comp` is value of number of components or variable to which value of number of components is assigned. `comp_array` is component array.

Mixture mole fraction and molar flow has to be specified. Other two variables are selected according to mode. These variables are written in equation section.

```
 equation
     mode_var_1 = value_of_mv1;
     mode_var_2 = value_of_mv2;
     compMolFrac[1, :] = mole_frac_array[no_of_comp];
     totMolFlo[1] = value_of_mol_flo;
```

`mode_var_1` and `mode_var_2` are mode variables. `CompMolFrac[1,:]` is mixture mole fraction array of all components. `TotMolFlo[1]` is the miture molar flow. Right hand side of these equations are values of respective variables.

Material_Stream support five modes:

PT: values of Pressure `(P)` and Temperature `(T)` are given

1. PH: value of Pressure `(P)` and Mixture molar enthalpy `(phasMolEnth[1])` are given
2. PVF: values of Pressure `(P)` and Vapor phase mole fraction `(vapPhasMolFrac)` are given.
3. TVF: values of Temperature `(T)` and Vapor phase mole fraction `(vapPhasMolFrac)` are given.
4. PS: values of Pressure `(P)` and Mixture molar entropy `(phasMolEntr[1])` are given.

Examples msTP, msPH, msPVF, msTVF, msPS, msTPbbp are available in Test section for different modes.

2.1.a Composite Material Stream:

When we extend material stream and assign values to its variables in one model, it cannot be used as general purpose material stream in simulations, hence we introduce composite material stream. Composite material stream is the model in which Material_Stream and thermodyanamic package is extended. Example is shown below;

```
model model_name
     extends Simulator.Streams.Material_Stream;
```

```
        extends
Simulator.Files.Thermodynamic_Packages.package_name;
    end model_name;
```

We can instance this composite material stream any number of times by dragging and dropping in one simulation. Parameters of material stream has to be specified while creating its instance. Final code looks like:

```
    model_name.model_name1(NOC = no_of_comp, comp =
comp_array)annotation(.....);
    model_name.model_name2(NOC = no_of_comp, comp =
comp_array)annotation(.....);
```

Composite material stream is connected to unit operation as input and output of that unit operation. Values of variables in composite material streams are also assigned in equation section.

```
    equation
        model_name1.mode_var_1 = value_of_mv1;
        model_name1.mode_var_2 = value_of_mv2;
        model_name1.compMolFrac[1, :] =
mole_frac_array[no_of_comp];
        model_name1.totMolFlo[1] = value_of_mol_flo;
```

`mode_var_1` and `mode_var_2` are mode variables of `model_name1`. `CompMolFrac[1,:]` is mixture mole fraction array of all components of `model_name1`. `TotMolFlo[1]` is the mixture molar flow of `model_name1`. Right hand side of these equations are values of respective variables.

2.2 Energy_Stream:

This model is used for showing energy connections in flowsheet. This model can be instantiated by dragging and dropping. The code looks like follows:

```
    Simulator.Streams.Energy_Stream energy_Stream1 annotation(....);
```

3. Unit_Operations:

This section contains all the unit operations available for simulation. Some of these unit operations can be instantiated directly, some of them needs to be extended with thermodynamic packages.

3.1 Mixer:

This is simulation of Steady state mixer. This model can have any number of input streams and one output stream. Mixer can be instantiated by dragging and dropping. Values of mixer parameters has to be specified while instantiation. Final code looks like follows:

```
Simulator.Unit_Operations.Mixer mixer1(NOC = no_of_comp , NI =
no_of_inlets , comp = comp_array, outPress = "selected_mode" )
annotation( ...);
```

`no_of_comp` is number of components. `NI` is number of inlet streams of mixer, `comp` is component array and `outPress` is mode for outlet pressure calculation. Since `outPress` is defined as string, mode needs to be given in double quotes("").
Modes available for `outPress` are:

1. `"Inlet_Minimum"`: outlet pressure is minimum of pressures in all inlet streams.
2. `"Inlet_Average"`: outlet pressure is average of pressures in all inlet streams.
3. `"Inlet_Maximum"`: outlet pressure is maximum of pressures in all inlet streams.

