Creation and Development of process flowsheets in OpenModelica

Introduction

Simulator is the OpenModelica library created for chemical process simulation at FOSSEE, IIT Bombay. The library comprises of a wide range of compound database along with a comprehensive set of thermodynamic packages and unit operations.

I. Working rule for the creation of process flowsheets in OpenModelica

1) Import the Chemsep database from Simulator→Files → Chemsep_Database

Syntax: import data = Simulator.Files.Chemsep_Database;

2) Instantiate the compounds required for simulation of the process flowsheet.

Syntax: parameter data.Methanol meth;

3) Define the number of compounds instantiated as an integer parameter.

Syntax: parameter NOC= 2;

4) Since the material stream is chosen along with a particular thermodynamic package, it may be done in a separate model and used in the flowsheet. The following subroutine describes the creation of the material stream model instantiation.

- ✓ Create a separate model with name Material_Stream.
- ✓ Extend the material stream model from Simulator→Streams → Material Stream
  Syntax: extends Simulator.Streams.Material_Stream
- ✓ Extend the thermodynamic package required for simulation along with the material stream from Simulator→Files → Thermodynamic_Packages
  Syntax: extends Simulator.Files.Thermodynamic_Packages.Raoults_Law

5) Define the component array which may be used to access all the general and thermophysical properties of the compounds selected in the flowsheet.

Syntax: parameter data.General_Properties comp[NOC] = {meth, eth, wat};

6) Switch from Text view to Diagram view in the modelling perspective. The conventional drag and drop approach will be used to build and develop the process flowsheet. This will be followed by providing appropriate parameters as arguments to the instantiated model.

7) Drag and drop the material stream model created using the subroutine followed in point-4. On doing the above, a pop up text box appears. The stream name can be given accordingly and thereby the model gets instantiated successfully.
Once the instantiation is done the NOC (Number of components) and comp(component array) defined earlier are provided as arguments.
In the example given below the stream name is Input.

Syntax: Meth_Wat_Distillation.Material_Streams Input(NOC=NOC, comp=comp)

8) Similar to the material stream model drag and drop the unit operation required and name it accordingly. The name of the instantiated Heater model is Heater and the arguments provided in the bracket are the parameters required to simulate the model.

Syntax: Simulator.Unit_Operations.Heater Heater (pressDrop = 0, eff = 1, NOC = NOC, comp=comp)

9) Now, an another instantiation of the material stream model is done to serve the outlet of the unit operation. This depends on whether the unit operation has a single/multiple outlet.

10) Once the inlet material stream, unit-operation model and outlet material stream are instantiated, the inlet and outlet of the unit operation model are connected to the respective material streams.

11)On connecting the model to the material streams the "Connect" equations are automatically written in the modelling perspective under the **equation** section.

12) This is followed by providing the variable values essential to satisfy the degrees of freedom of the system.

Syntax: Variables to be specified for the inlet material stream

```
//Input mole fraction of the stream
Input.compMolFrac[1, :] = {0.36,0.64};

 //Input temperature
Input.T = 300;
```

```
 //Input molar flow rate
Input.totMolFlo[1]= 60;

//Input pressure
 Input.P = 101325;
```

Variables to be specified for unit operation (Eg: Heater)

```
//Heat added
Heater.outT=325.15;
```

13) After providing the necessary variables required to satisfy the completion of the model, the **“Check Model”** button is clicked to check the equality of number of equations and variables of the model.

14) Now the model is complete and we can proceed to simulate by clicking the **“Simulate”** button at the top of the screen.

15) Similar procedure is followed and the respective model is accordingly instantiated to develop the flowsheet.

II. Error Rectification and Debugging

1) On simulation of the model, there are possibilities of model errors which may be of the following types

- ✓ Degrees of Freedom Mismatch.
- ✓ Convergence Issues
- ✓ Deviation in results due to improper initial guess values

1.1) Degrees of Freedom: These kind of errors normally arise when unwanted variables are defined without any equations corresponding to them or when equations are repeated more than once. Usually while handling array operations equations tend to get repeated due to multiple looping of the array.

Corrective Measure: Errors of such nature may be avoided by checking the model multiple times at each and every stage, as the model is constructed. This will help to identify the exact equation getting repeated and provide a rectification for the same.

1.2) Convergence Issues: The most complex type of error which occurs mainly due to any mathematical violation caused in the model. These kind of errors may occur at any part of the model.

Corrective Measure: The approach used to eliminate such errors is by handling minimal number of equations in a particular simulation run in turn achieved by simulating one block at a time while creating the flowsheet. In this manner we will be able to identify and correct the mathematical error occurring in the model.

In order to implement the above step one can make use of an inbuilt Debugger named **Transformational Debugger** in OpenModelica. This debugger exactly spots the variables which cause a mathematical violation along with the line in which it is placed.

The second supportive means to avoid these errors is to provide good initial guess to the variables which cause the problems. OpenModelica displays the list of variables in the model which can be provided with good guess values.

Syntax: T(start=300);