

# **Development and Commercialization of an IDAES-Based Power Plant Performance Monitoring and Optimization System**

**DOE FY2020 Phase I Release 2 SBIR Topic 25(b)**

Award Number: DE-SC0020794

By

Dr. Rodney R. Gay  
MapEx Software, Inc.

RodneyGay@PowerPlantPerformance.com



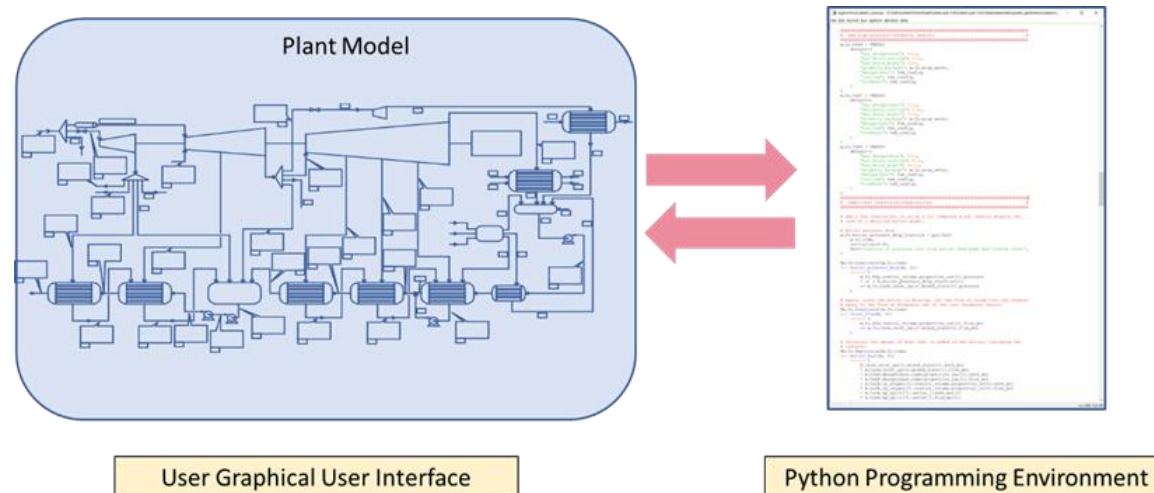
# SBIR Project Overview

- Introduction
  - IDAES (Institute for the Design of Advanced Energy Systems)
  - Open-source computational platform.
  - Extensive modeling environment for power plant and process applications.
  - Equation-oriented analytics
  - Supports dynamic analysis and optimization.
  - IDAES is ideally suited for the energy-system modeling challenges of the future
- Problem Identification:
  - Building IDAES models requires IDAES-specific expertise
  - No user interface to support IDAES model building
  - IDAES modeling is implemented via Python-language programs.
  - Many process-system engineers are not familiar with Python.
- Solution:
  - Develop a desktop software application to support IDAES users.
  - Implement a flowsheet-based, graphical-user-interface.
  - Eliminate the need for user to write Python programs.
  - Make IDAES easier to use.



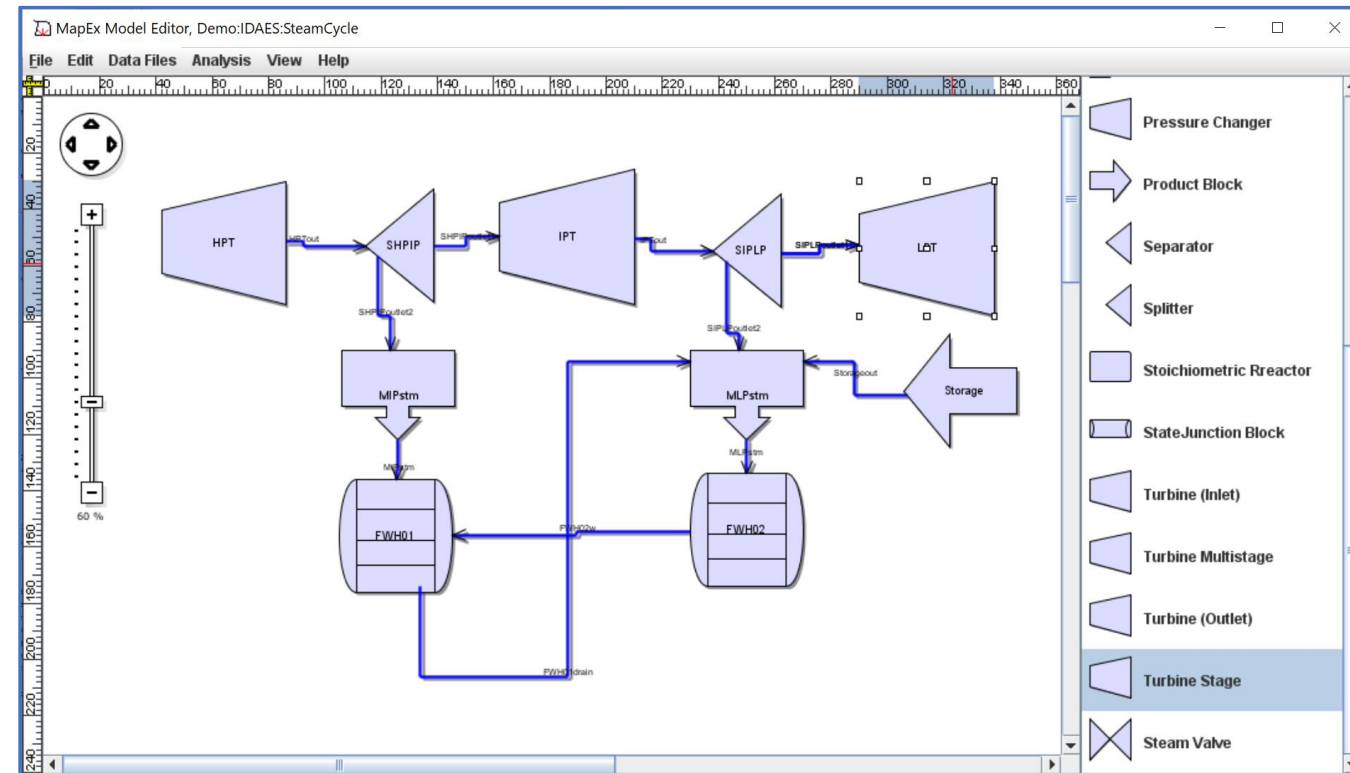
# Software Concept

- User builds model using flowsheet GUI.
- Software creates the Python code needed to run the IDAES analysis.
- Software runs analysis and retrieves results for display on flowsheet.



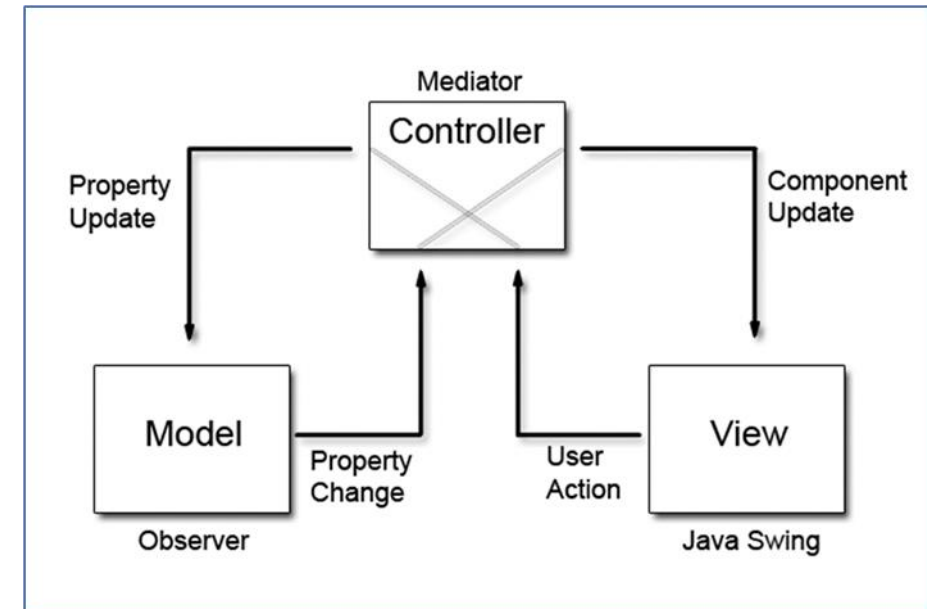
# User Experience

- User Interacts with the software via the flowsheet Graphical User Interface (GUI)
- Open an existing model or to create a new model
- Add/delete equipment icons (unit-models)
- Add/delete streams (arcs) connecting unit models
- Enter input data via dialogs assessed from flowsheet
- Run Analysis
  - Software generates IDAES Python code file
  - IDAES analysis runs and generates results
- Results are displayed on flowsheet
- Save model for future access

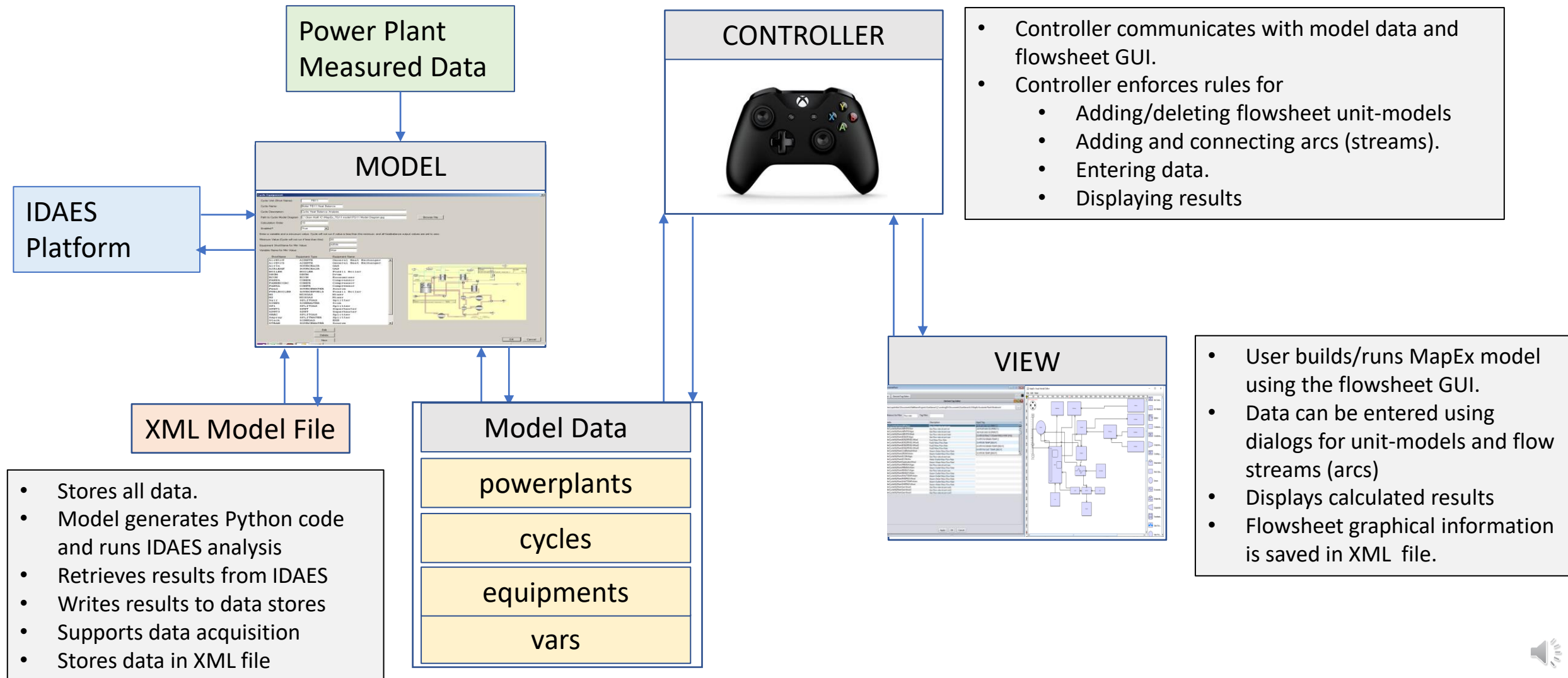


# Model-Controller-View Software Design

- The Model
  - Manages the data of the application
  - Implements the IDAES analysis
  - Receives user input from the Controller
  - Model does not communicate directly with the View.
- The View
  - Renders a graphical (GUI) presentation of the model data.
  - Handles interactions with the software user
  - View is independent of the Model.
  - The same View component is used for the hybrid analytics system being developed in DOE project DE-FE00031753.
- The Controller
  - Responds to the user input from the View and performs interactions on the data model objects
  - The Controller receives the input, optionally validates it, and then passes the input to the Model



