



Enterprise Architect

User Guide Series

# Parametric Simulation using OpenModelica

Sparx Systems Enterprise Architect provides integration with OpenModelica to support rapid and robust simulation of how a SysML Parametric model will behave under different circumstances; the simulation properties are defined in a Simulation Artifact.

Author: Sparx Systems

Date: 7/08/2019

Version: 1.0

CREATED WITH  ENTERPRISE  
ARCHITECT



# Table of Contents

Parametric Simulation using OpenModelica .....	4
Interfacing with OpenModelica .....	7
OpenModelica on Windows .....	8
OpenModelica on Linux .....	11
Creating a Parametric Model .....	17
Configure SysML Simulation Window .....	34
Model Analysis using Datasets .....	44
Modeling and Simulation with Modelica Library .....	49
SysML Simulation Examples .....	56
Electrical Circuit Simulation Example .....	58
Mass-Spring-Damper Oscillator Simulation Example .....	70
Water Tank Pressure Regulator .....	80
Troubleshooting OpenModelica Simulation .....	96

# Parametric Simulation using OpenModelica

Enterprise Architect provides integration with OpenModelica to support rapid and robust evaluation of how a SysML model will behave in different circumstances. This section describes the process of defining a Parametric model, annotating the model with additional information to drive a simulation, and running a simulation to generate a graph.

## Introduction to SysML Parametric Models

SysML Parametric models support the engineering analysis of critical system parameters, including the evaluation of key metrics such as performance, reliability and other physical characteristics. These models combine requirements models with system design models, by capturing executable constraints based on complex mathematical relationships. Parametric diagrams are specialized Internal Block diagrams that help you, the modeler, to combine behavior and structure models with engineering analysis models such as performance, reliability, and mass property models.

For further information on the concepts of SysML Parametric models, refer to the official OMG SysML website and its linked sources.

## SysMLSimConfiguration Artifact

Enterprise Architect helps you to extend the usefulness of your SysML parametric models by annotating them with extra information that allows the model to be simulated. The resulting model is then generated as a Modelica model that can be solved (simulated) using OpenModelica.

The simulation properties for your model are stored against a Simulation Artifact. This preserves your original model and supports multiple simulations being configured against a single SysML model. The Simulation Artifact can be found on the 'Artifacts' Toolbox page.

## User Interface

The user interface for the SysML simulation is described in the *Configure SysML Simulation Window* topic.

## OpenModelica Examples

To aid your understanding of how to create and simulate a SysML parametric model, three examples have been provided to illustrate three different domains. These

examples and what you are able to learn from them are described in the *SysML Simulation Examples* topic.

# Interfacing with OpenModelica

For details on installing OpenModelica and connecting Enterprise Architect to it, see the Help topic covering the platform where Enterprise Architect is installed.

## Installation

Platform	Detail
Windows	If Enterprise Architect is installed on a Windows platform, see the <i>OpenModelica on Windows</i> Help Topic.
Linux	If Enterprise Architect is installed on a Linux platform, see the <i>OpenModelica on Linux</i> Help Topic.

# OpenModelica on Windows

When installing OpenModelica for Enterprise Architect operating on a Windows platform, you firstly install the OpenModelica application, then configure the settings in Enterprise Architect to access OpenModelica.



## Install OpenModelica

Step	Action
1	Download the OpenModelica Installer from: <a href="https://openmodelica.org/download/download-windows">https://openmodelica.org/download/download-windows</a>
2	Double-click on the OpenModelica installer and follow the 'Wizard' instructions. We recommend that you accept the default path for installation.
3	Check that you can locate the executable omc.exe. For example: C:\OpenModelica1.9.2\bin\omc.exe



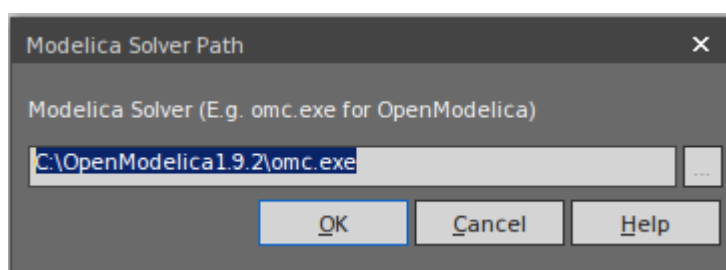
## Access

Use either of these access paths to display the 'Modelica Solver Path' dialog.

Method	Select
Ribbon	Simulate > System Behavior > Modelica > SysMLSim Configuration Manager >  > Configure Modelica Solver
Other	Double-click on an Artifact with the SysMLSimConfiguration stereotype >  > Configure Modelica Solver

## Configure the Solver

For Windows, the 'Modelica Solver Path' dialog resembles this:



Type or browse for the path to the Modelica solver to use OpenModelica in Enterprise Architect.

# OpenModelica on Linux

If Enterprise Architect is installed on Linux it is necessary to operate with OpenModelica installed on the same platform. The OpenModelica Linux installation is publicly documented for Debian and Ubuntu; however, it can also be installed under Linux Mint.

This Help topic provides guidance on:

1. Installation of OpenModelica on:
  - Linux Debian / Ubuntu
  - Linux Mint
2. Configuring Enterprise Architect to access OpenModelica.

## Linux Debian / Ubuntu

To install OpenModelica on a Linux Debian / Ubuntu system refer to the URL:

<https://openmodelica.org/download/download-linux>

This provides the instructions for Debian / Ubuntu Packages.

Run these scripts in a terminal:

Step	Action
1	To add OpenModelica to your additional repository list:

	<pre>for deb in deb deb-src; do echo "\$deb http://build.openmodelica.org/apt `lsb_release -cs` nightly"; done   sudo tee /etc/apt/sources.list.d/openmodelica.list</pre>
2	<p>Import the GPG key used to sign the releases:</p> <pre>wget -q http://build.openmodelica.org/apt/openm odelica.asc -O-   sudo apt-key add -</pre>
3	<p>Update and install OpenModelica:</p> <pre>sudo apt-get update sudo apt-get install openmodelica sudo apt-get install omlib-* # Installs optional Modelica libraries (most have not been tested with OpenModelica)</pre>
4	<p>To check this installation, ensure that you can find the file <code>/usr/bin/omc</code> by, for example, executing this command on the terminal:</p> <ul style="list-style-type: none"> <li>• <code>~ \$ /usr/bin/omc --version</code></li> </ul> <p>Your installation is successful if the command returns a string resembling this:</p> <ul style="list-style-type: none"> <li>• OpenModelica 1.13.0~dev-1322-g53a43cf</li> </ul>

## Linux Mint

To install OpenModelica on Linux Mint, you initially perform an install for Ubuntu and then modify the Linux Mint code name to match the Ubuntu code name.

This is a list of mappings of the Linux Mint code name to the Ubuntu code name (to be used in later steps):

- Linux Mint 17.3 (Rosa) = Ubuntu 14.04 (Trusty): **rosa = trusty**
- Linux Mint 18 (Sarah) = Ubuntu 16.04 (Xenial): **sarah = xenial**
- Linux Mint 18.1 (Serena) = Ubuntu 16.04 (Xenial): **serena = xenial**
- Linux Mint 18.2 (Sonya) = Ubuntu 16.04 (Xenial): **sonya = xenial**
- Linux Mint 18.3 (Sylvia) = Ubuntu 16.04 (Xenial): **sylvia = xenial**
- Linux Mint 19 (Tara) = Ubuntu 18.04 (Bionic): **tara = bionic**

Click [Here](#) for a full list of Linux Mint History and the mappings with Ubuntu.


Step	Action
1	Run this script in a terminal:

	<pre>for deb in deb deb-src; do echo "\$deb http://build.openmodelica.org/apt `lsb_release -cs` nightly"; done   sudo tee /etc/apt/sources.list.d/openmodelica.list</pre>
2	<p>To change the repository URL in Linux Mint:</p> <ul style="list-style-type: none"> <li>• On the Linux Mint main screen select: 'Menu   Search Bar   Software Sources (type in password)   Additional repositories   Select 'Openmodelica'   Edit URL'</li> <li>• Change the Linux Mint name (for example, <i>rosa</i>) to the corresponding Ubuntu name (for example, <i>trusty</i>) as in the list at the top of this table; that is: <pre>deb http://build.openmodelica.org/apt <b>rosa</b> nightly deb http://build.openmodelica.org/apt <b>trusty</b> nightly</pre> </li> <li>• Click on the OK button</li> </ul>
3	<ul style="list-style-type: none"> <li>• Select 'Openmodelica(Sources)'  Edit URL</li> <li>• Change the Linux Mint name according to the list at the top of the table</li> </ul>

	<p>For example, change the Linux Mint name <i>rosa</i> to the corresponding Ubuntu name <i>trusty</i></p> <ul style="list-style-type: none"> <li>• Click on the OK button</li> </ul>
4	<p>To update and install OpenModelica, run these scripts in a terminal:</p> <pre>sudo apt-get update sudo apt-get install openmodelica sudo apt-get install omlib-* # Installs optional Modelica libraries (most have not been tested with OpenModelica)</pre>

## Access

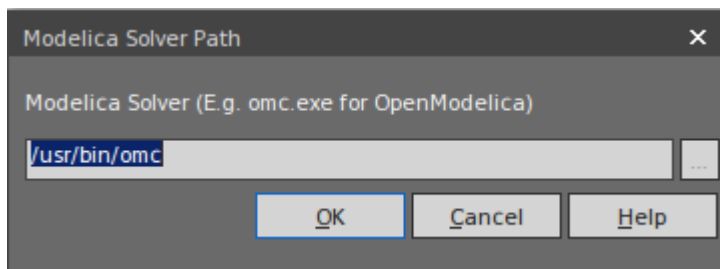
Use either of these access paths to display the 'Modelica Solver Path' dialog, to configure the solver.

Method	Select
Ribbon	<p>Simulate &gt; System Behavior &gt; Modelica &gt; SysMLSim Configuration Manager &gt;  &gt; Configure Modelica Solver</p>
Other	<p>Double-click on an Artifact with the SysMLSimConfiguration stereotype &gt;</p>

 > Configure Modelica Solver

## Configure the Solver

The 'Modelica Solver Path' dialog resembles this:



Type in or browse for the path to the Modelica solver to use.



# Creating a Parametric Model

In this topic we discuss how you might develop SysML model elements for simulation (assuming existing knowledge of SysML modeling), configure these elements in the Configure SysML Simulation window, and observe the results of a simulation under some of the different definitions and modeling approaches. The points are illustrated by snapshots of diagrams and screens from the SysML Simulation examples provided in this chapter.

When creating a Parametric Model, you can apply one of three approaches to defining Constraint Equations:

- Defining inline Constraint Equations on a Block element
- Creating re-usable Constraint Blocks, and
- Using connected constraint properties

You would also take into consideration:

- Flows in physical interactions
- Default Values and Initial Values
- Simulation Functions
- Value Allocation, and
- Packages and Imports

## Access

Ribbon	Simulate > System Behavior > Modelica
--------	---------------------------------------

---

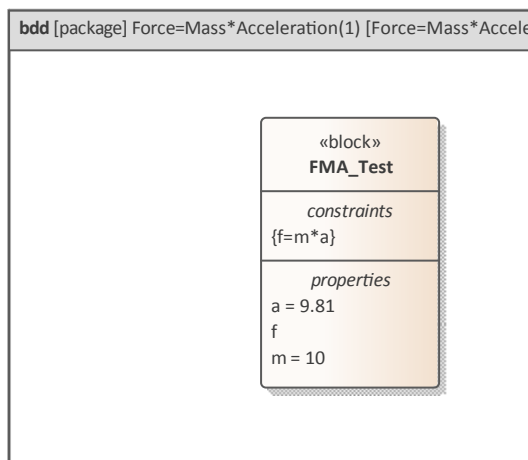
> SysMLSim Configuration Manager

---

## Defining inline Constraint Equations on a Block

Defining constraints directly in a Block is straightforward and is the easiest way to define constraint equations.

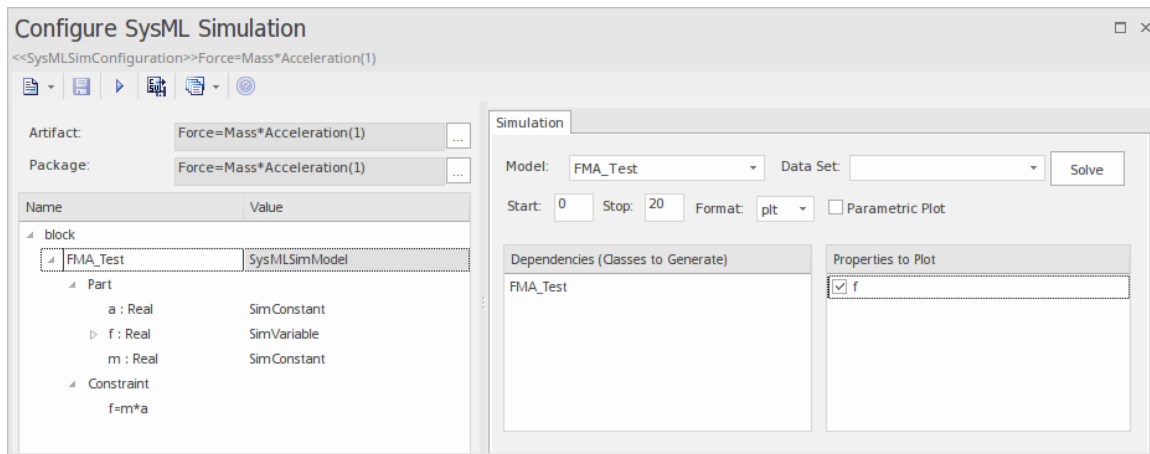
In this figure, constraint ' $f = m * a$ ' is defined in a Block element.



*Tip: You can define multiple constraints in one Block.*

1. Create a SysMLSim Configuration Artifact '**Force=Mass\*Acceleration(1)**' and point it to the Package '**FMA\_Test**'.
2. For '**FMA\_Test**', in the '**Value**' column set '**SysMLSimModel**'.

3. For Parts 'a', 'm' and 'f', in the 'Value' column set 'a' and 'm' to 'SimConstant' and (optionally) set 'f' to 'SimVariable'.
4. On the 'Simulation' tab, in the 'Properties to Plot' panel, select the checkbox against 'f'.
5. Click on the Solve button to run the simulation.

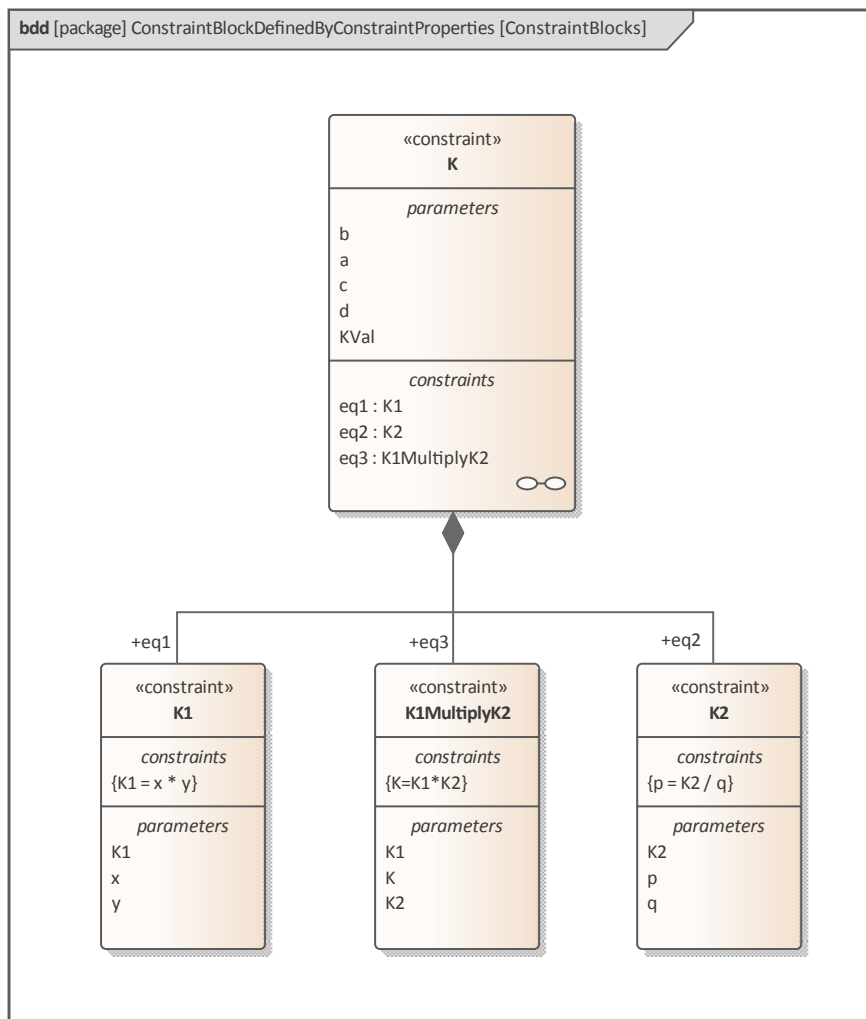


A chart should be plotted with  $f = 98.1$  (which comes from  $10 * 9.81$ ).

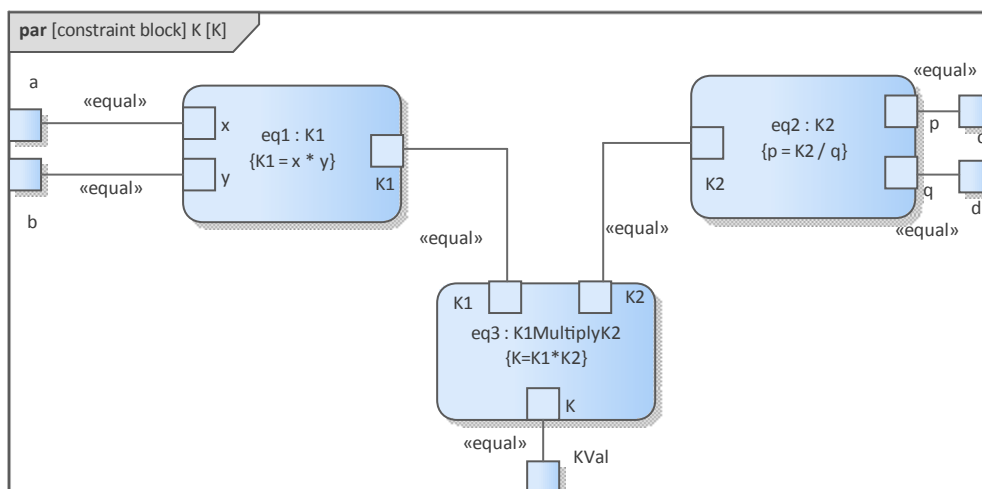
## Connected Constraint Properties

In SysML, constraint properties existing in Constraint Blocks can be used to provide greater flexibility in defining constraints.

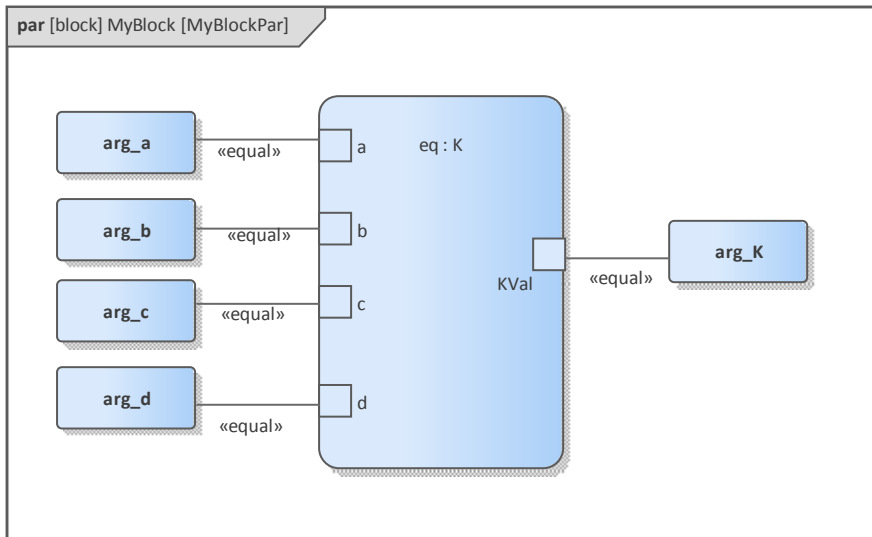
In this figure, Constraint Block 'K' defines parameters 'a', 'b', 'c', 'd' and 'KVal', and three constraint properties 'eq1', 'eq2' and 'eq3', typed to 'K1', 'K2' and 'K1MultiplyK2' respectively.



