Creating Unit Models

To demonstrate creation of a new unit model, we will create a constant-heat-capacity ideal-gas isentropic compressor. This will be a simple textbook model. We will utilize the mass and energy balances provided by IDAES control volumes, but we will write our own isentropic constraint based off of equations 7.18 and 7.23 from "Introduction to Chemical Engineering Thermodynamics" by J.M. Smith, H.C. Van Ness, and M.M. Abbott.

The outlet temperature of an ideal gas undergoing isentropic compression is given by

$$t_{out} = t_{in} + \frac{1}{\eta}\left(t_{in}\left(\frac{p_{out}}{p_{in}}\right)^{\frac{\gamma-1}{\gamma}} - t_{in}\right)$$

where p is pressure, t is temperature, and γ is the ratio of constant pressure heat capacity to constant volume heat capacity.

We will begin with relevant imports. We will need

- Pyomo for writing our energy balance constraints
- ConfigBlocks for specifying options for our compressor
- ControlVolume0DBlocks for creating the appropriate state blocks for the inlet and outlet and for defining mas balances
- IdealParameterBlock which provides a simple ideal-gas property package.
- A few other helpful functions and enums from IDAES

In [1]:

```
import pyomo.environ as pe
from pyomo.common.config import ConfigBlock, ConfigValue, In
from idaes.core import (ControlVolume0DBlock,
                        declare_process_block_class,
                        EnergyBalanceType,
                        MomentumBalanceType,
                        MaterialBalanceType,
                        UnitModelBlockData,
                        useDefault,
                        FlowsheetBlock)
from idaes.core.util.config import is_physical_parameter_block
from methanol_param_VLE import PhysicalParameterBlock
from idaes.core.util.misc import add_object_reference
```

Now, we can write a function to create a control volume for our compressor. The control volume will define the inlet and outlet streams along with the appropriate state variables (specified by the property package). We will also use the control volume to create mass and energy balance constraints.

Our function will take the compressor unit model object, the name of the control volume, and configuration options as arguments. Our compressor will only support steady-state models, so we will first ensure that `dynamic` and `has_holdup` are both `False` .

Next, we will create a 0D control volume. We are using a 0D control volume because our model does not depend on space. We then

1. Attach the control volume to the compressor
2. Create the appropriate state blocks with the control volume (for the inlet and outlet streams)
3. Use the control volume to add mass balance constraints
4. Use the control volume to add energy balance constraints

In [2]:

```
def make_control_volume(unit, name, config):
    if config.dynamic is not False:
        raise ValueError('IdealGasIsentropcCompressor does not support d
    if config.has_holdup is not False:
        raise ValueError('IdealGasIsentropcCompressor does not support h

    control_volume = ControlVolume0DBlock(default={"property_package": c
                                                   "property_package_arg

    setattr(unit, name, control_volume)

    control_volume.add_state_blocks(has_phase_equilibrium=config.has_pha
    control_volume.add_material_balances(balance_type=config.material_ba
                                         has_phase_equilibrium=config.ha
    control_volume.add_total_enthalpy_balances(has_heat_of_reaction=Fals
                                               has_heat_transfer=False,
                                               has_work_transfer=True)
```

Next, we will write a function to add constraints to specify that the compressor is isentropic.

1. Create a `pressure_ratio` variable to represent p_{out}/p_{in}. The lower bound is 1, because we only want to allow compression (and not expansion).
2. Create a `ConstraintList` to hold the constraints.
3. Add the `ConstraintList` to the compressor
4. Create the local variables `inlet` and `outlet` to reference the inlet and outlet state blocks.
5. Add a constraint relating the inlet pressure, outlet pressure, and pressure ratio variables:
$$p_{in} p_{ratio} = p_{out}$$
6. Add a constraint relating the inlet and outlet temperatures:
$$t_{out} = t_{in} + \frac{1}{\eta}\left(t_{in} p_{ratio}^{\frac{\gamma-1}{\gamma}} - t_{in}\right)$$