Composite material stream needs to be created and instatiated as inputs and output of the mixer. Output of mixer can be connected using diagram view to instantiated outlet material stream. Inlet connectors of mixer are defined as an array, which are not available in diagram view for connection, hence we need to write connect equations for connecting inlet material streams. Connect equations are written in equation section and they look like follows:

```
connect(mixer1.outlet, cmpst_strmNI+1.inlet) annotation( ...);
connect(cmpst_strm1.outlet, mixer1.inlet[1]);
connect(cmpst_strm2.outlet, mixer1.inlet[2]);
                                .
                                .
                                .
                                .
connect(cmpst_strmNI.outlet, mixer1.inlet[NI]);
```

Here first equation is automatically generated because of connection created using diagram view. Other equations needs to be written manually in text view.

`cmpst_strm` is created composite material stream. `cmpst_strmNI+1` is output stream. `cmpst_strm1.outlet, cmpst_strm2.outlet.... cmpst_strmNI.outlet` are outlet connectors of instatiated composite materiatl streams. `mixer1.inlet[1], mixer1.inlet[2]... mixer1.inlet[NI]` are inlet connectors of instantiated mixer model.

values of `compMolFrac[1, :], totMolFlo[1]` and two mode variables for all inlet streams needs to be assigned in equation section. Other variables are calculated in simulation. Code looks like follows:

```
     cmpst_strm1.mode_var_1 = value_of_mv1_of_strm1;
     cmpst_strm1.mode_var_2 = value_of_mv2_of_strm1;
     cmpst_strm1.compMolFrac[1, :] =
mole_frac_array_of_strm1[no_of_comp];
     cmpst_strm1.totMolFlo[1] = value_of_mol_flo_of_strm1;

     cmpst_strm2.mode_var_1 = value_of_mv1_of_strm2;
     cmpst_strm2.mode_var_2 = value_of_mv2_of_strm2;
     cmpst_strm2.compMolFrac[1, :] =
mole_frac_array_of_strm2[no_of_comp];
     cmpst_strm2.totMolFlo[1] = value_of_mol_flo_of_strm2;
                                    .
                                    .
                                    .
     cmpst_strmNI.mode_var_1 = value_of_mv1_of_strmNI;
     cmpst_strmNI.mode_var_2 = value_of_mv2_of_strmNI;
     cmpst_strmNI.compMolFrac[1, :] =
mole_frac_array_of_strmNI[no_of_comp];
     cmpst_strmNI.totMolFlo[1] = value_of_mol_flo_of_strmNI;
```

Left hand side of equations are variables of inlet streams and right hand side are their values. Test model mix1 is available in Test section.

3.2 Heater:

This is simulation of stream heater. This has one input material stream, one output material stream and one energy input. Heater can be instantiated by dragging and dropping. Values of heater parameters has to be specified while instantiation. Final code looks like follows:

```
     Simulator.Unit_Operations.Heater heater1(pressDrop =
value_of_pressDrop, eff = value_of_eff, NOC = no_of_comp, comp =
comp_array) annotation( ...);
```

`pressDrop` is the pressure drop and `eff` is the efficiency.

Composite material stream needs to be created and instanced for input and output of heater. One Energy_Stream also instantiated. All connections can be made in digram view.

Values for variables of input stream has to be given and value of one variable of heater also needs to be specified according to mode. Value for heater variable can be specified as below:

```
     heater1.mode_var = value_of_mode_var ;
```

Heater model supports four modes:

1. Heat added: value of heat added(`heatAdd`) is specified in Joules.
2. Outlet temperature: value of outlet temperature(`outT`) is specified in K.
3. Energy Stream: value of enFlo variable from Energy_Stream(`Energy_Stream_name.enFlo`) is specified in Joules.

4. Phase Molar Fraction: value of Vapor phase molar fraction of outlet(`outVapPhasMolFrac`) is specified.
5. Temperature increase: value of temperature change(`tempInc`) is specified.

Example heater1 is available in Test section for Heater.

3.3 Cooler:

This is simulation of stream cooler. This has one input material stream, one output material stream and one energy ouput. Cooler can be instantiated by dragging and dropping. Values of cooler parameters has to be specified while instantiation. Final code looks like follows:

```
Simulator.Unit_Operations.Cooler cooler1(pressDrop =
value_of_pressDrop, eff = value_of_eff, NOC = no_of_comp, comp =
comp_array) annotation( ...);
```

`pressDrop` is the pressure drop and `eff` is the efficiency.

Composite material stream needs to be created and instanced for input and output of cooler. One Energy_Stream also instantiated. All connections can be made in digram view.