# Model-View-Controller Implementation



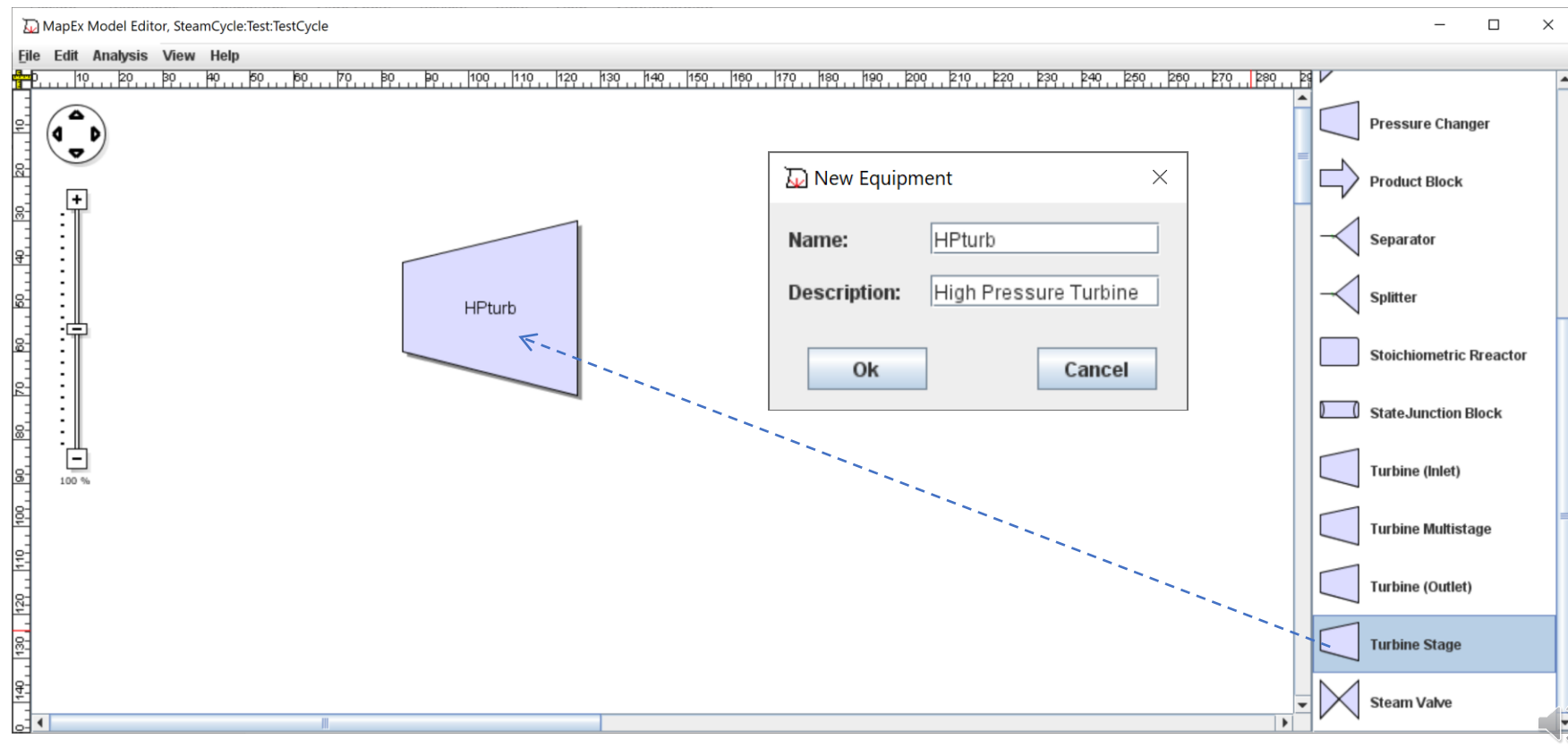
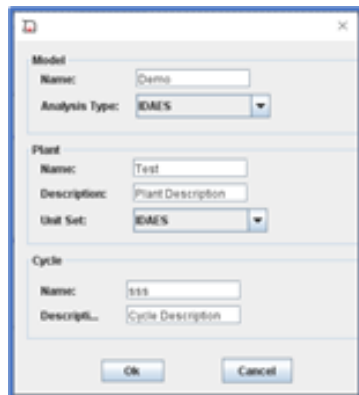
# Build and Run an IDAES Model

1. Construct flowsheet representation of the model
  - a) Add equipment icons to flowsheet
    1. Drag selected equipment icon onto flowsheet
    2. Give the equipment a name
  - b) Add flow streams between equipment icons
    1. Drag cursor from source equipment to target equipment
    2. Specify the source port and target port
    3. Give the stream a name
2. Enter/Edit Input Data
  - a) Double-click on equipment to view its input/output variables
  - b) Edit data for each variable by right-clicking “edit”
3. Click: Run Analysis
  - a) Software generates Python code file representing the model and its data
  - b) Open and run the file in your Python development environment (Idle or Spyder)
  - c) Running and retrieving results will be automated in Phase II



# Build Flowsheet Model of a Power Plant Cycle

- Create a new IDAES (cycle) model
  - Input names of Model, Plant and Cycle
- Add unit models
  - Drag Icons onto Flowsheet





# Edit Input Data (Outlet Pressure) for Turbine Stage

MapEx Model Editor, SteamCycle:Test:TestCycle

File Edit Data Files Analysis View Help

Equipment: HPturb

Variables Details

Variables and Measurements

Filter:

Variable	Description	Value	IDAES Input	Action	Units	Ta...	Ta...
addTPXport	Add an output port using TPS ...	0	iapws95.l...	unfix			0
efficiency...	Isentropic efficiency	0.9		fix	%		0
inletEnth...	inlet enthalpy in moles	65,000		unfix	kJ/kg...		0
inletFlow...	inlet flow in moles	10,000		unfix	kg-m...		0
inletPress...	inlet pressure	19,300,000		unfix	Pa		0
inletTemp...	inlet temperature	819		unfix	K		0
outletEnth...	outlet enthalpy in moles	50,000		unfix	kJ/kg...		0
outletFlow...	outlet flow in moles	10,000		unfix	kg-m...		0
outletPres...	outlet pressure	4,140,000		fix	Pa	Col... 4,1...	0
outletTem...	outlet temperature			unfix	K		0
pressure...	pressure ratio	10		unfix	ratio		0
propertyP...	Name of property package	0.m.fs.prop...		fix			0
workMech...	Work input or output	1		unfix	Watt		0

Ok Cancel

Steam Valve

Equipment Variable: outletPressure

Variable Details

Name: outletPressure

Description: outlet pressure

Var Type: double

Action: fix

Tag Name: ColdReheatPress

Tag Value: 4140000.0

Tag Units: Pa

Uncertainty: 0.01

Measured: 4140000.0

Value: 4140000.0

IDAES input:

Python Var Name: outlet[:].pressure

Units: Pa

Heat Balance: 0.0

Results Units:

Out Tag Name:

Minimum: 0.0

Maximum: 0.0

Reference: 0.0

Report Level: 2

Ok Cancel

Python variable name is outlet[:].pressure

# Some Inputs (property\_package) are strings

MapEx Model Editor, SteamCycle:Test:TestCycle

File Edit Data Files Analysis View Help

Equipment: HPturb

Variables Details

Variables and Measurements

Filter:

Variable	Description	Value	IDAESIn...	Action	Units	Ta...	Ta...
addTPXp...	Add an output port using TP...	0	iapws95...	unfix			0
efficiency...	Isentropic efficiency	0.9		fix	%		0
inletEnth...	inlet enthalpy in moles	65,000		unfix	kJ/kg...		0
inletFlow...	inlet flow in moles	10,000		unfix	kg-m...		0
inletPres...	inlet pressure	19,300,000		unfix	Pa		0
inletTem...	inlet temperature	819		unfix	K		0
outletEnt...	outlet enthalpy in moles	50,000		unfix	kJ/kg...		0
outletFlo...	outlet flow in moles	10,000		unfix	kg-m...		0
outletPre...	outlet pressure	4,140,000		fix	Pa	Col... 4,1...	
outletTe...	outlet temperature	700		unfix	K		0
pressure...	pressure ratio	10		unfix	ratio		0
propertyP...	Name of property package	0	m.fs.pro...	fix			0
workMec...	Work input or output	1		unfix	Watt		0

Ok Cancel

Steam Valve

Equipment Variable: propertyPackage

Variable Details

Name: propertyPackage

Description: Name of property package

Var Type: Arg

Action: fix

Tag Name:

Tag Value: 0.0

Tag Units:

Uncertainty: 0.01

Measured: 0.0

Value: 0.0

IDAES input: m.fs.prop\_water

Python Var Name: property\_package

Units:

Heat Balance: 0.0

Results Units:

Out Tag Name:

Minimum: 0.0

Maximum: 0.0

Reference: 0.0

Report Level: 3

Ok Cancel

IDAES Input is: m.fs.prop\_water

# Fix/Unfix Variable Values

- The user can choose to how the input Value is used.
  - fix – set the input value of the variable to the Value
  - unfix – leave the value of the variable unspecified, free to be determined by the analysis
  - value – input an initial value

Equipment Variable: inletPressure

**Variable Details**

Name:	inletPressure
Description:	inlet pressure
Var Type:	double
Action:	fix
Tag Name:	fix
Tag Value:	unfix
Tag Units:	value
Uncertainty:	0.01
Measured:	0.0
Value:	1.93E7
IDAES input:	
Python Var Name:	inlet[.].pressure
Units:	Pa
Heat Balance:	0.0
Results Units:	
Out Tag Name:	
Minimum:	0.0
Maximum:	0.0
Reference:	0.0
Report Level:	2

Ok Cancel



# Fix Multiple Inputs for HPturb

- Set input values for a sufficient number of input variables to enable prediction of HPturb power and outlet enthalpy.
- Fix values of
  - efficiency
  - inlet enthalpy
  - Inlet flow
  - Inlet pressure
  - outlet pressure
- At this point the analysis is ready to run!