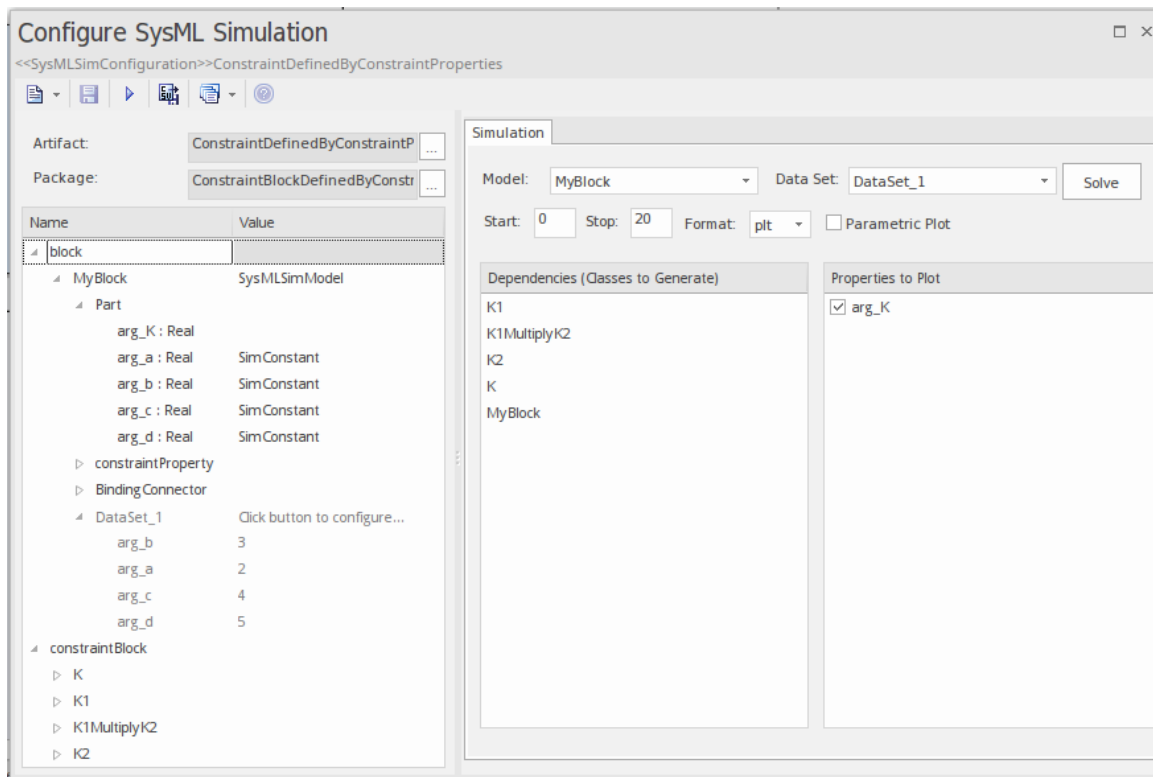
Create a Parametric diagram in Constraint Block 'K' and connect the parameters to the constraint properties with Binding connectors, as shown:



- Create a model MyBlock with five Properties (Parts)
- Create a constraint property 'eq' for MyBlock and show the parameters
- Bind the properties to the parameters



- Provide values ( $\text{arg\_a} = 2$ ,  $\text{arg\_b} = 3$ ,  $\text{arg\_c} = 4$ ,  $\text{arg\_d} = 5$ ) in a data set
- In the 'Configure SysML Simulation' dialog, set 'Model' to 'MyBlock' and 'Data Set' to 'DataSet\_1'
- In the 'Properties to Plot' panel, select the checkbox against 'arg\_K'
- Click on the Solve button to run the simulation



The result 120 (calculated as  $2 * 3 * 4 * 5$ ) will be computed and plotted. This is the same as when we do an expansion with pen and paper:  $K = K1 * K2 = (x*y) * (p*q)$ , then bind with the values  $(2 * 3) * (4 * 5)$ ; we get 120.

What is interesting here is that we intentionally define K2's equation to be  $p = K2 / q$  and this example still works.

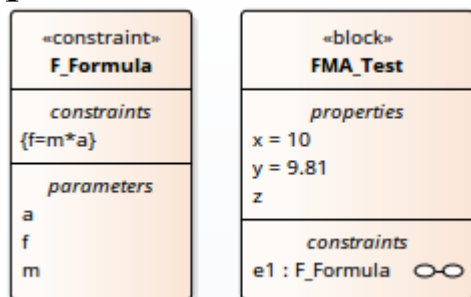
We can easily solve K2 to be  $p * q$  in this example, but in some complex examples it is extremely hard to solve a variable from an equation; however, the Enterprise Architect SysMLSim can still get it right.

In summary, the example shows you how to define a Constraint Block with greater flexibility by constructing the constraint properties. Although we demonstrated only one layer down into the Constraint Block, this mechanism could work on complex models for an arbitrary level of use.

## Creating Reuseable Constraint Blocks

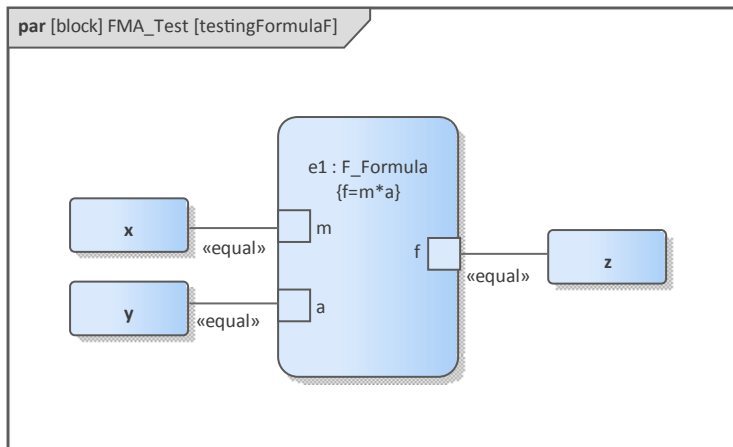
If one equation is commonly used in many Blocks, a Constraint Block can be created for use as a constraint property in each Block. These are the changes we make, based on the previous example:

- Create a Constraint Block element 'F\_Formula' with three parameters 'a', 'm' and 'f', and a constraint ' $f = m * a$ '

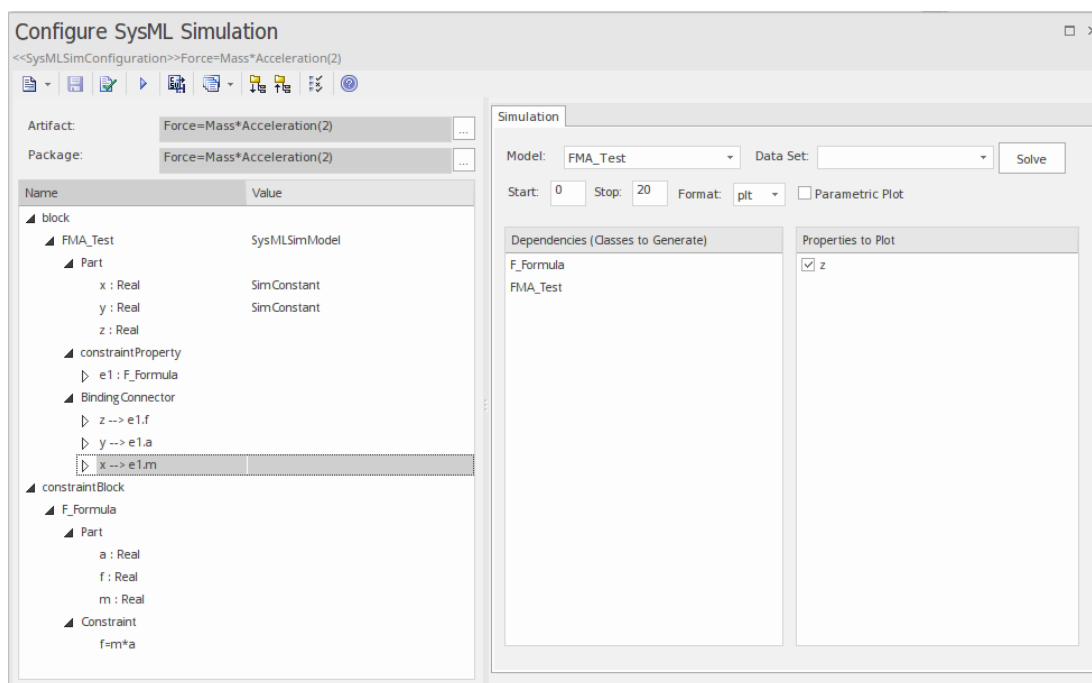


*Tip: Primitive type 'Real' will be applied if property types are empty*

- Create a Block 'FMA\_Test' with three properties 'x', 'y' and 'z', and give 'x' and 'y' the default values '10' and '9.81' respectively
- Create a Parametric diagram in 'FMA\_Test', showing the properties 'x', 'y' and 'z'
- Create a Constraint Property 'e1' typed to 'F\_Formula' and show the parameters
- Draw Binding connectors between 'x—m', 'y—a', and 'f—z' as shown:



- Create a SysMLSimConfiguration Artifact element and configure it as shown in the dialog illustration:
  - In the 'Value' column, set 'FMA\_Test' to 'SysMLSimModel'
  - In the 'Value' column, set 'x' and 'y' to 'SimConstant'
  - In the 'Properties to Plot' panel select the checkbox against 'Z'
  - Click on the Solve button to run the simulation



A chart should be plotted with  $f = 98.1$  (which comes from



10 \* 9.81).

## Flows in Physical Interactions

When modeling for physical interaction, exchanges of conserved physical substances such as electrical current, force, torque and flow rate should be modeled as flows, and the flow variables should be set to the attribute 'isConserved'.

Two different types of coupling are established by connections, depending on whether the flow properties are potential (default) or flow (conserved):

- Equality coupling, for potential (also called effort) properties
- Sum-to-zero coupling, for flow (conserved) properties; for example, according to Kirchoff's Current Law in the electrical domain, conservation of charge makes all charge flows into a point sum to zero

In the generated Modelica code of the 'ElectricalCircuit' example:

```
connector ChargePort
    flow Current i;           //flow keyword will be
generated if 'isConserved' = true
    Voltage v;
```

```
end ChargePort;

model Circuit
  Source source;
  Resistor resistor;
  Ground ground;
equation
  connect(source.p, resistor.n);
  connect(ground.p, source.n);
  connect(resistor.p, source.n);
end Circuit;
```

Each connect equation is actually expanded to two equations (there are two properties defined in ChargePort), one for equality coupling, the other for sum-to-zero coupling:

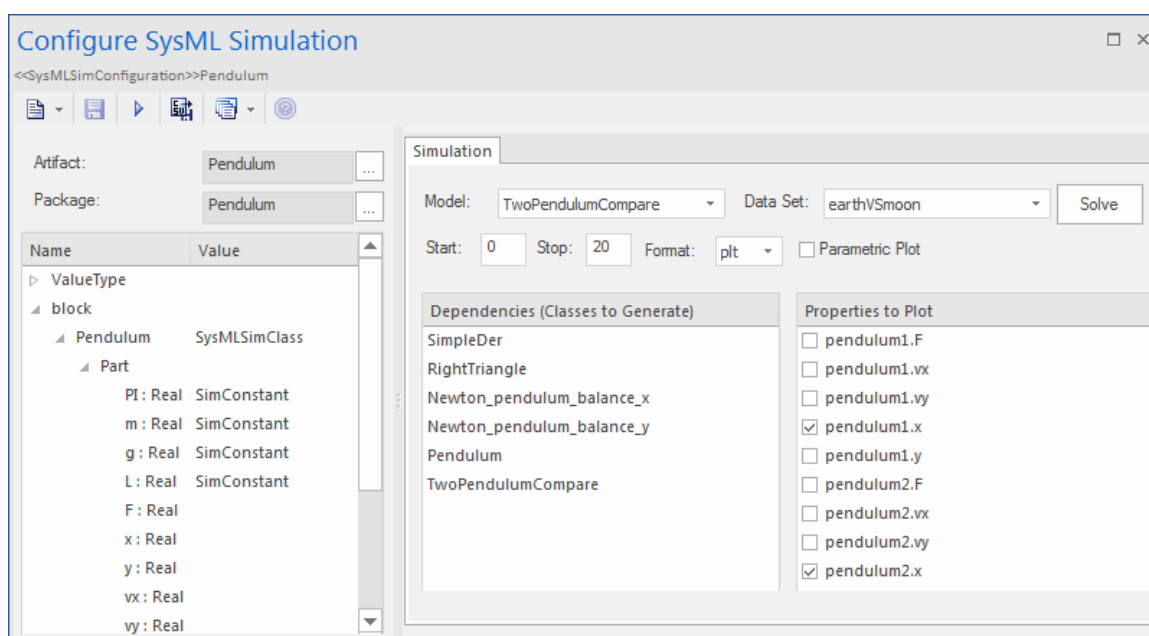
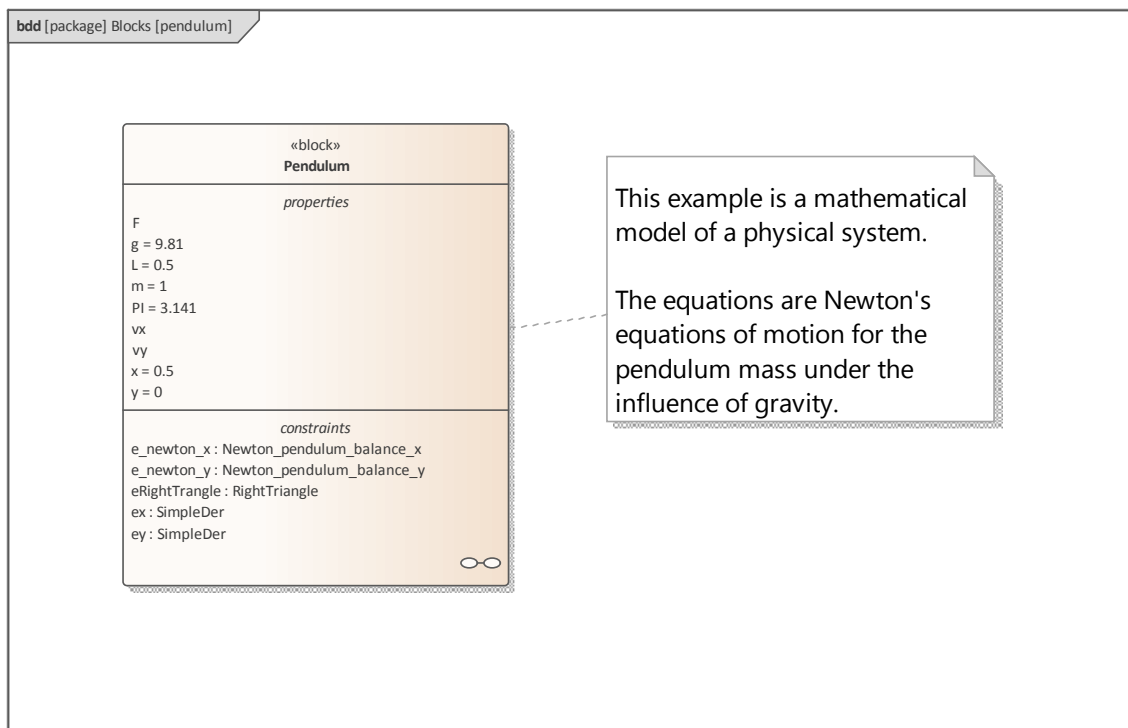
```
source.p.v = resistor.n.v;
source.p.i + resistor.n.i = 0;
```

## Default Value and Initial Values

If initial values are defined in SysML property elements ('Properties' dialog > 'Property' page > 'Initial' field), they can be loaded as the default value for a SimConstant or the initial value for a SimVariable.

In this Pendulum example, we have provided initial values

for properties 'g', 'L', 'm', 'PI', 'x' and 'y', as seen on the left hand side of the figure. Since 'PI' (the mathematical constant), 'm' (mass of the Pendulum), 'g' (Gravity factor) and 'L' (Length of Pendulum) do not change during simulation, set them as 'SimConstant'.



The generated modelica code resembles this:

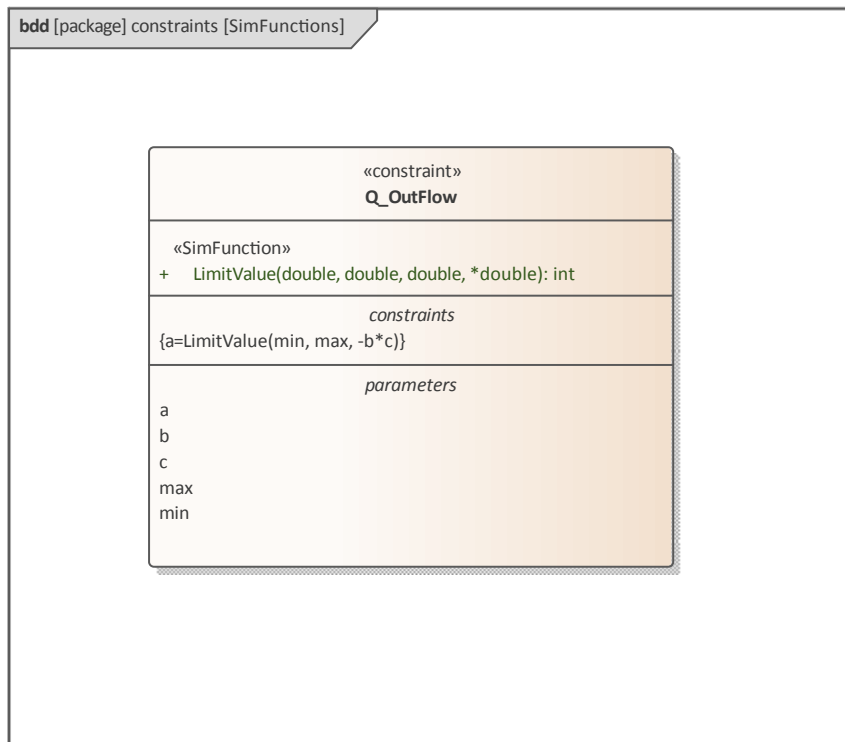
```
class Pendulum
  parameter Real PI = 3.141;
  parameter Real m = 1;
  parameter Real g = 9.81;
  parameter Real L = 0.5;
  Real F;
  Real x (start=0.5);
  Real y (start=0);
  Real vx;
  Real vy;
  .....
equation
  .....
end Pendulum;
```

- Properties 'PI', 'm', 'g' and 'L' are constant, and are generated as a declaration equation
- Properties 'x' and 'y' are variable; their starting values are 0.5 and 0 respectively, and the initial values are generated as modifications

## Simulation Functions

A Simulation function is a powerful tool for writing complex logic, and is easy to use for constraints. This section describes a function from the TankPI example.

In the Constraint Block 'Q\_OutFlow', a function 'LimitValue' is defined and used in the constraint.