Values for variables of input stream has to be given and value of one variable of heater also needs to be specified according to mode. Value for heater variable can be specified as below:

```
cooler1.mode_var = value_of_mode_var ;
```

Heater model supports four modes:

1. Heat removed: value of heate removed (`heatRem`) is specified in Joules.
2. Outlet temperature: value of outlet temperature(`outT`) is specified in K.
3. Energy Stream: enFlo variable from Energy_Stream(`Energy_Stream_name.enFlo`) is specified in Joules.
4. Phase Molar Fraction: Vapor phase molar fraction of outlet(`outVapPhasMolFrac`) is specified.
5. Temperature drop: value of temperature change(`tempDrop`) is specified.

Example cooler1 is available in Test section for Cooler.

3.4 Valve:

Valve is used for constant pressure drop in simulation. Outlet material stream properties are calculated for isenthalpic pressure drop. Valve has one inlet and one outlet for material streams. Valve can be instantiated by dragging and dropping in diagram view. Value to its parameters is given while instantiating. The code looks like follows:

```
Simulator.Unit_Operations.Valve valve1(NOC = no_of_comp , comp =
comp_array) annotation( ...);
```

Composite material stream needs to be created and instanced as input and output. All connections can be done in diagram view. Values of variables of input stream needs to be given and one variable of valve needs to be specified according to mode. Code for valve looks like follows:

```
valve1.mode_var = value_of_mode_var;
```

Valves supports two modes:

1. Pressure drop: value of pressure drop(`pressDrop`) is specified.
2. Outlet pressure: value of outlet pressure(`outP`) is specified.

Test example valve1 is available in Test section.

3.5 Shortcut_Column:

This model is the implementation of Fenske-Underwood-Gilliland method. This model is used for calculating separation, temperatures and heat loads in distillation. To use Shortcut_Column we need to first extend it with thermodynamic and then use that model in simulation. This can be done as follows:

```
model model_name
     extends Simulator.Unit_Operations.Shortcut_Column;
     extends
Simulator.Files.Thermodynamic_Packages.package_name;
model  model_name;
```

This model is then instantiated as shortcut column. Some parameters needs to be specified while instantiating. Final code for for instantiating shortcut column looks like follows:

```
model_name model_name1(NOC = no_of_comp, comp = comp_array,
condType = "codensor_type", HKey = index_of_HKey_in_comp_array, LKey
= index_of_LKey_in_comp_array);
```

`condType`: This is condensor Type. This is defined as string. `condType` can be `"Partial"` or `"Total"`. Default is `"Total"`.

`HKey`: This is the index of heavy key component in compenent array.

`LKey`: This is the index of light key component in component array.

Composite material stream needs to be created and instantiated for Feed, Bottoms and Top. Energy_Stream is instantiated for Condesor_Duty and Reboiler_Duty. All connections can be done in diagram view. Values of variables of feed stream needs to be given and some variables of shortcut column needs to be specified in equation section. Code for shortcut column looks like follows:

```
model_name1.shortP[2] = value_of_bottoms_press ;
model_name1.shortP[3] = value_of_top_press;
model_name1.mixMolFrac[2, model_name1.LKey] = value_of
Lkey_molfrac_in_top ;
model_name1.mixMolFrac[3, model_name1.HKey] =
value_of_HKey_molfrac_in_bottoms ;
```

```
model_name1.actR = value_of_reflux_ratio;
```

`model_name1.shortP[2]` is bottoms pressure. `model_name1.shortP[3]` is top pressure. `model_name1.mixMolFrac[2, model_name1.LKey]` is mole fraction of light key component in bottoms. `model_name1.mixMolFrac[3, model_name1.HKey]` is mole fraction of heavy key component in top. `model_name1.actR` is reflux ratio. Right hand side of equations are values of these variables respectively.

Example shortcut1 is available in Test section for Shortcut_Column.