In [3]:
```
def add_isentropic(unit, name, config):
    unit.pressure_ratio = pe.Var(initialize=1.0, bounds=(1, None))
    cons = pe.ConstraintList()
    setattr(unit, name, cons)
    inlet = unit.control_volume.properties_in[0.0]
    outlet = unit.control_volume.properties_out[0.0]
    gamma = inlet._params.gamma
    cons.add(inlet.pressure * unit.pressure_ratio == outlet.pressure)
    cons.add(outlet.temperature ==
             (inlet.temperature +
              1/config.compressor_efficiency *
              (inlet.temperature * unit.pressure_ratio**((gamma - 1) / g
               inlet.temperature)))
```

We also need a function to specify configuration options for the compressor.

In [4]:
```
def make_compressor_config_block(config):
    config.declare("material_balance_type", ConfigValue(default=Material
    config.declare("energy_balance_type", ConfigValue(default=EnergyBala
    config.declare("momentum_balance_type", ConfigValue(default=Momentum
    config.declare("has_phase_equilibrium", ConfigValue(default=False, d
    config.declare("has_pressure_change", ConfigValue(default=False, dom
    config.declare("property_package", ConfigValue(default=useDefault, d
    config.declare("property_package_args", ConfigBlock(implicit=True))
    config.declare("compressor_efficiency", ConfigValue(default=0.75, do
```

Finally, we can define the ideal-gas isentropic compressor. To do so, we create a class called `IdealGasIsentropicCompressorData` and use the `declare_process_block_class` decorator. For now, just consider the decorator to be boiler-plate. We then need to define the config block and write the `build` method. The `build` method should alwasy call `super`. Next, we simply call the functions we wrote to build the control volume, energy balance, and electricity requirement performance equation. Finally, we need to call `self.add_inlet_port()` and `self.add_outlet_port()`. These methods need to be called in order to create the ports which are used for connecting the unit to other units.

In [5]:
```
@declare_process_block_class("IdealGasIsentropicCompressor")
class IdealGasIsentropicCompressorData(UnitModelBlockData):
    CONFIG = UnitModelBlockData.CONFIG()
    make_compressor_config_block(CONFIG)

    def build(self):
        super(IdealGasIsentropicCompressorData, self).build()

        make_control_volume(self, "control_volume", self.config)
        add_isentropic(self, "isentropic", self.config)

        self.add_inlet_port()
        self.add_outlet_port()

        add_object_reference(self, 'work', self.control_volume.work[0.0]
```

The compressor model is complete and can now be used like other IDAES unit models:

In [6]:
```
m = pe.ConcreteModel()
m.fs = fs = FlowsheetBlock(default={"dynamic": False})
fs.properties = props = PhysicalParameterBlock(default={'Cp': 0.038056,

fs.compressor = IdealGasIsentropicCompressor(default={"property_package"
                                                     "has_phase_equilib
fs.compressor.inlet.flow_mol.fix(1)
fs.compressor.inlet.mole_frac[0, 'CH3OH'].fix(0.25)
fs.compressor.inlet.mole_frac[0, 'CH4'].fix(0.25)
fs.compressor.inlet.mole_frac[0, 'H2'].fix(0.25)
fs.compressor.inlet.mole_frac[0, 'CO'].fix(0.25)
fs.compressor.inlet.pressure.fix(0.14)
fs.compressor.inlet.temperature.fix(2.9315)
fs.compressor.outlet.pressure.fix(0.56)

opt = pe.SolverFactory('ipopt')
opt.options['linear_solver'] = 'mumps'
res = opt.solve(m, tee=False)
print(res.solver.termination_condition)
fs.compressor.outlet.display()
print('work: ', round(fs.compressor.work.value, 2), ' MJ')
```

```
optimal
outlet : Size=1
    Key  : Name        : Value
    None :    flow_mol : {0.0: 1.0}
         :   mole_frac : {(0.0, 'CH3OH'): 0.25, (0.0, 'CH4'): 0.25, (0
.0, 'CO'): 0.25, (0.0, 'H2'): 0.25}
         :    pressure : {0.0: 0.56}
         : temperature : {0.0: 4.314094393272423}
work:  5.26  MJ
```

In [ ]: