

# Tutorial

## Modelica Buildings Library and and

# Best Practices for Modeling of Thermofluid Flow Systems

Michael Wetter  
Simulation Research Group

September 23, 2020



**Lawrence Berkeley National Laboratory**

Overview  
of  
Modelica Buildings Library

# Primary use of Buildings library

## Main applications

- Model repository for building and district energy simulation, see <https://lbl-srg.github.io/soep/>
- Development of 5th generation district heating and cooling
- Controls design and performance evaluation.
- Repository of control sequences in the Control Description Language, <https://obc.lbl.gov/>.
- Model-based design process.
- Development and testing of FDD algorithms.

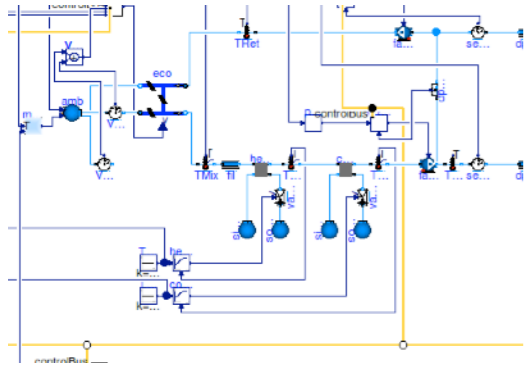
## License

- All development is open-source under BSD.

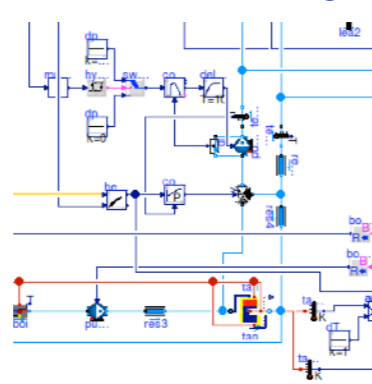
# Model repository: Modelica Buildings Library.

## Open-source repository of 1000+ models and functions.

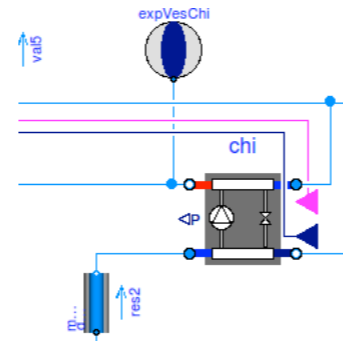
Air-based HVAC



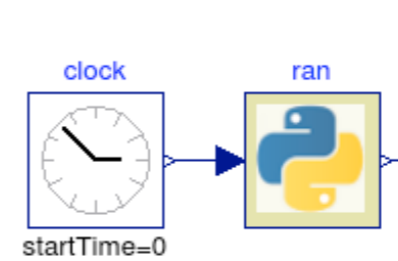
Hydronic heating



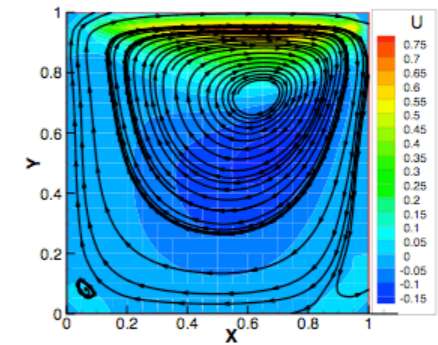
Chiller plants



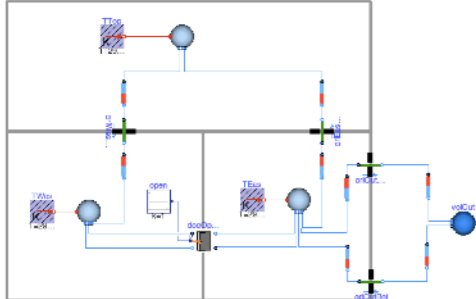
Embedded Python



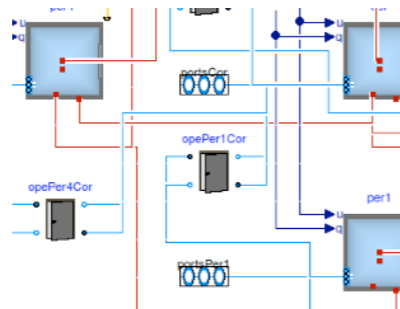
Room air flow



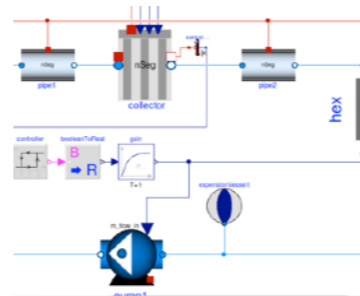
Natural ventilation, multizone air exchange, contaminant transport



Room heat transfer, incl. window (TARCOG)



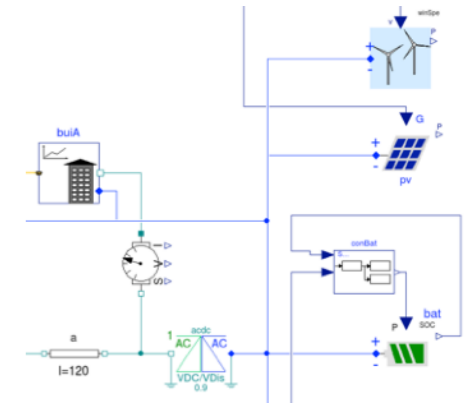
Solar collectors



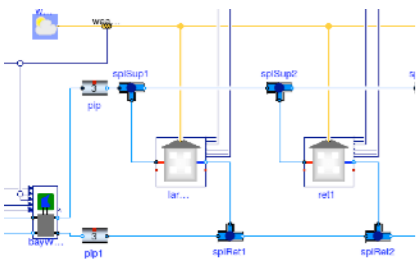
FLEXLAB



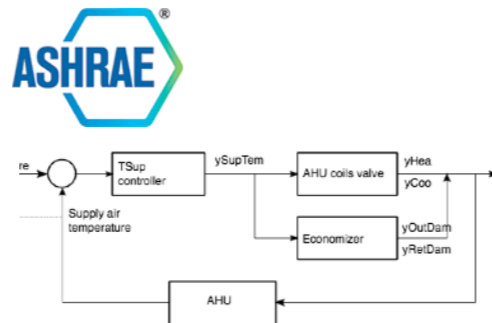
Electrical systems



District heating and cooling systems



Control design & deployment, including ASHRAE G36



### Current developments



Make it the core of the Spawn of EnergyPlus.

Use for real-time building control (OpenBuildingControl)

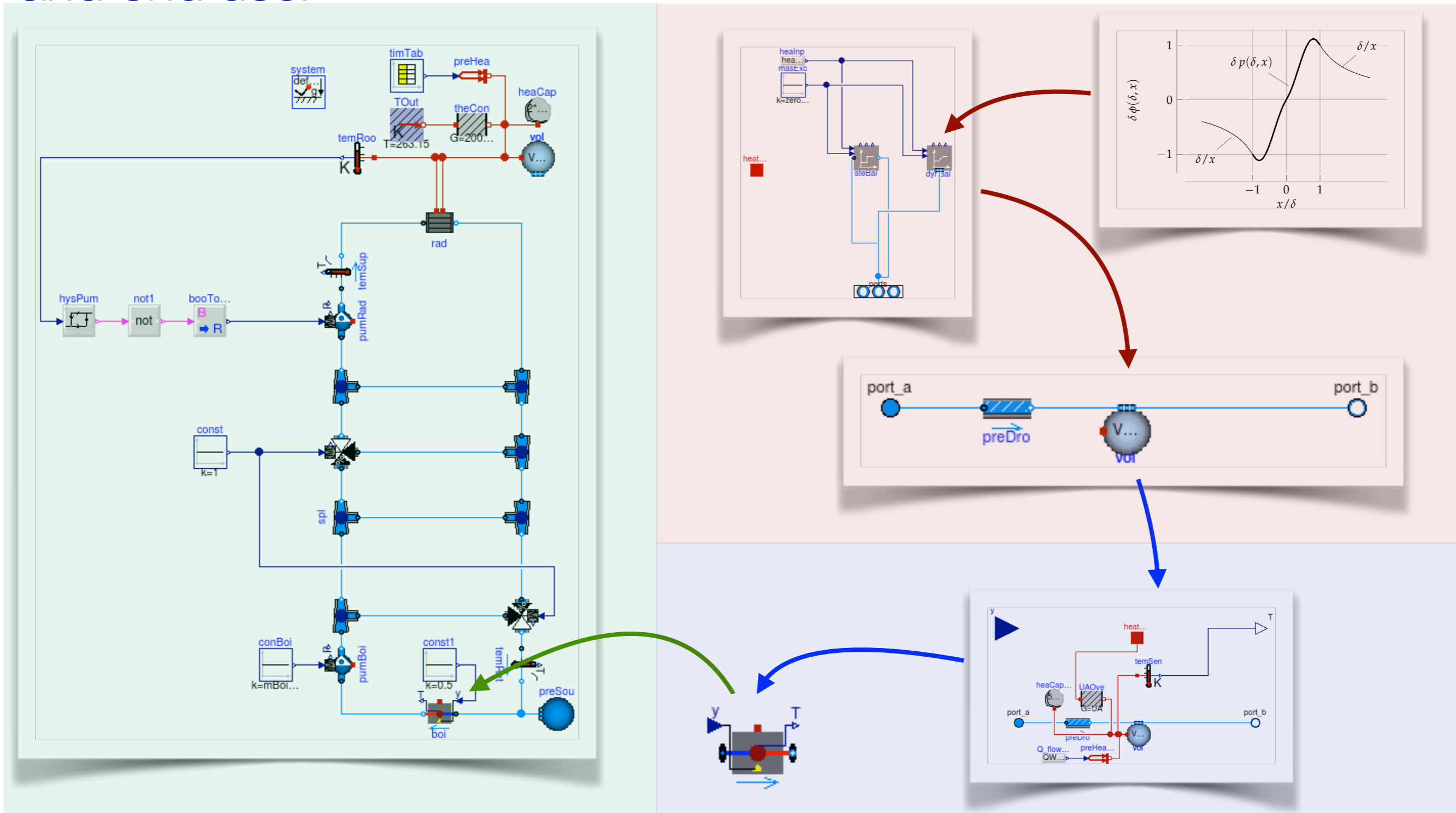
Emulators for testing and comparison of advanced building control sequences, including MPC (BOPTTEST)






Co-develop with IBPSA Modelica library, including district heating and cooling systems

[simulationresearch.lbl.gov/modelica](http://simulationresearch.lbl.gov/modelica)

# Separation between library developer, component developer and end user



## Legend:

-  Library developer
-  Component developer
-  End user

# Main modeling assumptions

<b>Media</b>	Can track moisture (X) and contaminants (C).
<b>HVAC equipment</b>	Most equipment based on performance curve, or based on nominal conditions and similarity laws. Refrigerant is not modeled. Most equipment optional steady-state or 1st order transient.
<b>Flow resistances</b>	Based on $m_{\text{flow\_nominal}}$ and $dp_{\text{nominal}}$ plus similarity law. Optional flag to linearize or to set $dp=0$ .
<b>Room model</b>	Any number of constructions are possible. Layer-by-layer window model (similar to Window 6). Optional flag to linearize radiation and/or convection.
<b>Electrical systems</b>	DC. AC 1-phase and 3-phase (dq, dq0). Quasi-stationary or dynamic phase angle (but not frequency).

# Documentation and distribution

## Documentation

- General [user guide](#) (getting started, best practice, developer instructions, ...).
- 19 [user guides](#) for individual packages.
- 3 [tutorials](#) with step-by-step instructions.
- All models contain an “info” section.
- Small test models for all classes, large test cases for “smoke tests,” and various validation cases.

## Distribution

- For users:  
<http://simulationresearch.lbl.gov/modelica>
- For developers:  
<https://github.com/lbl-srg/modelica-buildings>

**Buildings**

- UsersGuide
- Air
- Airflow
- Applications
- BoundaryConditions
- Controls
- Electrical
- Fluid
  - UsersGuide
  - Actuators
    - UsersGuide
    - Dampers
      - Exponential
      - MixingBox
      - MixingBoxMinimumFlow
      - PressureIndependent**
    - Examples
    - Validation
  - Motors

**INFORMATION**

Model for an air damper whose airflow is proportional to the input signal, assuming that at  $y = 1$ ,  $m\_flow = m\_flow\_nominal$ . This is unless the pressure difference  $dp$  is too low, in which case a  $k_{Dam} = m\_flow\_nominal / \sqrt{dp\_nominal}$  characteristic is used.

The model is similar to [Buildings.Fluid.Actuators.Valves.TwoWayPressureIndependent](#), except for adaptations for damper parameters. Please see that documentation for more information.

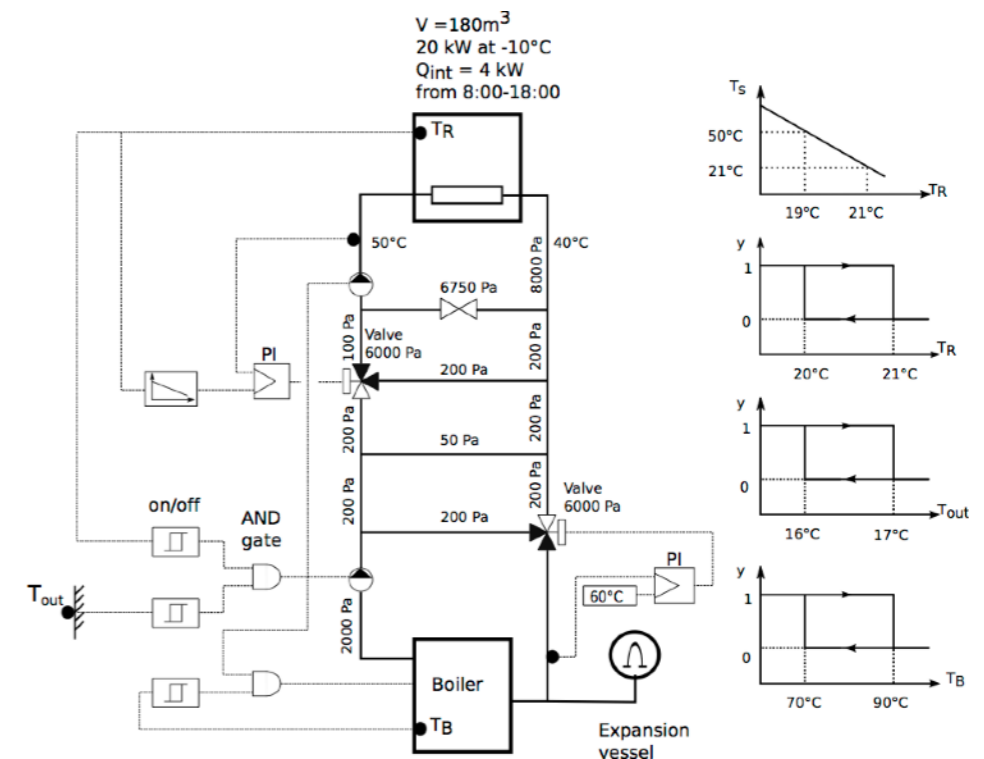
**Computation of the damper opening**

The fractional opening of the damper is computed by

- inverting the quadratic flow function to compute the flow coefficient from the flow rate and the pressure drop values (under the assumption of a turbulent flow regime);
- inverting the exponential characteristics to compute the fractional opening from the loss coefficient value (directly derived from the flow coefficient).

The quadratic interpolation used outside the exponential domain in the function [Buildings.Fluid.Actuators.BaseClasses.exponentialDamper](#) yields a local extremum. Therefore, the formal inversion of the function is not possible. A cubic spline is used instead to fit the inverse of the damper characteristics. The central domain of the characteristics having a monotonous exponential profile, its inverse can be properly approximated with three equidistant support points. However, the quadratic functions used outside of the exponential domain can have various profiles depending on the damper coefficients. Therefore, five linearly distributed support points are used on each side domain to ensure a good fit of the inverse.

Note that below a threshold value of the input control signal (fixed at 0.02), the fractional opening is forced to zero and no more related to the actual flow coefficient of the damper. This avoids steep transients of the



# Best practice and modeling hints

See also

<https://simulationresearch.lbl.gov/modelica/userGuide/bestPractice.html>



# Building large system models

## 1. Understand the problem:

1. What question do you want to answer?
2. Know what you want to model.
  1. Draw system schematics.
  2. Identify control input.
  3. Draw the control loops.
  4. Determine the control sequences.

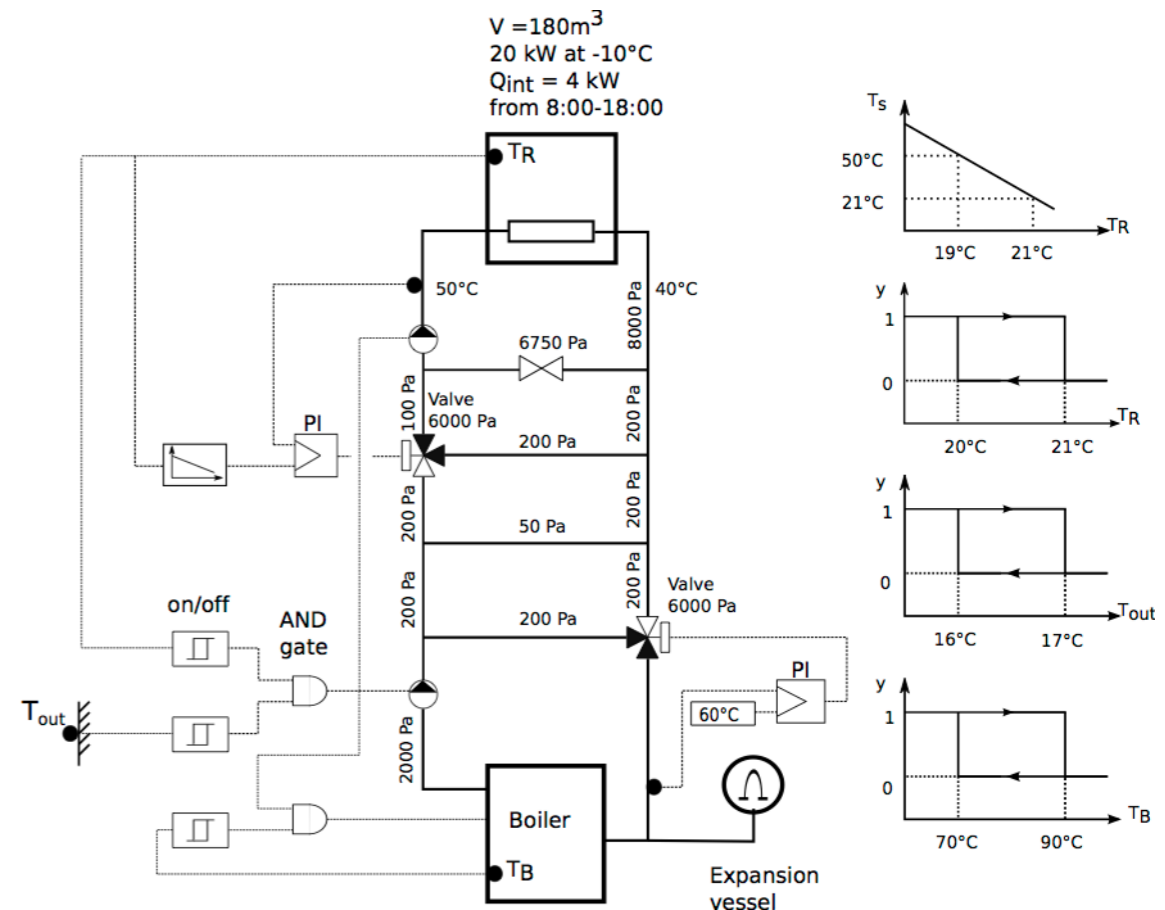
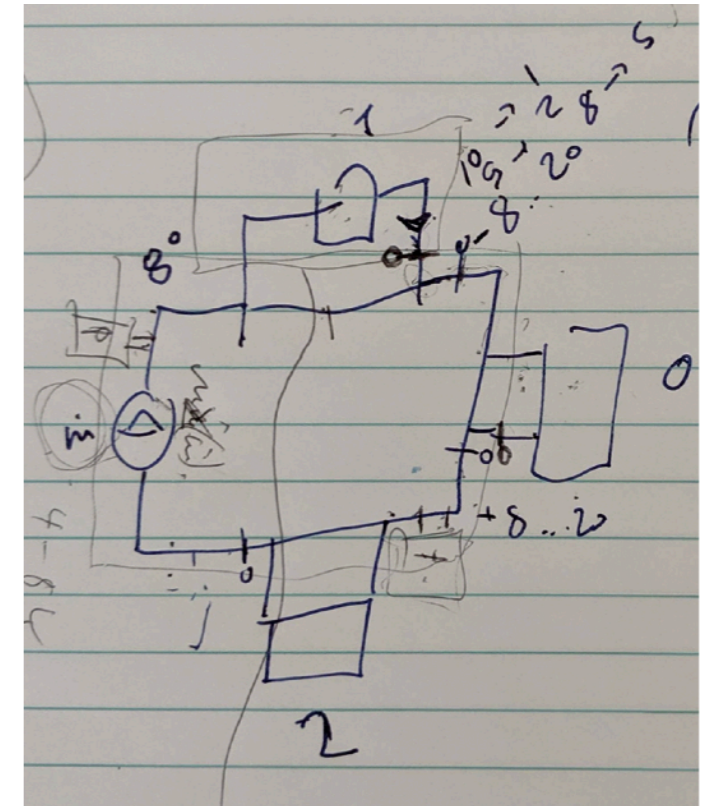
## 2. Compartmentalize: Split the system into subcomponents that can be tested in isolation.

## 3. Implement: Now, and only now, start implementing in software.

1. Document and build test cases as you go along.

Errors are easy to detect in small models, but hard in large models. If you add unit tests, you make sure what has been tested remains intact as the model evolves.

2. Assemble the subcomponents to build the full model.

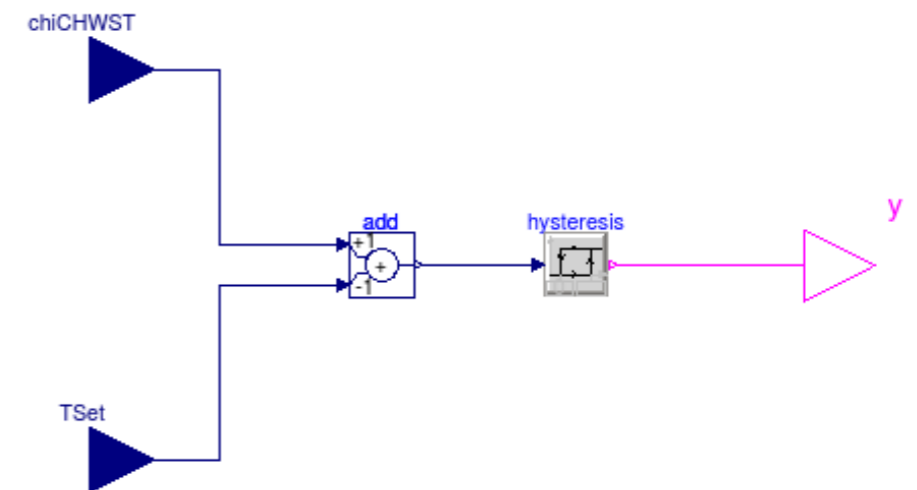


# Building large system models

How do you build and debug a large system model?

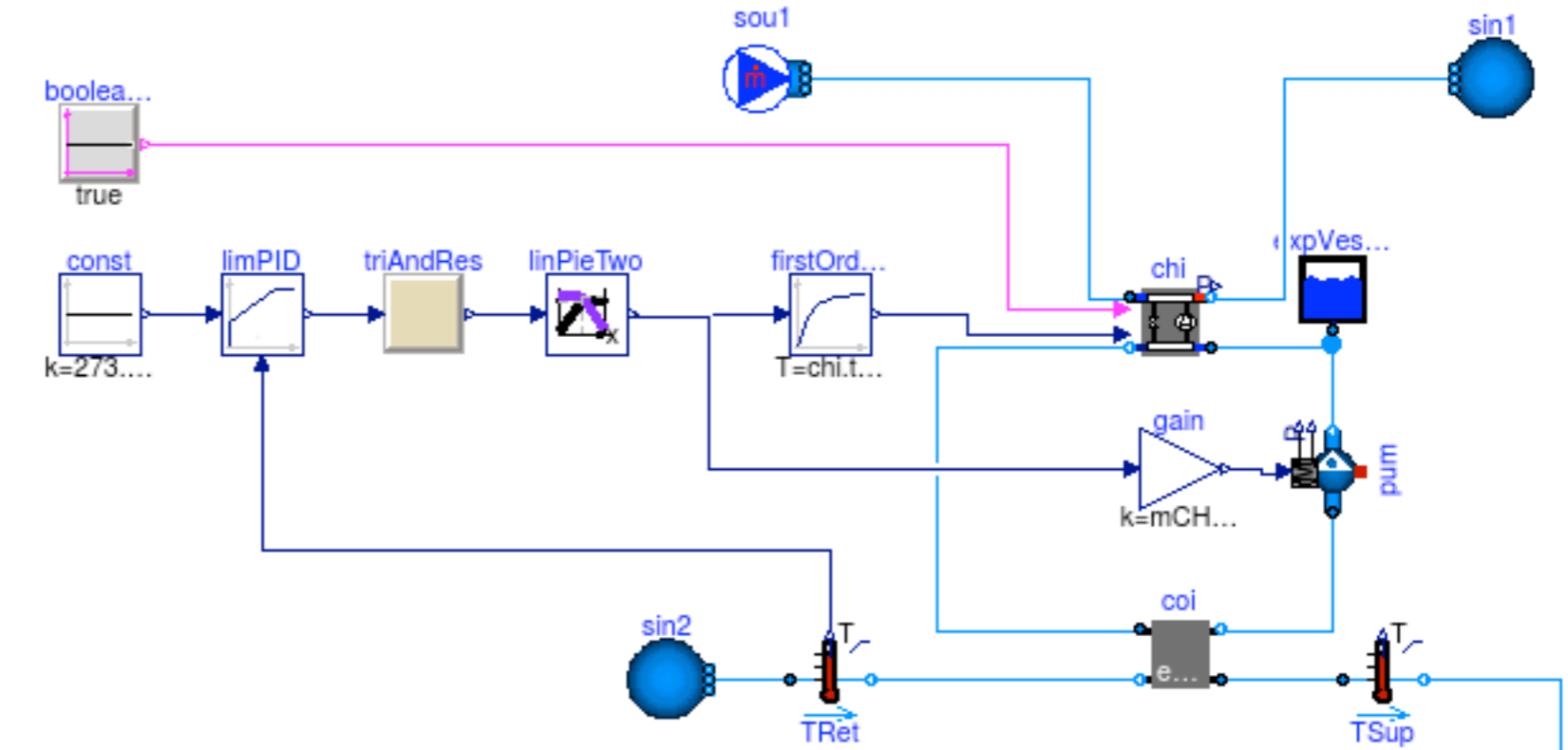
1. Split the model into small models — or better, architect the large model from the beginning to be based on smaller models
2. Test the smaller models for well known conditions.
3. Add smaller models to unit tests.

For example, see [Chiller Plant](#),  
in which each small models contains a simple unit test.

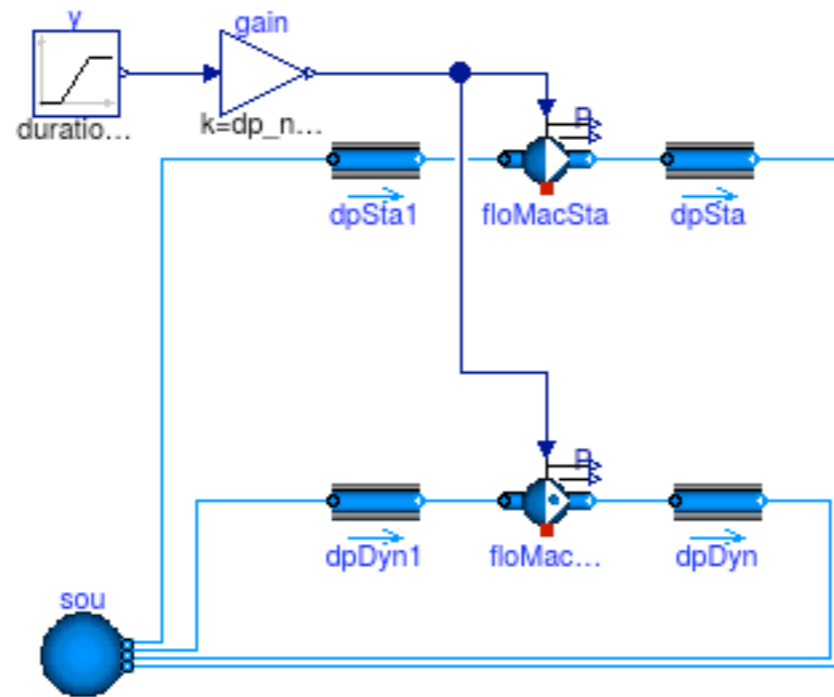


# Use small unit tests, as in

## Chiller plant base classes



## Pumps



# Don't Repeat Yourself: Propagate common parameters

Don't assign the same values to multiple parameters:

```
Pump pum(m_flow_nominal=0.1) "Pump";  
TemperatureSensor sen(m_flow_nominal=0.1) "Sensor";
```

Instead, propagate parameters and assign the value once:

```
Modelica.SIunits.MassFlowRate m_flow_nominal = 0.1  
  "Nominal mass flow rate";  
Pump pum(final m_flow_nominal=m_flow_nominal) "Pump";  
TemperatureSensor sen(final m_flow_nominal=m_flow_nominal) "Sensor";
```

Assignments can include computations, such as

```
Modelica.SIunits.HeatFlowRate QHea_nominal = 3000  
  "Nominal heating power";  
Modelica.SIunits.TemperatureDifference dT = 10  
  "Nominal temperature difference";  
Modelica.SIunits.MassFlowRate m_flow_nominal = QHea_nominal/dT/4200  
  "Nominal mass flow rate";  
...
```

# Don't Repeat Yourself:

Always define the media at the top-level

Top-level system-model

```
replaceable package Medium = Buildings.Media.Air  
  "Medium model";
```

Propagate medium to instance of model

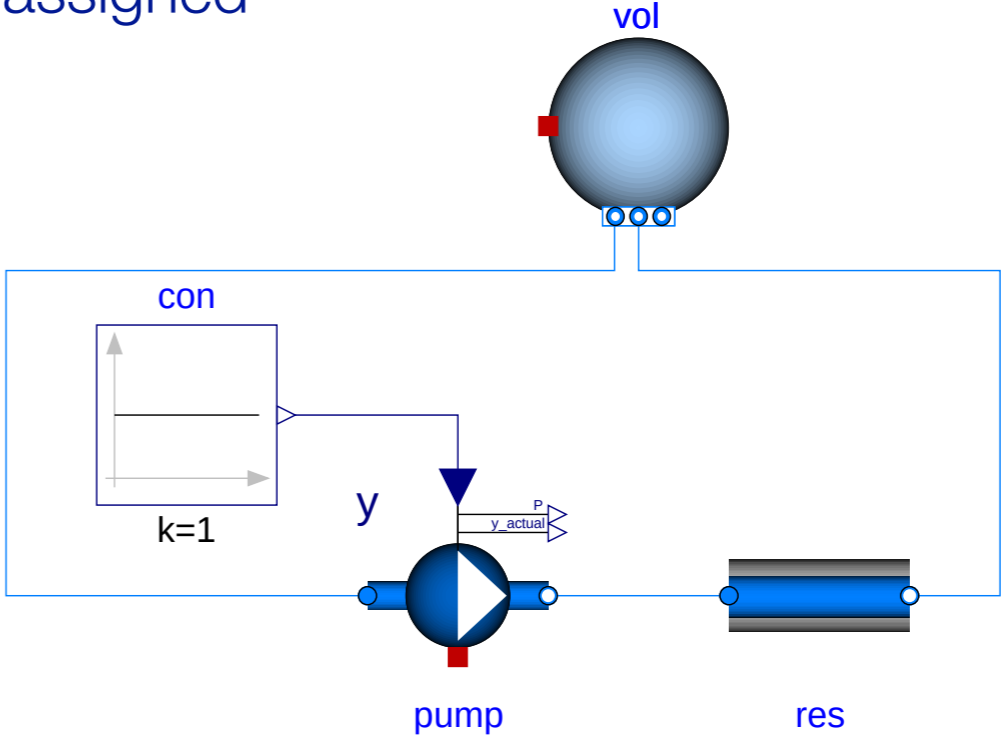
```
TemperatureSensor sen(  
  redeclare final package Medium = Medium,  
  final m_flow_nominal=m_flow_nominal) "Sensor";
```

Note: For arrays of parameters, use the **each** keyword, as in

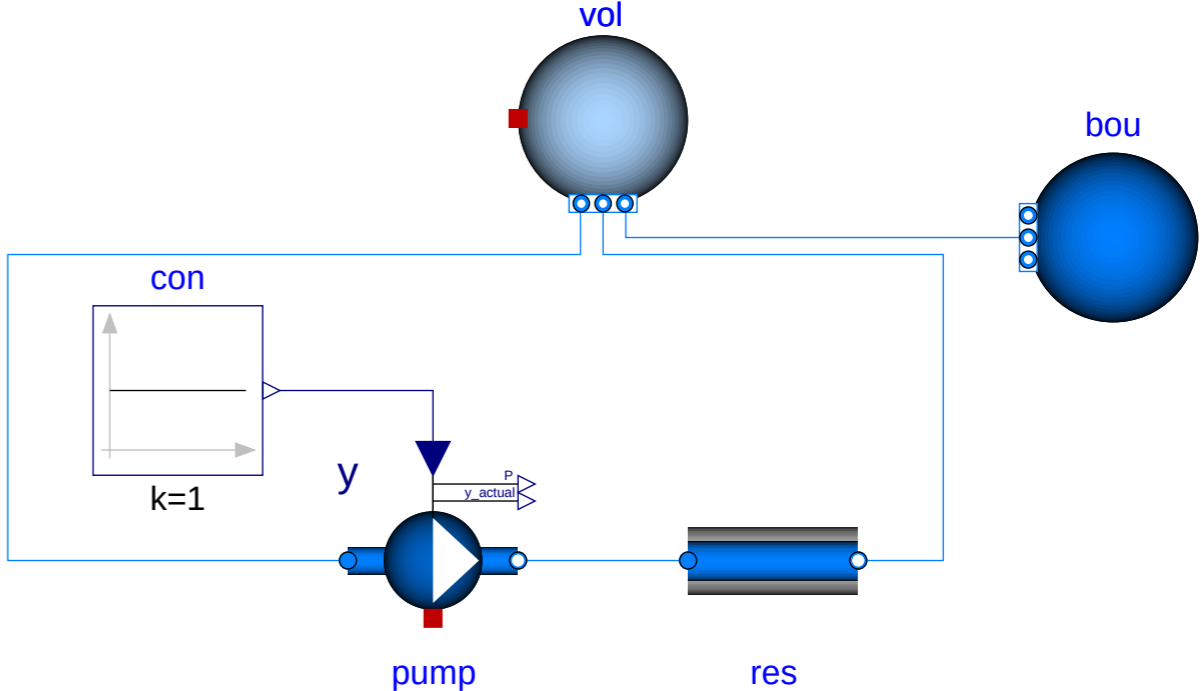
```
TemperatureSensor sen[2](  
  each final m_flow_nominal=m_flow_nominal)  
  "Sensor";
```

# All system models must have a reference pressure

Underdetermined model as no pressure state is assigned

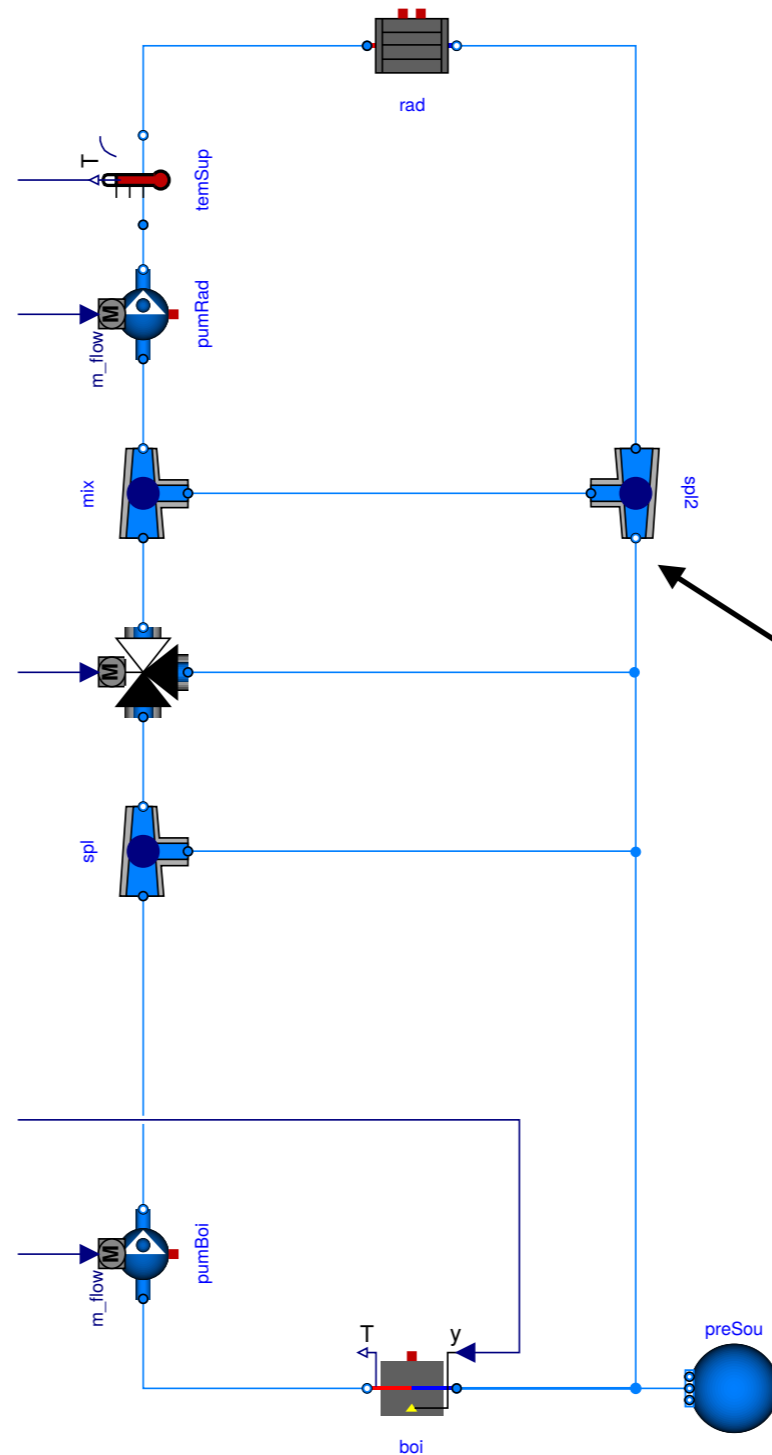


Model that provides a reference pressure through the instance **bou**.