3.6 Component_Separator:

Component_Separator is the mass balance unit operation. You need to specify absolute mole flow or percentage of moles in inlet for each component. Mass balance and energy balance is done in Component_Separator using those values. Component_Separator can be instantiated as follows:

```
Simulator.Unit_Operations.Compound_Separator
compound_Separator1(NOC = no_of_comp , comp = comp_array, sepFact =
{"comp1_sep_factor", "comp2_sep_factor",...,"compNOC_sep_factor"},
sepStrm = sep_stream_no);
```

`sepFact` is separation factor array of index `NOC`. Each component have its own separation factor. These are string values. You can choose separation factor from below:

1. `"Molar_Flow"`: Absolute molar flow value of component in stream in mol/s
2. `"Mass_Flow"`: Absolute mass flow value of component in stream in gm/s
3. `"Inlet_Molar_Flow_Percent"`: Molar flow percent of inlet mole flow of respective component
4. `"Inlet_Mass_Flow_Percent"`: Mass flow percent of inlet mole flow of respective component

`sepStrm` is the separation stream. `Component_Separator` separates one stream into two streams. `SepStrm` is the stream for which separation factors are given. It is an integer value.

Composite stream needs to be created and instantiated for one input and two outputs of Component_Separator. One Energy_Stream is also instantiated. All connections can be done in diagram view. Values of input streams has to be specified in equation section. Values for separation factors has to be given in equation section. Code for separation values assigning looks like follows:

```
compound_Separator1.sepFactVal = {value_of_sep_fact_comp1,
value_of_sep_fact_comp2,...., value_of_sep_fact_compNOC};
```

`compound_Separator1.sepFactVal` is an array of index `NOC` which contains separation factor values of separation stream for all components.

Example comp_sep1 is available in Test section for Component_Separator.

3.7 Flash:

Flash model separates input stream into vapor stream and liquid stream at given temperature and pressure. For using flash we need to first extend Flash with Thermodynamic model and use that model in simulation. The code looks like follows:

```
model model_name
     extends Simulator.Unit_Operations.Flash;
     extends
Simulator.Files.Thermodynamic_Packages.package_name;
end model_name;
```

We need to instantiate this model in final model of simulation. For instantiating this model we need to write:

```
model_name  model_name1(NOC = no_of_comp, comp = comp_array);
```

Composite material stream needs to be created and instantiated for feed stream, liquid stream and vapor stream. All connections can be done in diagram view. Values of feed stream has to be specified in equation section.

Example Flash is available in Test section for Flash model.

3.8 Splitter:

This is the model for steady state stream splitter. It have one input stream and can have any number of output stream. Splitter can be instantiated by dragging and dropping. Values of splitter parameters has to be specified while instantiation. Final code looks like follows:

```
Simulator.Unit_Operations.Splitter splitter1(NOC = no_of_comp ,
comp = comp_array, NO = no_of_output_streams , calcType =
"calculation_type") annotation( ...);
```

`NO` is number of output streams. `CalcType` is splitter calculation type. Following calculation types are available.

1. `"Split_Ratio"`: Array of split ratio of all output stream
2. `"Molar_Flow"`: Array of molar flows of all output stream in mol/s
3. `"Mass_Flow"`: Array of mass flows of all output stream in g/s

Composite material stream needs to be created and instatiated as input and outputs of the splitter. Input of splitter can be connected using diagram view to instantiated input material stream. Outlet connectors of splitter are defined as an array, which are not available in diagram view for connection, hence we need to write connect equations for connecting outlet material streams. Connect equations are written in equation section and they look like follows:

```
connect(cmpst_strmNO+1.outlet, split.inlet) annotation( ...);
connect(splitter1.outlet[1], cmpst_strm1.inlet);
```

```
connect(splitter1.outlet[2], cmpst_strm2.inlet);
                                .
                                .
                                .
connect(splitter1.outlet[NO], cmpst_strmNO.inlet);
```

Here first equation is automatically generated because of connection created using diagram view. Other equations needs to be written manually in text view.

`cmpst_strm` is created composite material stream. `cmpst_strmNO+1` is input stream. `cmpst_strm1.outlet, cmpst_strm2.outlet.... cmpst_strmNO.outlet` are inlet connectors of instatiated composite material streams. `splitter1.outlet[1], splitter1.outlet[2]... splitter1.outlet[NO]` are outlet connectors of instantiated splitter model.

Composite material stream needs to be created and instantiated for input and outputs. Values for input stream needs to be assigned. `specVal` array should be specified in equation section. Code looks like follows:

```
splitter1.specVal = specVal_array;
```

`splitter1.specVal` is specification value array and `specVal_array` is array of splitter specification values. Example split is available in Test section.