Equipment: HPturb

Variables Details

Variables and Measurements

Filter:  X

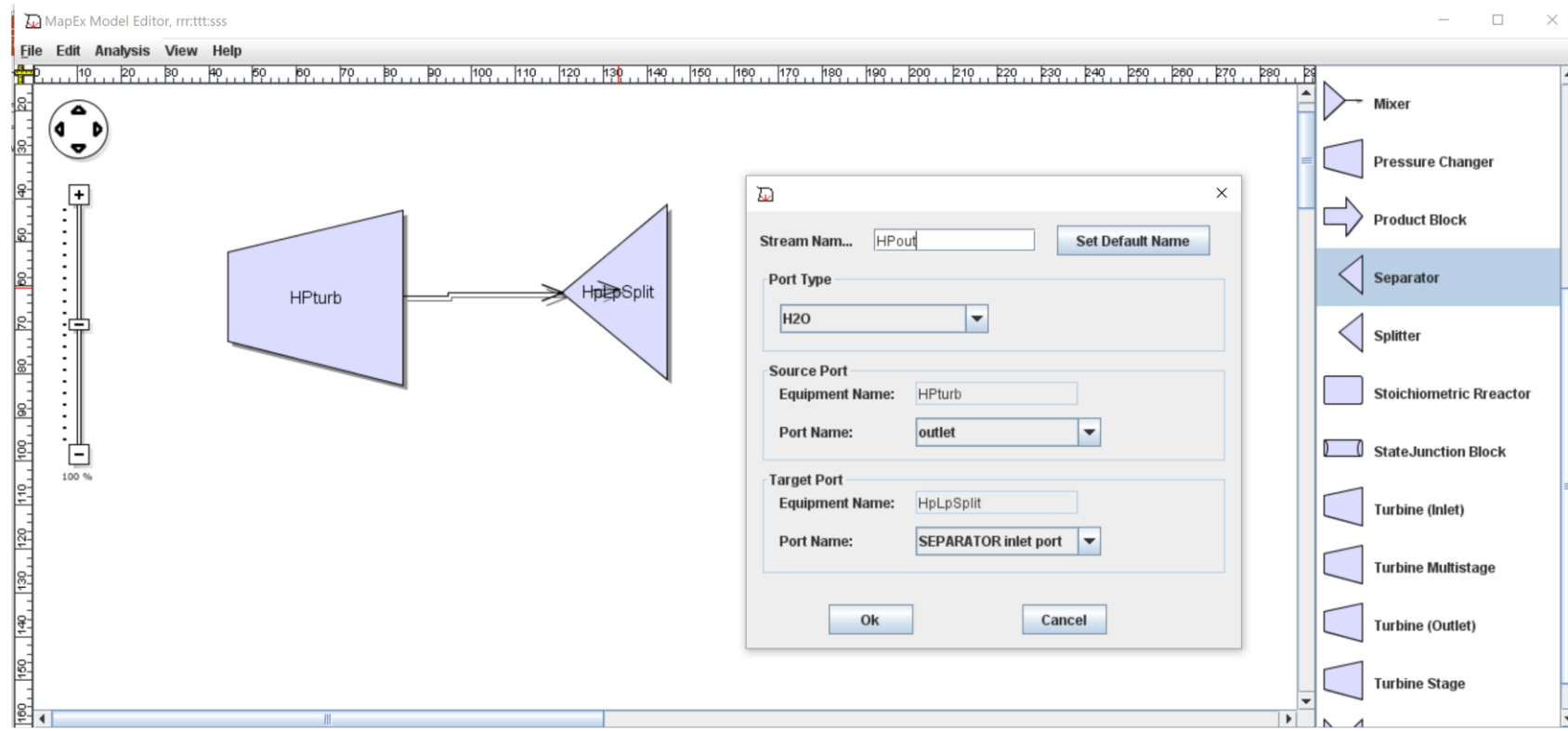
Variable	Description	Value	IDAESInput	Action	Units	TagNa...	TagVal...
efficiencyIsent...	Isentropic efficiency	0.9		fix	%		0
inletEnthmol	inlet enthalpy in moles	65,000		fix	kJ/kg-mo...		0
inletFlowmol	inlet flow in moles	10,000		fix	kg-mole/...		0
inletPressure	inlet pressure	19,300,000		fix	Pa		0
outletPressure	outlet pressure	4,140,000		fix	Pa		0
propertyPacka...	Name of property package	0	m.fs.prop_water	fix			0
addTPXport	Add an output port using TPS properties	0	iapws95.iapws9...	unfix			0
inletTemperat...	inlet temperature	819		unfix	K		0
outletEnthmol	outlet enthalpy in moles	50,000		unfix	kJ/kg-mo...		0
outletFlowmol	outlet flow in moles	10,000		unfix	kg-mole/...		0
outletTemper...	outlet temperature	700		unfix	K		0
pressureRatio	pressure ratio	10		unfix	ratio		0
workMechanical	Work input or output	1		unfix	Watt		0

Ok Cancel



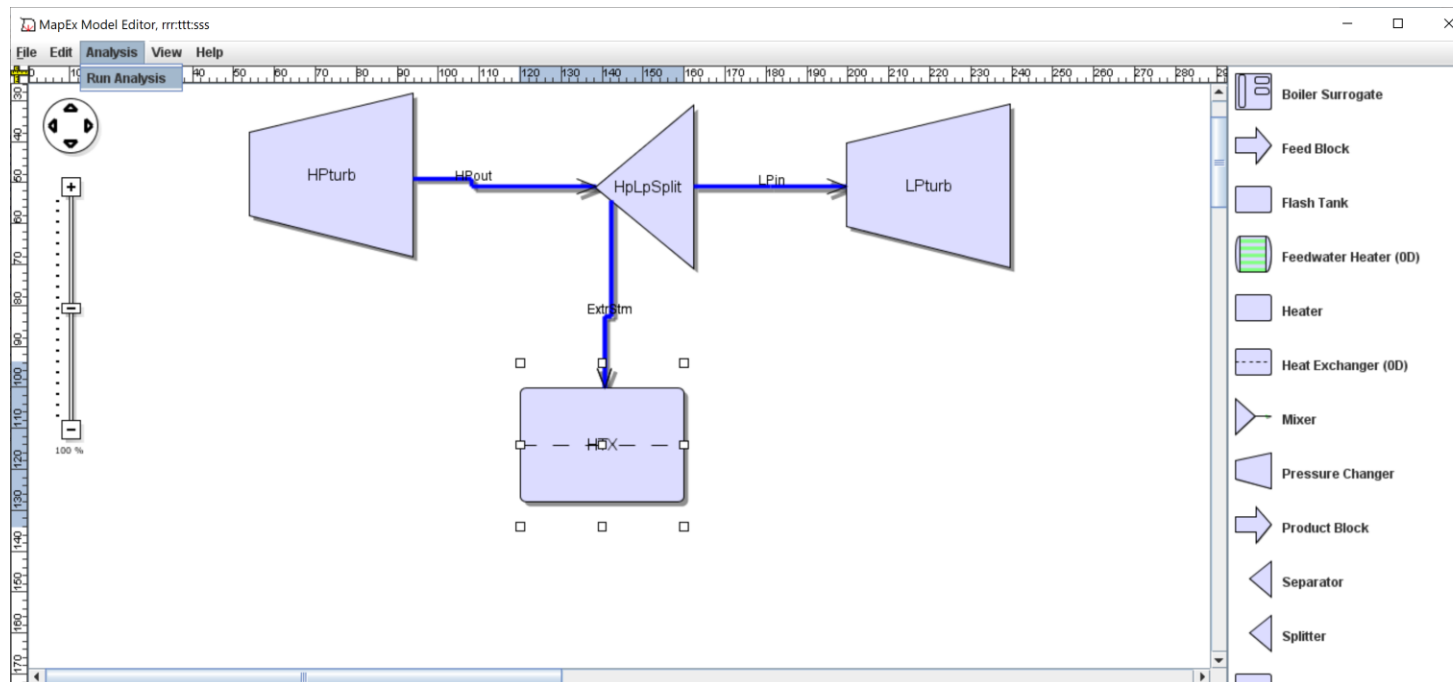
# Add Splitter to the Cycle

- Add a Separator equipment icon to cycle and name it HPIPsplit
- Click on HPturb, then drag stream to HpLpSplit
- Choose ports at stream inlet and outlet
- Give the stream a name



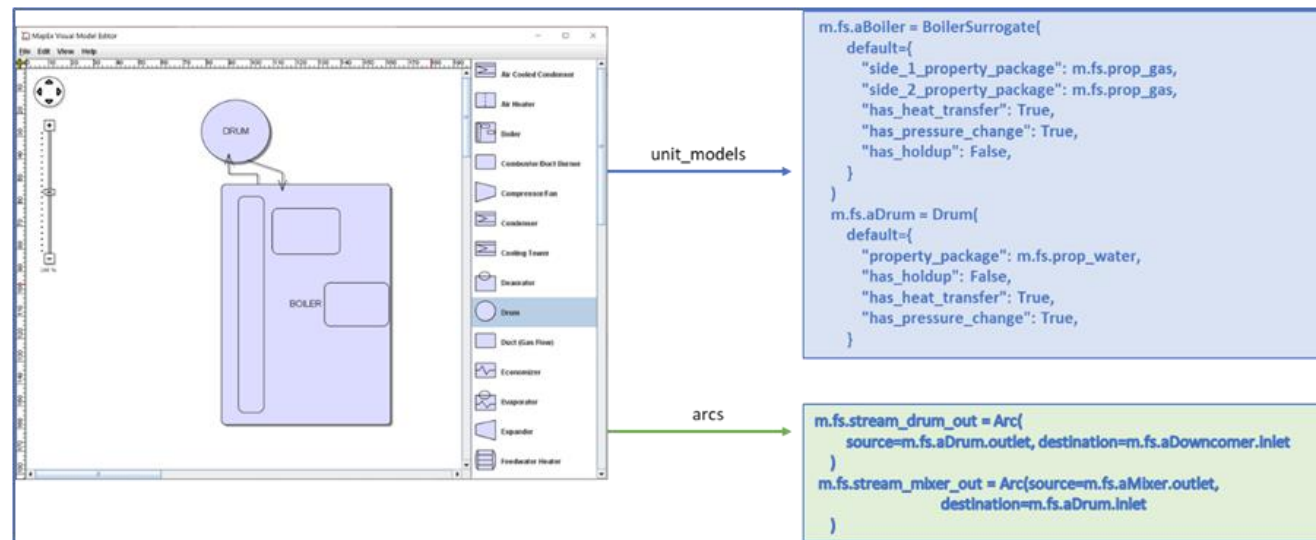
# Generate IDAES Python Code

- Add LPturb and HTX equipment icons to cycle and draw flow streams
- Click “Run Analysis”
- Python code file representing the current flowsheet model is generated
  - Default name is: `CycleName_PythonCode.py`
- Analysis does not yet run automatically
  - Load file into Spyder (or your chosen Python development environment) to run manually



