# Tutorial

# Modelica Buildings Library
# and
# Best Practices for Modeling of Thermofluid Flow Systems

Michael Wetter
Simulation Research Group

October 9, 2018

BERKELEY LAB Lawrence Berkeley National Laboratory

# Overview
# of
# Modelica Buildings Library

# Intended use of Buildings library

Users

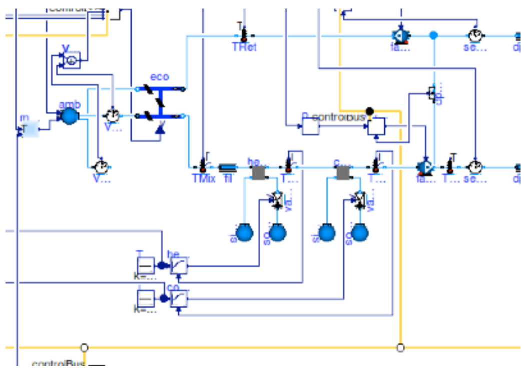
- Engine for "Spawn of EnergyPlus" HVAC and controls
- Equipment manufacturers, design firms, academia.
- Model-based design process.
- FDD algorithms.

License

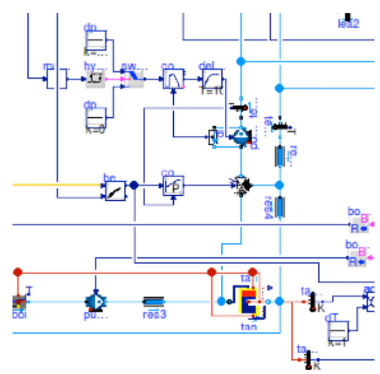
- All development is open-source under BSD.

# Scope of the Modelica Buildings Library

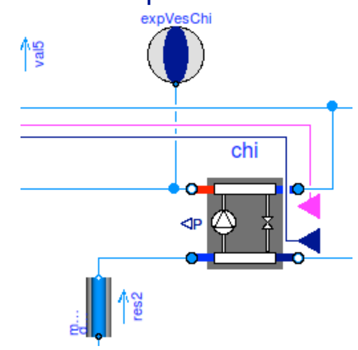
Air-based HVAC



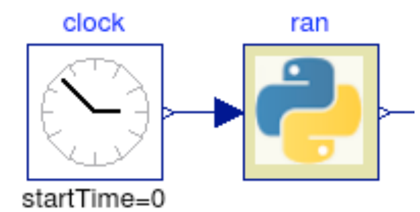
Hydronic heating



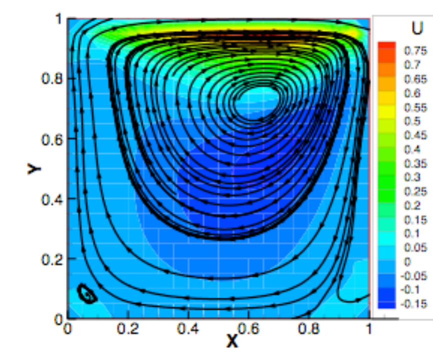
Chiller plants



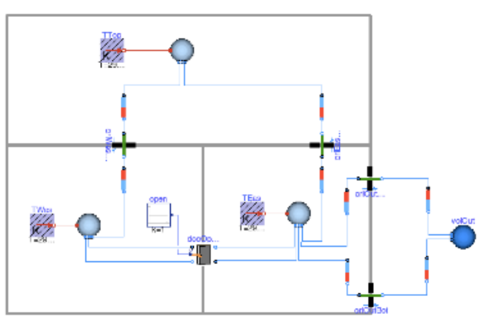

Embedded Python



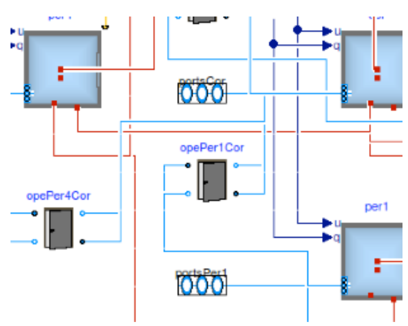

Room air flow



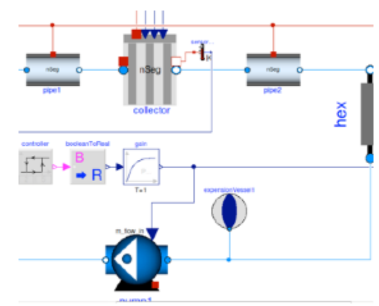

Natural ventilation, multizone air exchange, contaminant transport



Room heat transfer, incl. window (TARCOG)

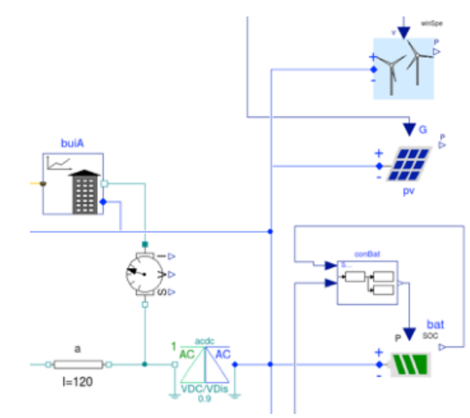


Solar collectors

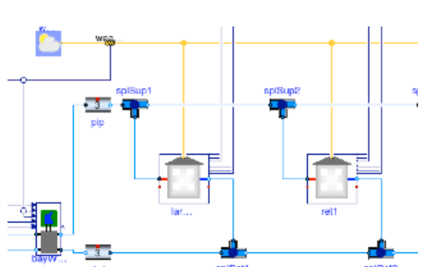


FLEXLAB

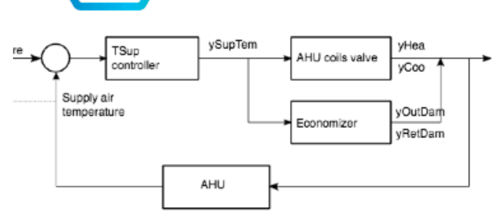
Electrical systems



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 (BOPTEST)

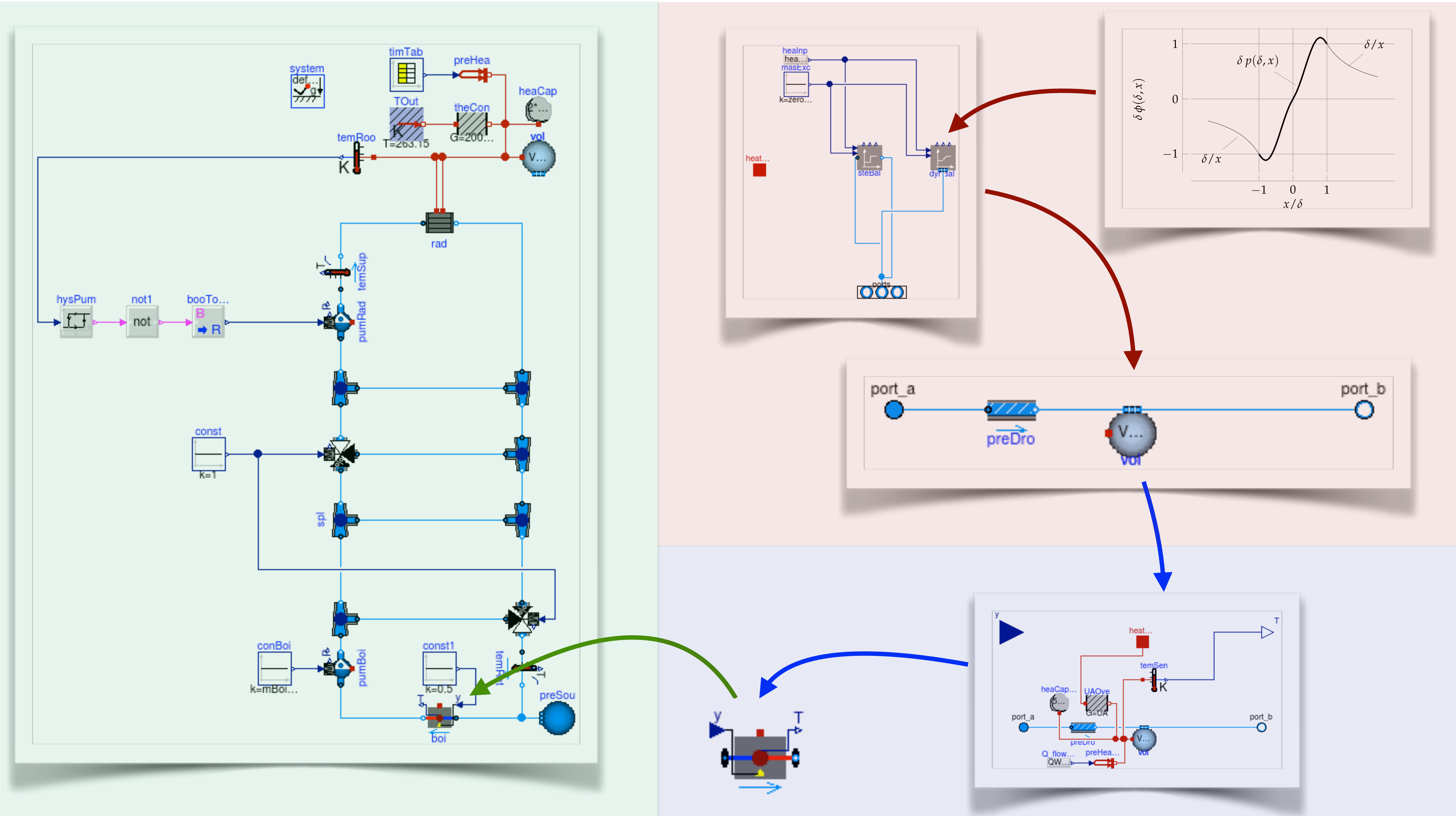
Co-develop with IBPSA Modelica library, including district heating and cooling systems

simulationresearch.lbl.gov/modelica



Michael Wetter, Wangda Zuo, Thierry S. Nouidui and Xiufeng Pang. Modelica Buildings library. Journal of Building Performance Simulation, 7(4):253-270, 2014.

# Separation between library developer, component developer and end user




**Legend:**

Library developer

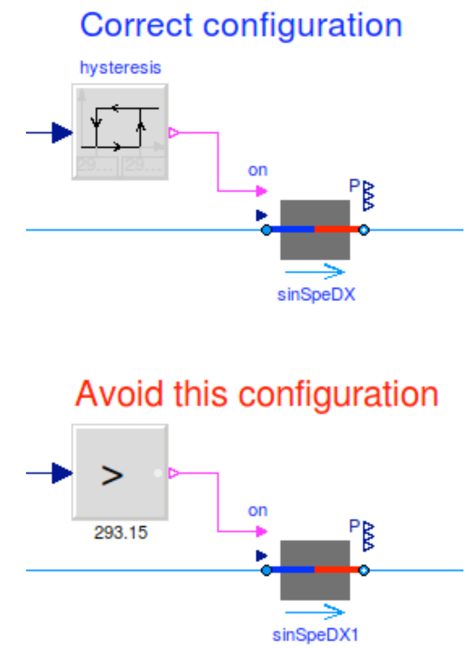
Component developer

End user

# Main modeling assumptions

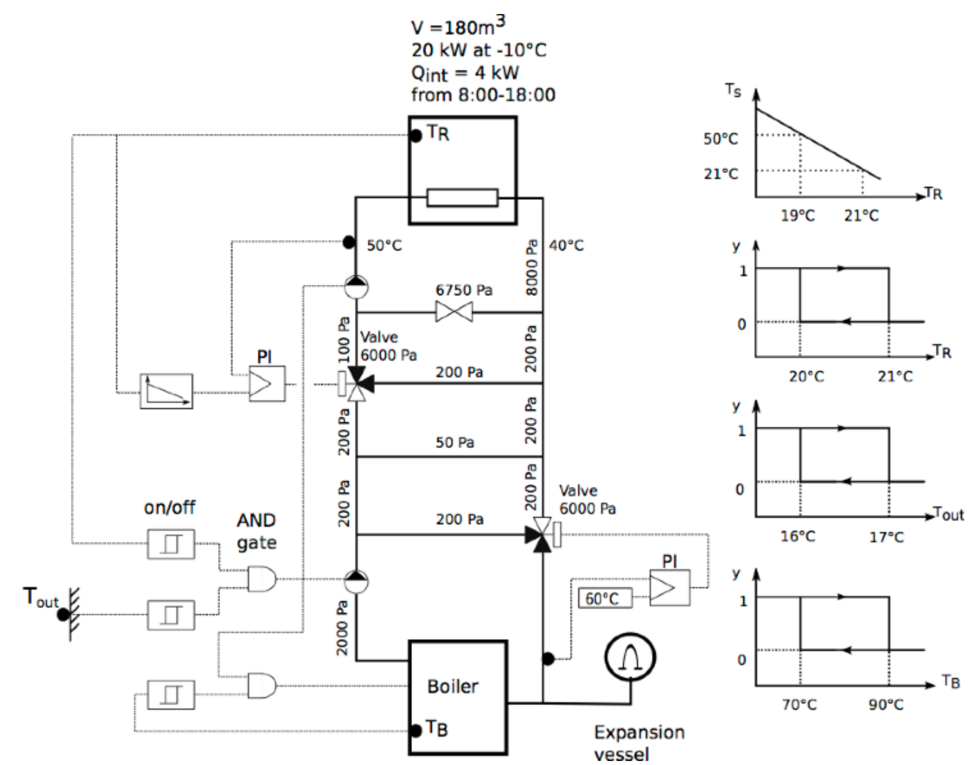
| | |
|---|---|
| **Media** | Can track moisture (X) and contaminants (C). |
| **HVAC equipment** | Most equipment based on performance curve, or based on nominal conditions and similarity laws.<br>Refrigerant is not modeled.<br>Most equipment optional steady-state or 1st order transient. |
| **Flow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

# Documentation and distribution




## Documentation

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




## Distribution

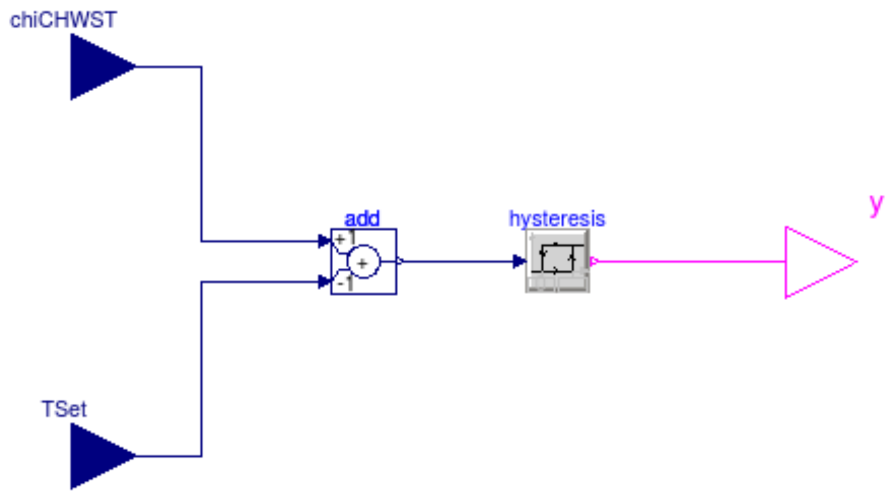
- Main site http://simulationresearch.lbl.gov/modelica
- Development site with version control, wiki and issue tracker: https://github.com/lbl-srg/modelica-buildings

# Best practice and modeling hints

# Building large system models

How do you build and debug a large system model?

1. Split the model into smaller models.
2. Test the smaller models for well known conditions.
3. Add smaller models to unit tests.