- On a Block or Constraint Block, create an operation ('LimitValue' in this example) and open the 'Operations' tab of the Features window
- Give the operation the stereotype 'SimFunction'
- Define the parameters and set the direction to 'in/out'

*Tips: Multiple parameters could be defined as 'out', and the caller retrieves the value in format of:*

*(out1, out2, out3) = function\_name(in1, in2, in3,*

```

in4, ...); //Equation form
      (out1, out2, out3) := function_name(in1, in2,
in3, in4, ...); //Statement form

```

- Define the function body in the text field of the 'Code' tab of the Properties window, as shown:

```

pLim :=
    if p > pMax then
        pMax
    else if p < pMin then
        pMin
    else
        p;

```

When generating code, Enterprise Architect will collect all the operations stereotyped as 'SimFunction' defined in Constraint Blocks and Blocks, then generate code resembling this:

```

function LimitValue
    input Real pMin;
    input Real pMax;
    input Real p;
    output Real pLim;
    algorithm
        pLim :=
            if p > pMax then

```

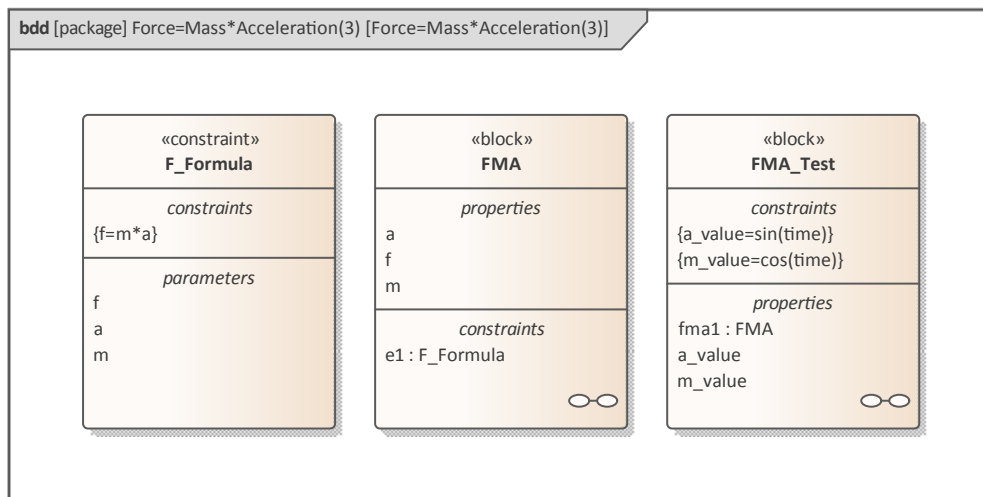
```

        pMax
    else if p < pMin then
        pMin
    else
        p;
end LimitValue;

```

## Value Allocation

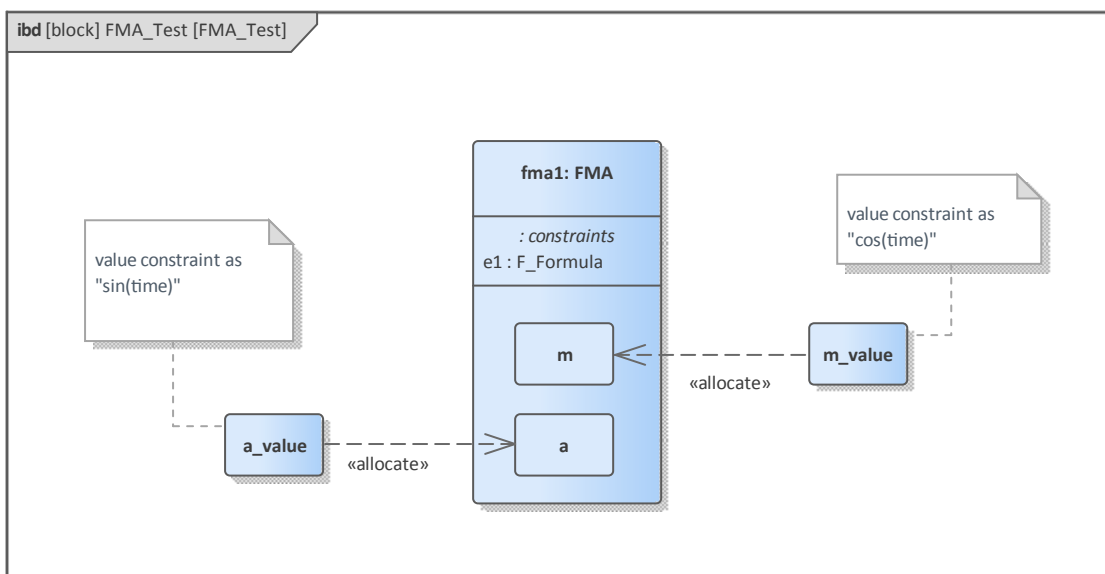
This figure shows a simple model called 'Force=Mass\*Acceleration'.



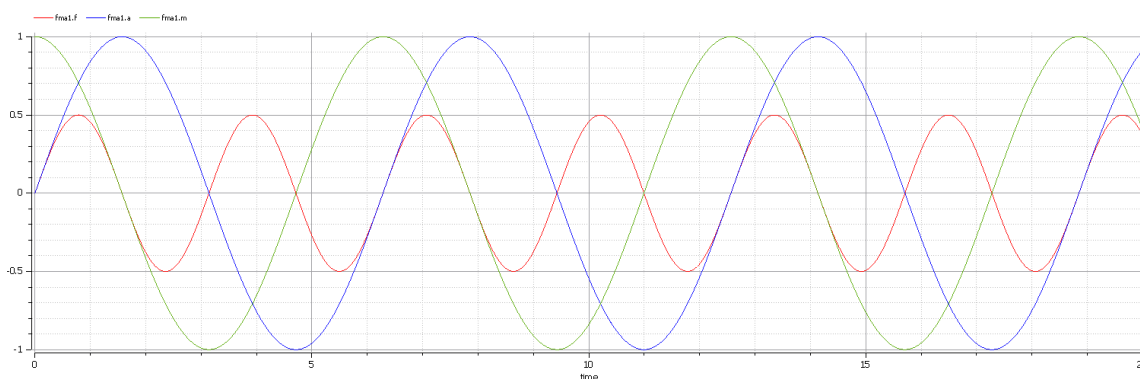
- A Block 'FMA' is modeled with properties 'a', 'f', and 'm' and a constraintProperty 'e1', typed to Constraint Block 'F\_Formula'
- The Block 'FMA' does not have any initial value set on its properties, and the properties 'a', 'f' and 'm' are all variable, so their value change depends on the environment in

which they are simulated

- Create a Block 'FMA\_Test' as a SysMLSimModel and add the property 'fma1' to test the behavior of Block 'FMA'
- Constraint 'a\_value' to be 'sin(time)'
- Constraint 'm\_value' to be 'cos(time)'
- Draw Allocation connectors to allocate values from environment to the model 'FMA'



- Select the 'Properties to Plot' checkboxes against 'fma1.a', 'fma1.m' and 'fma1.f'
- Click on the Solve button to simulate the model

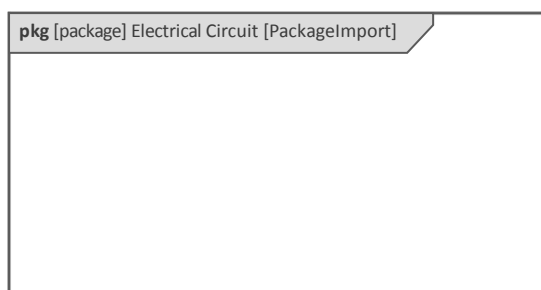




## Packages and Imports

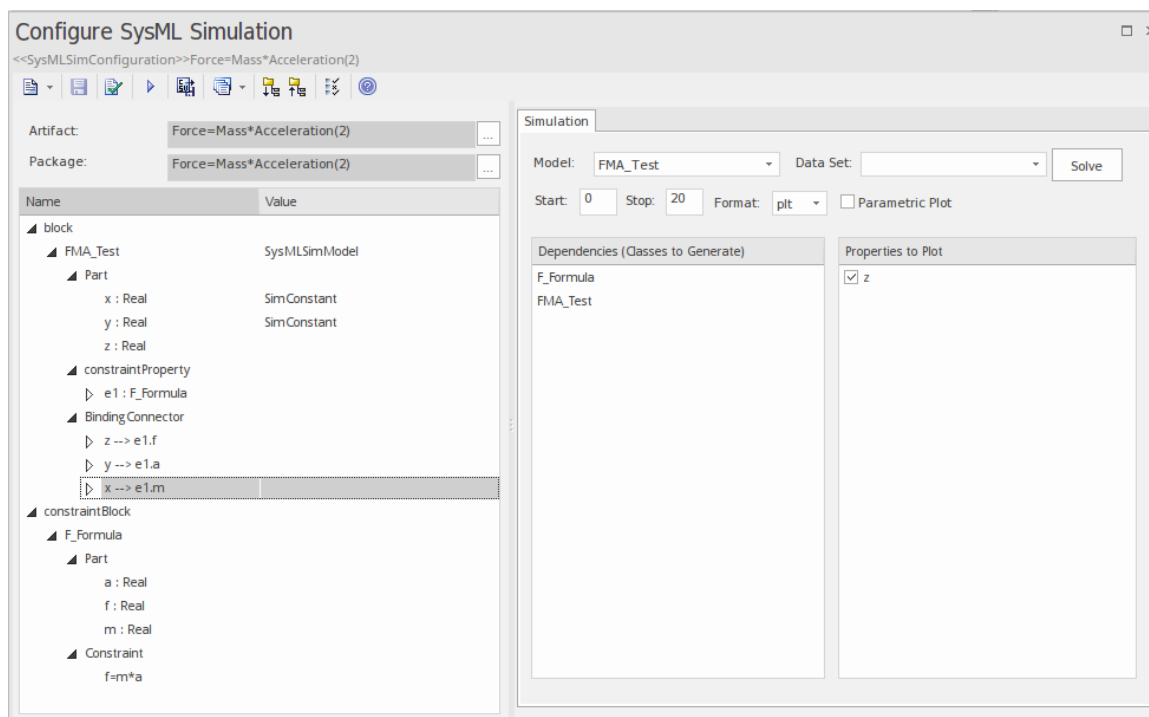
The SysMLSimConfiguration Artifact collects the elements (such as Blocks, Constraint Blocks and Value Types) of a Package. If the simulation depends on elements not owned by this Package, such as Reusable libraries, Enterprise Architect provides an Import connector between Package elements to meet this requirement.

In the Electrical Circuit example, the Artifact is configured to the Package 'ElectricalCircuit', which contains almost all of the elements needed for simulation. However, some properties are typed to value types such as 'Voltage', 'Current' and 'Resistance', which are commonly used in multiple SysML models and are therefore placed in a Package called 'CommonlyUsedTypes' outside the individual SysML models. If you import this Package using an Import connector, all the elements in the imported Package will appear in the SysMLSim Configuration Manager.



# Configure SysML Simulation Window


The Configure SysML Simulation window is the interface through which you can provide run-time parameters for executing the simulation of a SysML model. The simulation is based on a simulation configuration defined in a SysMLSimConfiguration Artifact element.

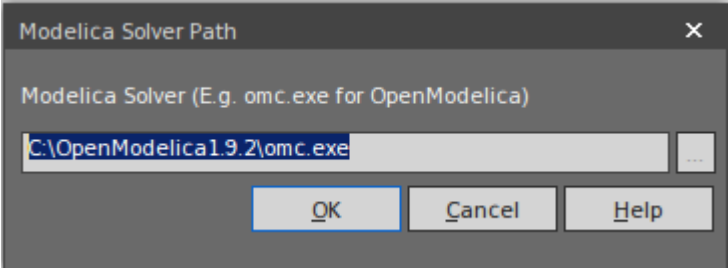






## Access

Ribbon	Simulate > System Behavior > Modelica > SysMLSim Configuration Manager
Other	Double-click on an Artifact with the SysMLSimConfiguration stereotype.



## Toolbar Options

Option	Description
	<p>Click on the drop-down arrow and select from these options:</p> <ul style="list-style-type: none"><li>• Select Artifact — Select and load an existing configuration from an Artifact with the SysMLSimConfiguration stereotype (if one has not already been selected)</li><li>• Create Artifact — Create a new SysMLSimConfiguration or select and load an existing configuration artifact</li><li>• Select Package — Select a Package to scan for SysML elements to configure for simulation</li><li>• Reload — Reload the Configuration Manager with changes to the current Package</li><li>• Configure Modelica Solver — Display the 'Modelica Solver Path' dialog, in which you type or browse for the path to the Modelica solver to use</li></ul>

	
	<p>Click on this button to save the configuration to the current Artifact.</p>
	<p>Click on this button to generate, compile and run the current configuration, and display the results.</p>
	<p>After simulation, the result file is generated in either plt, mat or csv format. That is, with the filename:</p> <ul style="list-style-type: none"> <li>• ModelName_res.plt</li> <li>• ModelName_res.mat or</li> <li>• ModelName_res.csv</li> </ul> <p>Click on this button to specify a directory into which Enterprise Architect will copy the result file.</p>
	<p>Click on this button to select from these options:</p> <ul style="list-style-type: none"> <li>• Generate Modelica Code — Generate the code without compiling or running it</li> <li>• Open Modelica Simulation Directory</li> </ul>


	<p>— Open the directory into which Modelica code will be generated</p> <ul style="list-style-type: none"> <li>• Edit Modelica Templates — Customize the code generated for Modelica, using the Code Template Editor</li> </ul>
--	--

## Simulation Artifact and Model Selection

Field	Action
Artifact	Click on the  icon and either browse for and select an existing SysMLSimConfiguration Artifact, or create a new Artifact.
Package	<p>If you have specified an existing SysMLSimConfiguration Artifact, this field defaults to the Package containing the SysML model associated with that Artifact.</p> <p>Otherwise, click on the  icon and browse for and select the Package containing the SysML model to configure for simulation. You must specify (or create) the Artifact before selecting the Package.</p>

## Package Objects

This table discusses the types of object from the SysML model that will be listed under the 'Name' column in the Configure SysML Simulation window, to be processed in the simulation. Each object type expands to list the named objects of that type, and the properties of each object that require configuration in the 'Value' column.

Many levels of the object types, names and properties do not require configuration, so the corresponding 'Value' field does not accept input. Where input is appropriate and accepted, a drop-down arrow displays at the right end of the field; when you click on this a short list of possible values displays for selection. Certain values (such as 'SimVariable' for a Part) add further layers of parameters and properties, where you click on the  button to, again, select and set values for the parameters. For datasets, the input dialog allows you to type in or import values, such as initial or default values; see the *Model Analysis using Datasets* topic.

Element Type	Behavior
ValueType	ValueType elements either generalize from a primitive type or are substituted by SysMLSimReal for simulation.

Block	<p>Block elements mapped to SysMLSimClass or SysMLSimModel elements support the creation of data sets. If you have defined multiple data sets in a SysMLSimClass (which can be generalized), you must identify one of them as the default (using the context menu option 'Set as Default Dataset').</p> <p>As a SysMLSimModel is a possible top-level element for a simulation, and will not be generalized, if you have defined multiple datasets the dataset to use is chosen during the simulation.</p>
Properties	<p>Properties within a Block can be configured to be either SimConstants or SimVariables. For a SimVariable, you configure these attributes:</p> <ul style="list-style-type: none"><li>• isContinuous — determines whether the property value varies continuously ('true', the default) or discretely ('false')</li><li>• isConserved — determines whether values of the property are conserved ('true') or not ('false', the default); when modeling for physical interaction, the interactions include exchanges of conserved physical substances such as electrical current, force or fluid flow</li><li>• changeCycle — specifies the time</li></ul>

	<p>interval at which a discrete property value changes; the default value is '0'</p> <ul style="list-style-type: none"> <li>- changeCycle can be set to a value other than 0 only when isContinuous = 'false'</li> <li>- The value of changeCycle must be positive or equal to 0</li> </ul>
Port	No configuration required.
SimFunction	<p>Functions are created as operations in Blocks or Constraint Blocks, stereotyped as 'SimFunction'.</p> <p>No configuration is required in the Configure SysML Simulation window.</p>
Generalization	No configuration required.
Binding Connector	<p>Binds a property to a parameter of a constraint property.</p> <p>No configuration required.</p>
Connector	<p>Connects two Ports.</p> <p>No configuration required in the Configure SysML Simulation view.</p> <p>However, you might have to configure the properties of the Port's type by determining whether the attribute</p>



	isConserved should be set as 'False' (for potential properties, so that equality coupling is established) or 'True' (for flow/conserved properties, so that sum-to-zero coupling is established).
Constraint Block	No configuration required.

## Simulation Tab

This table describes the fields of the 'Simulation' tab on the Configure SysML Simulation view.

Field	Action
Model	Click on the drop-down arrow and select the top-level node (a SysMLSimModel element) for the simulation. The list is populated with the names of the Blocks defined as top-level, model nodes.
Data Set	Click on the drop-down arrow and select the dataset for the selected model.
Start	Type in the initial wait time before which the simulation is started, in seconds

	(default value is 0).
Stop	Type in the number of seconds for which the simulation will execute.
Format	Click on the drop-down arrow and select either 'plt', 'csv' or 'mat' as the format of the result file, which could potentially be used by other tools.
Parametric Plot	<ul style="list-style-type: none"><li>• Select this checkbox to plot Legend A on the y-axis against Legend B on the x-axis.</li><li>• Deselect the checkbox to plot Legend(s) on the y-axis against time on the x-axis</li></ul> <p>Note: With the checkbox selected, you must select two properties to plot.</p>
Dependencies	Lists the types that must be generated to simulate this model.
Properties to Plot	Provides a list of variable properties that are involved with the simulation. Select the checkbox against each property to plot.



# Model Analysis using Datasets



Every SysML Block used in a Parametric model can, within the Simulation configuration, have multiple datasets defined against it. This allows for repeatable simulation variations using the same SysML model.

A Block can be typed as a SysMLSimModel (a top-level node that cannot be generalized or form part of a composition) or as a SysMLSimClass (a lower-level element that can be generalized or form part of a composition). When running a simulation on a SysMLSimModel element, if you have defined multiple datasets, you can specify which dataset to use. However, if a SysMLSimClass within the simulation has multiple datasets, you cannot select which one to use during the simulation and must therefore identify one dataset as the default for that Class.

## Access

Ribbon	Simulate > System Behavior > Modelica > SysMLSim Configuration Manager > in "block" group > Name column > Context menu on block element > Create Simulation DataSet
--------	---

## Dataset Management

Task	Action
Create	To create a new dataset, right-click on a Block name and select the 'Create Simulation Dataset' option. The dataset is added to the end of the list of components underneath the Block name. Click on the  button to set up the dataset on the 'Configure Simulation Data' dialog (see the <i>Configure Simulation Data</i> table).
Duplicate	To duplicate an existing dataset as a base for creating a new dataset, right-click on the dataset name and select the 'Duplicate' option. The duplicate dataset is added to the end of the list of components underneath the Block name. Click on the  button to edit the dataset on the 'Configure Simulation Data' dialog (see the <i>Configure Simulation Data</i> table).
Delete	To remove a dataset that is no longer required, right-click on the dataset and

	select the 'Delete Dataset' option.
Set Default	To set the default dataset used by a SysMLSimClass when used as a property type or inherited (and when there is more than one dataset), right-click on the dataset and select the 'Set as Default' option. The name of the default dataset is highlighted in bold. The properties used by a model will use this default configuration unless the model overrides them explicitly.

## Configure Simulation Data

This dialog is principally for information. The only column in which you can directly add or change data is the 'Value' column.

Configure Simulation Data				
Attribute	Stereotype	Type	Default Value	Value
▷ pendulum1	SimVariable	Pendulum		
▾ pendulum2	SimVariable	Pendulum		
PI	SimConstant	Real	3.1415926	
m	SimConstant	Real	1	
g	SimConstant	Real	9.81	1.6
L	SimConstant	Real	0.5	0.8
F	SimVariable	Real		
x	SimVariable	Real	0.5	0.8
y	SimVariable	Real	0	
vx	SimVariable	Real		
vy	SimVariable	Real		

Column	Description
Attribute	The 'Attribute' column provides a tree view of all the properties in the Block being edited.
Stereotype	The 'Stereotype' column identifies, for each property, if it has been configured to be a constant for the duration of the simulation or variable, so that the value is expected to change over time.
Type	The 'Type' column describes the type used for simulation of this property. It can be either a primitive type (such as 'Real') or a reference to a Block contained in the model. Properties referencing Blocks will show the child properties specified by the referenced Block below

	them.
Default Value	The 'Default Value' column shows the value that will be used in the simulation if no override is provided. This can come from the 'Initial Value' field in the SysML model or from the default dataset of the parent type.
Value	The 'Value' column allows you to override the default value for each primitive value.
Export / Import	Click on these buttons to modify the values in the current dataset using an external application such as a spreadsheet, and then re-import them to the list.



# Modeling and Simulation with Modelica Library

This feature is available from Enterprise Architect Release 14.1.

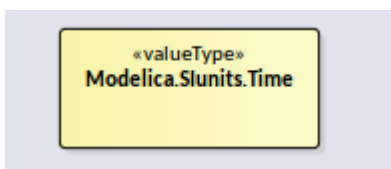
The Modelica Libraries are comprehensive resources that provide many useful types, functions and models. When creating SysML models in Enterprise Architect, you can reference resources available in the Modelica Libraries.

## Referencing a Type defined in Modelica Library

To configure a simulation to reference a Modelica Library, you first create a ValueType element pointing to the Modelica library, and register this in the Simulation configuration.

### **First, create an element for a Referenced Modelica Type**

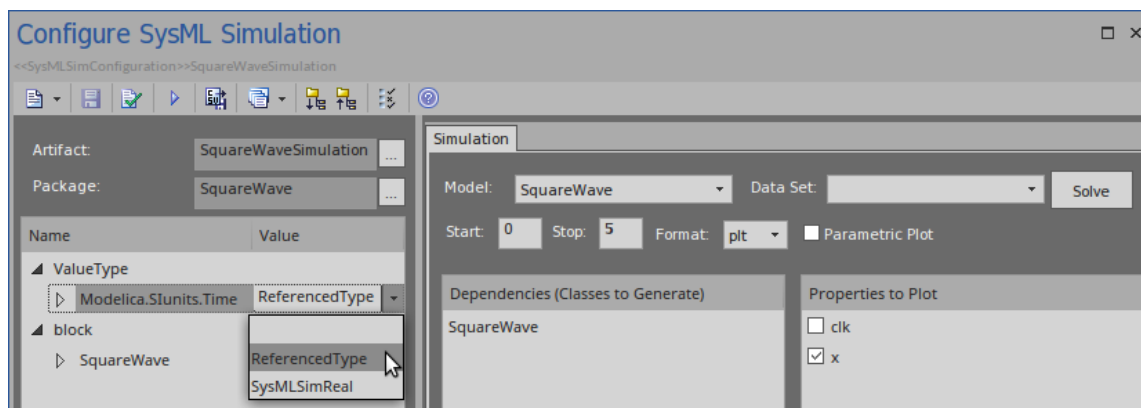
- Create a ValueType element with the full name of the Modelica Library path:



Configure the ValueType element as 'ReferencedType':

- Double-click on the SysMLSimConfiguration element to open the 'Configure SysML Configuration' tab

- Navigate to the ValueType element
- In the drop-down field set the value to 'ReferencedType'



As the ValueType element is configured as 'ReferencedType', the element will not display in the 'Dependencies' list and will not be generated as a new Class definition to the Modelica file.

### **Next, set the type for a Property to the ValueType element**

In Enterprise Architect, a SysML Property can be set to be a primitive type or an element such as a Block or a ValueType.

#### **Option 1:**

- Select the Property (Part or Port)
- Press Ctrl+2 to open the Properties window
- Switch to the 'Property' tab and choose 'Select Type...'
- Browse to the ValueType element you created

#### **Option 2:**

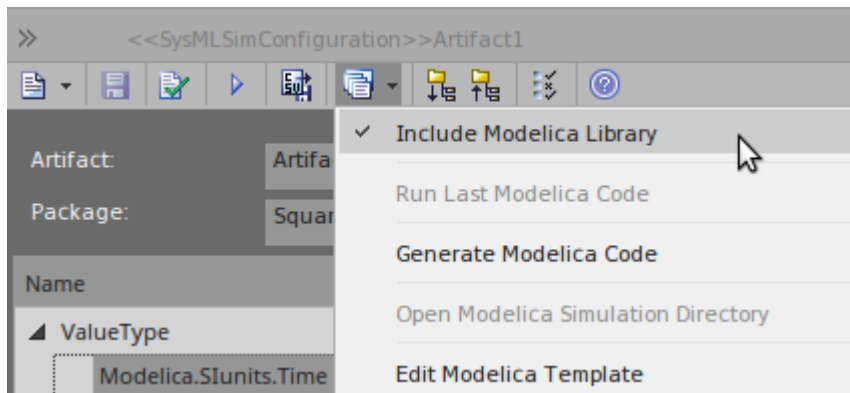
- Select the Property (Part or Port)
- Press Ctrl+L on the Property

- Browse to the ValueType element you created

## Including a Modelica Library in a Simulation

When using referenced types from a Modelica library in a model, you must load the Modelica model in the environment for the simulation to work.

- Expand the menu option and select 'Include Modelica Library'



- If this option is ticked, this function will be generated to 'Solve.mos' by default:

*loadModel(Modelica);*

Click [Here](#) for a detailed description of the *loadModel()* scripting function.

## Customize the Modelica Script Template

You can modify the Modelica script template to add extra libraries required by the model and simulation.

## Access

Ribbon: Code | Configure | Options | Edit Code Templates

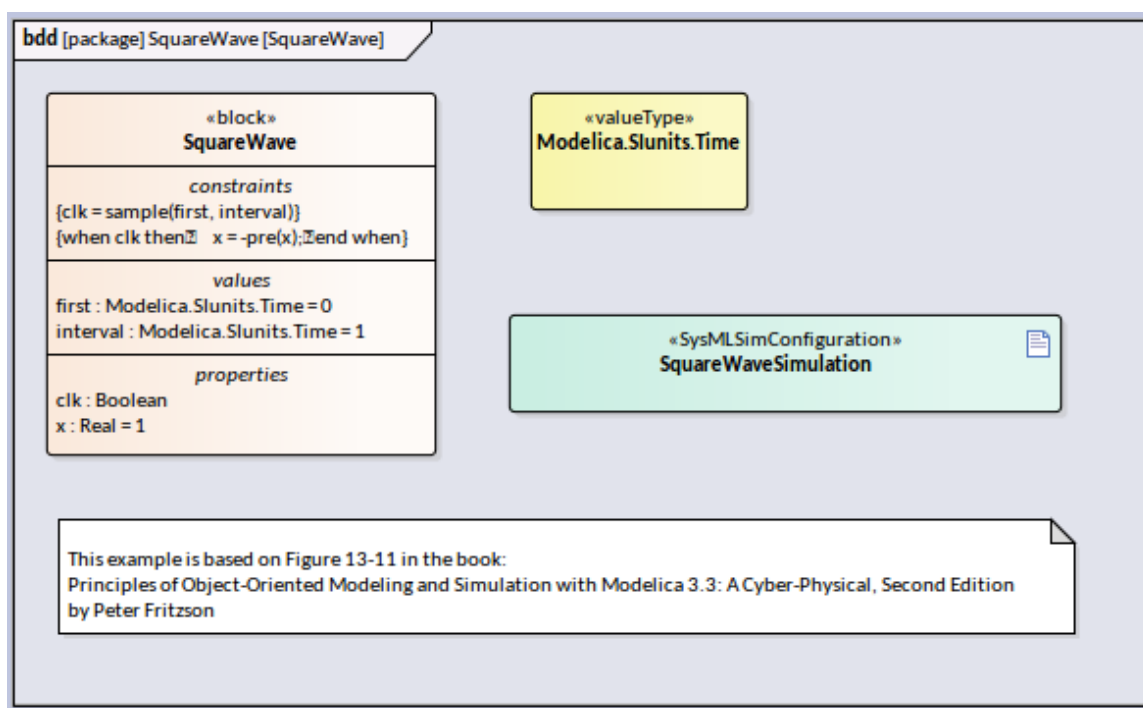
In the 'Language' field select 'Modelica' and in the Scripts list select 'SysMLSim Script'

```
10 %if sysmlSim_IncludeLibrary == "T"%
11 msg := "----- Loading Modelica Model... -----";
12 loadModel(Modelica);
13 %endIf%
```

As you are appending extra libraries after 'loadModel(Modelica)', the libraries' resources can be referenced by your model.

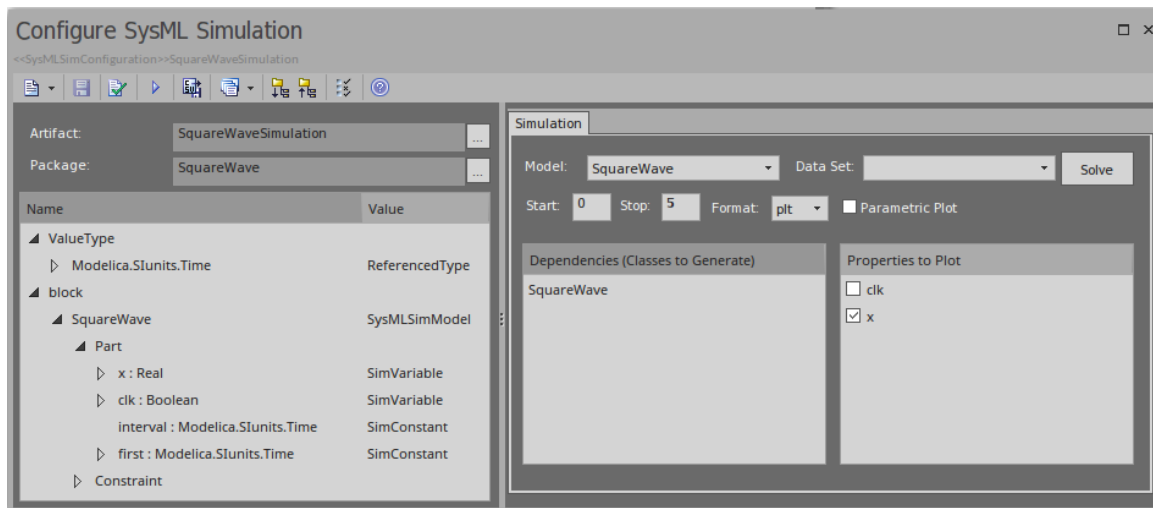
## SquareWave Example

This example is based on Figure 13-11 in: *Principles of Object-Oriented Modeling and Simulation with Modelica 3.3: A Cyber-Physical*, Second Edition, by Peter Fritzson.

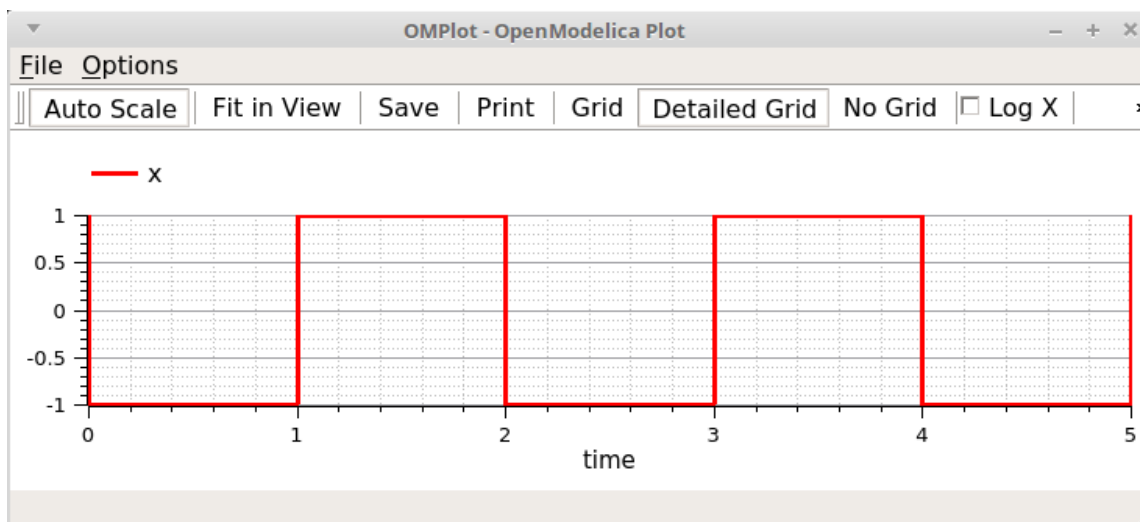


In this example:

- We create a ValueType *Modelica.SIunits.Time*, which is used for the property *first* and *interval* of the Block *SquareWave*
- ValueType *Modelica.SIunits.Time* is configured as 'ReferencedType' in the SysML Simulation window
- Select the menu item 'Include Modelica Library'

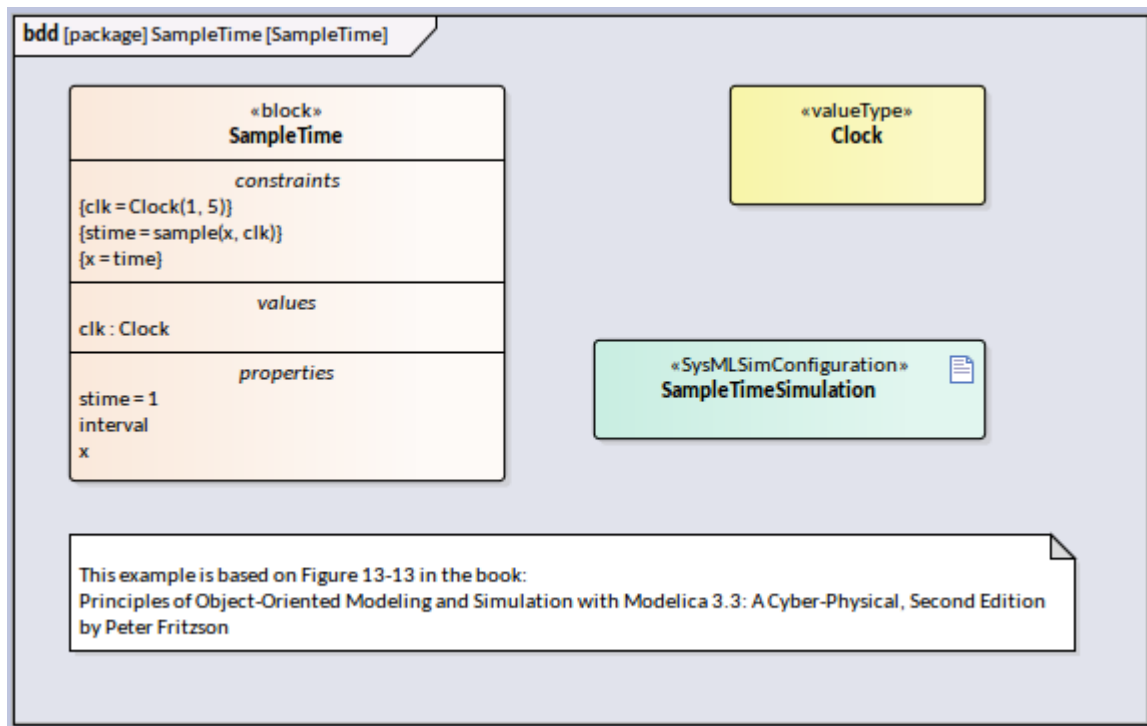


Run the simulation; the variable *x* is plotted like this:



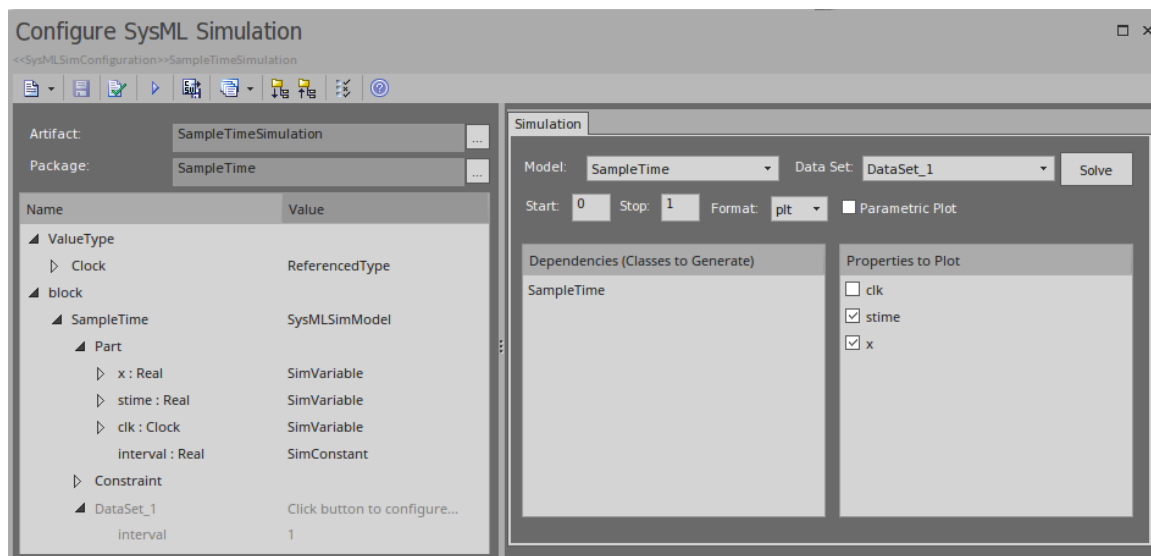
## SampleTime Example

This example is based on Figure 13-13 in: *Principles of Object-Oriented Modeling and Simulation with Modelica 3.3: A Cyber-Physical*, Second Edition, by Peter Fritzson.

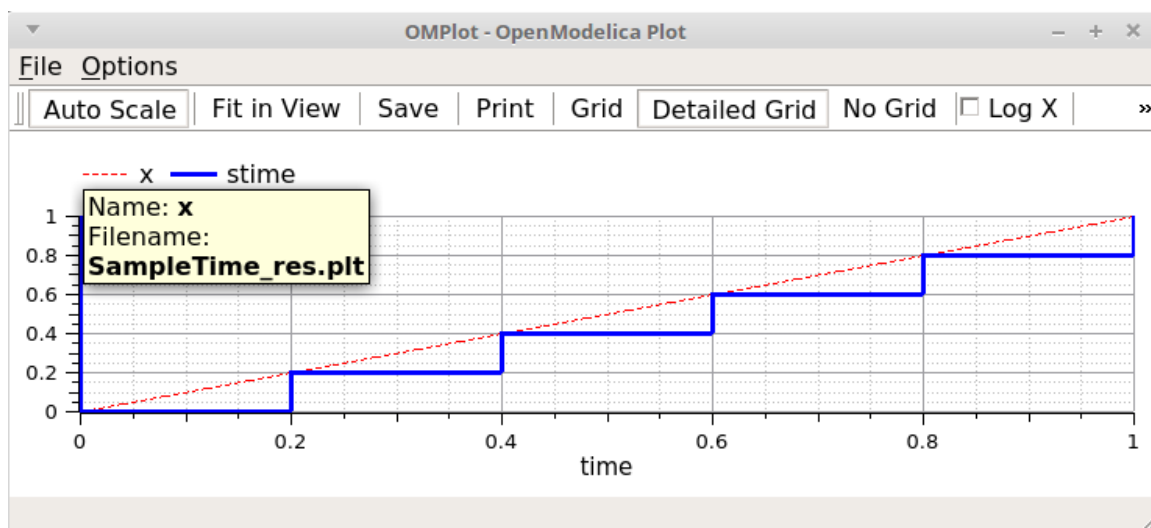


In this example:

- We created a Value Type *Clock*, which is used for the property *clk* of Block *SampleTime*
- Value Type *Clock* is configured as 'ReferencedType' in the SysML Simulation window
- The menu item 'Include Modelica Library' is unselected



Run the simulation; the variable  $x$  and  $stime$  plot resembles this:



# SysML Simulation Examples

This section provides a worked example for each of: creating a SysML model for a domain, simulating it, and evaluating the results of the simulation. These examples apply the information discussed in the earlier topics.

## Electrical Circuit Simulation Example

The first example is of the simulation of an electrical circuit. The example starts with an electrical circuit diagram and converts it to a parametric model. The model is then simulated and the voltage at the source and target wires of a resistor are evaluated and compared to the expected values.

[Electrical Circuit Simulation Example](#)

## Mass-Spring-Damper Oscillator Simulation Example

The second example uses a simple physical model to demonstrate the oscillation behavior of a string under tension.

[Mass-Spring-Damper Oscillator Simulation Example](#)



## Water Tank Pressure Regulator

The final example shows the water levels of two water tanks where the water is being distributed between them. We first simulate a well-balanced system, then we simulate a system where the water will overflow from the second tank.

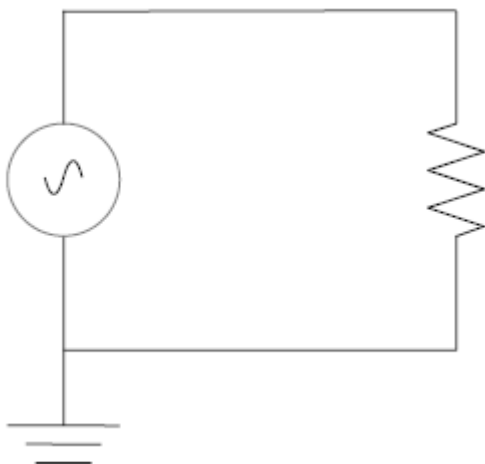
[Water Tank Pressure Regulator](#)

# Electrical Circuit Simulation Example

In this section, we will walk through the creation of a SysML parametric model for a simple electrical circuit, and then use a parametric simulation to predict and chart the behavior of that circuit.

## Circuit Diagram

The electrical circuit we are going to model, shown here, uses a standard electrical circuit notation.

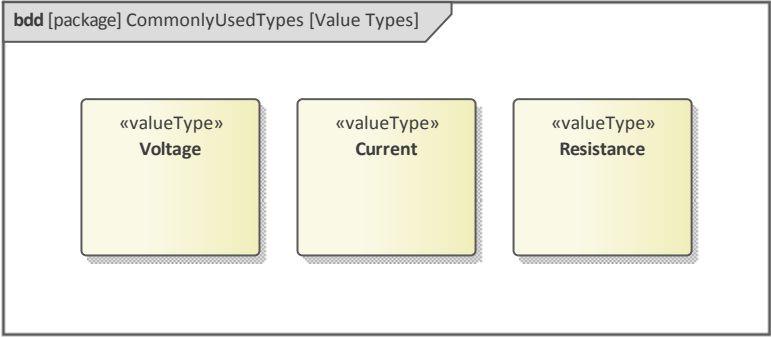


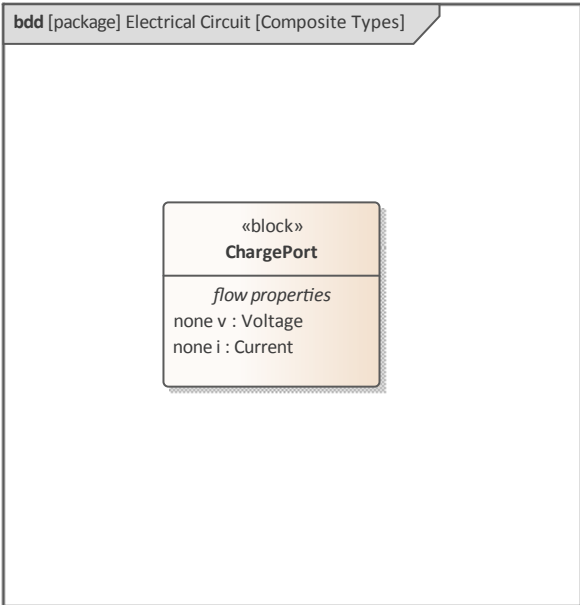
The circuit includes an AC power source, a ground and a resistor, connected to each other by wires.