3.9 Centrifugal_Pump:

This is simulation of centrifugal pump. This has one input stream one output stream and one energy connection. Centrifugal pump can be instantiated by dragging and dropping. Values of some parameters of centrifugal pump needs to be given in bracket after instantiation. Code for centrifugal pump looks like follows:

```
Simulator.Unit_Operations.Centrifugal_Pump
centrifugal_Pump1(comp = comp_array, NOC = no_of_comp, eff =
value_of_eff) annotation( ...);
```

`eff` is the efficiency of the pump.

Composite material stream has to be created and instantiated as input and output of the pump. Energy_Stream is also instantiated. All connections can be done in diagram view. Values of variables for input stream needs to be specified in equation section. Value of one variable is also needs to be speicified according to mode. Code for centrifugal pump looks like follows:

```
centrifugal_Pump1.mode_var = value_of_mode_var;
```

`centrifugal_Pump1.mode_var` is mode variable and right hand side is its value. `Centrifugal_Pump` supports following modes:

1. Pressure increase: Value of increase in pressure(`pressInc`) in Pa is specified.
2. outlet Pressure: Value of outlet pressure(`outP`) in Pa is specified.
3. power required: Value of power required(`reqPow`) in Watts is specified

4. Energy Stream: Value of energy flow in energy stream (`Energy_Stream_Name.enFlo`) in Watts is specified.

Example pump is available in Test section for Centrifugal_Pump.

3.10 Adiabatic Compressor:

Compressor provides energy to vapor stream by increasing pressure. Ideal process is isentropic. Non ideality is considered by taking compressor efficiency which is provided by user. This model needs one input stream, one output stream and one energy stream. To use Adiabatic_Compressor we need to first extend it with thermodynamic and then use that model in simulation. This can be done as follows:

```
model model_name
     extends Simulator.Unit_Operations.Adiabatic_Compressor;
     extends
Simulator.Files.Thermodynamic_Packages.package_name;
end  model_name;
```

This model is then instantiated as shortcut column. Some parameters needs to be specified while instantiating. Final code for for instantiating shortcut column looks like follows:

```
model_name model_name1(NOC = no_of_comp, comp = comp_array, eff
= value_of_eff);
```

`eff` is the efficiency of Compressor.

Composite material stream has to be created and instantiated as input and output of the compressor. Energy_Stream is also instantiated. All connections can be done in diagram view. Values of variables for input stream needs to be specified in equation section. Value of one variable is also needs to be speicified according to mode. Code for adiabatic compressor looks like follows:

```
model_name1.mode_var = value_of_mode_var;
```

`model_name1.mode_var` is mode variable and right hand side is its value. `Adiabatic_Compressor` supports following modes:

1. Pressure increase: Value of increase in pressure(`pressInc`) in Pa is specified.
2. outlet Pressure: Value of outlet pressure(`outP`) in Pa is specified.
3. power required: Value of power required(`reqPow`) in Watts is specified

Example adia_comp1 is available in Test section for Adiabatic_Compressor.

3.11 Adiabatic Expander:

Expander generates energy from high pressure vapor stream. Ideal process is isentropic. Non ideality is considered by taking expander efficiency which is provided by user. This model needs one

input stream, one output stream and one energy stream. To use Adiabatic_Expander we need to first extend it with thermodynamic and then use that model in simulation. This can be done as follows:

```
model model_name
     extends Simulator.Unit_Operations.Adiabatic_Expander;
     extends
Simulator.Files.Thermodynamic_Packages.package_name;
end  model_name;
```

This model is then instantiated as shortcut column. Some parameters needs to be specified while instantiating. Final code for for instantiating shortcut column looks like follows:

```
model_name model_name1(NOC = no_of_comp, comp = comp_array, eff
= value_of_eff);
```

`eff` is the efficiency of Expander.

Composite material stream has to be created and instantiated as input and output of the expander. Energy_Stream is also instantiated. All connections can be done in diagram view. Values of variables for input stream needs to be specified in equation section. Value of one variable is also needs to be speicified according to mode. Code for adiabatic expander looks like follows:

```
model_name1.mode_var = value_of_mode_var;
```

`model_name1.mode_var` is mode variable and right hand side is its value. `Adiabatic_Expander` supports following modes:

4. Pressure drop: Value of increase in pressure(`pressDrop`) in Pa is specified.
5. outlet Pressure: Value of outlet pressure(`outP`) in Pa is specified.
6. power generated: Value of power generated(`genPow`) in Watts is specified

Example adia_exp1 is available in Test section for Adiabatic_Expander.