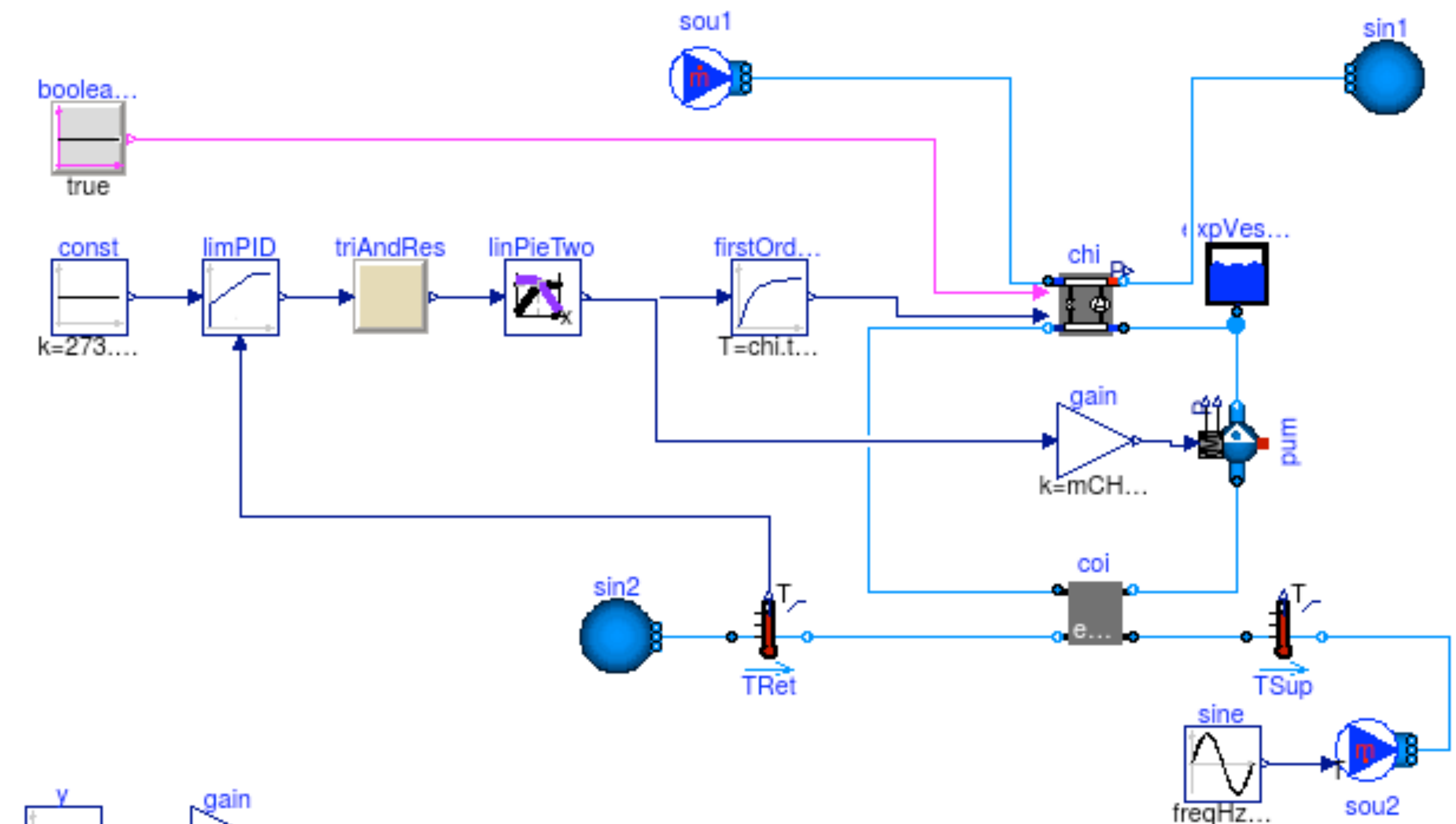



For example, see Chiller Plant

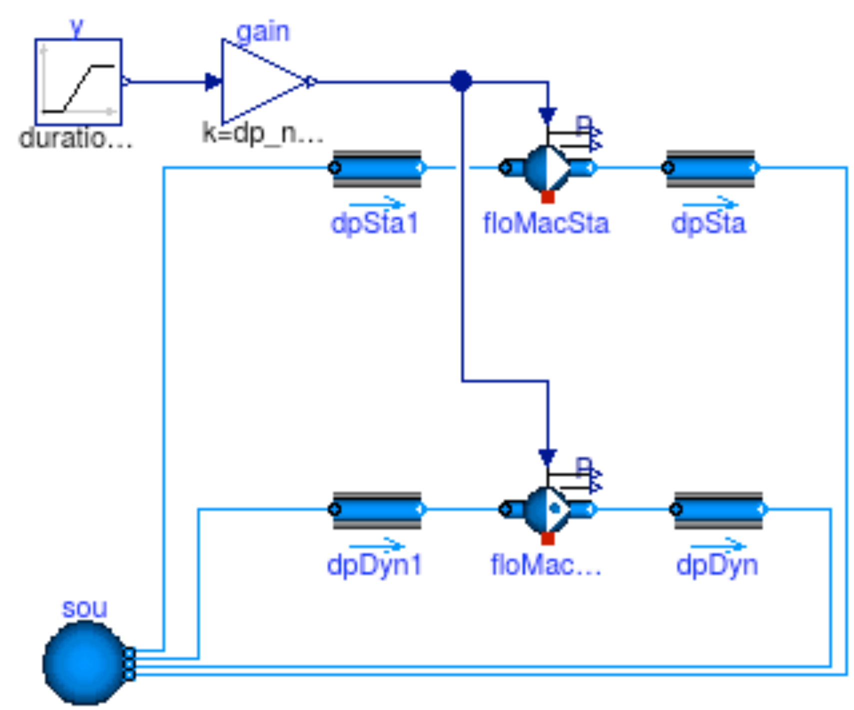
Each small models contains a simple unit test.

# Use small unit tests, as in

Chiller plant base classes




Pumps



# Propagate common parameters

Don't assign values to the same parameters

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

Instead, propagate parameters

```
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";
...
```

# 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

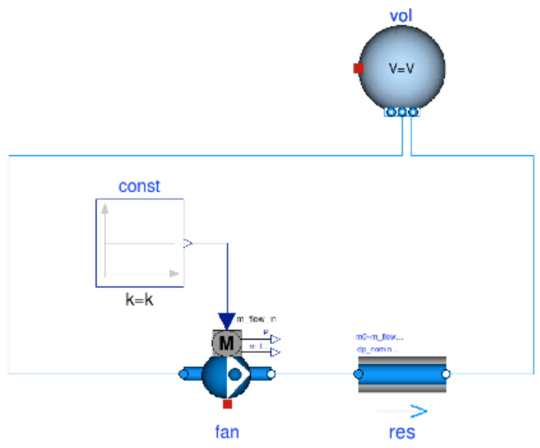
```
TemperatureSensor sen(
  redeclare final package Medium = Medium,
  final m_flow_nominal=m_flow_nominal) "Sensor";
```

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

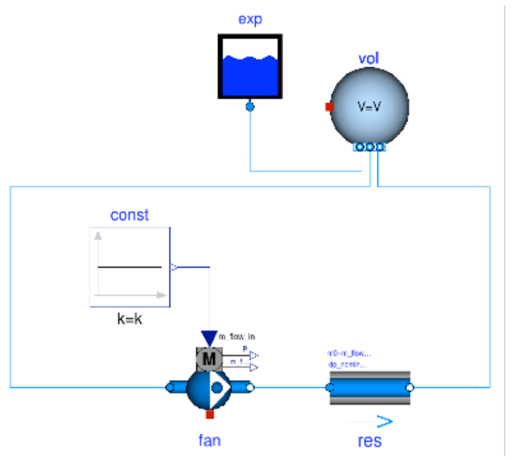
```
TemperatureSensor sen[2](
  each final m_flow_nominal=m_flow_nominal)
"Sensor";
```