## Create SysML Model

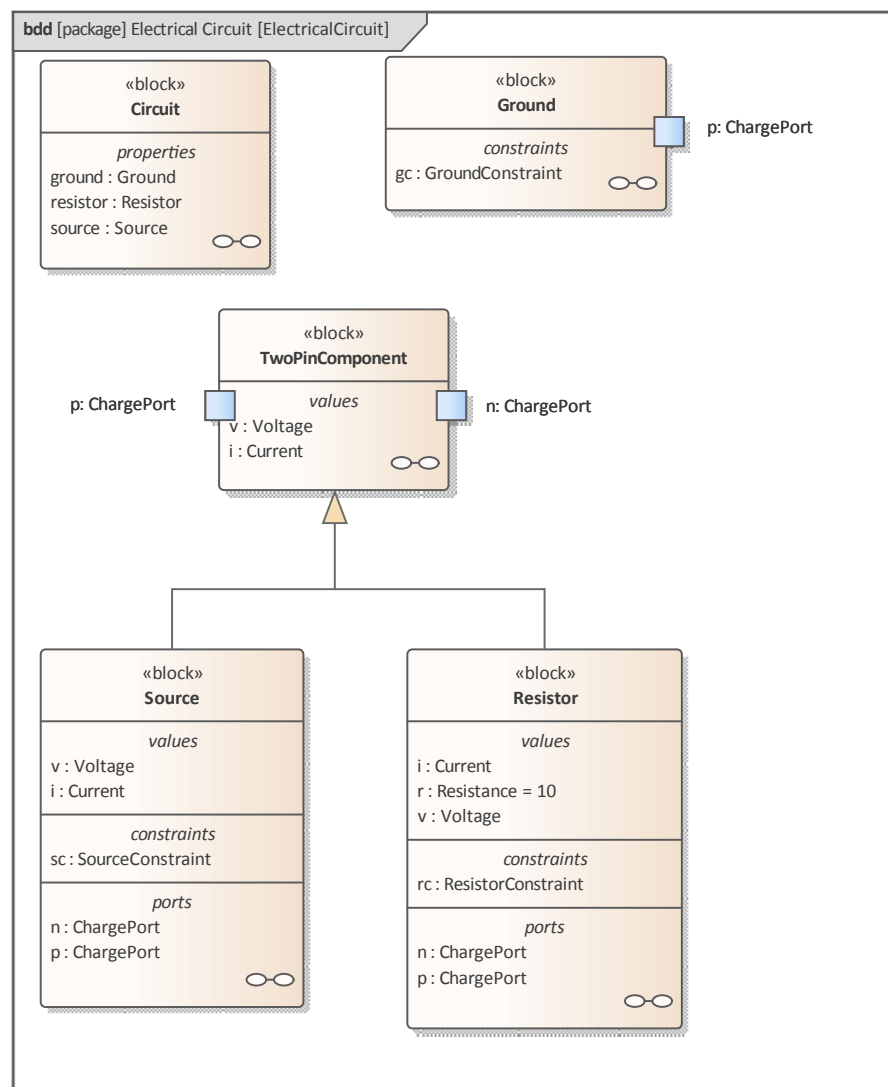
This table shows how we can build up a complete SysML

model to represent the circuit, starting at the lowest level types and building up the model one step at a time.

Component	Action
Types	<p>Define Value Types for the Voltage, Current and Resistance. Unit and quantity kind are not important for the purposes of simulation, but would be set if defining a complete SysML model. These types will be generalized from the primitive type 'Real'. In other models, you can choose to map a Value Type to a corresponding simulation type separate from the model.</p>  <p>bdd [package] CommonlyUsedTypes [Value Types]</p> <pre> classDiagram     class Voltage["«valueType» Voltage"]     class Current["«valueType» Current"]     class Resistance["«valueType» Resistance"]   </pre> <p>Additionally, define a composite type called ChargePort, which includes properties for both Current and Voltage. This type allows us to represent the electrical energy at the connectors between components.</p> <p>ElectricalCircuit : Composite Types</p>

	
Blocks	<p>In SysML, the circuit and each of the components will be represented as Blocks. Create a Circuit Block in a Block Definition diagram (BDD). The circuit has three parts: a source, a ground, and a resistor. These parts are of different types, with different behaviors. Create a Block for each of these part types. The three parts of the Circuit Block are connected through Ports, which represent electrical pins. The source and resistor have a positive and a negative pin. The ground has only one pin, which is positive. Electricity (electric charge) is transmitted through the pins. Create an abstract block 'TwoPinComponent' with two Ports (pins). The two Ports are named 'p' (positive) and 'n' (negative), and they are of type ChargePort.</p>

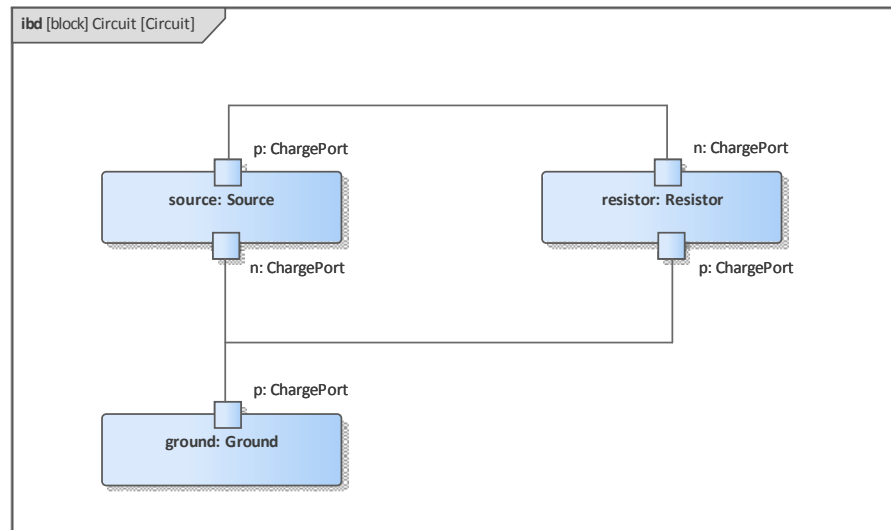
This figure shows what the BDD should look like, with the Blocks Circuit, Ground, TwoPinComponent, Source and Resistor.



## Internal Structure

Create an Internal Block diagram (IBD) for Circuit. Add properties for the Source, Resistor and Ground, typed by the corresponding Blocks. Connect the Ports with connectors. The positive pin of the

Source is connected to the negative pin of the Resistor. The positive pin of the Resistor is connected to the negative pin of the Source. The Ground is also connected to the negative pin of the Source.



Notice that this follows the same structure as the original circuit diagram, but the symbols for each component have been replaced with properties typed by the Blocks we have defined.

## Constraints

Equations define mathematical relationships between numeric properties. In SysML, equations are represented as constraints in Constraint Blocks. Parameters of Constraint Blocks correspond to SimVariables and SimConstants of Blocks ('i', 'v', 'r' in this

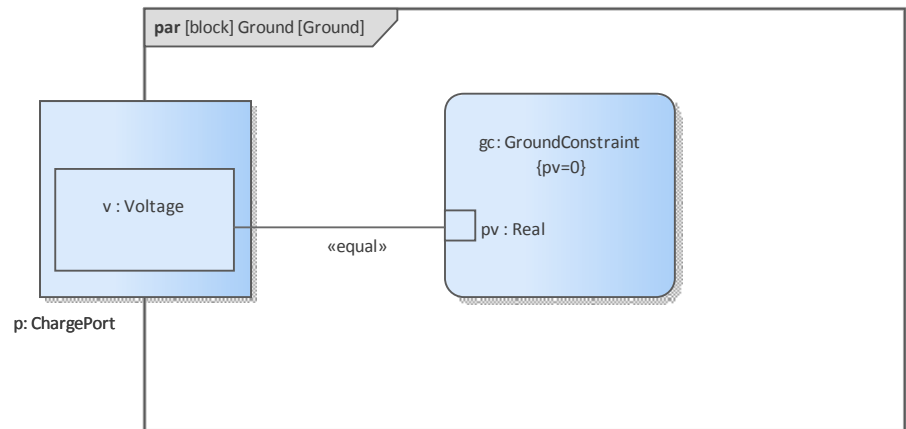
example), as well as to SimVariables present in the type of the Ports ('pv', 'pi', 'nv', 'ni' in this example).

### Create a Constraint Block

'TwoPinComponentConstraint' to define parameters and equations common to sources and resistors. The equations should state that the voltage of the component is equal to the difference between the voltages at the positive and negative pins. The current of the component is equal to the current going through the positive pin. The sum of the currents going through the two pins must add up to zero (one is the negative of the other). The Ground constraint states that the voltage at the Ground pin is zero. The Source constraint defines the voltage as a sine wave with the current simulation time as a parameter. This figure shows how these constraints should look in a BDD.

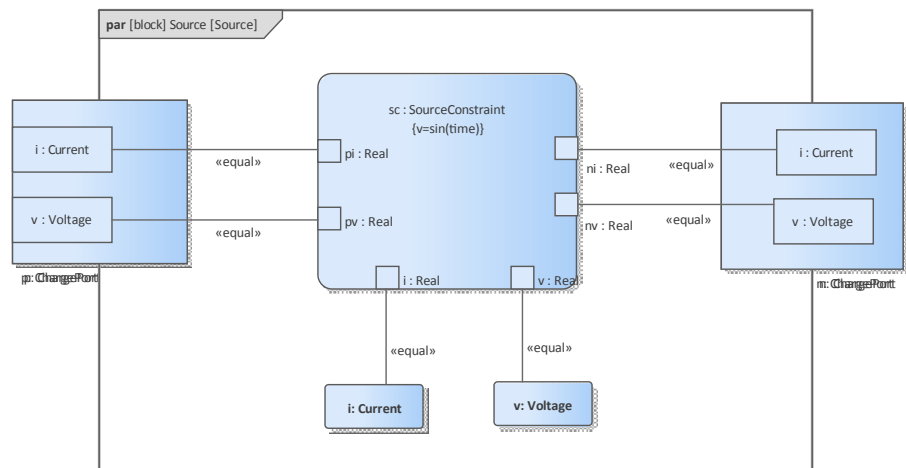
	<pre> classDiagram     class TwoPinComponentConstraint {         &lt;&lt;constraint&gt;&gt;         constraints {             pi+ni=0             i=pi             v=pv-nv         }         parameters {             i: Real             ni: Real             nv: Real             pi: Real             pv: Real             v: Real         }     }     class ResistorConstraint {         &lt;&lt;constraint&gt;&gt;         constraints {             v=r*i         }         parameters {             i: Real             ni: Real             nv: Real             pi: Real             pv: Real             r: Real             v: Real         }     }     class SourceConstraint {         &lt;&lt;constraint&gt;&gt;         constraints {             v=sin(time)         }         parameters {             i: Real             ni: Real             nv: Real             pi: Real             pv: Real             v: Real         }     }     class GroundConstraint {         &lt;&lt;constraint&gt;&gt;         constraints {             pv=0         }         parameters {             pv: Real         }     }     TwoPinComponentConstraint &lt; -- ResistorConstraint     TwoPinComponentConstraint &lt; -- SourceConstraint   </pre>
<p><b>Bindings</b></p>	<p>The values of Constraint parameters are equated to variable and constant values with binding connectors. Create Constraint properties on each Block (properties typed by Constraint Blocks) and bind the Block variables and constants to the Constraint parameters to apply the Constraint to the Block. These figures show the bindings for the Ground, the Source and the Resistor respectively. For the Ground constraint, bind gc.pv to p.v.</p>





For the Source constraint, bind:

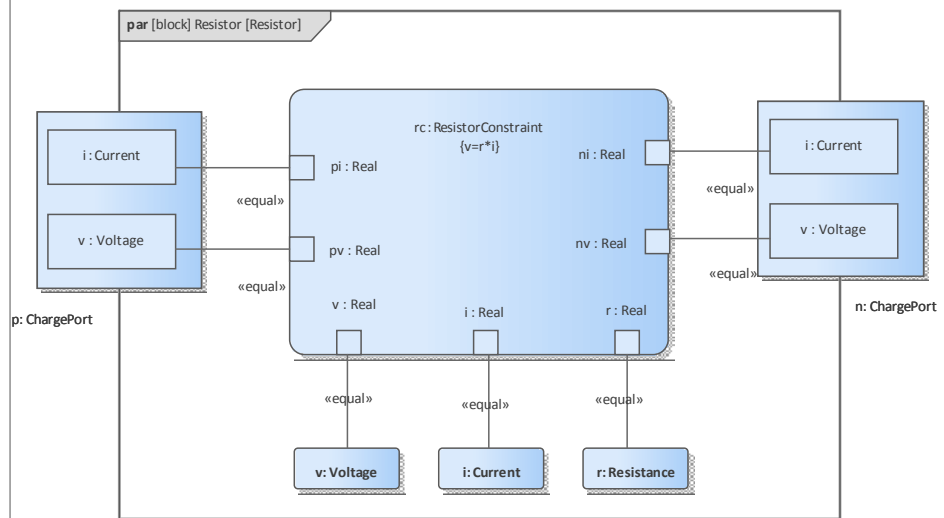
- sc.pi to p.i
- sc.pv to p.v
- sc.v to v
- sc.i to i
- sc.ni to n.i and
- sc.nv to n.v



For the Resistor constraint, bind:

- rc.pi to p.i
- rc.pv to p.v


- rc.v to v
- rc.i to i
- rc.ni to n.i
- rc.nv to n.v and
- rc.r to r



## Configure Simulation Behavior

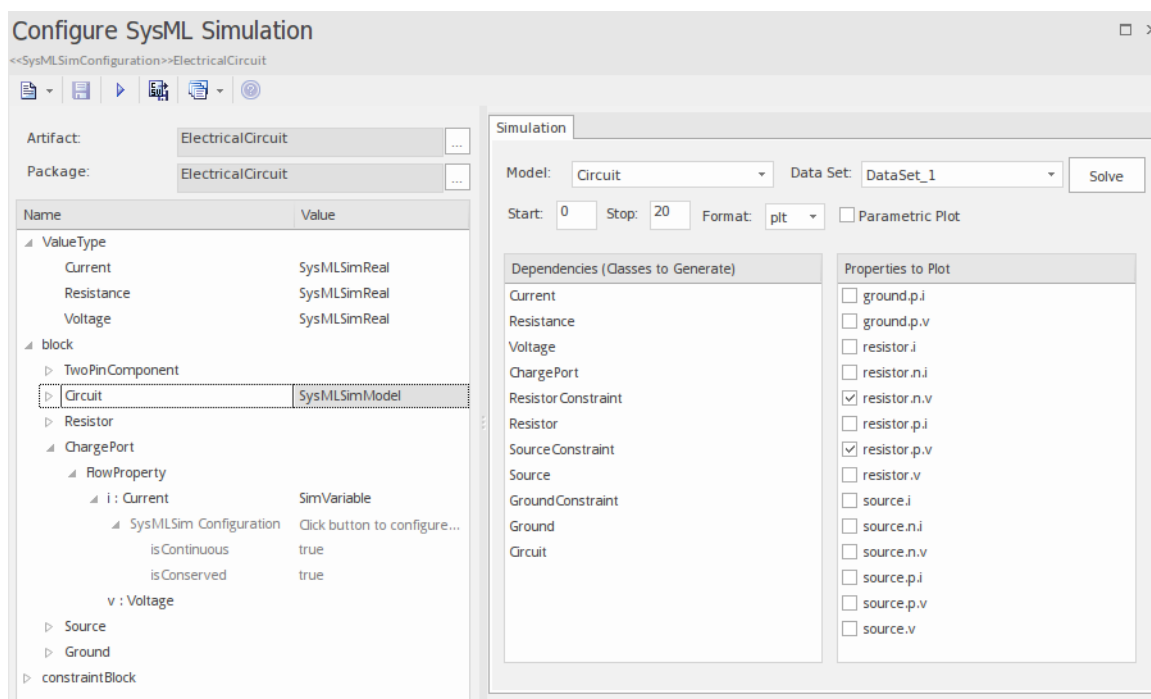
This table shows the detailed steps of the configuration of SysMLSim.

Step	Action
SysMLSimC onfiguration Artifact	<ul style="list-style-type: none"> <li>• Select 'Simulate &gt; System Behavior &gt; Modelica &gt; SysMLSim Configuration Manager'</li> <li>• From the first toolbar icon drop-down, select 'Create Artifact' and create the</li> </ul>

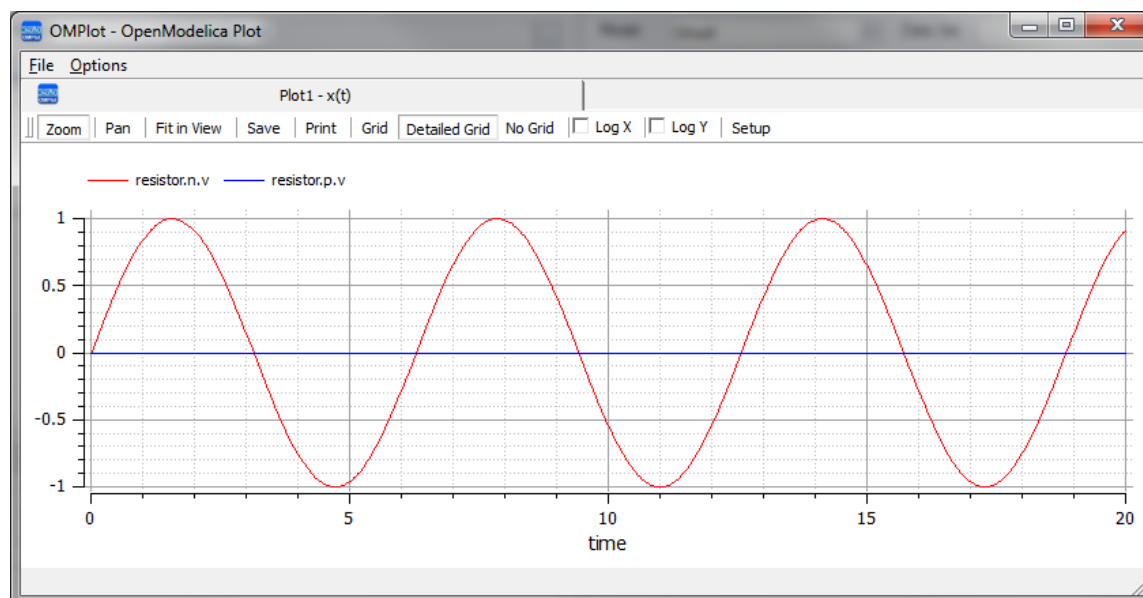
	<p>Artifact element</p> <ul style="list-style-type: none"> <li>• Select the Package that owns this SysML Model</li> </ul>
Create Root elements in Configuration Manager	<ul style="list-style-type: none"> <li>• ValueType</li> <li>• block</li> <li>• constraintBlock</li> </ul>
ValueType Substitution	Expand ValueType and for each of Current, Resistance and Voltage select 'SysMLSimReal' from the 'Value' combo box.
Set property as flow	<ul style="list-style-type: none"> <li>• Expand 'block' to ChargePort   FlowProperty   i : Current and select 'SimVariable' from the 'Value' combo box</li> <li>• For 'SysMLSimConfiguration' click on the  button to open the 'Element Configurations' dialog</li> <li>• Set 'isConserved' to 'True'</li> </ul>
SysMLSimModel	This is the model we want to simulate: set the Block 'Circuit' to be 'SysMLSimModel'.

## Run Simulation

In the 'Simulation' page, select the checkboxes against 'resistor.n.v' and 'resistor.p.v' for plotting and click on the Solve button.



The two legends 'resistor.n.v' and 'resistor.p.v' are plotted, as shown.