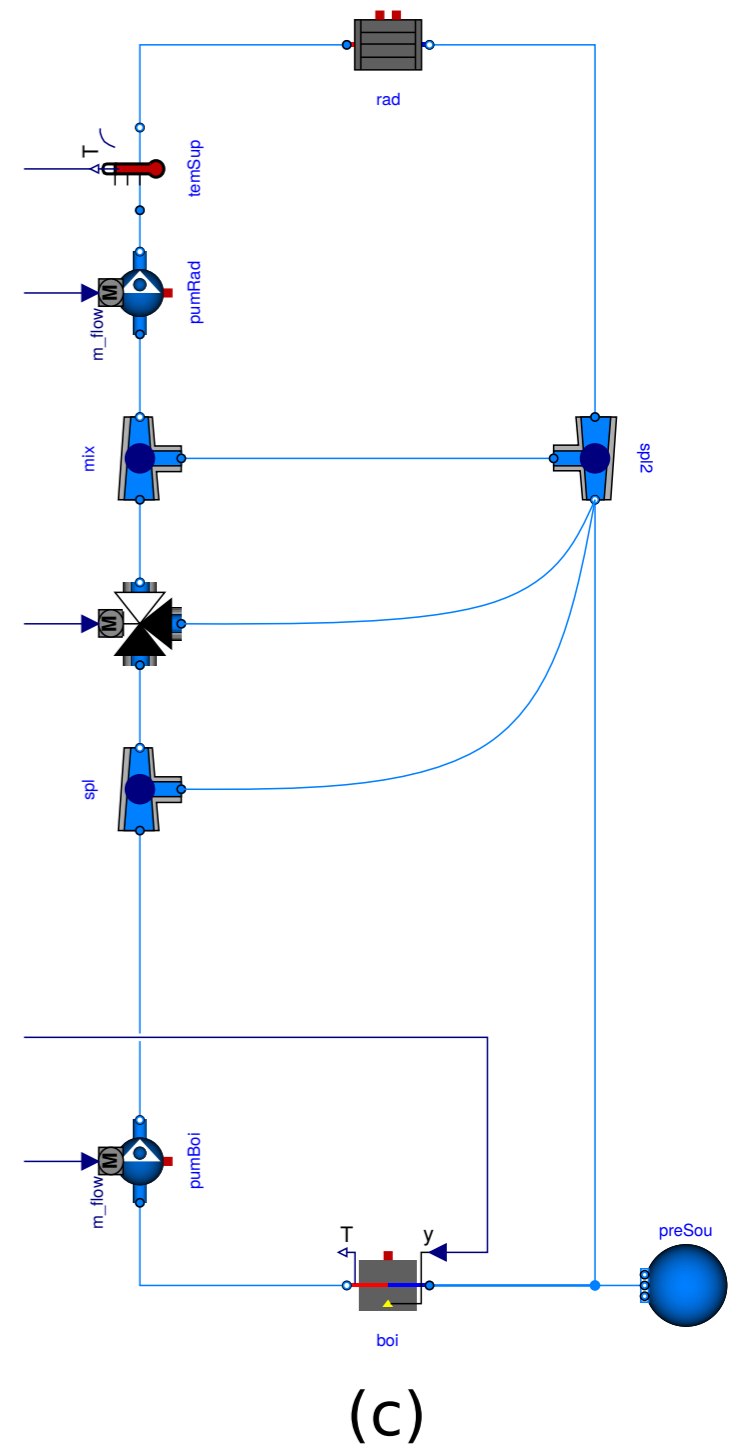
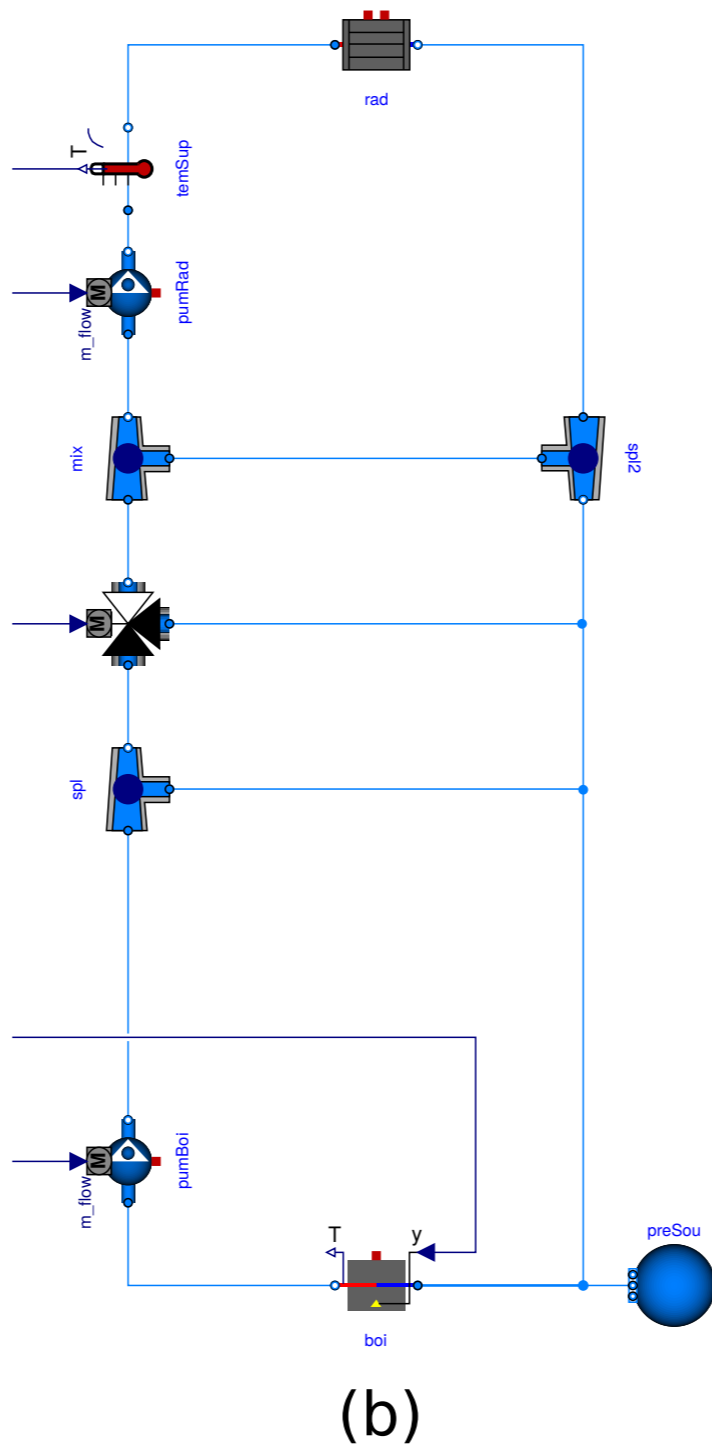
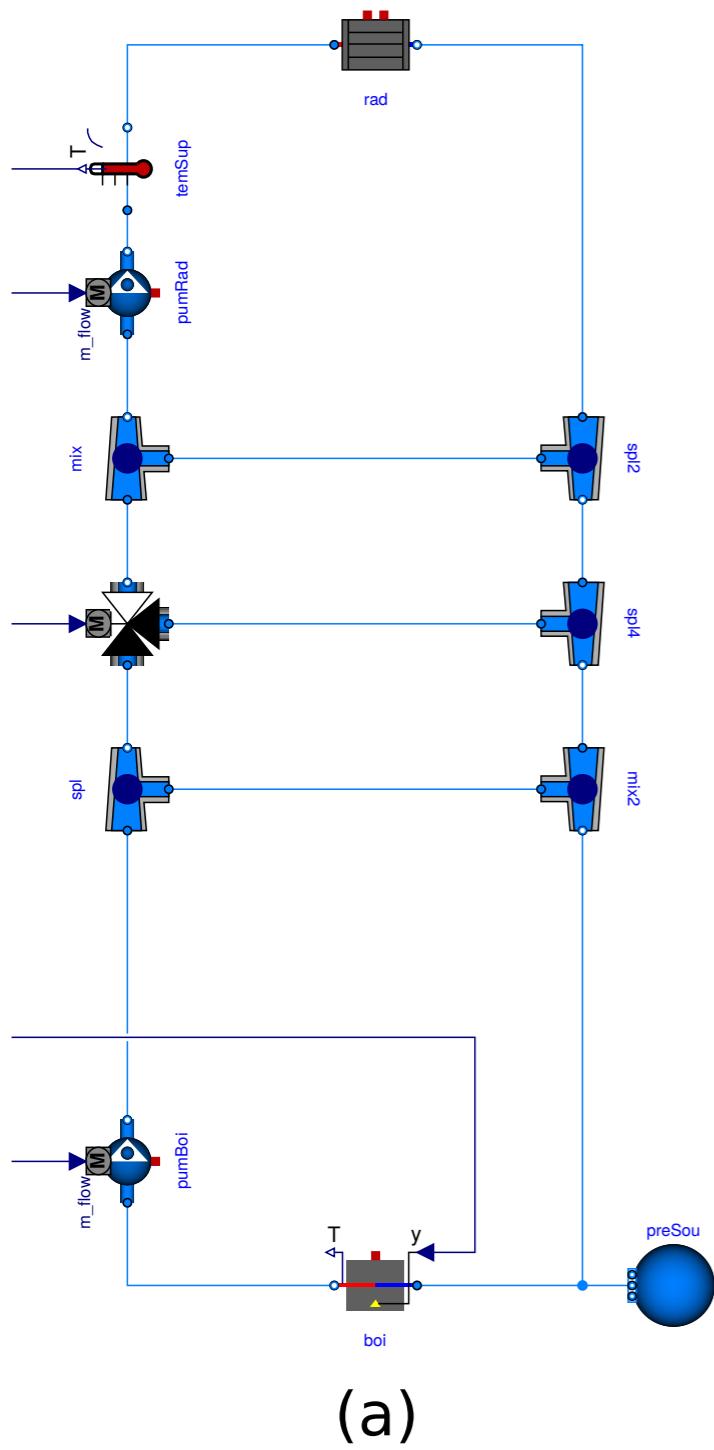
# Modeling of fluid junctions

What is wrong with this model?



$$h = \frac{\sum_i \max(0, \dot{m}_i) h_i}{\sum_i \max(0, \dot{m}_i)}$$

# Modeling of fluid junctions

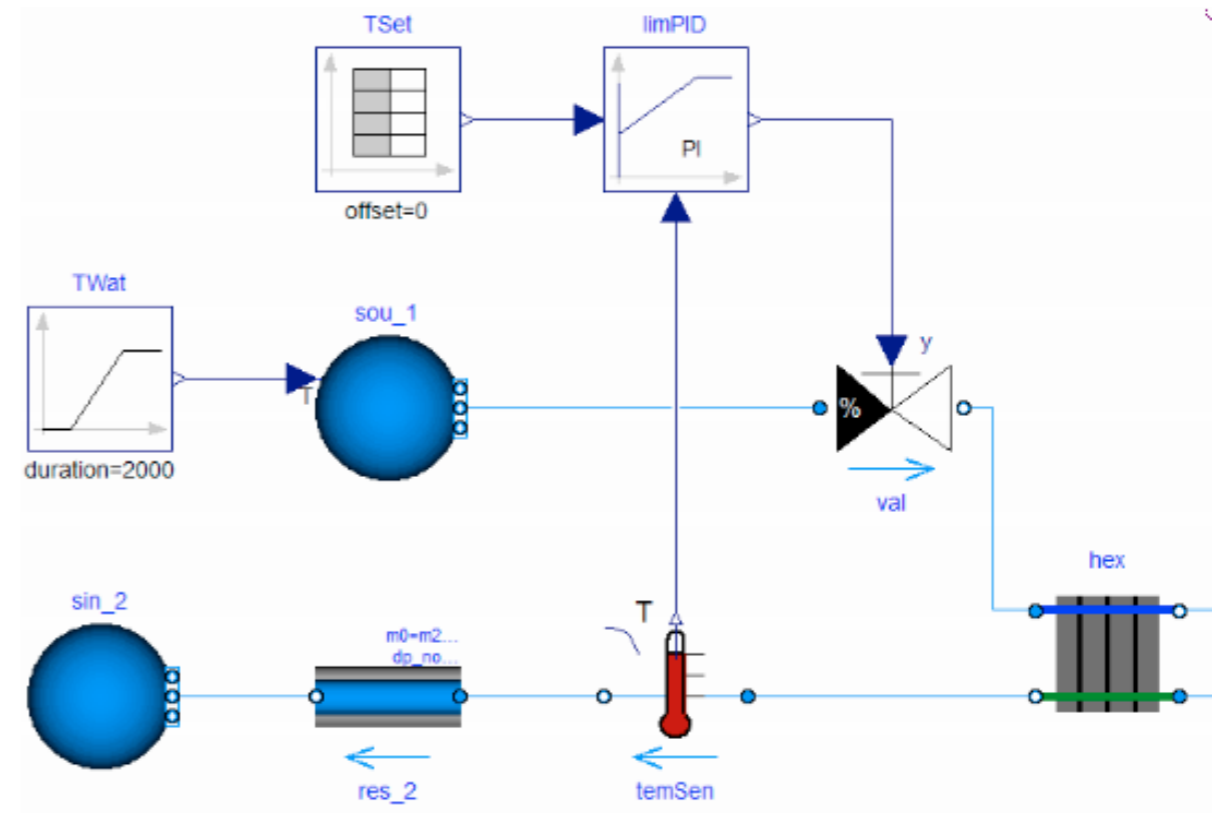




# Avoid oscillations of sensor signal

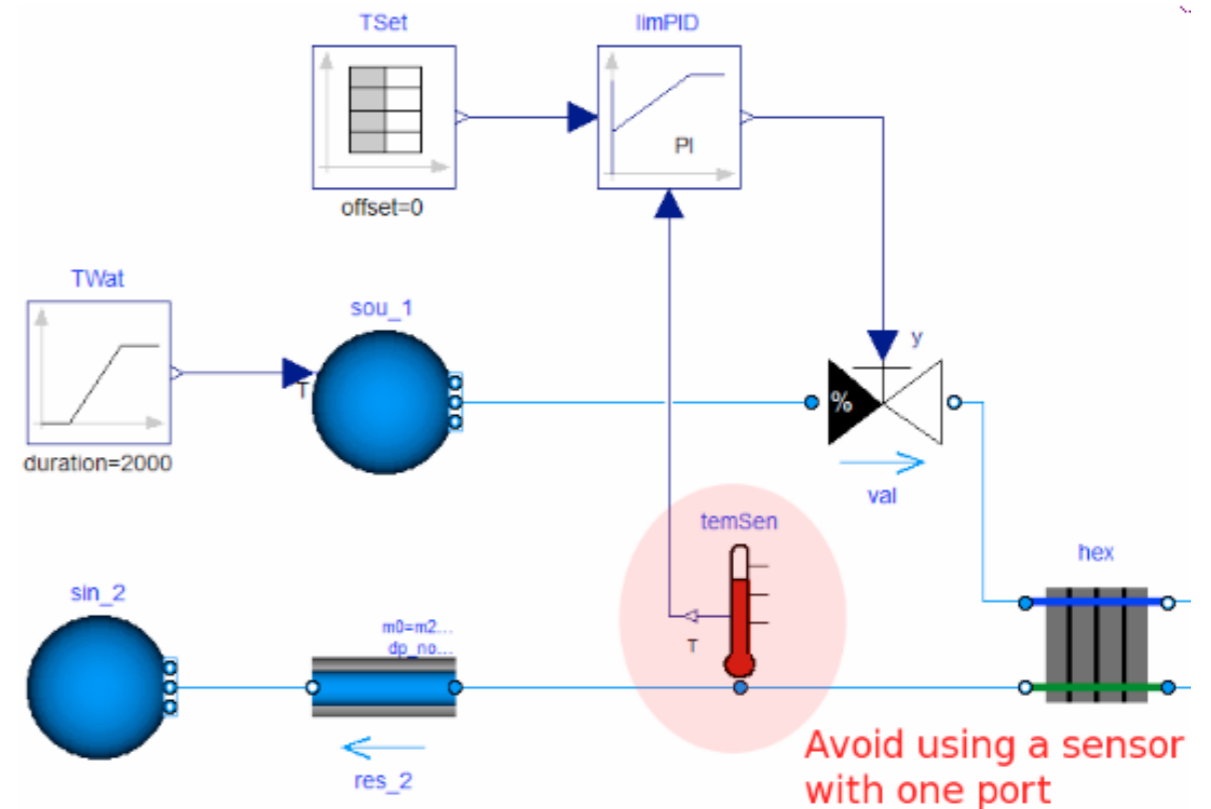
Correct use because

$$\tau \frac{dT}{dt} = \frac{|\dot{m}|}{\dot{m}_0} (\theta - T)$$



Incorrect, as sensor output oscillates if mass flow rate changes sign. This happens for example if the mass flow rate is near zero and approximated by a solver.

See also [User Guide](#).



# Nature is smooth, so avoid events

This triggers events:

```
h = if port_a.m_flow > 0  
    then inStream(port_a.h_outflow) else port_a.h_outflow;
```

Avoid events using regularization:

```
h = Modelica.Fluid.Utilities.regStep(  
    x = port_a.m_flow,  
    y1 = inStream(port_a.h_outflow),  
    y2 = port_a.h_outflow,  
    x_small = m_flow_nominal*1E-4);
```

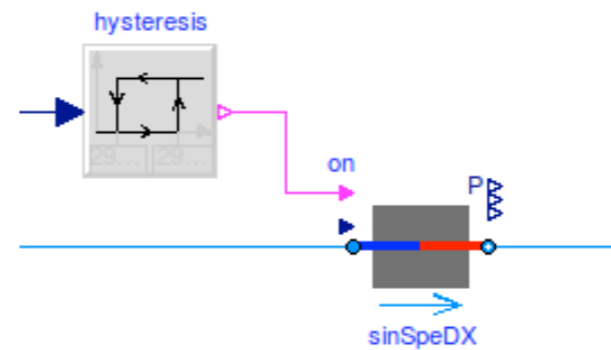
See also [User Guide](#).

# Beware of oscillating control, and guard against noise

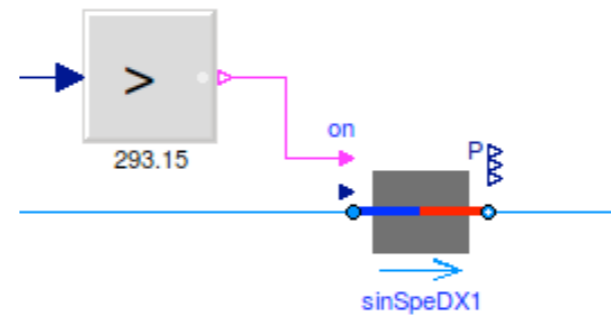
If the control input oscillates around zero, then this model stalls

What happens if this model is simulated with an adaptive time step?

Correct configuration



Avoid this configuration



```
model Test
  Real x(start=0.1);
equation
  der(x) = if x > 0 then -1 else 1;
end Test;
```

# Setting of nominal values is important for scaling of residuals

If pressure is around 1E5 Pa, set `p(nominal=1E5)`.

Nominal values are used to scale residuals.

Modelica simulation tools typically control the local integration error as

$$\epsilon \leq t_{rel} |x^i| + t_{abs}$$

where the absolute tolerance is scaled with the nominal value as

$$t_{abs} = t_{rel} |x_{nom}^i|.$$

# Exercise: Modeling of a simple thermofluid flow system

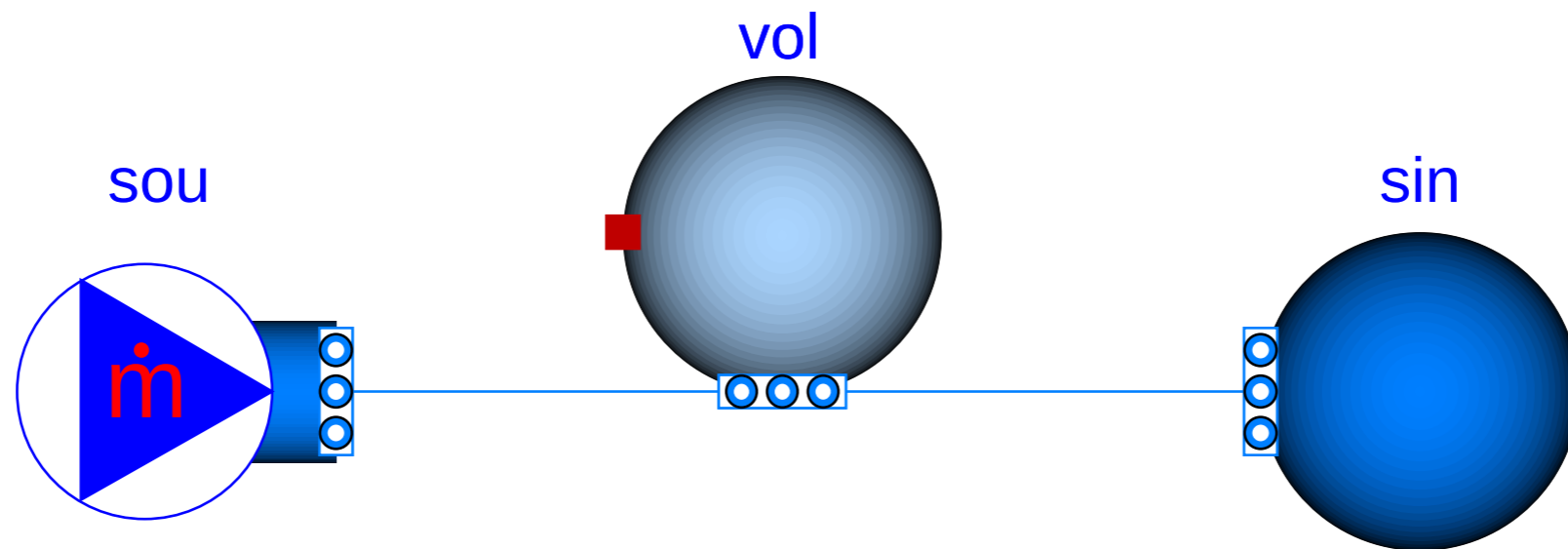
How do you implement a model with

1. a flow source of  $1 \text{ kg/s}$  of water at  $30^\circ\text{C}$ ,
2. a well stirred tank with no heat loss and a volume of  $2 \text{ m}^3$ , and
3. an infinite sink that is at atmospheric pressure?



# Exercise: Modeling of a simple thermofluid flow system

1. Make instances using models from `Buildings.Fluid.Sources` and `Buildings.Fluid.MixingVolumes`.
2. Assign the parameters.
3. Check and simulate the model.



# Further resources

## Tutorials

- [Buildings.Examples.Tutorial](#)

## User guides

- [User guides for specific packages of models.](#)
- [User guide with general information.](#)

# Developer Guide

See also

<https://simulationresearch.lbl.gov/modelica/userGuide/prePostProcessing.html>



# Overview

## Main topics

- Coding style and conventions
- Requirements
- Organization of the library
- Adding a new model
- Adding regression tests

## Further literature

- [User Guide -> Development](#)
- [Style guide](#)

# Coding style and conventions

Based on Modelica Standard Library.

Most variables are 3 letter camel case to avoid too long names, for example,  
**TOut**, **yDamMax**, ...

Code duplication avoided where practical.

Additional information at

<https://simulationresearch.lbl.gov/modelica/userGuide/development.html#style-guide>  
and

[https://simulationresearch.lbl.gov/modelica/releases/latest/help/  
Buildings\\_UsersGuide.html#Buildings.UsersGuide.Conventions](https://simulationresearch.lbl.gov/modelica/releases/latest/help/Buildings_UsersGuide.html#Buildings.UsersGuide.Conventions)

# Organization of individual packages

Packages are typically structured as shown on the right.

To add a new class, look first at **Interfaces** and **BaseClasses**.

You probably will never implement a component without extending a base class, such as from **Buildings.Fluid.Interfaces**

```
Tutorial  
UsersGuide
```

```
Any other classes (models,  
functions etc.)
```

```
Data  
Types  
Examples  
Validation  
Benchmarks  
Experimental  
Interfaces  
BaseClasses  
Internal  
Obsolete
```

# Implementing new thermofluid flow devices

[Buildings.Fluid.Interface](#) provides base classes.

[Buildings.Fluid.Interface.UsersGuide](#) describes these classes.

Alternatively, simple models such as the models below may be used as a starting point for implementing new models for thermofluid flow devices:

[Buildings.Fluid.HeatExchangers.HeaterCooler\\_u](#)

For a device that adds heat to a fluid stream.

[Buildings.Fluid.MassExchangers.Humidifier\\_u](#)

For a device that adds humidity to a fluid stream.

[Buildings.Fluid.Chillers.Carnot](#)

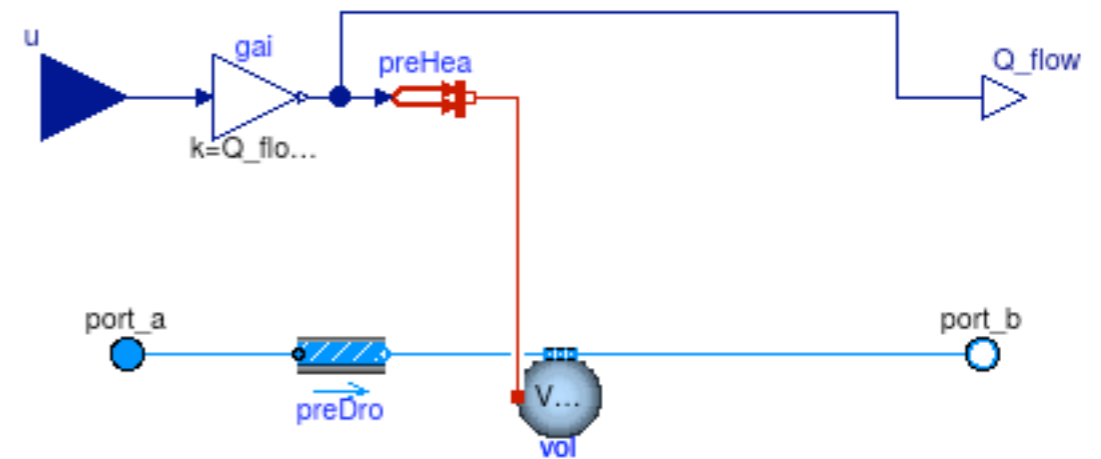
For a device that exchanges heat between two fluid streams.

[Buildings.Fluid.MassExchangers.ConstantEffectiveness](#)

For a device that exchanges heat and humidity between two fluid streams.

# Adding a heat exchanger

See [HeaterCooler\\_u](#)



```
within Buildings.Fluid.HeatExchangers;
```

```
model HeaterCooler_u "Heater or cooler with prescribed heat flow rate"  
  extends Buildings.Fluid.Interfaces.TwoPortHeatMassExchanger(  
    redeclare final Buildings.Fluid.MixingVolumes.MixingVolume vol(  
      prescribedHeatFlowRate=true);
```

```
  parameter Modelica.SIunits.HeatFlowRate Q_flow_nominal  
    "Heat flow rate at u=1, positive for heating";
```

```
  Modelica.Blocks.Interfaces.RealInput u "Control input";  
  Modelica.Blocks.Interfaces.RealOutput Q_flow(unit="W")  
    "Heat added to the fluid";
```

```
protected
```

```
  Buildings.HeatTransfer.Sources.PrescribedHeatFlow preHea  
    "Prescribed heat flow";  
  Modelica.Blocks.Math.Gain gai(k=Q_flow_nominal) "Gain";
```

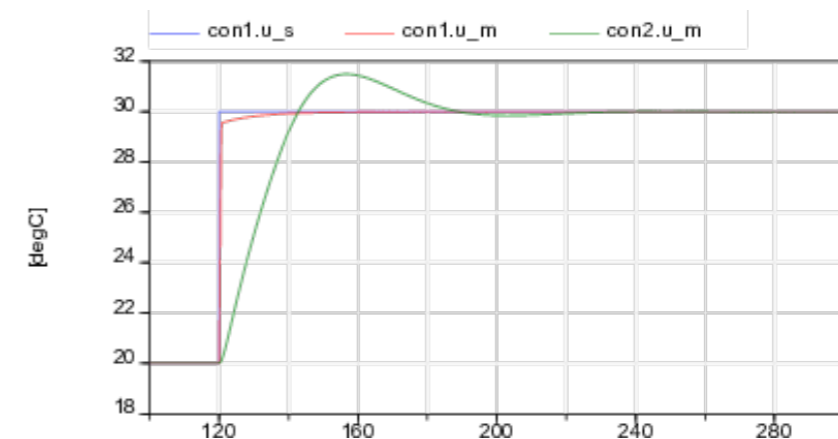
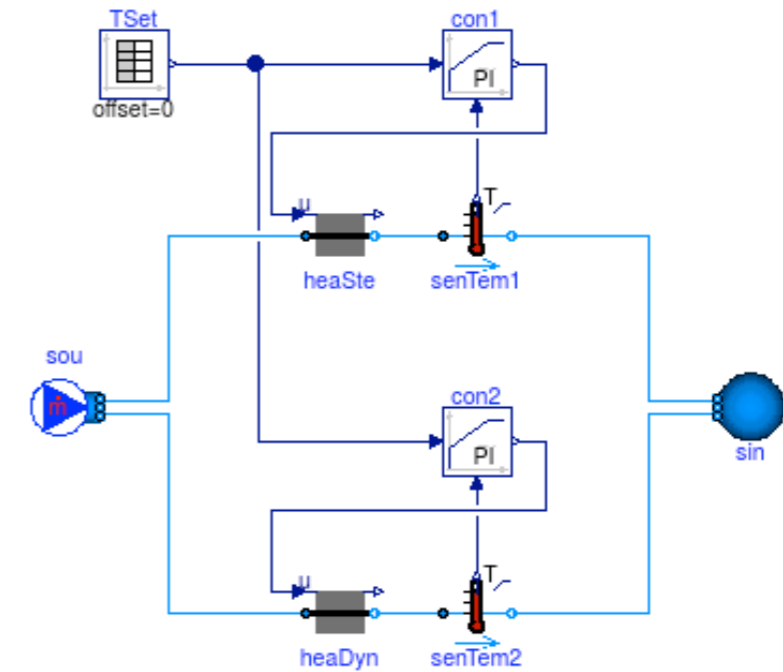
```
equation
```

```
  connect(u, gai.u); ... // other connect statements  
  annotation (...); // documentation  
end HeaterCooler_u;
```

# Add examples and validations to unit testing framework

1. Add validation and stress tests for different model configurations.
2. Validate results and add main outputs to plot script. These variables become part of the regression tests.
3. Run  
`modelica-buildings/bin/runUnitTests.py`
4. Update `Buildings/package.mo` [release notes](#).
5. Issue pull request on <https://github.com/lbl-srg/modelica-buildings>.

See [Unit Test documentation](#).



?