# Setting a reference pressure

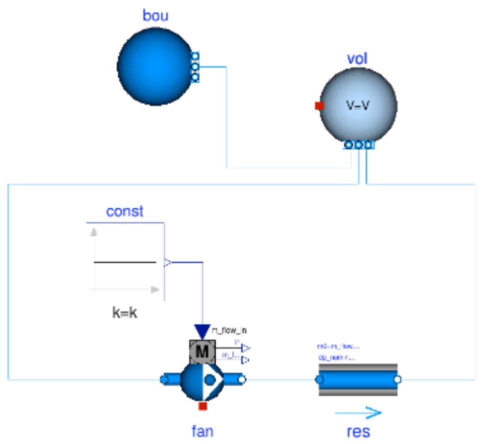
Underdetermined model as no pressure is assigned




Well defined model, but additional state for pressure as reservoir *p/p0=V0/p*

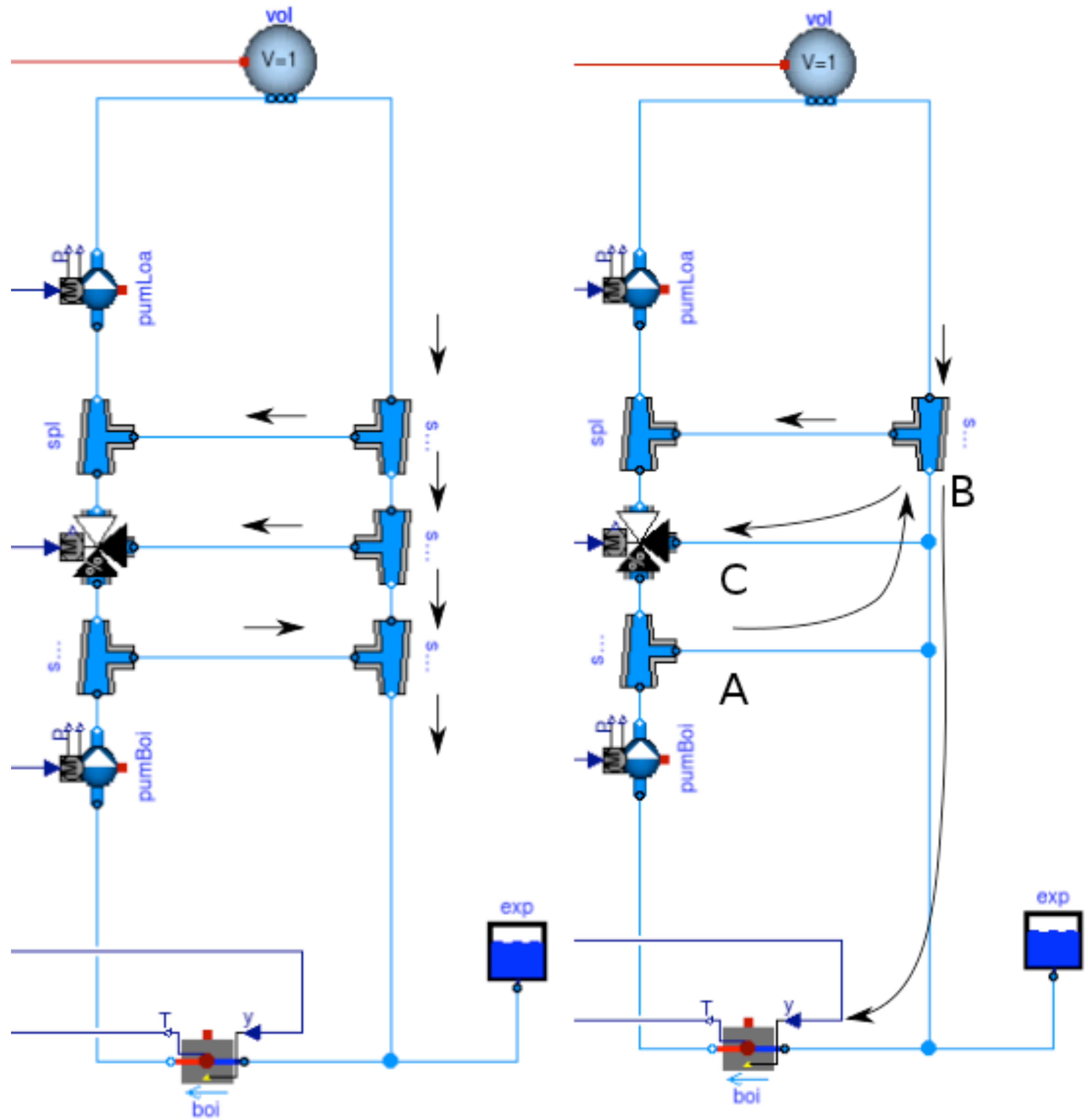



Most efficient model as reservoir *p* is constant



# Modeling of fluid junctions

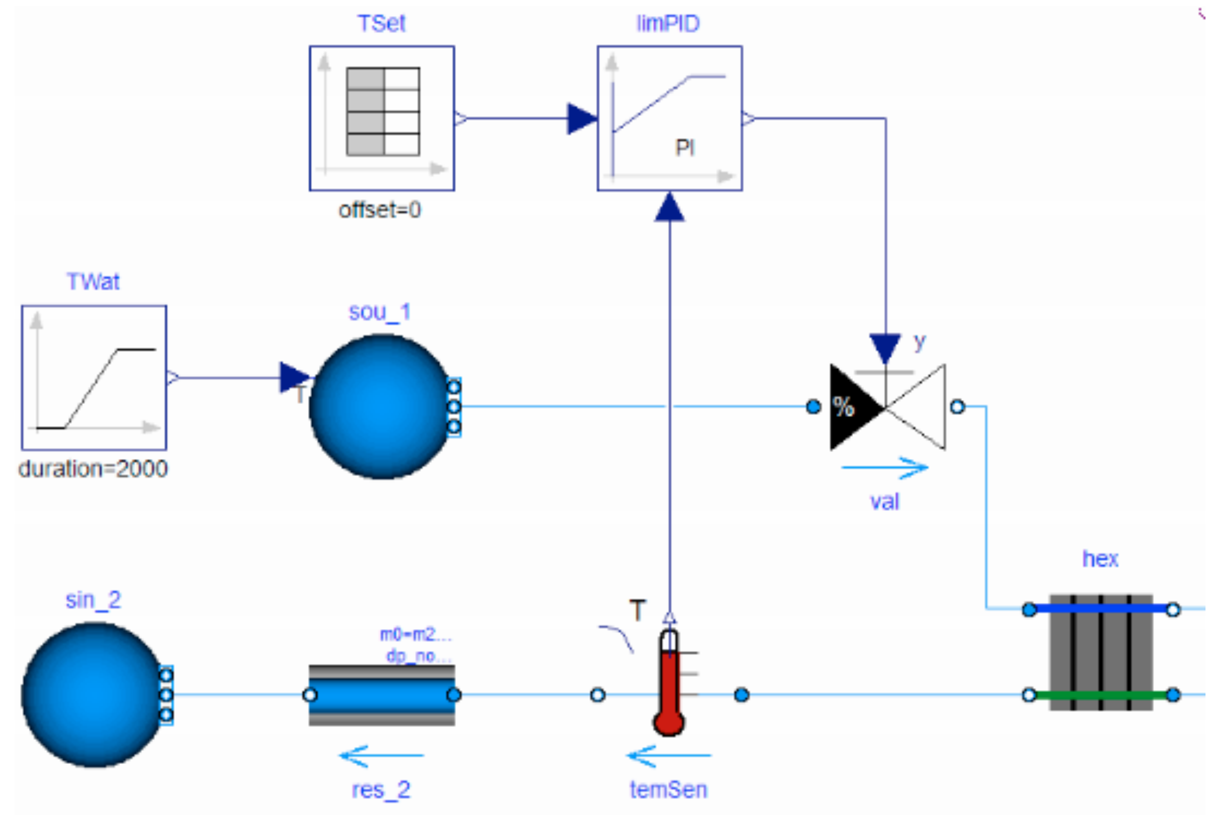