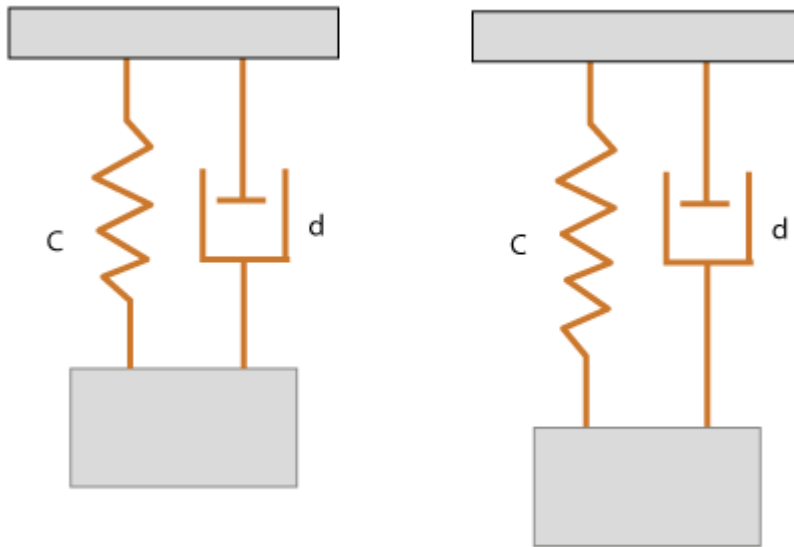
# Mass-Spring-Damper Oscillator

## Simulation Example

In this section, we will walk through the creation of a SysML parametric model for a simple Oscillator composed of a mass, a spring and a damper, and then use a parametric simulation to predict and chart the behavior of this mechanical system. Finally, we perform what-if analysis by comparing two oscillators provided with different parameter values through data sets.

### System being modeled

A mass is hanging on a spring and damper. The first state shown here represents the initial point at  $\text{time}=0$ , just when the mass is released. The second state represents the final point when the body is at rest and the spring forces are in equilibrium with gravity.

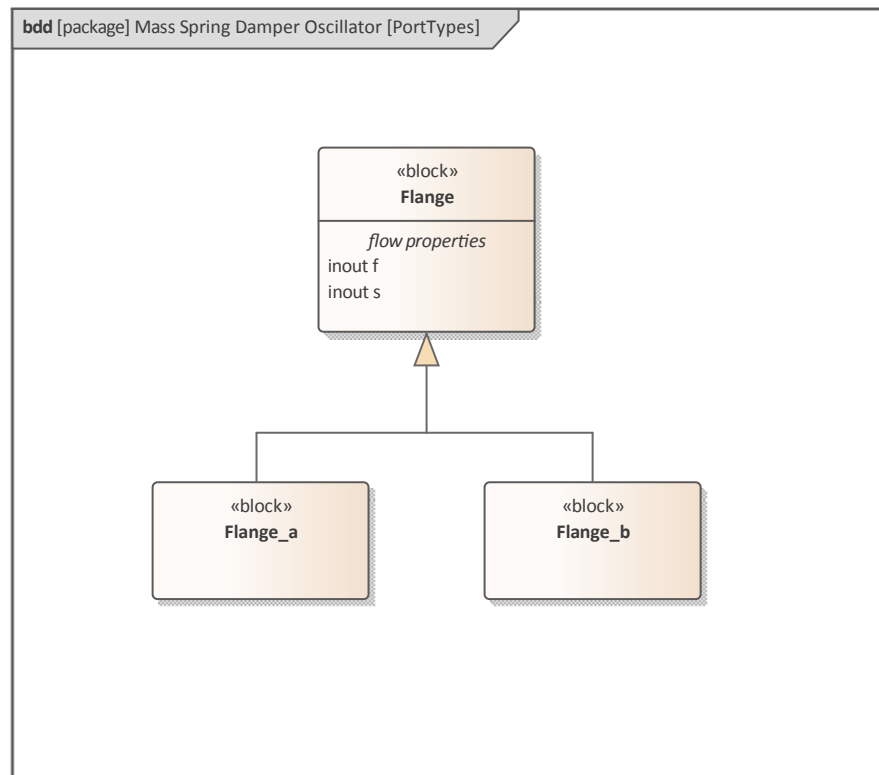


## Create SysML Model

The MassSpringDamperOscillator model in SysML has a main Block, the *Oscillator*. The Oscillator has four parts: a fixed *ceiling*, a *spring*, a *damper* and a *mass body*. Create a Block for each of these parts. The four parts of the Oscillator Block are connected through Ports, which represent mechanical flanges.

Components	Description
Port Types	The Blocks 'Flange_a' and 'Flange_b' used for flanges in the 1D transitional mechanical domain are identical but have slightly different roles, somewhat analogous to the roles of PositivePin and NegativePin in the electrical domain. Momentum is transmitted through the

flanges. So the attribute *isConserved* of flow property *Flange.f* should be set to True.



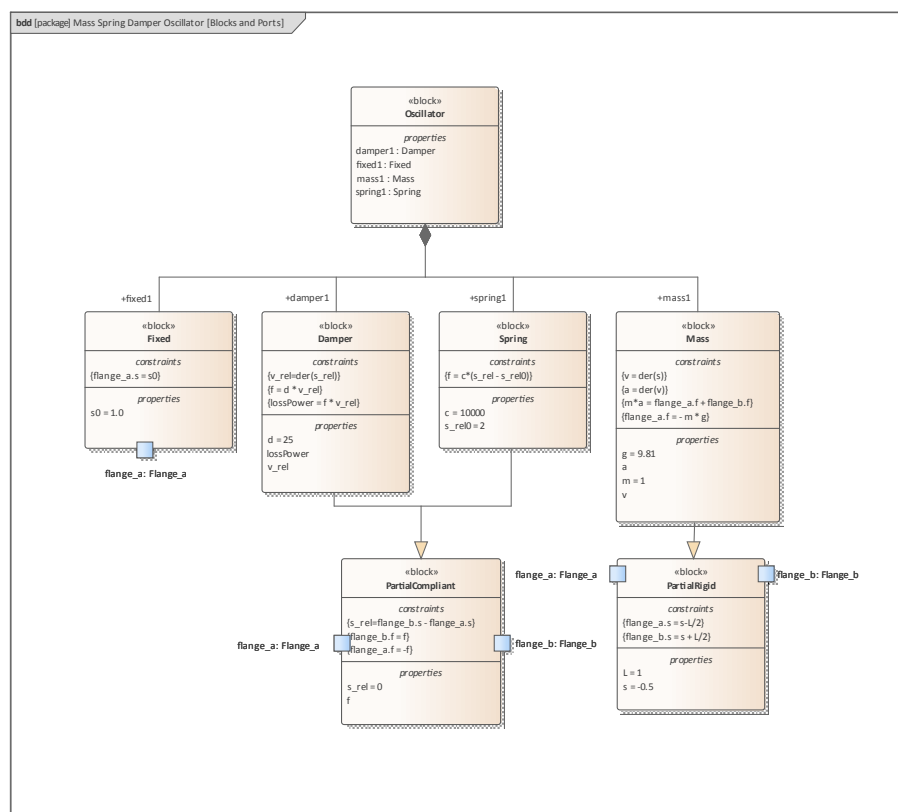
## Blocks and Ports

- Create Blocks 'Spring', 'Damper', 'Mass' and 'Fixed' to represent the spring, damper, mass body and ceiling respectively
- Create a Block 'PartialCompliant' with two Ports (flanges), named 'flange\_a' and 'flange\_b' — these are of type Flange\_a and Flange\_b respectively; the 'Spring' and 'Damper' Blocks generalize from 'PartialCompliant'
- Create a Block 'PartialRigid' with two



Ports (flanges), named 'flange\_a' and 'flange\_b' — these are of type Flange\_a and Flange\_b respectively; the 'Mass' Block generalizes from 'PartialRigid'

- Create a Block 'Fixed' with only one flange for the ceiling, which only has the Port 'flange\_a' typed to Flange\_a

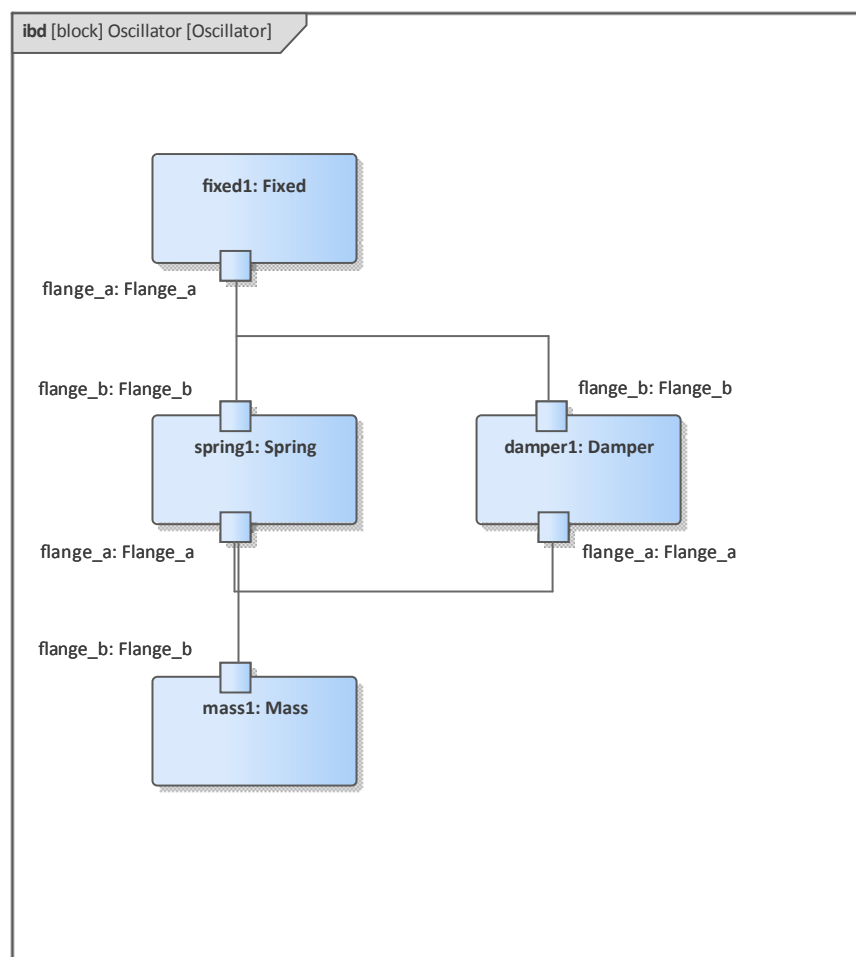


Internal structure

Create an Internal Block diagram (IBD) for 'Oscillator'. Add properties for the fixed ceiling, spring, damper and mass body, typed by the corresponding Blocks. Connect the Ports with connectors.

- Connect 'flange\_a' of 'fixed1' to

- 'flange\_b' of 'spring1'
- Connect 'flange\_b' of 'damper1' to 'flange\_b' of 'spring1'
- Connect 'flange\_a' of 'damper1' to 'flange\_a' of 'spring1'
- Connect 'flange\_a' of 'spring1' to 'flange\_b' of 'mass1'



## Constraints

For simplicity, we define the constraints directly in the Block elements; optionally you can define Constraint Blocks, use constraint properties in the Blocks, and

	bind their parameters to the Block's properties.
--	--

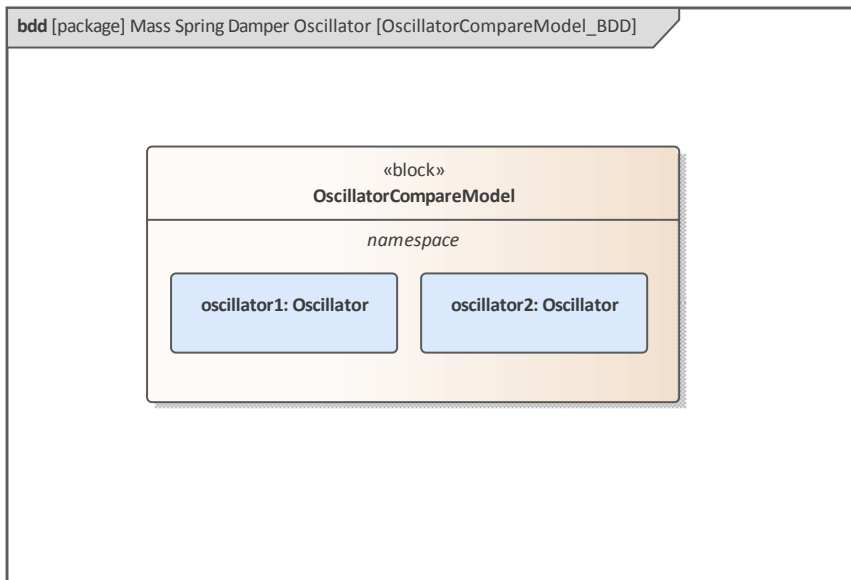
## Two Oscillator Compare Plan

After we model the Oscillator, we want to do some what-if analysis. For example:

- What is the difference between two oscillators with different dampers?
- What if there is no damper?
- What is the difference between two oscillators with different springs?
- What is the difference between two oscillators with different masses?

Here are the steps for creating a comparison model:

- Create a Block named 'OscillatorCompareModel'
- Create two Properties for 'OscillatorCompareModel', called *oscillator1* and *oscillator2*, and type them with the Block *Oscillator*



## Setup DataSet and Run Simulation

Create a SysMLSim Configuration Artifact and assign it to this Package. Then create these data sets:

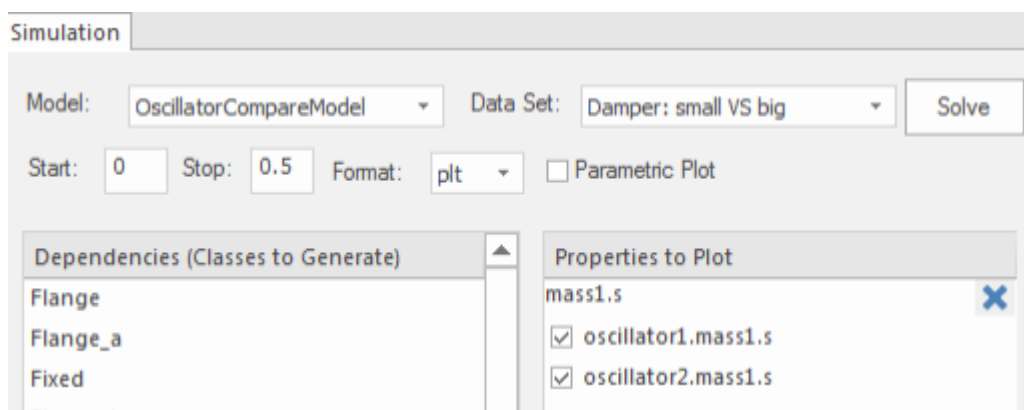
- Damper: small VS big  
provide 'oscillator1.damper1.d' with the value 10 and 'oscillator2.damper1.d' with the larger value 20
- Damper: no vs yes  
provide 'oscillator1.damper1.d' with the value 0;  
( 'oscillator2.damper1.d' will use the default value 25)
- Spring: small vs big  
provide 'oscillator1.spring1.c' with the value 6000  
and 'oscillator2.spring1.c' with the larger value 12000
- Mass: light vs heavy  
provide 'oscillator1.mass1.m' with the value 0.5 and  
'oscillator2.mass1.m' with the larger value 2

## The configured page resembles this:

OscillatorCompareModel	SysMLSimModel
Part	
Damper: small VS big oscillator2.damper1.d oscillator1.damper1.d	Click button to configure... 20 10
Spring: small VS big oscillator2.spring1.c oscillator1.spring1.c	Click button to configure... 12000 6000
Damper: no VS yes oscillator1.damper1.d	Click button to configure... 0
Mass: light VS Heavy oscillator2.mass1.m oscillator1.mass1.m	Click button to configure... 2 0.5

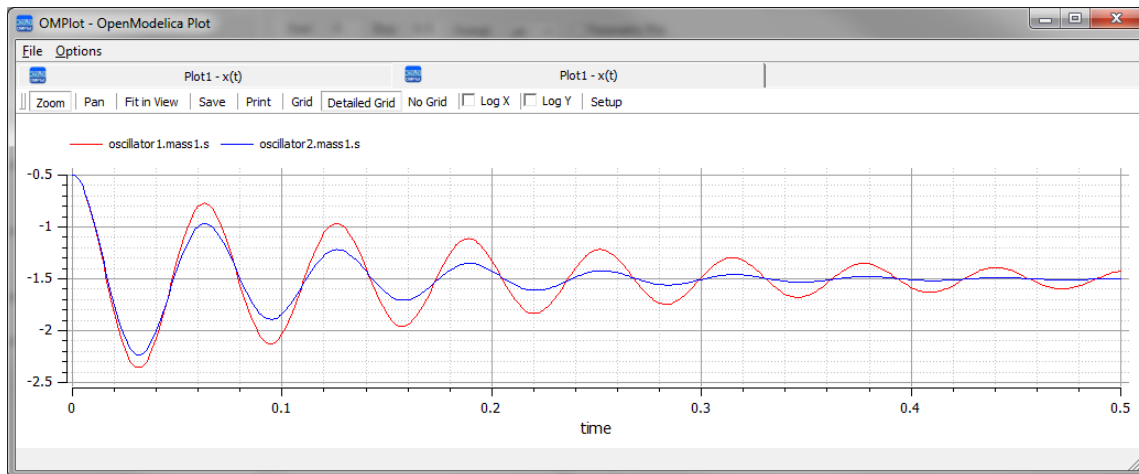
On the 'Simulation' page, select 'OscillatorCompareModel', plot for 'oscillator1.mass1.s' and 'oscillator2.mass1.s', then choose one of the created datasets and run the simulation.

*Tip: If there are too many properties in the plot list, you can toggle the Filter bar using the context menu on the list header, then type in 'mass1.s' in this example.*

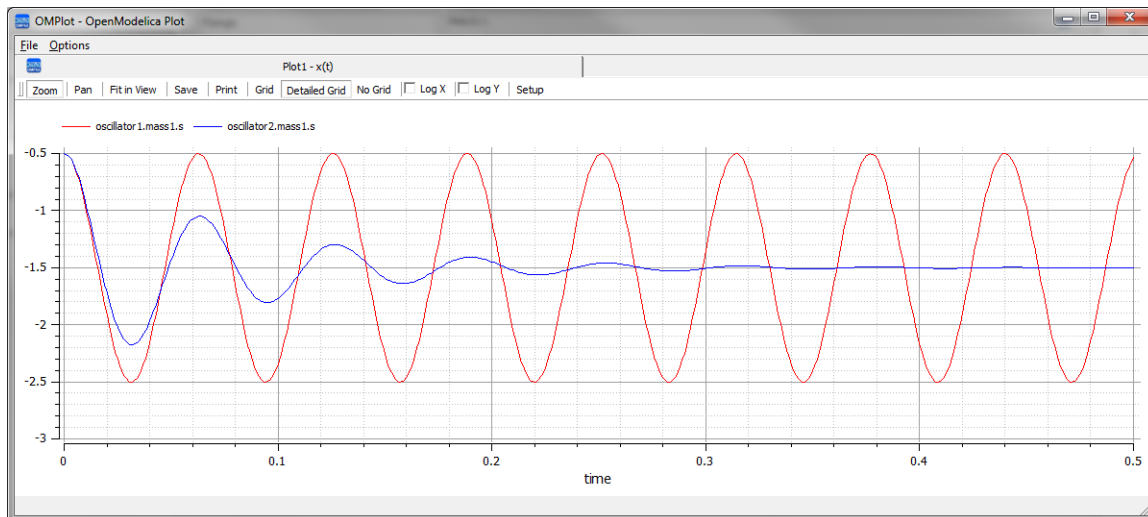


## These are the simulation results:

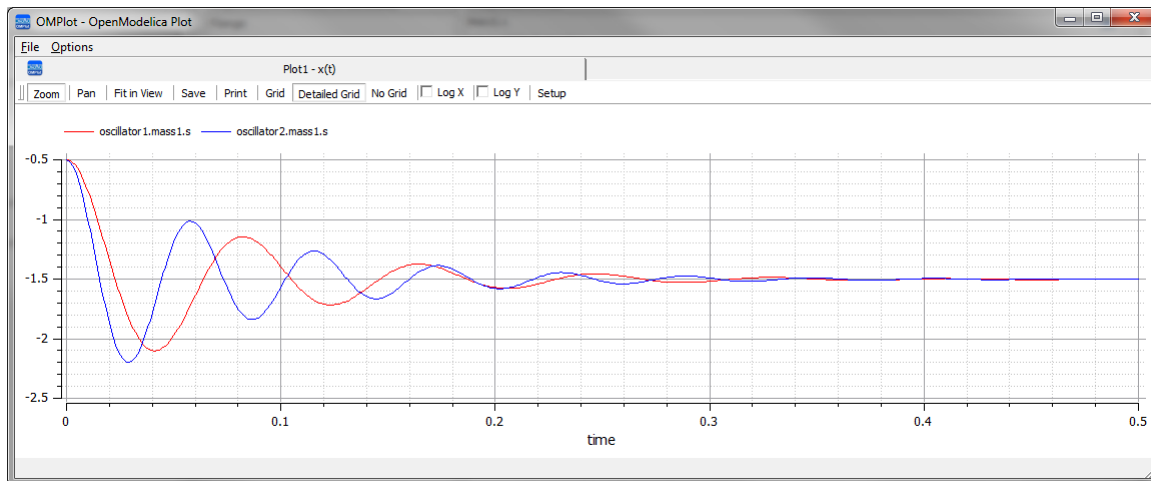
- Damper, small vs big: the smaller damper makes the body oscillate more



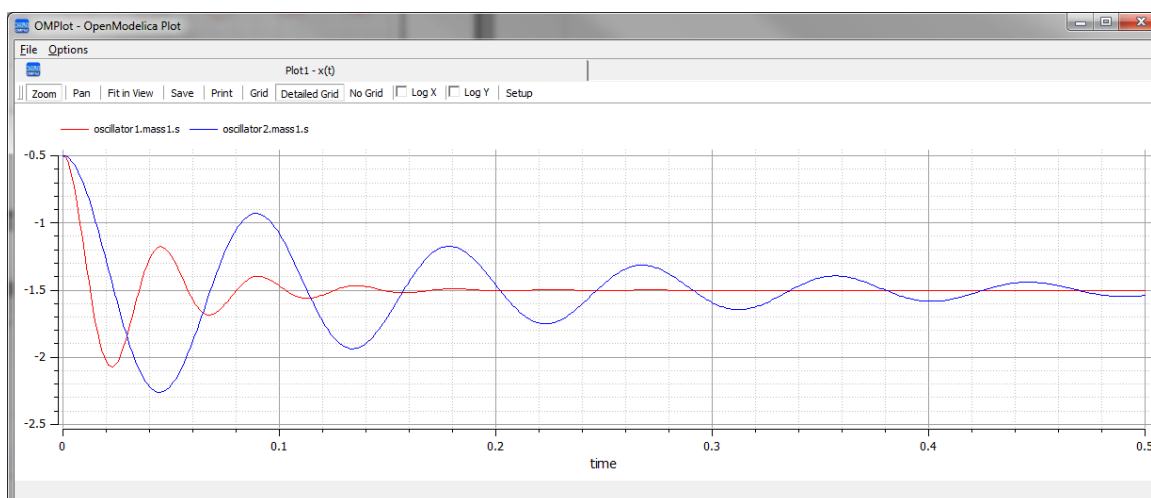
- Damper, no vs yes: the oscillator never stops without a damper



- Spring, small vs big: the spring with smaller 'c' will oscillate more slowly



- Mass, light vs heavy: the object with smaller mass will oscillate faster and regulate quicker



# Water Tank Pressure Regulator

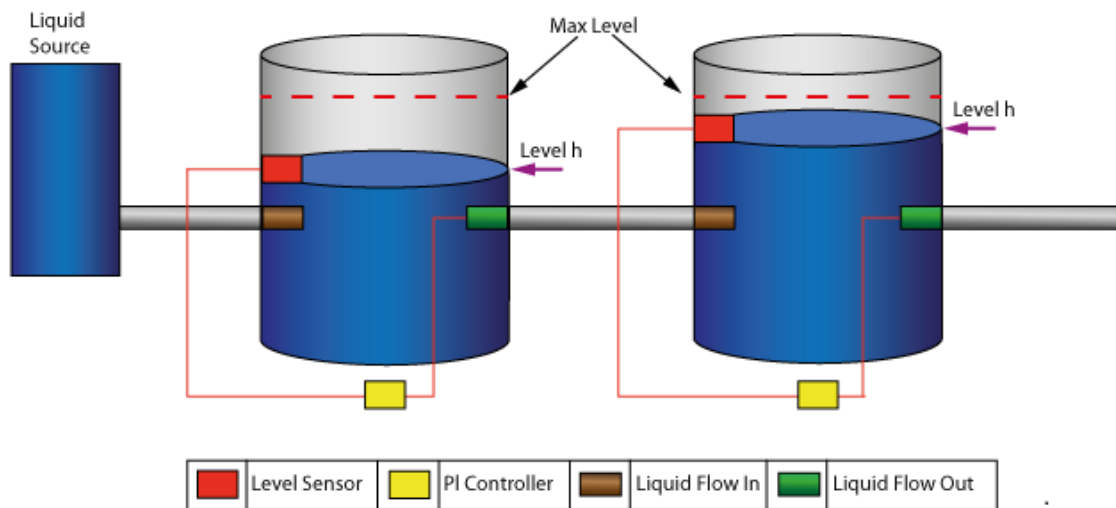
In this section we will walk through the creation of a SysML parametric model for a Water Tank Pressure Regulator, composed of two connected tanks, a source of water and two controllers, each of which monitors the water level and controls the valve to regulate the system.

We will explain the SysML model, create it and set up the SysMLSim Configurations. We will then run the simulation with OpenModelica.

## System being modeled

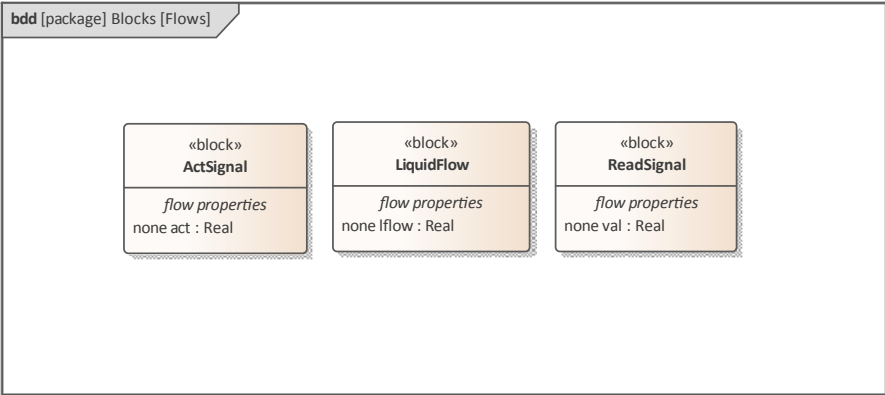
This diagram depicts two tanks connected together, and a water source that fills the first tank. Each tank has a proportional–integral (PI) continuous controller connected to it, which regulates the level of water contained in the tanks at a reference level. While the source fills the first tank with water, the PI continuous controller regulates the outflow from the tank depending on its actual level. Water from the first tank flows into the second tank, which the PI continuous controller also tries to regulate. This is a natural and non domain-specific physical problem.





## Create SysML Model

Component	Discussion
Port Types	<p>The tank has four ports; they are typed to these three blocks:</p> <ul style="list-style-type: none"> <li>• ReadSignal: Reading the fluid level; this has a property 'val' with unit 'm'</li> <li>• ActSignal: The signal to the actuator for setting valve position</li> <li>• LiquidFlow: The liquid flow at inlets or outlets; this has a property 'lflow' with unit 'm<sup>3</sup>/s'</li> </ul>

	 <pre> classDiagram     class ActSignal {         &lt;&lt;block&gt;&gt;         flow properties         none act : Real     }     class LiquidFlow {         &lt;&lt;block&gt;&gt;         flow properties         none lflow : Real     }     class ReadSignal {         &lt;&lt;block&gt;&gt;         flow properties         none val : Real     } </pre>
<p>Block Definition Diagram</p>	<p>LiquidSource: The water entering the tank must come from somewhere, therefore we have a liquid source component in the tank system, with the property <i>flowLevel</i> having a unit of 'm<sup>3</sup>/s'. A Port 'qOut' is typed to 'LiquidFlow'.</p> <p>Tank: The tanks are connected to controllers and liquid sources through Ports.</p> <ul style="list-style-type: none"> <li>• Each Tank has four Ports:             <ul style="list-style-type: none"> <li>- qIn: for input flow</li> <li>- qOut: for output flow</li> <li>- tSensor: for providing fluid level measurements</li> <li>- tActuator: for setting the position of the valve at the outlet of the tank</li> </ul> </li> <li>• Properties:             <ul style="list-style-type: none"> <li>- area (unit='m<sup>2</sup>'): area of the tank, involved in the <i>mass balance</i> equation</li> <li>- h (unit = 'm'): water level,</li> </ul> </li> </ul>

involved in the *mass balance* equation; its value is read by the sensor

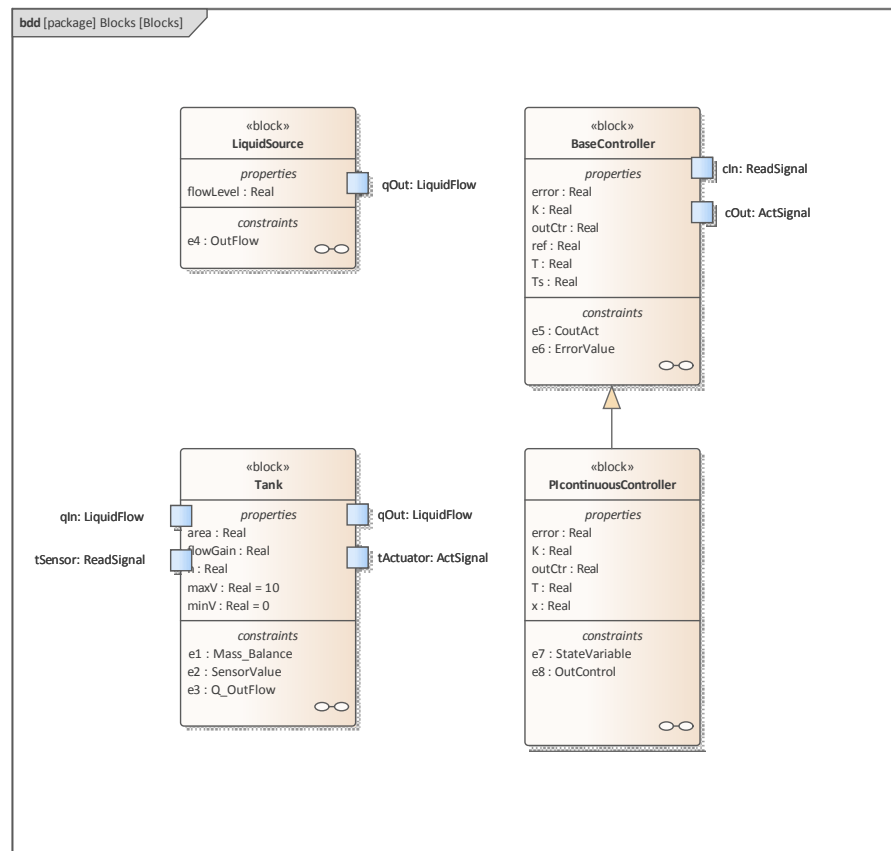
- flowGain (unit = 'm<sup>2</sup>/s'): the output flow is related to the valve position by *flowGain*
- minV, maxV: Limits for output valve flow

BaseController: This Block could be super of a PI Continuous Controller and PI Discrete Controller.

- Ports:
  - cIn: Input sensor level
  - cOut: Control to actuator
- Properties:
  - Ts (unit = 's'): Time period between discrete samples (not used in this example)
  - K: Gain factor
  - T (unit = 's'): Time constant of controller
  - ref: reference level
  - error: difference between the reference level and the actual level of water, obtained from the sensor
  - outCtr: control signal to the actuator for controlling the valve position

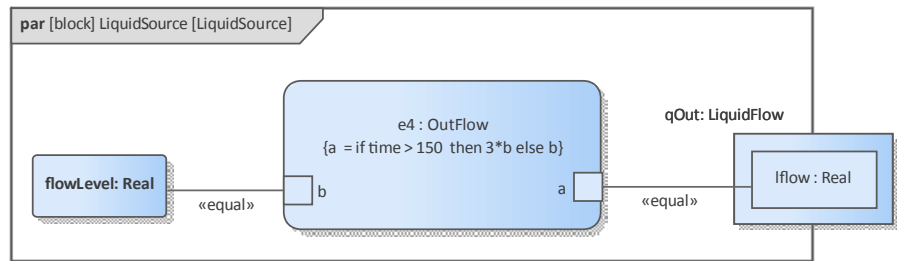
## PIcontinuousController: generalize from BaseController

- Properties:
  - x: the controller state variable



### Constraint Blocks

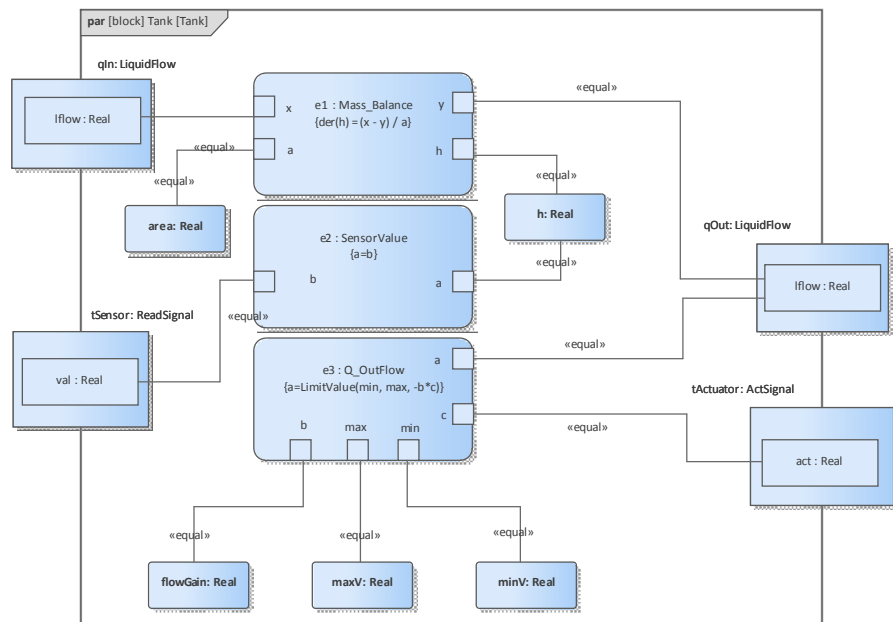
The flow increases sharply at time=150 to a factor of three of the previous flow level, which creates an interesting control problem that the controller of the tank has to handle.



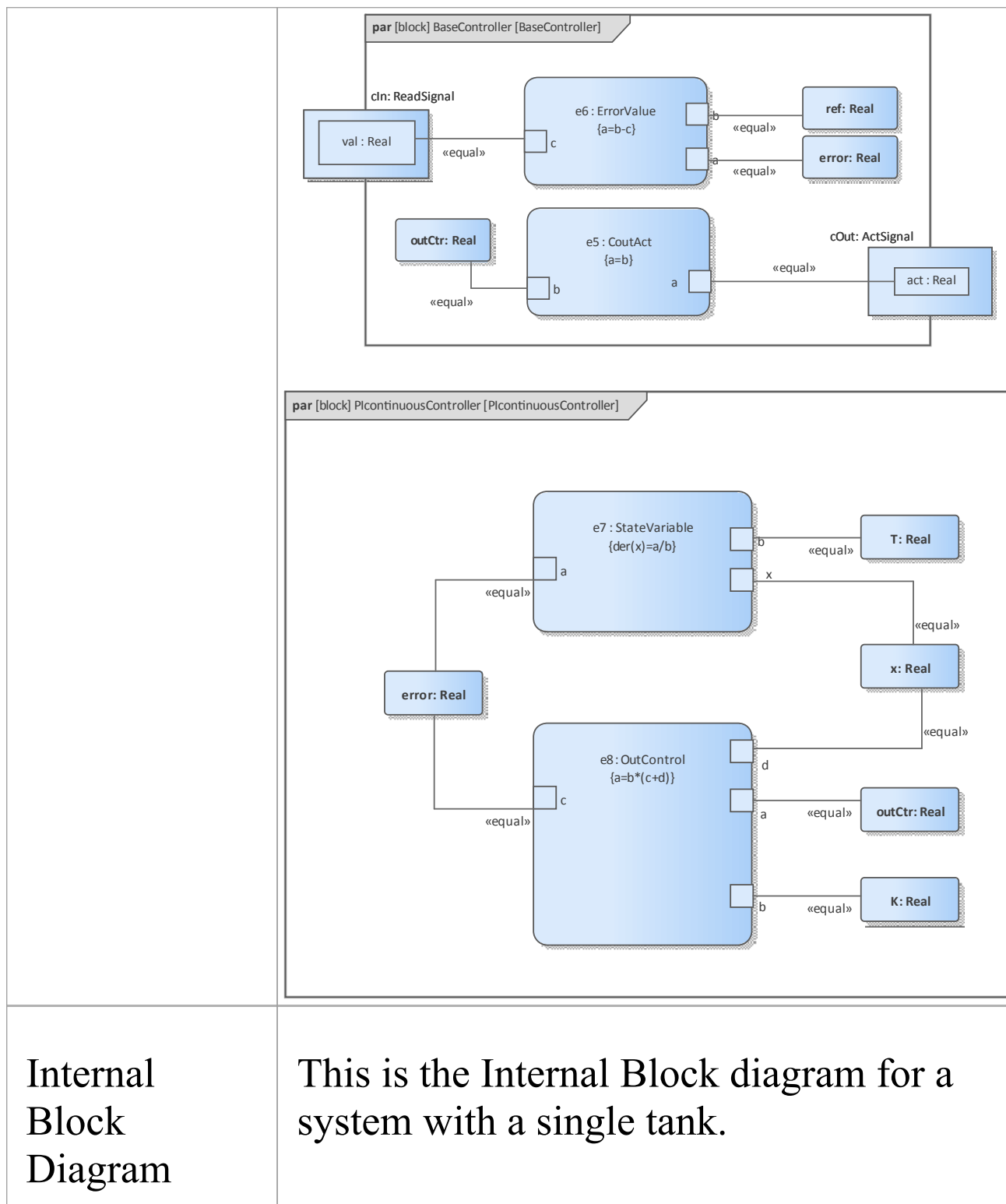
The central equation regulating the behavior of the tank is the *mass balance* equation.

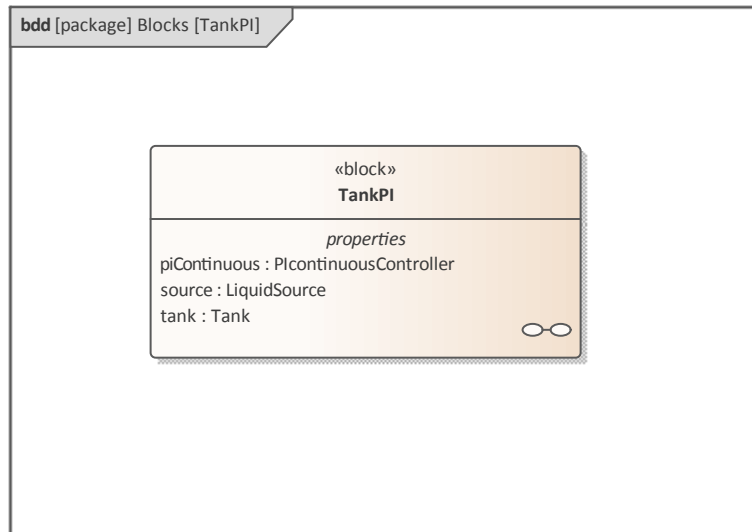
The output flow is related to the valve position by a 'flowGain' parameter.

The sensor simply reads the level of the tank.

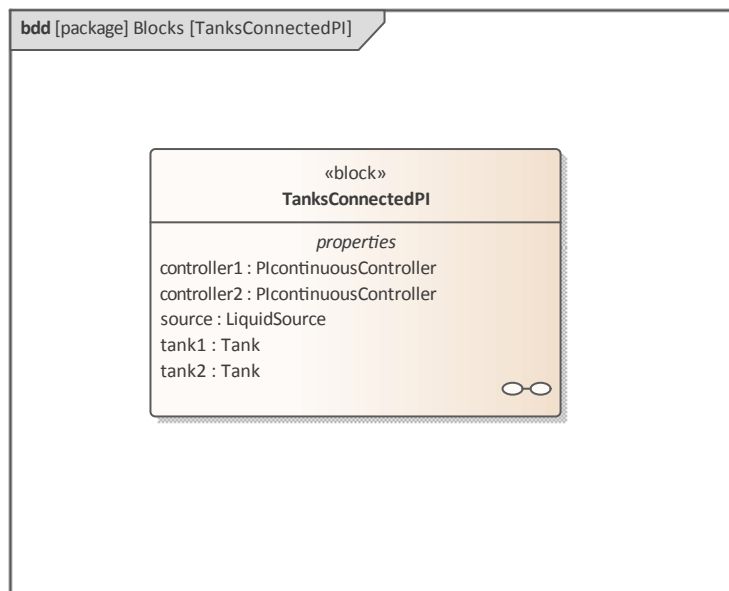


The Constraints defined for 'BaseController' and 'PIcontinuousController' are illustrated in these figures.