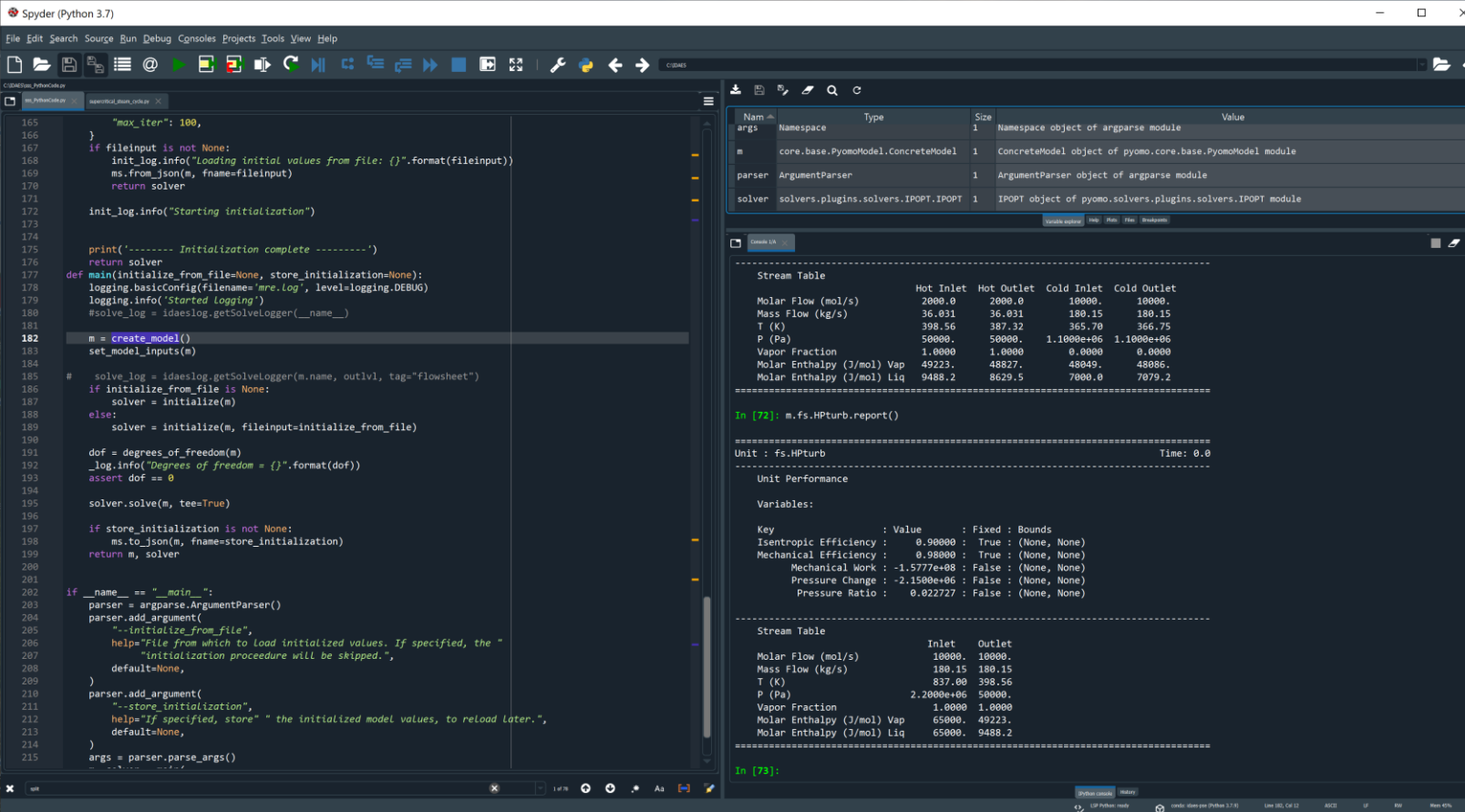
# Python Code File

- Software creates a file containing Python code to run the flowsheet cycle.
- Diagram below shows Python code to:
  - Add Boiler and Drum unit-model and specify optional inputs
  - Add connecting Arcs (flow streams) between Boiler and Drum



# Manually Open and Run Python File

- Use Spyder to open the file: C:\IDAES\CycleName\_PythonCode.py
- Click “Run”
- Screen below shows the model and calculated results in Spyder



The screenshot displays the Spyder Python IDE interface. The left pane shows a Python script with the following key sections:

```
165     "max_iter": 100,
166 }
167 if fileinput is not None:
168     init_log.info("Loading initial values from file: {}".format(fileinput))
169     ms.from_json(m, fname=fileinput)
170     return solver
171
172 init_log.info("Starting initialization")
173
174 print('----- Initialization complete -----')
175
176 return solver
177
178 def main(initialize_from_file=None, store_initialization=None):
179     logging.basicConfig(filename='mre.log', level=logging.DEBUG)
180     logging.info("Started Logging")
181     #solve_log = idaeslog.getSolveLogger(__name__)
182
183     m = create_model()
184     set_model_inputs(m)
185
186     # solve_log = idaeslog.getSolveLogger(m.name, outlvl, tag="flowsheet")
187     if initialize_from_file is None:
188         solver = initialize(m)
189     else:
190         solver = initialize(m, fileinput=initialize_from_file)
191
192     dof = degrees_of_freedom(m)
193     log.info("Degrees of freedom = {}".format(dof))
194     assert dof == 0
195
196     solver.solve(m, tee=True)
197
198     if store_initialization is not None:
199         ms.to_json(m, fname=store_initialization)
200     return m, solver
201
202 if __name__ == "__main__":
203     parser = argparse.ArgumentParser()
204     parser.add_argument(
205         "--initialize_from_file",
206         help="File from which to load initialized values. If specified, the "
207             "initialization procedure will be skipped.",
208         default=None,
209     )
210     parser.add_argument(
211         "--store_initialization",
212         help="If specified, store " the initialized model values, to reload later.",
213         default=None,
214     )
215     args = parser.parse_args()
```

The right pane shows the IPython console output, including a table of calculated results:

Stream	Hot Inlet	Hot Outlet	Cold Inlet	Cold Outlet
Molar Flow (mol/s)	2000.0	2000.0	10000.	10000.
Mass Flow (kg/s)	36.031	36.031	180.15	180.15
T (K)	398.56	387.32	365.70	366.75
P (Pa)	50000.	50000.	1.1000e+06	1.1000e+06
Vapor Fraction	1.0000	1.0000	0.0000	0.0000
Molar Enthalpy (J/mol) Vap	49223.	48827.	48049.	48086.
Molar Enthalpy (J/mol) Liq	9488.2	8629.5	7000.0	7079.2

The console also shows the execution of the `m.fs.HPTurb.report()` command, which outputs the unit performance and variables for the HPTurb unit.

Unit Performance

Variables:

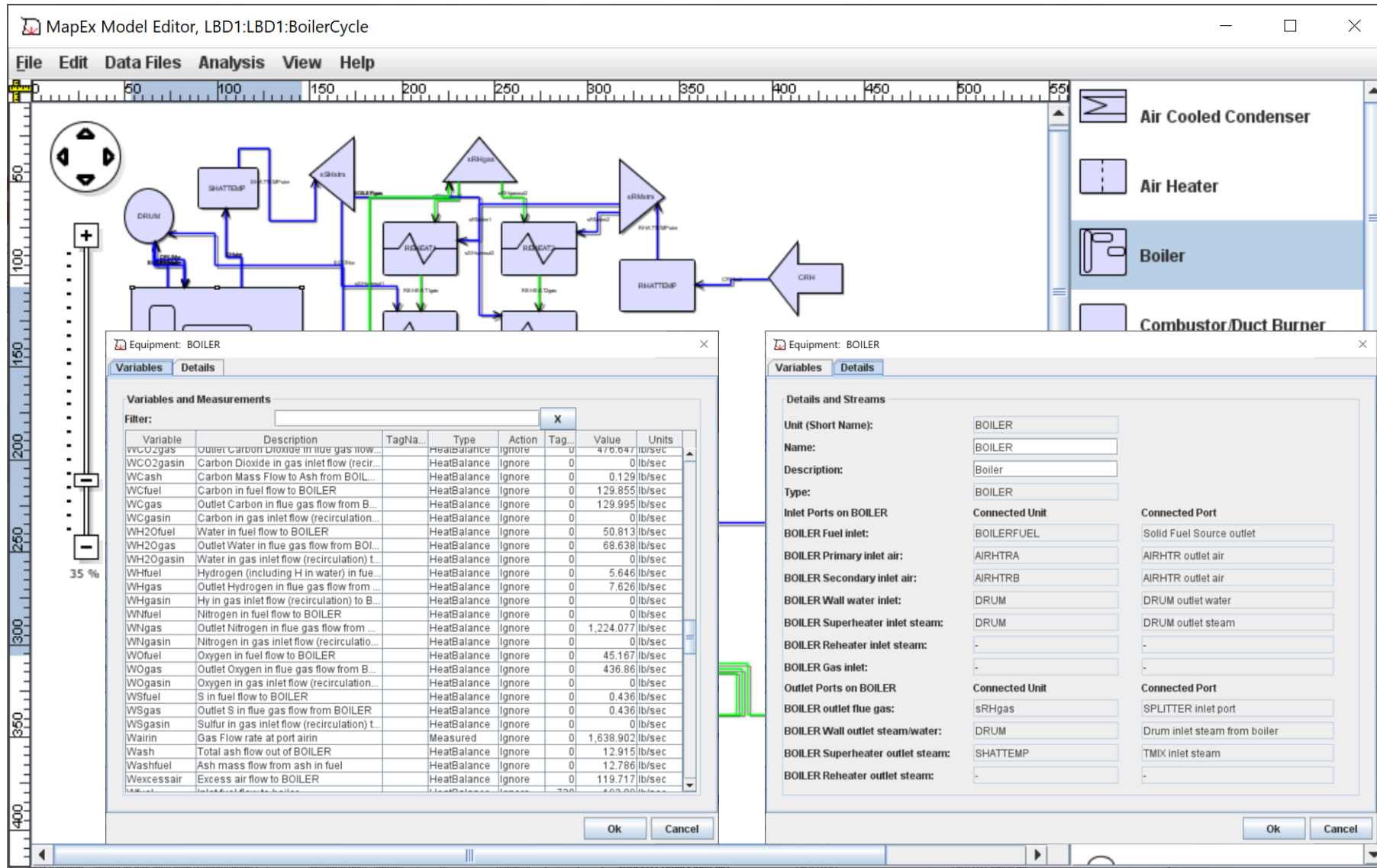
Key	Value	Fixed	Bounds
Isentropic Efficiency	0.90000	True	(None, None)
Mechanical Efficiency	0.90000	True	(None, None)
Mechanical Work	-1.5777e+08	False	(None, None)
Pressure Change	-2.1500e+06	False	(None, None)
Pressure Ratio	0.022727	False	(None, None)

Stream Table

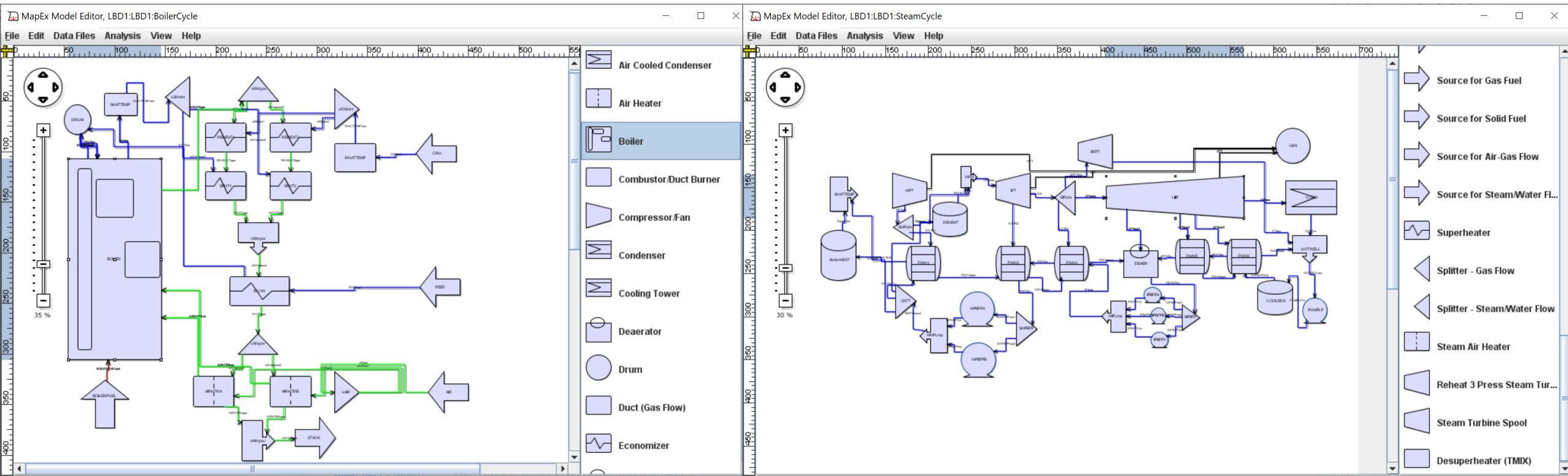
Stream	Inlet	Outlet
Molar Flow (mol/s)	10000.	10000.
Mass Flow (kg/s)	180.15	180.15
T (K)	837.00	398.56
P (Pa)	2.2000e+06	50000.
Vapor Fraction	1.0000	1.0000
Molar Enthalpy (J/mol) Vap	65000.	49223.
Molar Enthalpy (J/mol) Liq	65000.	9488.2



# Review Calculated Results (Boiler)



# Example: Model of a Coal-Fired-Power-Plant



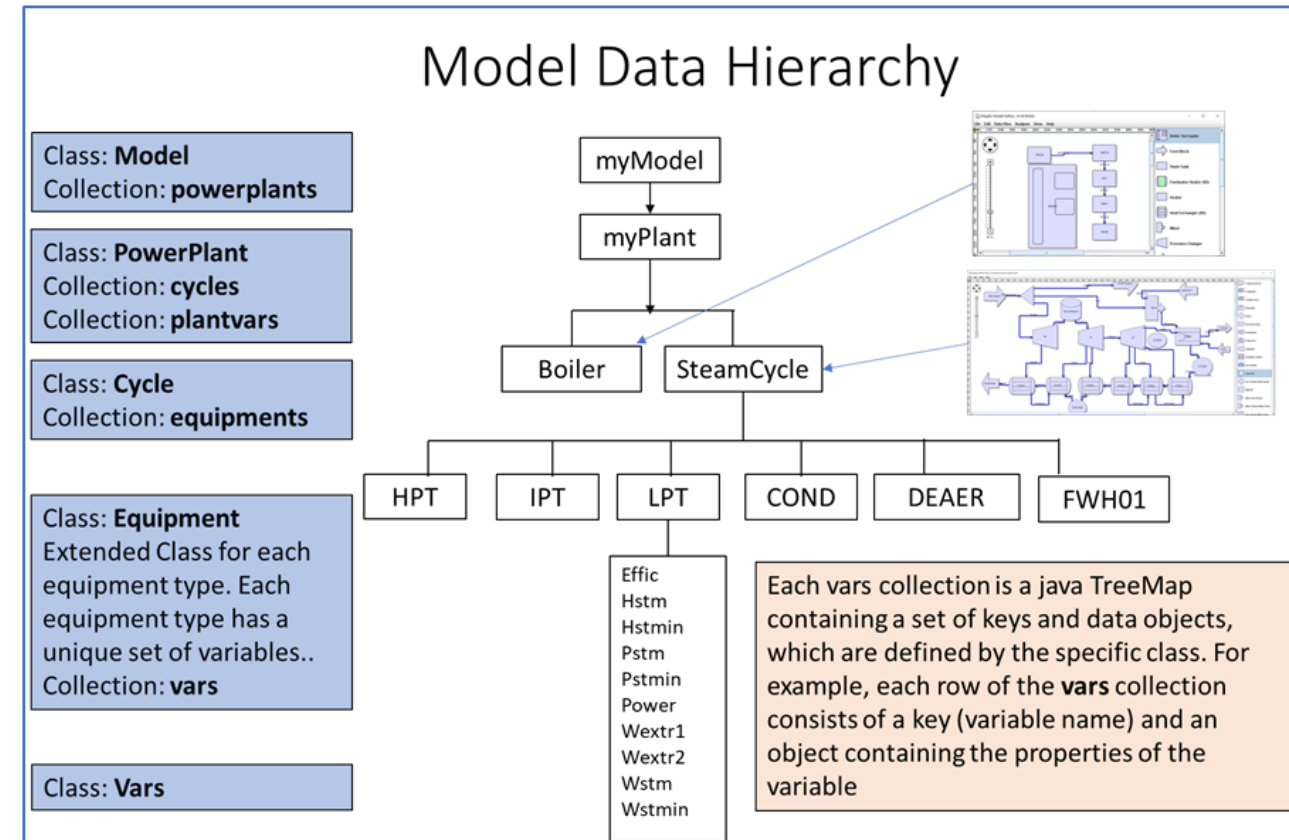
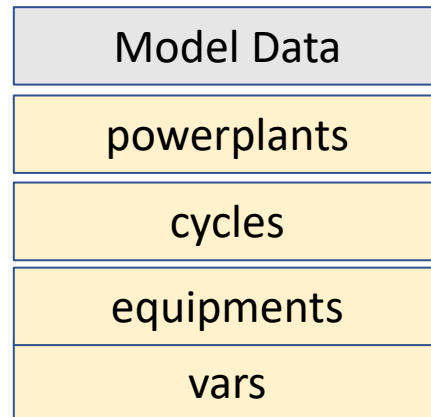
Boiler Cycle

Steam Cycle



# Model Data

- Model data is Hierarchical



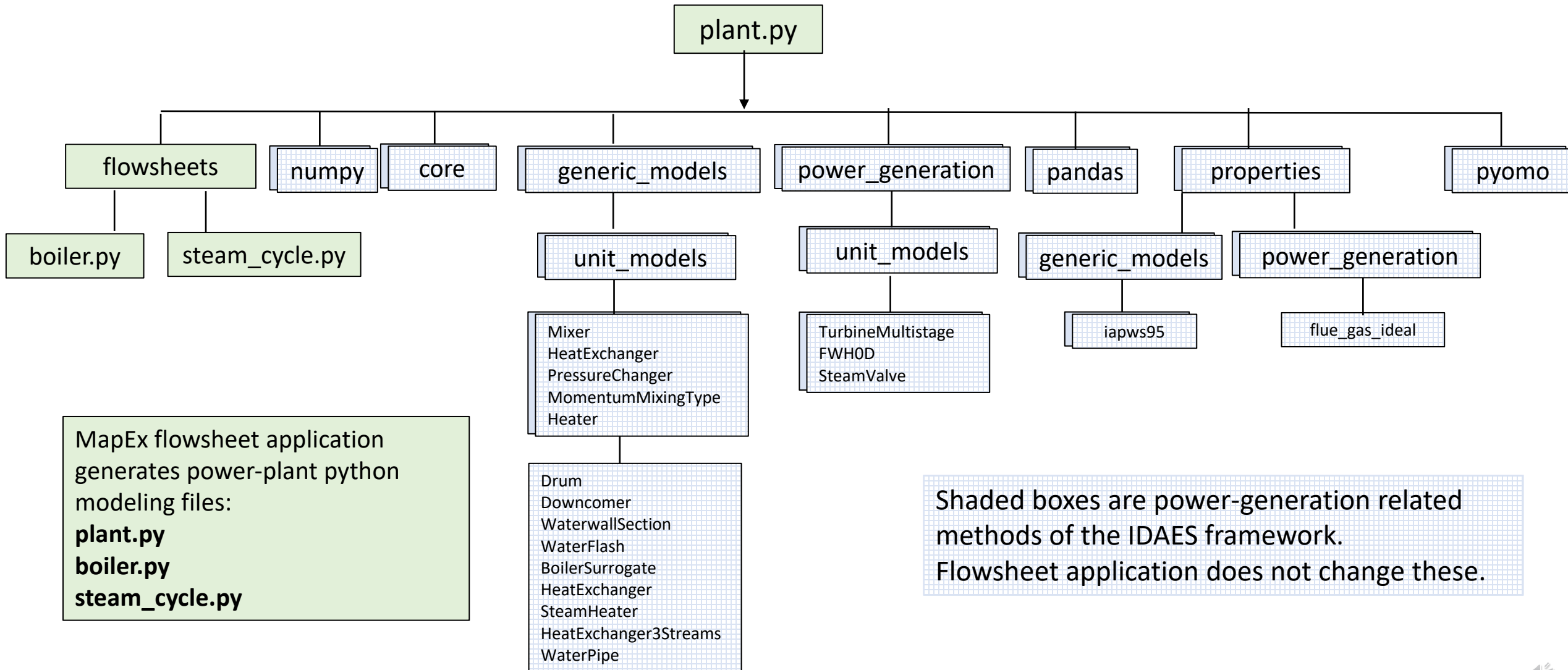
# Python Code File

The IDAES Python code file includes up to ten blocks of coding:

1. Import Python-IDAES-Pyomo Environment
2. Create Flowsheet and Pyomo Model
3. Add Unit Models
4. Add Arcs Connecting Unit Models
5. Set Model Parameters, Fix Variables and Initial Guesses
6. Add Custom User Variables/Constraints/Expressions
7. Initialize Unit Models
8. Solve Pyomo Model
9. Define Outputs
10. Write Results

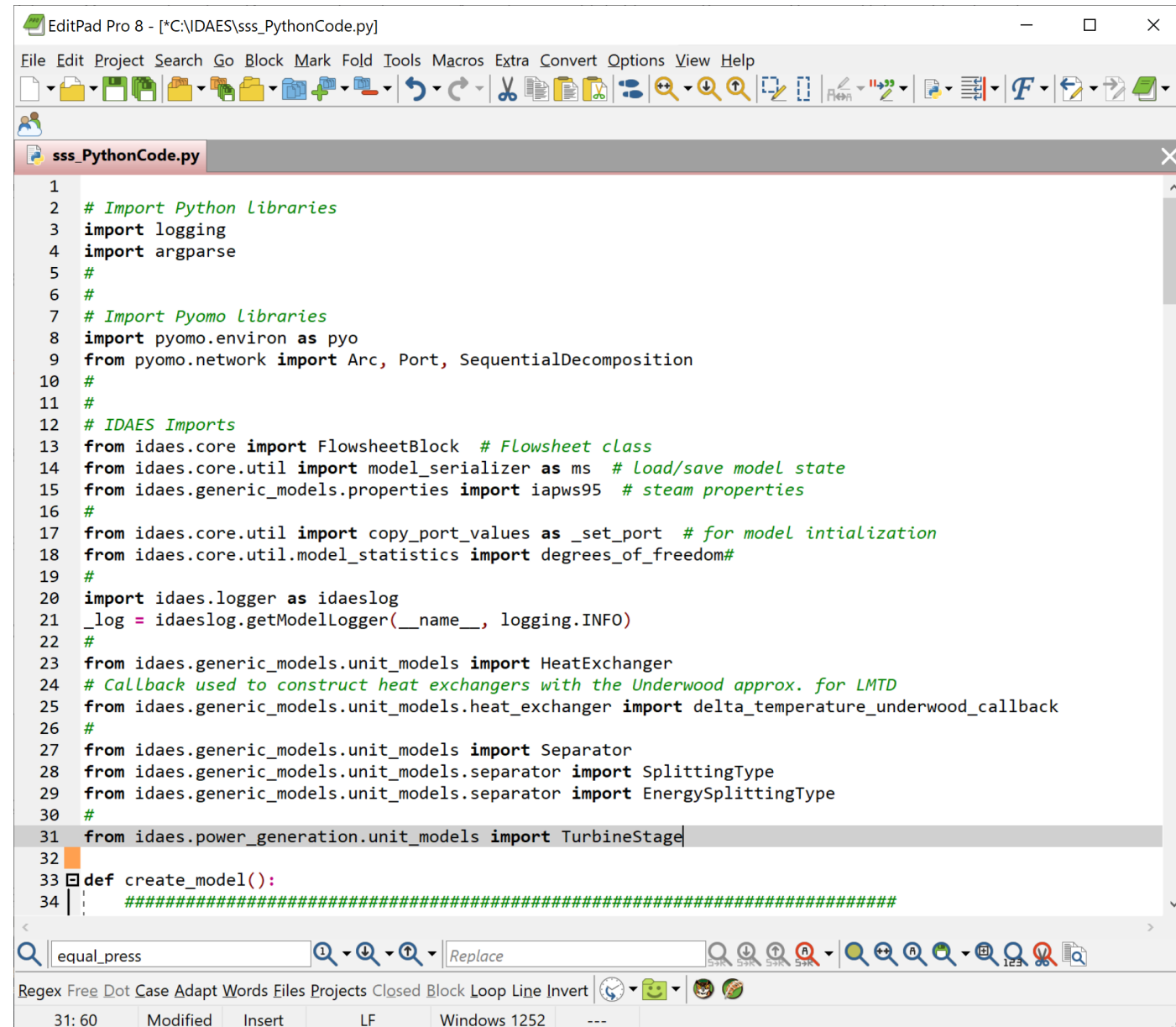


# IDAES Environment: Escalante Power Plant Model



# 1. Import Python-IDAES Environment

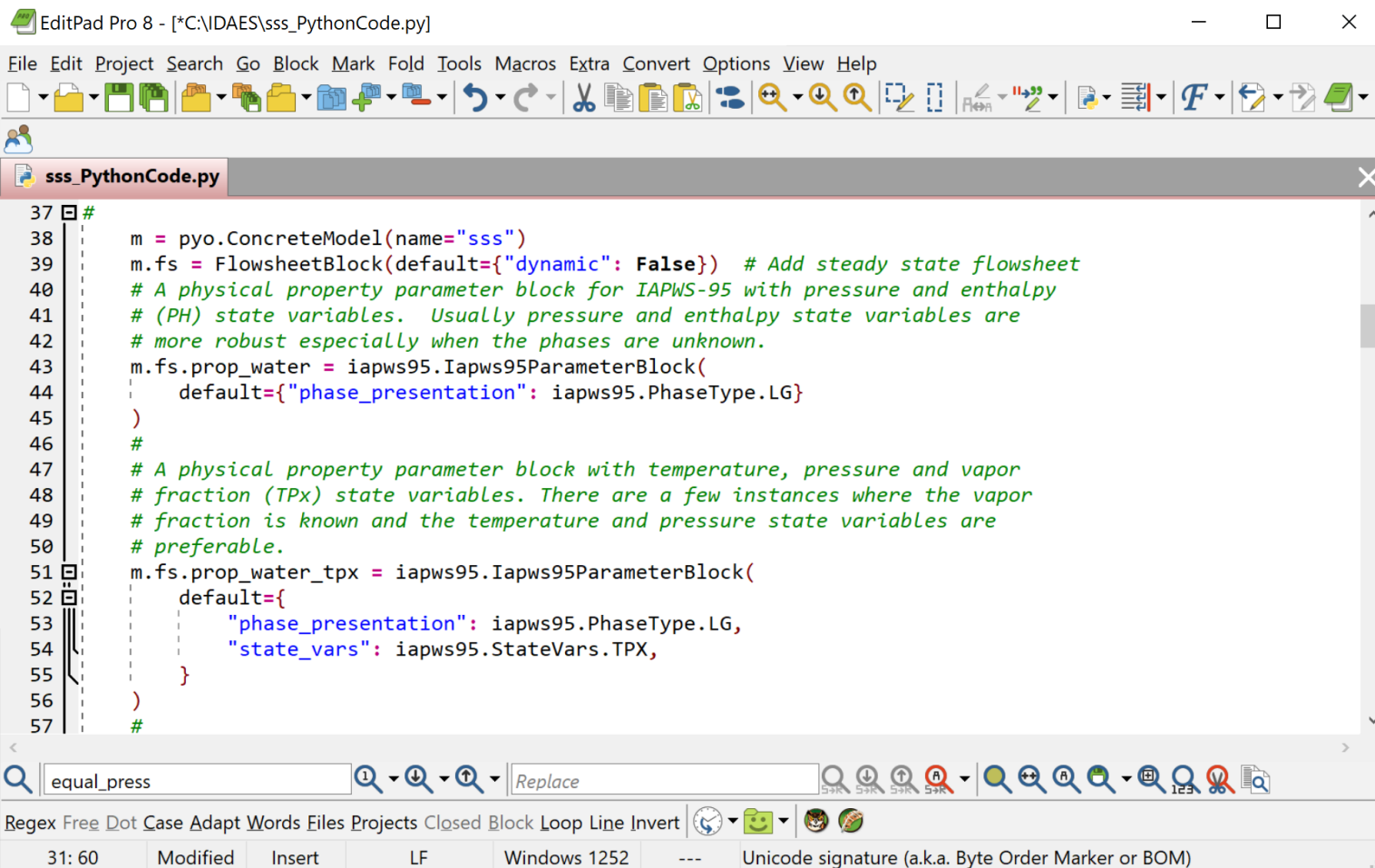
- Python import statements are required for IDAES classes
- Locations of IDAES files may change over time
  - IDAES models from previous releases of IDAES may not run with the current release
  - Backward compatibility will be desired in the future



```
1
2 # Import Python Libraries
3 import logging
4 import argparse
5 #
6 #
7 # Import Pyomo Libraries
8 import pyomo.environ as pyo
9 from pyomo.network import Arc, Port, SequentialDecomposition
10 #
11 #
12 # IDAES Imports
13 from idaes.core import FlowsheetBlock # Flowsheet class
14 from idaes.core.util import model_serializer as ms # Load/save model state
15 from idaes.generic_models.properties import iapws95 # steam properties
16 #
17 from idaes.core.util import copy_port_values as _set_port # for model initialization
18 from idaes.core.util.model_statistics import degrees_of_freedom#
19 #
20 import idaes.logger as idaeslog
21 _log = idaeslog.getModelLogger(__name__, logging.INFO)
22 #
23 from idaes.generic_models.unit_models import HeatExchanger
24 # Callback used to construct heat exchangers with the Underwood approx. for LMTD
25 from idaes.generic_models.unit_models.heat_exchanger import delta_temperature_underwood_callback
26 #
27 from idaes.generic_models.unit_models import Separator
28 from idaes.generic_models.unit_models.separator import SplittingType
29 from idaes.generic_models.unit_models.separator import EnergySplittingType
30 #
31 from idaes.power_generation.unit_models import TurbineStage
32
33 def create_model():
34     #####
```

## 2. Create Pyomo Model and Flowsheet

- Application creates a pyomo concrete model:  $m$
- Then Inserts a flowsheet:  $fs$
- Steam/water properties:  $iapws95$



```
37 #
38 m = pyo.ConcreteModel(name="sss")
39 m.fs = FlowsheetBlock(default={"dynamic": False}) # Add steady state flowsheet
40 # A physical property parameter block for IAPWS-95 with pressure and enthalpy
41 # (PH) state variables. Usually pressure and enthalpy state variables are
42 # more robust especially when the phases are unknown.
43 m.fs.prop_water = iapws95.Iapws95ParameterBlock(
44     default={"phase_presentation": iapws95.PhaseType.LG}
45 )
46 #
47 # A physical property parameter block with temperature, pressure and vapor
48 # fraction (TPx) state variables. There are a few instances where the vapor
49 # fraction is known and the temperature and pressure state variables are
50 # preferable.
51 m.fs.prop_water_tpx = iapws95.Iapws95ParameterBlock(
52     default={
53         "phase_presentation": iapws95.PhaseType.LG,
54         "state_vars": iapws95.StateVars.TPX,
55     }
56 )
57 #
```

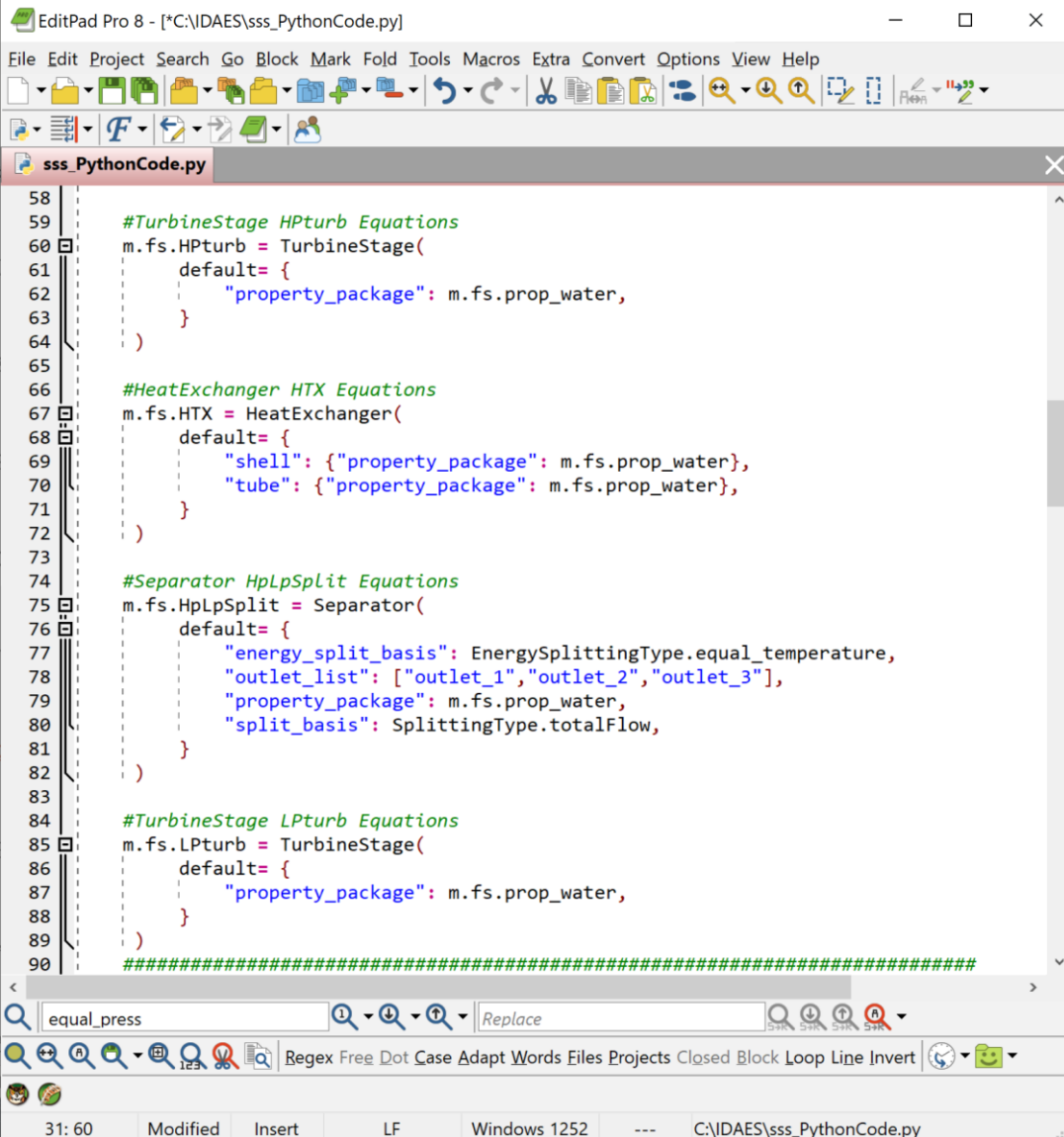
equal\_press

Regex Free Dot Case Adapt Words Files Projects Closed Block Loop Line Invert

31: 60 Modified Insert LF Windows 1252 --- Unicode signature (a.k.a. Byte Order Marker or BOM)

# 3. Add Unit Models to Flowsheet

- Unit Models:
  - TurbineStage
  - Separator
  - HeatExchanger
- Default options are supported



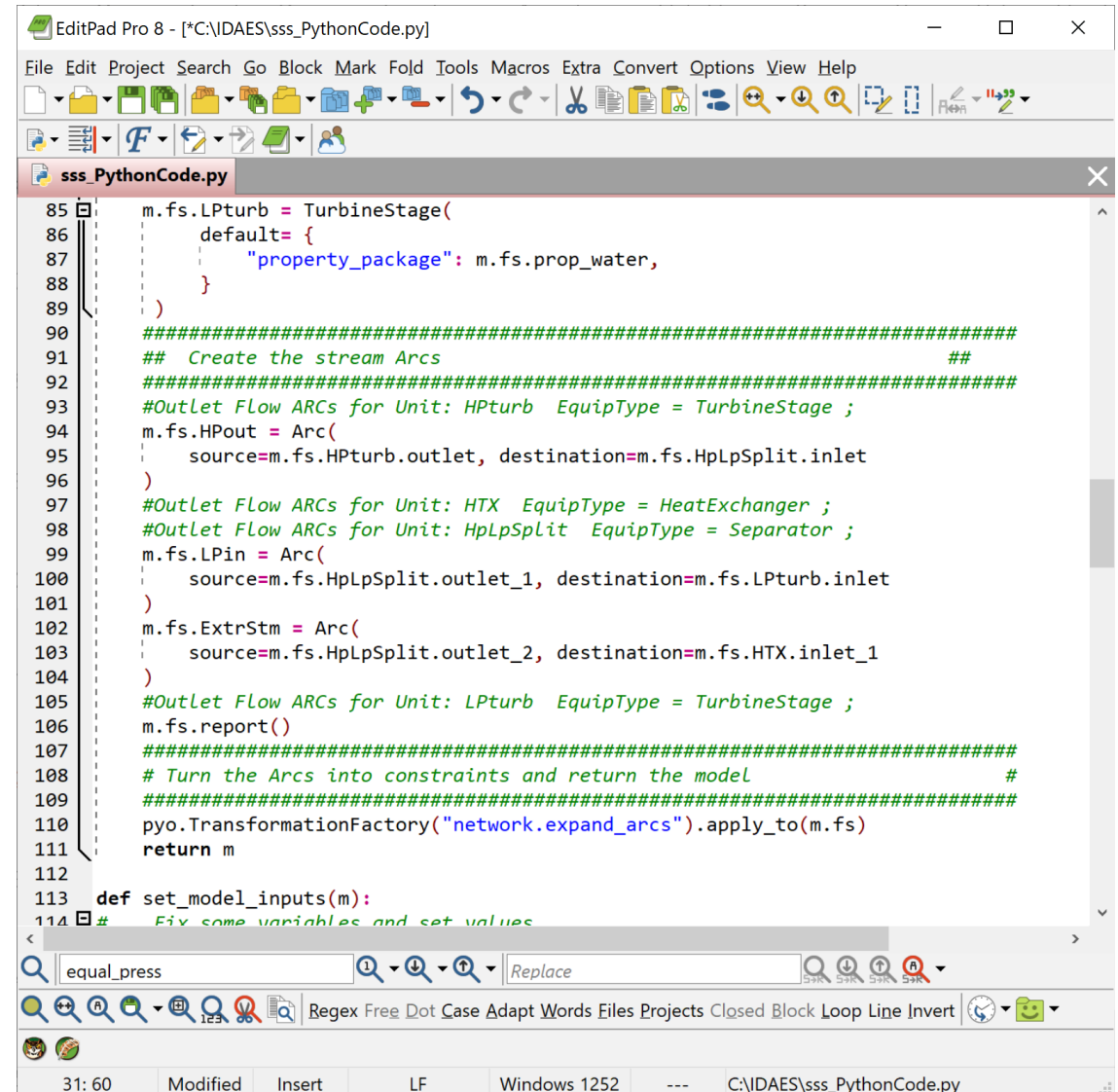
```
58
59 #TurbineStage HPturb Equations
60 m.fs.HPturb = TurbineStage(
61     default= {
62         "property_package": m.fs.prop_water,
63     }
64 )
65
66 #HeatExchanger HTX Equations
67 m.fs.HTX = HeatExchanger(
68     default= {
69         "shell": {"property_package": m.fs.prop_water},
70         "tube": {"property_package": m.fs.prop_water},
71     }
72 )
73
74 #Separator HpLpSplit Equations
75 m.fs.HpLpSplit = Separator(
76     default= {
77         "energy_split_basis": EnergySplittingType.equal_temperature,
78         "outlet_list": ["outlet_1", "outlet_2", "outlet_3"],
79         "property_package": m.fs.prop_water,
80         "split_basis": SplittingType.totalFlow,
81     }
82 )
83
84 #TurbineStage LPturb Equations
85 m.fs.LPturb = TurbineStage(
86     default= {
87         "property_package": m.fs.prop_water,
88     }
89 )
90 #####
```

The screenshot shows the EditPad Pro 8 interface with a Python file named `sss_PythonCode.py`. The code defines four unit models: `HPturb`, `HTX`, `HpLpSplit`, and `LPturb`. Each model is instantiated with a `default` dictionary specifying options like `property_package`, `energy_split_basis`, `outlet_list`, and `split_basis`. The `HpLpSplit` model includes a list of outlet names. The code is syntax-highlighted, and the window includes a standard menu bar and toolbar. At the bottom, a search bar contains the text `equal_press`, and the status bar shows the file path `C:\IDAES\sss_PythonCode.py`.



## 4. Add Arcs to Flowsheet

- User-specified arc names are used
- Application generates arcs using IDEAS port names

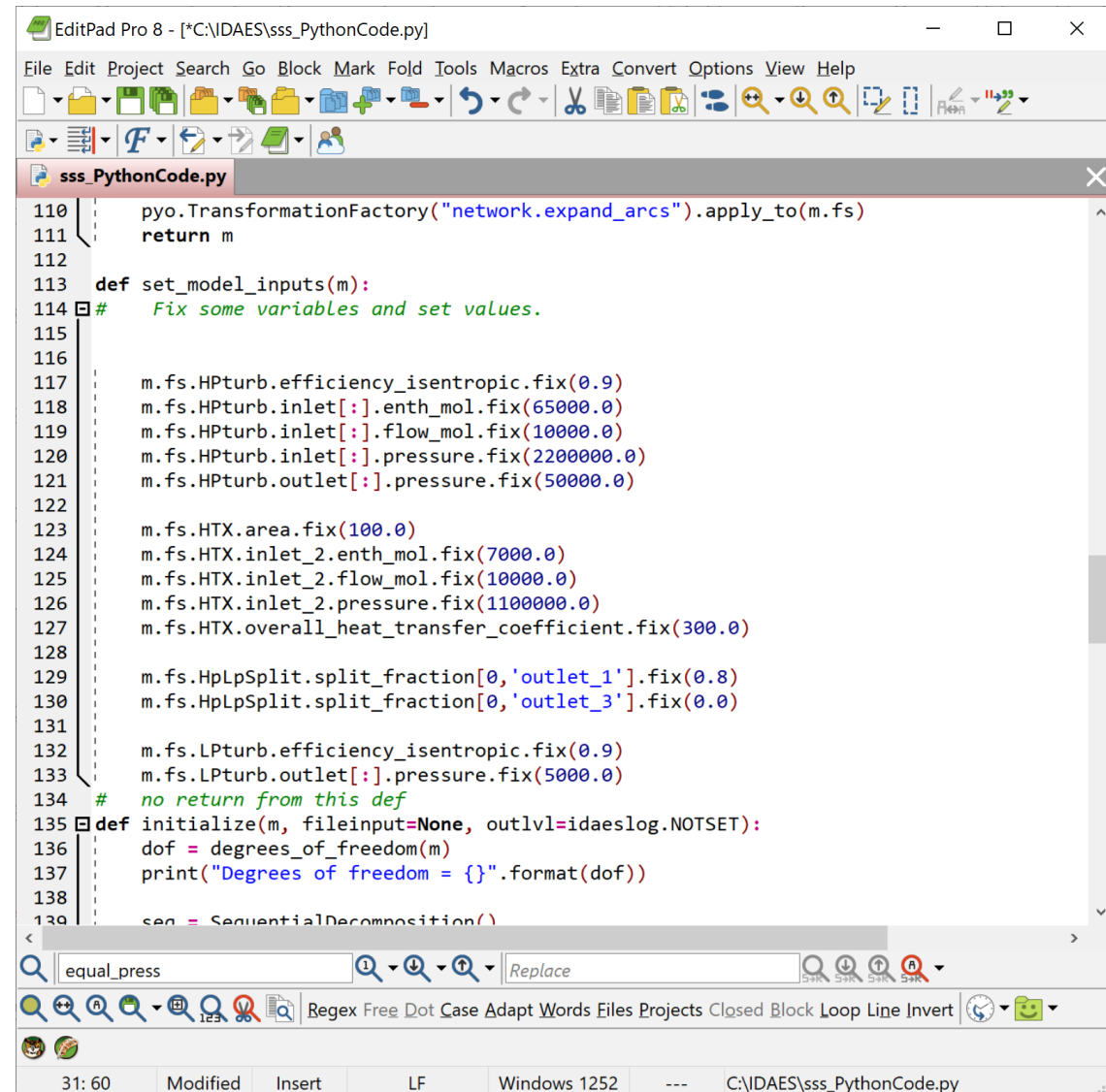


The screenshot shows the EditPad Pro 8 interface with a Python file named `sss_PythonCode.py`. The code defines a function `set_model_inputs(m)` that configures a flowsheet model `m`. It creates several arcs between units: `HPturb` to `HpLpSplit`, `HTX`, `LPin` to `LPturb`, and `ExtrStm`. The code also includes comments for creating stream arcs and turning them into constraints. The status bar at the bottom indicates the file is modified and the cursor is at line 31, column 60.

```
85 m.fs.LPturb = TurbineStage(  
86     default= {  
87         "property_package": m.fs.prop_water,  
88     }  
89 )  
90 #####  
91 ## Create the stream Arcs #####  
92 #####  
93 #Outlet Flow ARCs for Unit: HPturb EquipType = TurbineStage ;  
94 m.fs.HPout = Arc(  
95     source=m.fs.HPturb.outlet, destination=m.fs.HpLpSplit.inlet  
96 )  
97 #Outlet Flow ARCs for Unit: HTX EquipType = HeatExchanger ;  
98 #Outlet Flow ARCs for Unit: HpLpSplit EquipType = Separator ;  
99 m.fs.LPin = Arc(  
100     source=m.fs.HpLpSplit.outlet_1, destination=m.fs.LPturb.inlet  
101 )  
102 m.fs.ExtrStm = Arc(  
103     source=m.fs.HpLpSplit.outlet_2, destination=m.fs.HTX.inlet_1  
104 )  
105 #Outlet Flow ARCs for Unit: LPturb EquipType = TurbineStage ;  
106 m.fs.report()  
107 #####  
108 # Turn the Arcs into constraints and return the model #  
109 #####  
110 pyo.TransformationFactory("network.expand_arcs").apply_to(m.fs)  
111 return m  
112  
113 def set_model_inputs(m):  
114     # Fix some variables and set values
```