In the model on the right, mixing takes place in the fluid port B because the boiler, port A and port C all connect to port B.



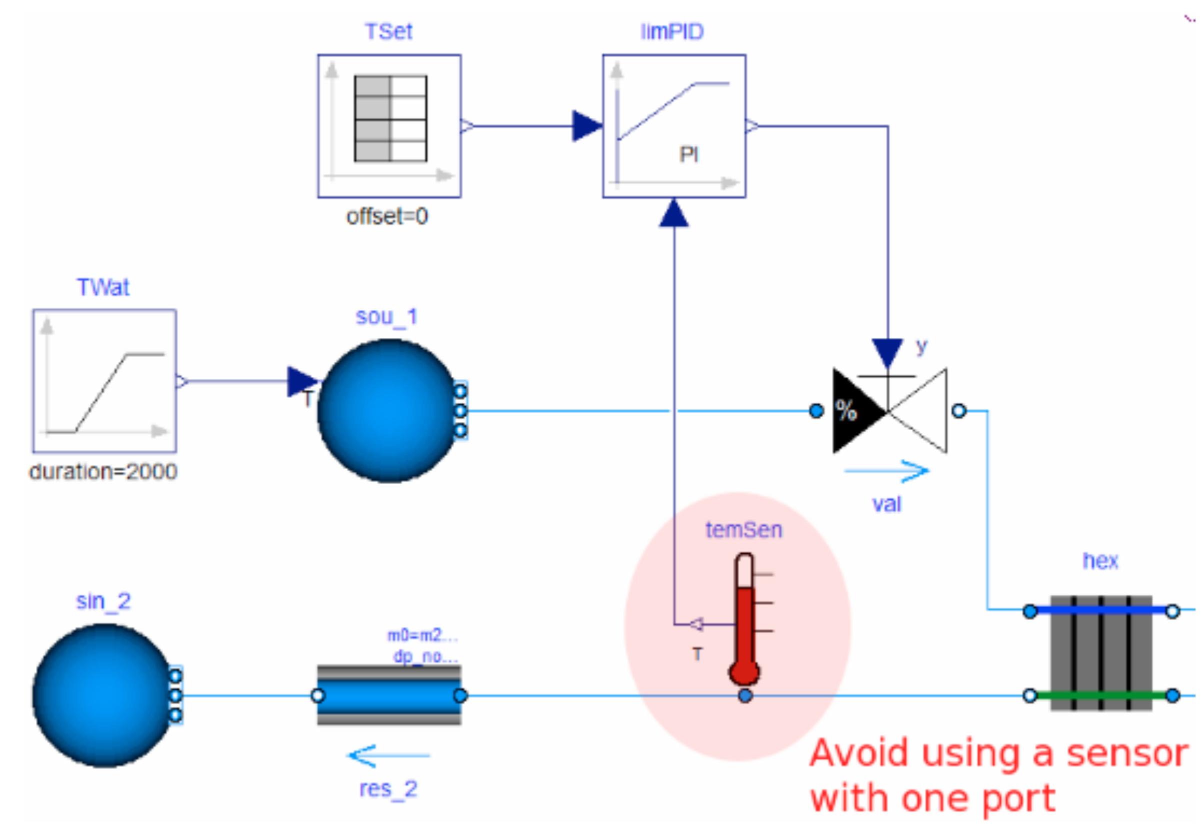
# Avoid oscillations of sensor signal

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.

# Avoid events

This triggers events:

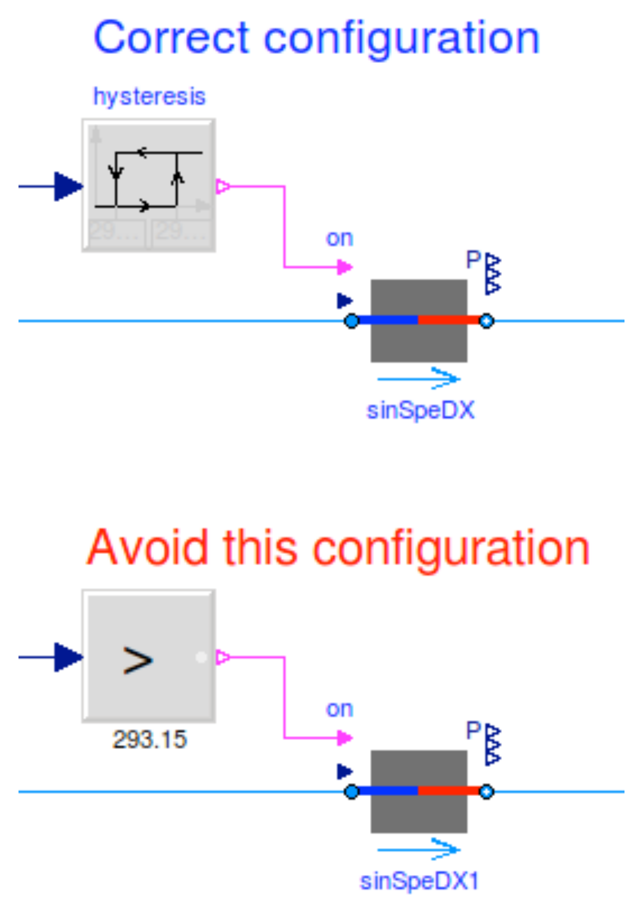
```
T_in = if port_a.m_flow > 0  then port_a.T else port_b.T;
```

Avoid events using regularization:

```
T = Modelica.Fluid.Utilities.regStep(
      x = port_a.m_flow,
      y1 = T_a_inflow,
      y2 = T_b_inflow,
      x_small = m_flow_nominal*1E-4);
```

See also User Guide.

# Beware of oscillating control




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

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

```
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, such as in Dymola's **`dsmodel.c`**:

```
{ /* Non-linear system of equations to solve. */
  ...
const char*const varnames_[]={"floMac1.VMachine_flow",
                              "floMac2.VMachine_flow"};
const double nominal_[]={0.001, 0.001};
...
```

In Dymola, the local integration error is

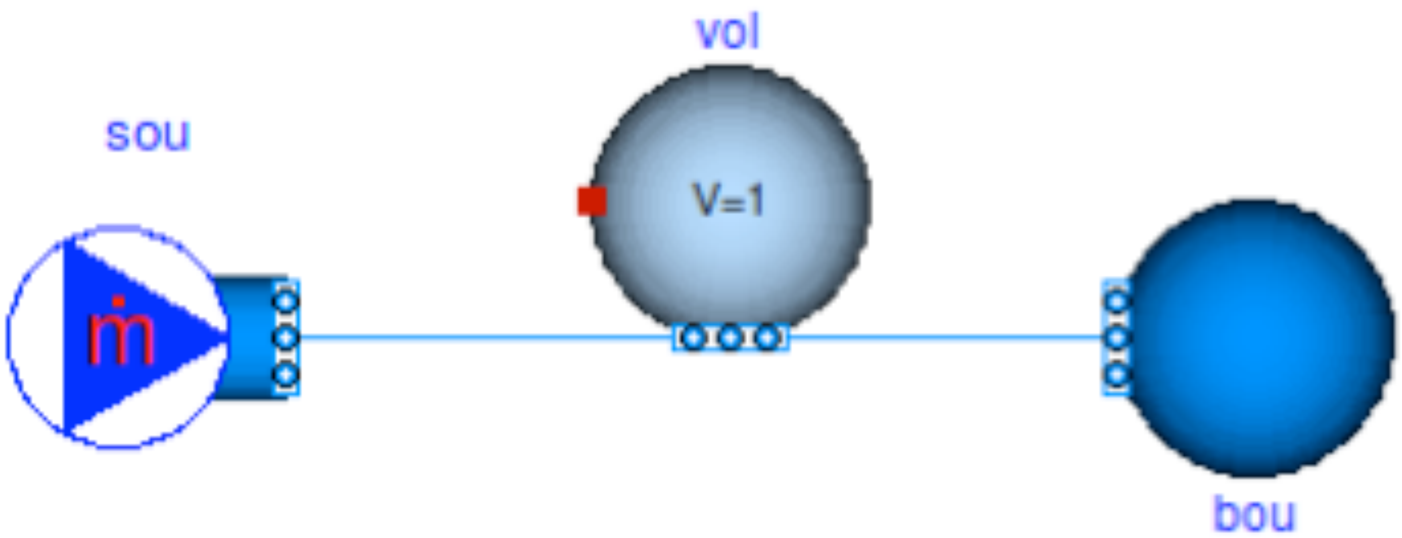
$\epsilon \leq t_{rel} \, |x^i| + t_{abs}$

where the absolute tolerance is scaled with the nominal value as

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

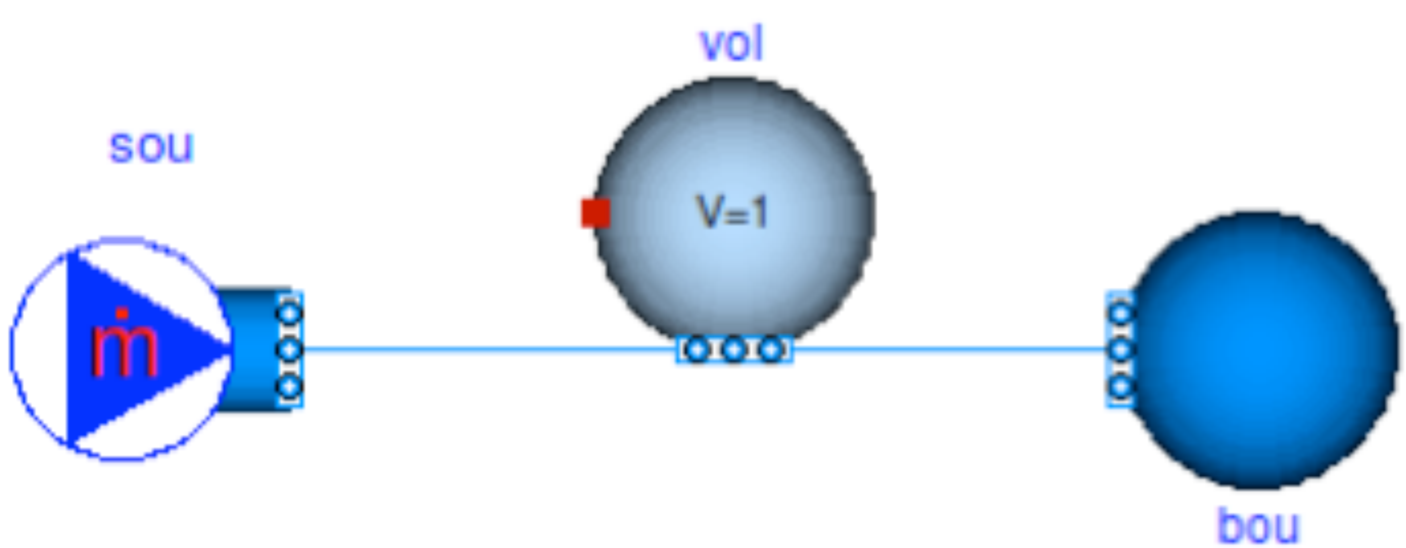
# Exercise: Modeling of a simple thermofluid flow system

How do you implement a source and boundary condition with a tank in between to create the model below:

# 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

# 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 convention

# Coding style and conventions

Based on Modelica Standard Library.

Most variables are 3 letter camel case to avoid too long names.

Code duplication avoided where practical.

Additional information at
https://github.com/lbl-srg/modelica-buildings/wiki/Style-Guide and
http://simulationresearch.lbl.gov/modelica/releases/latest/help/Buildings_UsersGuide.html

# Requirements

Physical requirements

Mathematical requirements

# 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:

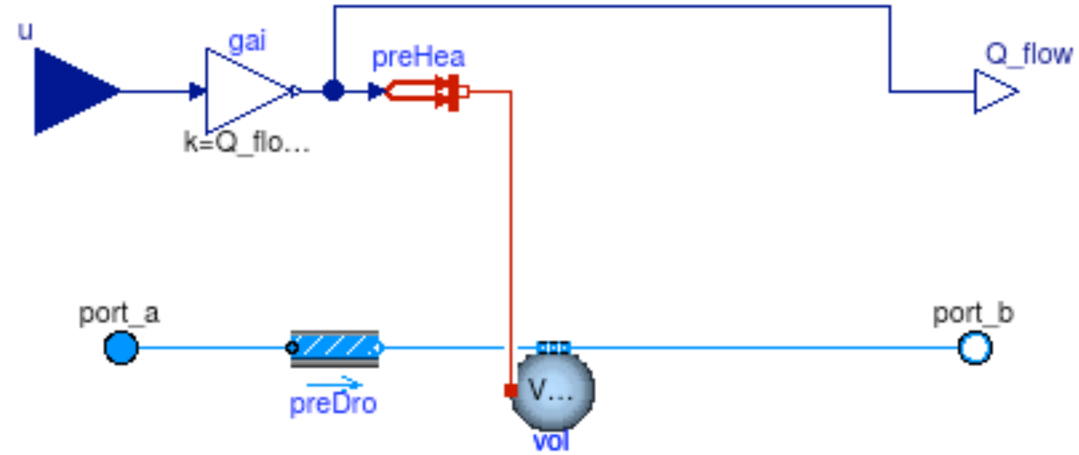
Buildings.Fluid.HeatExchangers.HeaterCooler_u
For a device that adds heat to a fluid stream.

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.

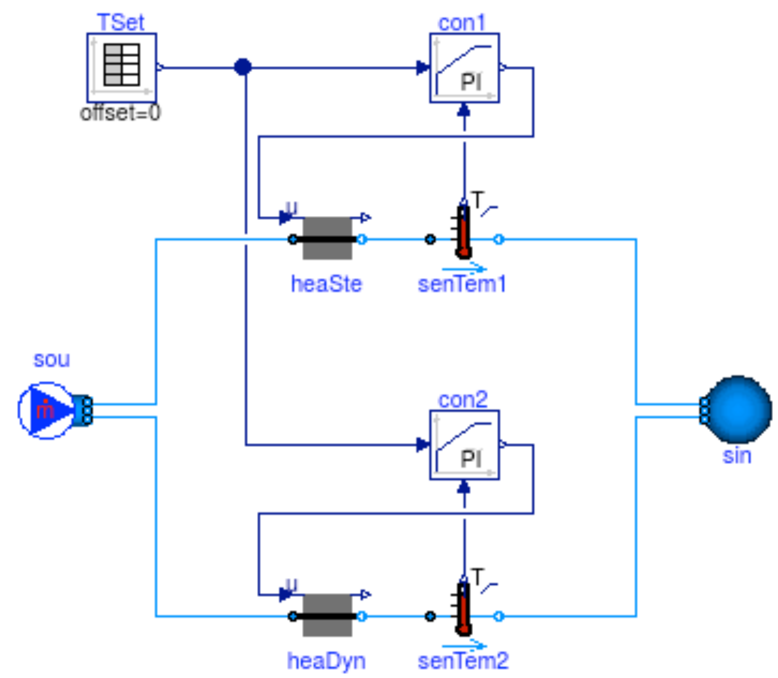
# Adding a heat exchanger




See [HeaterCooler_u]

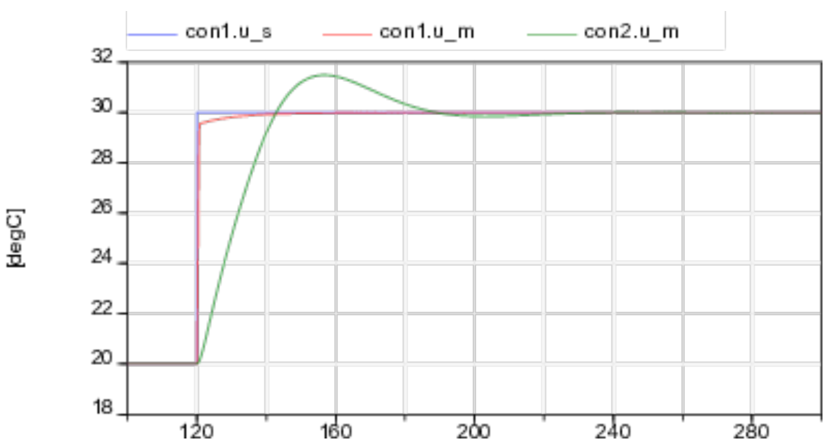
```
within Buildings.Fluid.HeatExchangers;

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.

?