This is the Internal Block diagram for a system with two connected tanks.



## Run Simulation

Since *TankPI* and *TanksConnectedPI* are defined as 'SysMLSimModel', they will be filled in the combo box of 'Model' on the 'Simulation' page.

Select *TanksConnectedPI*, and observe these GUI changes happening:

- 'Data Set' combobox: will be filled with all the data sets defined in *TanksConnectedPI*
- 'Dependencies' list: will automatically collect all the Blocks, Constraints, SimFunctions and ValueTypes directly or indirectly referenced by *TanksConnectedPI* (these elements will be generated as Modelica code)
- 'Properties to Plot': a long list of 'leaf' variable properties (that is, they don't have properties) will be collected; you can choose one or multiple to simulate, and they will become legends of the plot

## Create Artifact and Configure

Select 'Simulate > System Behavior > Modelica > SysMLSim Configuration Manager'

The elements in the Package will be loaded into the Configuration Manager.

Configure these Blocks and their properties as shown in this table.

Note: Properties not configured as 'SimConstant' are



'SimVariable' by default.

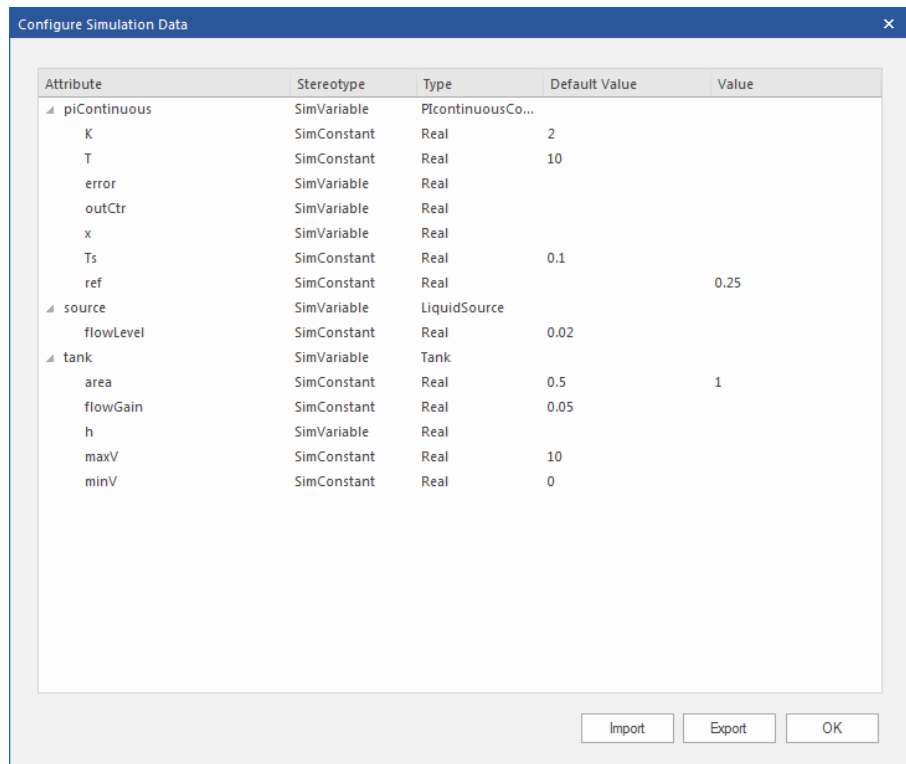
Block	Properties
LiquidSource	<p>Configure as 'SysMLSimClass'.</p> <p>Properties configuration:</p> <ul style="list-style-type: none"> <li>• flowLevel: set as 'SimConstant'</li> </ul>
Tank	<p>Configure as 'SysMLSimClass'.</p> <p>Properties configuration:</p> <ul style="list-style-type: none"> <li>• area: set as 'SimConstant'</li> <li>• flowGain: set as 'SimConstant'</li> <li>• maxV: set as 'SimConstant'</li> <li>• minV: set as 'SimConstant'</li> </ul>
BaseController	<p>Configure as 'SysMLSimClass'.</p> <p>Properties configuration:</p> <ul style="list-style-type: none"> <li>• K: set as 'SimConstant'</li> <li>• T: set as 'SimConstant'</li> <li>• Ts: set as 'SimConstant'</li> <li>• ref: set as 'SimConstant'</li> </ul>
PIcontinuous Controller	<p>Configure as 'SysMLSimClass'.</p>

TankPI	Configure as 'SysMLSimModel'.
TanksConnectedPI	Configure as 'SysMLSimModel'.

## Setup DataSet

Right-click on each element, select the 'Create Simulation Dataset' option, and configure the datasets as shown in this table.

Element	Dataset
LiquidSource	flowLevel: 0.02
Tank	h.start: 0 flowGain: 0.05 area: 0.5 maxV: 10 minV: 0
BaseController	T: 10 K: 2 Ts: 0.1

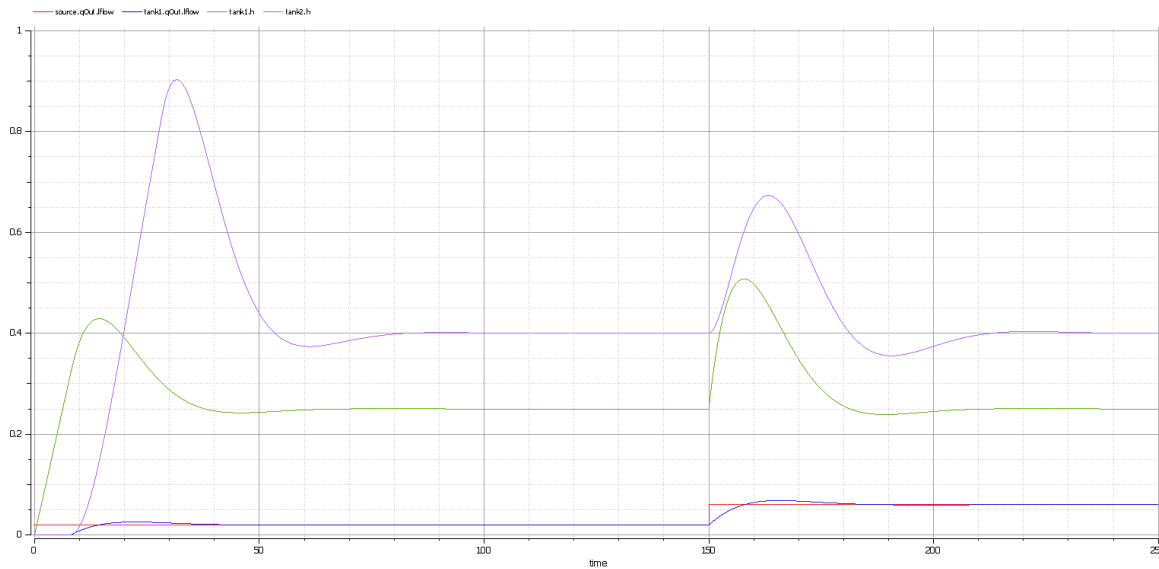
PIcontinuous Controller	<p>No configuration needed.</p> <p>By default, the specific Block will use the configured values from super Block's default dataSet.</p>
TankPI	<p>What is interesting here is that the default value could be loaded in the 'Configure Simulation Data' dialog. For example, the values we configured as the default dataSet on each Block element were loaded as default values for the properties of TankPI. Click the icon on each row to expand the property's internal structures to arbitrary depth.</p> <div></div>

	<p>Click on the OK button and return to the Configuration Manager. Then these values are configured:</p> <ul style="list-style-type: none"><li>• tank.area: 1 this overrides the default value 0.5 defined in the Tank Block's data set</li><li>• piContinuous.ref: 0.25</li></ul>
TanksConnectedPI	<ul style="list-style-type: none"><li>• controller1.ref: 0.25</li><li>• controller2.ref: 0.4</li></ul>

## Simulation and Analysis 1

Select these variables and click on the Solve button. This plot should prompt:

- source.qOut.lflow
- tank1.qOut.lflow
- tank1.h
- tank2.h



## Here are the analyses of the result:

- The liquid flow increases sharply at time=150, to 0.06 m<sup>3</sup>/s, a factor of three of the previous flow level (0.02 m<sup>3</sup>/s)
- Tank1 regulated at height 0.25 and tank2 regulated at height 0.4 as expected (we set the parameter value through the data set)
- Both tank1 and tank2 regulated twice during the simulation; the first time regulated with the flow level 0.02 m<sup>3</sup>/s; the second time regulated with the flow level 0.06 m<sup>3</sup>/s
- Tank2 was empty before flow came out from tank1

## Simulation and Analysis 2

We have set the tank's properties 'minV' and 'maxV' to

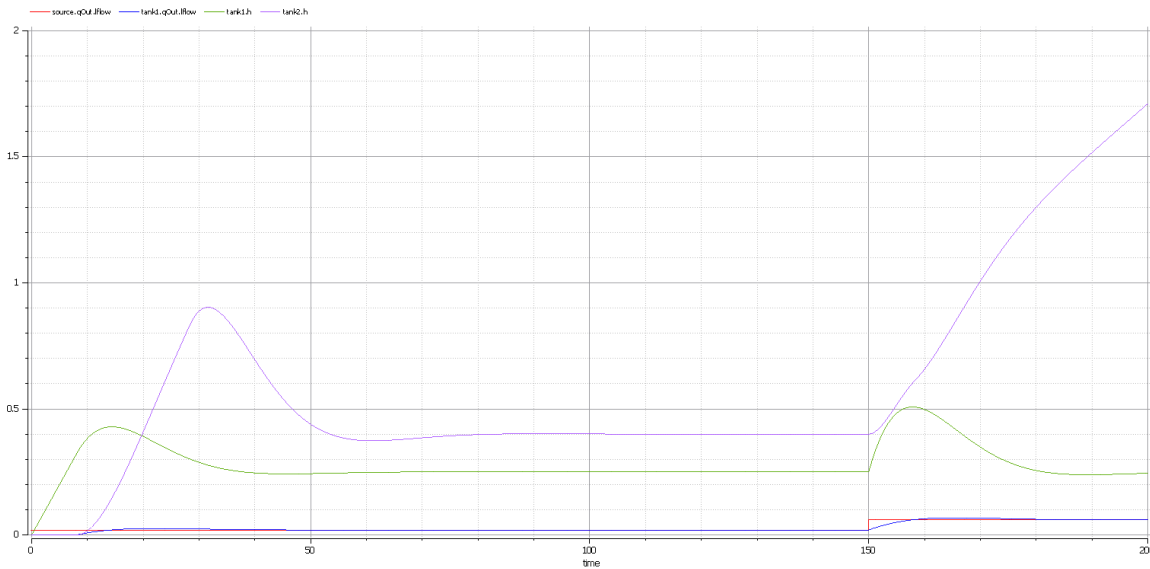
values 0 and 10, respectively, in the example.

In the real world, a flow speed of 10 m<sup>3</sup>/s would require a very big valve to be installed on the tank.

What would happen if we changed the value of 'maxV' to 0.05 m<sup>3</sup>/s ?

Based on the previous model, we might make these changes:

- On the existing 'DataSet\_1' of TanksConnectedPI, right-click and select 'Duplicate DataSet', and re-name to 'Tank2WithLimitValveSize'
- Click on the button to configure, expand 'tank2' and type '0.05' in the 'Value' column for the property 'maxV'
- Select 'Tank2WithLimitValveSize' on the 'Simulation' page and plot for the properties
- Click on the Solve button to execute the simulation



### Here are the analyses of the result:

- Our change only applies to tank2; tank1 can regulate as before on 0.02 m<sup>3</sup>/s and 0.06 m<sup>3</sup>/s
- When the source flow is 0.02 m<sup>3</sup>/s, tank2 can regulate as before
- However, when the source flow increases to 0.06 m<sup>3</sup>/s, the valve is too small to let the out flow match the in flow; the only result is that the water level of tank2 increases
- It is then up to the user to fix this problem; for example, change to a larger valve, reduce the source flow or make an extra valve

In summary, this example shows how to tune the parameter values by duplicating an existing DataSet.

# Troubleshooting OpenModelica Simulation

## Common Simulation Issues

This table describes some common issues that can prevent a model being simulated. Check the output in the 'Build' tab of the System Output window. The messages are dumped from the OpenModelica compiler (omc.exe), which normally points you to the lines of the Modelica source code. This will help you locate most of the errors.

Issue
The number of equations is less than the number of variables. You might have forgotten to set some properties to 'SimConstant', which means the value doesn't change during simulation. You might have to provide the 'SimConstant' property values before the simulation is started. (Set the values through a Simulation Data Set.)
The Blocks that are typing to Ports might not contain conserved properties. For example, a Block 'ChargePort' contains two parts — 'v : Voltage' and 'i: Current'. The property 'i : Current' should be defined as SimVariable with the attribute 'isConserved' set to 'True'.



SimConstants might not have default values — they should be provided with them.

A SimVariable might not have an initial value to start with — one should be provided.

The properties might be typed by elements (Blocks or Value Type) external to the configured Package; use a Package Import connector to fix this.

## SysML Simulation Configuration Filters

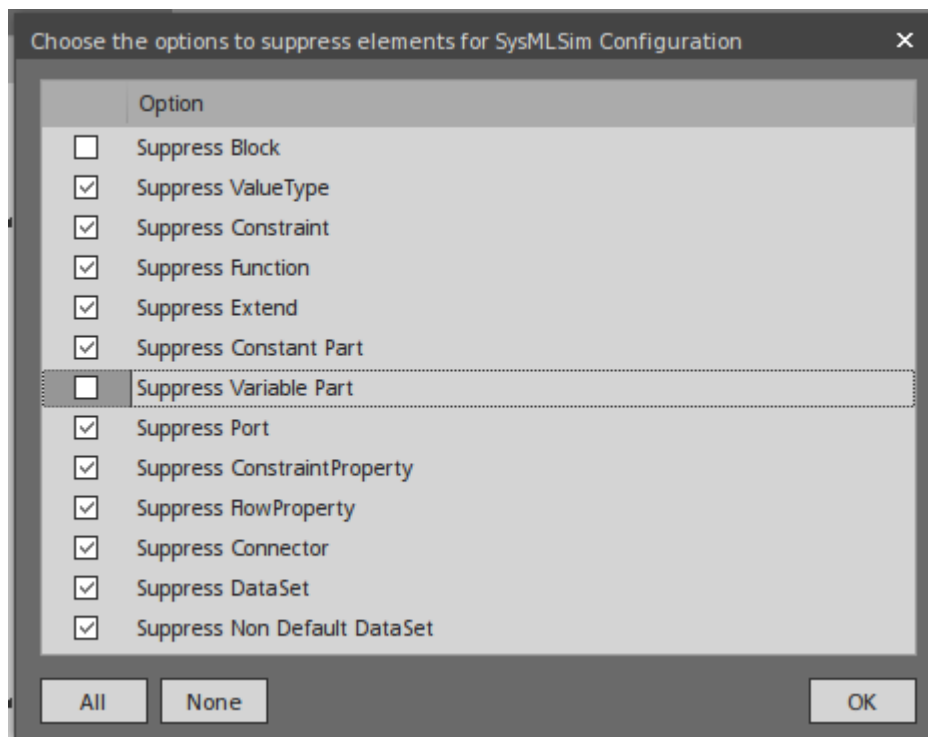
The 'SysML Simulation Configuration' dialog shows all the elements in the Package by default, including Value Types, Blocks, Constraint Blocks, Parts and Ports, Constraint Properties, Connectors, Constraints and Data Sets. For a medium-sized model, the full list can be quite long and it can be difficult for the user to find a potential modeling error.

In the TwoTanks example, if we clear the Tank.area property 'SimConstant' and then do a validation, we will find this error:

*Error: Too few equations, under-determined system. The model has 11 equation(s) and 13 variable(s).*

This error indicates that we might have forgotten to set some properties to 'SimConstant'.

What we can do now is click on the second button from the right on the toolbar (Filter for the configuration) and open the dialog shown here. Click on the All button, then deselect the 'Suppress Block' and 'Suppress Variable Part' checkboxes and click on the OK button.



Now we will have a much shorter list of variables, from which we can find that 'area' does not change during simulation. Then we define this as a 'SimConstant' and provide an initial value to fix the issue.

## Model Validation Examples

Message	Discussion
---------	------------

## Variable not defined in Constraint

In the TwoTanks example, when we browse to 'constraintBlock.Outcontrol.Constraint', suppose we find a typing error: we typed 'v' instead of 'b' in the constraint.

So, instead of:

$$a=b*(c+d)$$

We typed:

$$a=v*(c+d)$$

Click on the Validate button on the toolbar. These error messages will appear in the 'Modelica' tab:

*Validating model...*

*Error: Variable v not found in scope OutControl. (Expression: "a=v\*(c+d);")*

*Error: Error occurred while flattening model TanksConnectedPI*

*Number of Errors and Warnings found: 2*

Double-click on the error line; the configuration list displays with the constraint highlighted.

Change 'v' back to 'b' and click on the Validate button again. No errors should be found and the issue is fixed.

	<p><i>Tips:</i> Using the SysML Simulation Configuration view is a shortcut way of changing the constraints for a Block or Constraint Block. You can:</p> <ul style="list-style-type: none"> <li>• Change a constraint in place</li> <li>• Delete using the context menu of a constraint</li> <li>• Add a new constraint using the context menu of a Block or Constraint Block</li> </ul>
Duplicate Variable Names	<p>In the TwoTanks example, browse to <i>block.tank.constraintProperty.e1</i>. Suppose we gave two properties the same name:</p> <ul style="list-style-type: none"> <li>• Right-click on <i>e1</i>, select 'Find in Project Browser', and change the name to <i>e2</i>; reload the 'SysML Simulation Configuration' dialog</li> </ul> <p>Click on the Validate button on the toolbar; these error messages appear in the 'Modelica' tab:</p> <p><i>Validating model...</i></p> <p><i>Error: Duplicate elements (due to inherited elements) not identical: (Expression: "SensorValue e2;")</i></p> <p><i>Error: Error occurred while flattening model TanksConnectedPI</i></p> <p><i>Number of Errors and Warnings</i></p>

	<p><i>found: 2</i></p> <p>Double-click on the error line; the configuration list displays with the constraint properties highlighted.</p> <p>Change the name of one of them from <i>e2</i> back to <i>e1</i> and click on the Validate button again; no errors should be found and the issue is fixed.</p>
Properties defined in Constraint Blocks not used	<p>In the TwoTanks example, in the Browser window, we browse to the element 'Example Model.Systems Engineering.ModelicaExamples.TwoTanks.constraints.OutFlow'.</p> <p>Suppose we add a property '<i>c</i>' and potentially a new constraint, but we forget to synchronize for the instances — the constraint properties. This will cause a <i>Too few equations, under-determined system</i> error if we don't run validation.</p> <p>Reload the Package in the 'SysML Simulation Configuration' dialog and click on the Validate button on the toolbar. These error messages will appear in the 'Modelica' tab:</p> <p><i>Validating model...</i></p> <p><i>Error: ConstraintProperty 'e4' is</i></p>

*missing parameters defined in the typing ConstraintBlock 'OutFlow'. (Missing: c)*

*Error: Too few equations, under-determined system. The model has 11 equation(s) and 12 variable(s).*

*Number of Errors and Warnings found: 2*

Double-click on the error line; the configuration list displays with the constraint property highlighted. The constraint property is typed to *outFlow* and the new parameter '*c*' is missing.

Right-click on the constraint property in the configuration list, select 'Find in all Diagrams', then right-click on the Constraint property on the diagram and select 'Features | Parts / Properties' and select the 'Show Owned / Inherited' checkbox, then click on '*c*'.

Reload the model in the 'SysML Simulation Configuration' dialog and click on the Validate button. These error messages will appear in the 'Modelica' tab:

*Validating model...*

*Error: ConstraintProperty 'e4' does not have any incoming or outgoing*

*binding connectors for parameter 'c'.*

*Error: Too few equations,  
under-determined system. The model has  
11 equation(s) and 12 variable(s).*

*Number of Errors and Warnings  
found: 2*

In order to fix this issue, we can do either  
of two things based on the real logic:

1. If the property 'c' is necessary in the  
Constraint Block and a constraint is  
defined by using 'c', then we need to  
add a property in the context of the  
constraint property and bind to the  
parameter 'c'.
2. If the property 'c' is not required, then  
we can click on this property in the  
Constraint Block and press Ctrl+D.  
(The corresponding constraint  
properties will have 'c' deleted  
automatically.)