# 5. Fix Values of IDAES Inputs

- All values fixed by the user are fixed via Python code statements



The screenshot shows the EditPad Pro 8 interface with a Python script titled "sss\_PythonCode.py". The script defines a function `set_model_inputs(m)` that fixes various parameters of a model `m`. The code is as follows:

```
110 pyo.TransformationFactory("network.expand_arcs").apply_to(m.fs)
111 return m
112
113 def set_model_inputs(m):
114     # Fix some variables and set values.
115
116
117     m.fs.HPturb.efficiency_isentropic.fix(0.9)
118     m.fs.HPturb.inlet[:].enth_mol.fix(65000.0)
119     m.fs.HPturb.inlet[:].flow_mol.fix(10000.0)
120     m.fs.HPturb.inlet[:].pressure.fix(2200000.0)
121     m.fs.HPturb.outlet[:].pressure.fix(50000.0)
122
123     m.fs.HTX.area.fix(100.0)
124     m.fs.HTX.inlet_2.enth_mol.fix(7000.0)
125     m.fs.HTX.inlet_2.flow_mol.fix(10000.0)
126     m.fs.HTX.inlet_2.pressure.fix(1100000.0)
127     m.fs.HTX.overall_heat_transfer_coefficient.fix(300.0)
128
129     m.fs.HpLpSplit.split_fraction[0,'outlet_1'].fix(0.8)
130     m.fs.HpLpSplit.split_fraction[0,'outlet_3'].fix(0.0)
131
132     m.fs.LPturb.efficiency_isentropic.fix(0.9)
133     m.fs.LPturb.outlet[:].pressure.fix(5000.0)
134     # no return from this def
135 def initialize(m, fileinput=None, outlvl=idaeslog.NOTSET):
136     dof = degrees_of_freedom(m)
137     print("Degrees of freedom = {}".format(dof))
138
139     seq = SequentialDecomposition()
```

The bottom of the window shows a search bar with the text "equal\_press", a status bar with "31: 60", "Modified", "Insert", "LF", "Windows 1252", and the file path "C:\IDAES\sss\_PythonCode.py".

## 6. Define New Pyomo Variables and Constraints

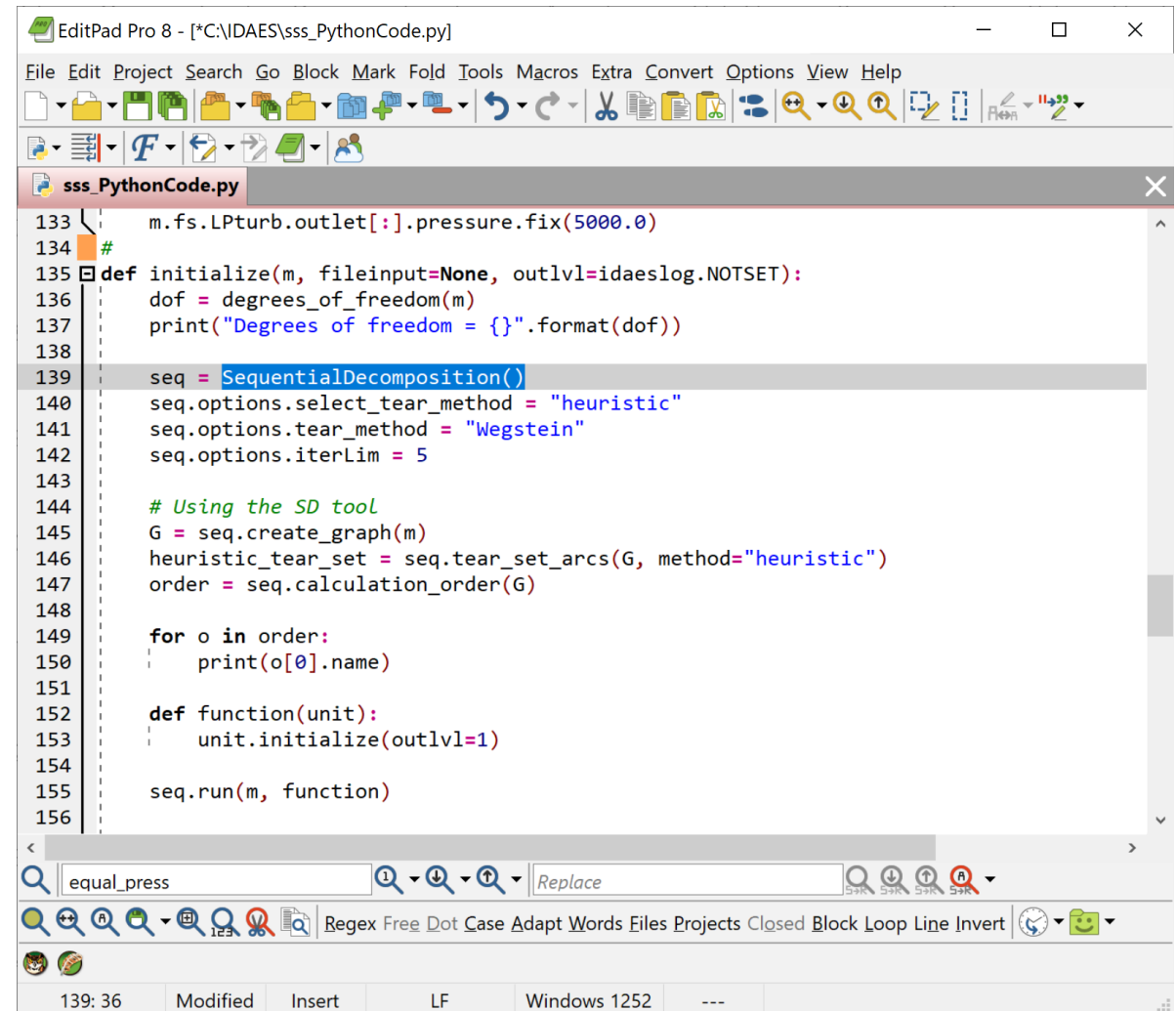
- Add Variable TTD to Feedwater heater

```
# add a variable for FWH1 TTD
m.fs.fwh1.ttd = pyo.Var(
    m.fs.time,
    initialize=10,
    doc="TTD for FWH (condensing T - BFW exit T). (C)",
)
# Add a constraint for FWH1 TTD
@m.fs.fwh1.Constraint(m.fs.time)
def set_ttd(b, t):
    return (
        b.condense.tube.properties_out[t].temperature
        == b.condense.shell.properties_in[t].temperature_sat - b.ttd[t]
    )

#Fix the value of TTD in the IDAES model
# fwh1 inputs
m.fs.fwh1.ttd.fix(10)
```

# 7. Initialization

- Three possible methods
  - Serial Decomposition
  - Data from file
  - Custom path from icon to icon
- Current software uses Sequential Decomposition
- Other methods will be added soon



The screenshot shows the EditPad Pro 8 interface with a file named `ssss_PythonCode.py` open. The code is a Python script for initializing a model. It starts with a line `m.fs.LPturb.outlet[:].pressure.fix(5000.0)` on line 133. Line 134 is a comment `#`. Line 135 defines a function `initialize(m, fileinput=None, outlvl=idaeslog.NOTSET):`. Inside this function, line 136 calculates `dof = degrees_of_freedom(m)` and line 137 prints `"Degrees of freedom = {}".format(dof)`. Line 138 is a comment. Line 139 creates a `seq = SequentialDecomposition()` object. Line 140 sets `seq.options.select_tear_method = "heuristic"`. Line 141 sets `seq.options.tear_method = "Wegstein"`. Line 142 sets `seq.options.iterLim = 5`. Line 143 is a comment `# Using the SD tool`. Line 144 creates a graph `G = seq.create_graph(m)`. Line 145 sets `heuristic_tear_set = seq.tear_set_arcs(G, method="heuristic")`. Line 146 sets `order = seq.calculation_order(G)`. Line 147 is a comment. Line 148 is a comment. Line 149 starts a `for o in order:` loop. Line 150 prints `o[0].name`. Line 151 is a comment. Line 152 defines a function `def function(unit):`. Line 153 calls `unit.initialize(outlvl=1)`. Line 154 is a comment. Line 155 calls `seq.run(m, function)`. Line 156 is a comment. The status bar at the bottom shows `139: 36`, `Modified`, `Insert`, `LF`, `Windows 1252`, and `---`.

```
133 m.fs.LPturb.outlet[:].pressure.fix(5000.0)
134 #
135 def initialize(m, fileinput=None, outlvl=idaeslog.NOTSET):
136     dof = degrees_of_freedom(m)
137     print("Degrees of freedom = {}".format(dof))
138
139     seq = SequentialDecomposition()
140     seq.options.select_tear_method = "heuristic"
141     seq.options.tear_method = "Wegstein"
142     seq.options.iterLim = 5
143
144     # Using the SD tool
145     G = seq.create_graph(m)
146     heuristic_tear_set = seq.tear_set_arcs(G, method="heuristic")
147     order = seq.calculation_order(G)
148
149     for o in order:
150         print(o[0].name)
151
152     def function(unit):
153         unit.initialize(outlvl=1)
154
155     seq.run(m, function)
156
```

# Summary

- Flowsheet based GUI is functional
  - Desktop software provides flowsheet-based user interface for building IDAES models
  - Flowsheet is graphical representation of the model (picture is worth a thousand words)
  - Building IDAES models is easier and faster
  - User is freed from writing Python code
- Some Desired Features Not Yet Implemented
  - Application does not yet automatically run Python code or retrieve results
  - Initialization needs to be improved
  - User defined variables/constraints/expressions not yet implemented
- Various Modes of analysis will be implemented in Phase II
  - Data Reconciliation
  - Parameter Estimation
  - Optimization
  - Dynamic Predictive Analysis and Optimization



**Acknowledgment:**

"This material is based upon work supported by the Department of Energy Award Number DE-SC0020794."

**Disclaimer:**

"This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof."